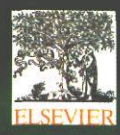
Лилиан Хоббс
Сьюзан Хилсон
Шилпа Лоуенд



Oracle9i R2

РАЗРАБОТКА и ЭКСПЛУАТАЦИЯ ХРАНИЛИЩ БАЗ ДАННЫХ

вступительное слово
Чак Розуот
исполнительный вице-президент Server Technologies
Oracle Corporation

КУДИЦ-ОБРАЗ

Oracle 9iR2 Data Warehousing

Lilian Hobbs
Susan Hilson
Shilpa Lawande

Digital Press
An imprint of Elsevier Science
Amsterdam • Boston • London • New York • Oxford • Paris • San Diego
San Fransisco • Singapore • Sydney • Tokyo

Симону, к сожалению ушедшему от нас
-Лилиан

Стейси и Эмили, моим золотым девочкам
-Сьюзан

Моим родителям и родителям моего мужа
-Шилпа

Лилиан Хобbс
Сьюзан Хилсон
Шилпа Лоуенд

Oracle 9i R2

РАЗРАБОТКА и ЭКСПЛУАТАЦИЯ ХРАНИЛИЩ БАЗДАННЫХ

ПРАКТИЧЕСКОЕ ПОСОБИЕ

вступительное слово
Чак Розуот
исполнительный вице-президент Server Technologies
Oracle Corporation

КУДИЦ-ОБРАЗ
Москва • 2004

ББК 32.973.26-018.2

Хоббс Л., Хилсон С., Лоуенд Ш.

Oracle9*i*R2: разработка и эксплуатация хранилищ баз данных / Пер. с англ. - М.: КУДИЦ-ОБРАЗ, 2004. - 592 с.

Книга посвящена теоретическим и практическим вопросам использования хранилищ данных (WareHouse). Проектирование хранилищ данных - один из сложных разделов программной инженерии. На него распространяются все современные методологии проектирования ПО, но их выполнение усложняется молодостью данного направления и очень острой нехваткой соответствующих специалистов. Рассмотрены особенности построения хранилищ на базе СУБД Oracle9*i*.

Книга будет полезна широкому кругу ИТ-специалистов, а также маркетологов, аналитиков и других специалистов, использующих в своей деятельности хранилища данных.

ISBN 5-9579-0036-2
ISBN 1-55558-287-7

Лилиан Хоббс, Сьюзан Хилсон, Шилпа Лоуенд / Oracle9*i*R2: разработка и эксплуатация хранилищ баз данных

Учебно-справочное издание

Перевод с англ. С. М. Лунин
Научный редактор Д.А. Бондарев
Макет В. Г. Клименко

Лицензия ЛР № 071806 от 02.03.99. «ИД КУДИЦ-ОБРАЗ».
119034, Москва, Гагаринский пер., д. 21, стр. 1. Тел.: 333-82-11, ok@kudits.ru

Подписано в печать 16.07.2004
Формат 70x90/16. Бум. газетная. Печать офсетная
Усл. печ. л. 43,3. Тираж 2000. Заказ 1559

Отпечатано в ОАО «Щербинская типография»
117623, г. Москва, ул. Типографская, д. 10

ISBN 5-9579-0036-2 © «ИД КУДИЦ-ОБРАЗ», 2004
ISBN 1-55558-287-7

This edition of Oracle9iR2 Data Warehousing by Lillian Hobbs, Susan Hillson and Shilpa Lawande is published by arrangement with Elsevier Ltd, The Boulevard, Langford Lane, Kidlington, OX5 1GB, England OR.

Настоящее издание книги Oracle9iR2 Data Warehousing авторов Lillian Hobbs, Susan Hillson and Shilpa Lawande публикуется по соглашению с Elsevier Ltd, The Boulevard, Langford Lane, Kidlington, OX5 1GB, England.

Русское издание опубликовано издательством КУДИЦ-ОБРАЗ, © 2004.

Вступительное слово

Три года назад, на заре революционного развития Интернет, я написал вступительное слово к первому изданию этой книги. Сейчас та инфраструктура, которую мы видели в зачаточном состоянии, прочно утвердилась и используется так, как мы и не могли и представить каких-то 36 месяцев назад.

«Умные» системы управления информацией необходимы для решения тех проблем, с которыми мы сталкиваемся в бизнесе и в обеспечении надежности и безопасности нашей повседневной жизни. Только системы хранения информации могут объединять информацию, чтобы выявить тенденции и понять, как максимально эффективно использовать ресурсы.

Возможность фиксировать информацию точно, практично и вовремя, интегрировать ее с другими данными, анализировать ее и доставлять через Интернет своим служащим, заказчикам, поставщикам и партнерам - вот ключ к успешному ведению бизнеса. Раньше системы хранения информации использовались в процессе принятия стратегических бизнес-решений. Сейчас они применяются при принятии тактических решений и в повседневной работе. Когда-то считалось, что систему хранения информации «было бы неплохо иметь», но организации двадцать первого века уже не могут эффективно работать без них.

Рост и размеров и интенсивности использования хранилищ данных все больше ускоряется.

Сегодня системы хранения данных Oracle часто содержат более терабайта данных. В течение следующих пяти лет некоторые организации могут перейти к управлению петабайтами данных. Интернет обеспечил мгновенный доступ к данным по требованию намного большему числу пользователей. Некоторые системы хранения информации Oracle уже сегодня имеют тысячи пользователей онлайн.

Дни, когда приходилось ждать отчета целую ночь, давно ушли, вместе с соответственно медленным принятием решений. При использовании Oracle 9i предварительное суммирование данных и сохранение их в материализованных представлениях (views), то есть те отчеты, выполнение которых ранее занимало 2 часа, теперь готовы уже через две минуты. Некоторые сайты Oracle обрабатывают более 150000 отчетов и запросов за день. С новыми аналитическими возможностями базы данных Oracle 9i, запросы OLAP могут выполняться непосредственно над теми данными, которые хранятся в хранилище Oracle.

Хранилища данных Oracle 9i теперь загружаются ежедневно, и даже несколько раз в день с помощью переносимых табличных областей, SQL*Loader, внешних таблиц и операций по обслуживанию разделов, а не путем периодических пакетных обновлений базы в конце каждой недели.

Системы хранения информации становятся источниками данных для критических приложений, таких как системы управления взаимодействием с заказчиками (Customer relationship management, CRM) и управления цепью поставок (Supply chain management, SCM). Также они все больше применяются для обнаружения несанкционированного доступа. Возможности информационных проходок (Data Mining), встроенные в базу данных Oracle, могут использоваться для того, чтобы обнаружить внезапное повышение активности учетной записи или изменение схемы работы.

Разрабатываете ли вы крупную систему хранения данных, небольшую лавку данных (data mart) или ищите информацию о том, как улучшить свою производительность с помощью Oracle 9i Release 2, в этой книге вы найдете советы и примеры, которые можно использовать.

Рекомендованные методы иллюстрируются в каждой главе множеством рабочих примеров. Эта книга, написанная Лиллиан Хоббс (Lillian Hobbs) Сьюзен Хилсон (Susan Hillson) и Шилпой Лоуэнд, членами команды разработчиков Oracle Server Technologies, подробно описывает свойства систем хранения информации Oracle 9i Release 2, средства Oracle 9i Application Server и Oracle 9i Developer Suite, используемые для создания, доступа и управления хранилищами информации.

Чак Розуот (Chuck Rozwat),
Исполнительный вице-президент
Server Technologies
Oracle Corporation

Введение

Были времена, когда компании редко имели систему хранения информации, но теперь они — необходимый источник сведений для принятия верных решений в бизнесе. Первоначально, хранилища информации могли создавать и обслуживать только крупные корпорации, однако последние достижения в области программного обеспечения баз данных и создания компьютеров означают, что теперь любая организация может создать хранилище информации или лавку данных (data mart).

Эта книга знакомит читателя с хранилищами информации и лавками данных при реализации их с помощью базы данных Oracle 9i Release 2 (9i2R). Эта книга создавалась так, чтобы каждый новичок в данном вопросе смог получить общее представление о хранении информации и обо всем том, что применяется при проектировании, создании, реализации и управлении данным решением в Oracle 9i2R. Структура глав книги следующая:

- В Главе 1 представлено введение в предмет хранения данных — знакомство с терминологией и обсуждение областей применения хранилищ информации и лавок данных.
- В Главе 2 показывается, как спроектировать хранилище информации и иллюстрируется создание измерений и информационных таблиц. На этом этапе впервые приводится пример базы данных для торговли EASYDW. Он используется во всех примерах этой книги.
- В Главе 3 описываются различные методы, которые можно использовать для повышения производительности обработки запросов в вашем хранилище. Даются рекомендации о том, как лучше всего использовать такие свойства, как индексы и разделы.
- В Главе 4 описывается, как можно добиться значительного улучшения времени ответа на запрос с помощью материализованных представлений (views) и перезаписи запросов.
- В Главе 5 показывается, как извлекать данные, трансформировать их, а затем загружать эти данные в хранилище.
- В Главе 6 рассматриваются некоторые инструменты от Oracle, которые можно использовать для создания хранилищ информации и составления отчетов. Особо описываются Oracle Warehouse Builder и Oracle Discoverer.

- В Главе 7 обсуждаются методы управления хранилищем информации, какие как резервное копирование, восстановление и управление разделами.
- В Главе 8 описывается то, как Web может сыграть ключевую роль для хранилища информации, и то, как это можно использовать на благо вашего бизнеса.
- В Главе 9 обсуждаются Oracle OLAP и информационные проходки Oracle (data mining), которые теперь являются частью базы 9i2R, и которые могут помочь вам лучше понять свой бизнес с помощью системы хранения информации.

Благодарности

При создании книги в разных стадиях производственного цикла участвует множество людей. Очень важную, но часто незаметную роль играют те, кто тратят свое драгоценное свободное время на просмотр книги. Мы выражаем глубокую признательность следующим людям, потратившим свое время на рецензирование данной книги:

Сринвасу Бала (Srinivas Bala), старшему менеджеру и специалисту по проверке качества в Narad Networks. У Сринваса имеется многолетний опыт разработки баз данных Oracle и приложений в телекоммуникационной отрасли.

Джею Дэвисону (Jay Davison), директору по разработке программ и утилит баз данных в Oracle Corporation.

Жан-Пьеру Диксу (Jean-Pierre Dijcks), менеджеру продукта Oracle Warehouse Builder, Oracle Corporation.

Майку Дюрану (Mike Durran), менеджеру продукта Discoverer, Oracle Corporation.

Баду Эндресу (Bud Endress), директору по продуктам Oracle OLAP Products, который участвовал в создании скриншотов и примеров для главы, посвященной OLAP.

Абхинаву Гупте, старшему сотруднику технического отдела Oracle Data Mining Group в Oracle.

Рамкумару Кришнану (Ramkumar Krishnan), старшему сотруднику технического отдела Oracle Data Mining Group в Oracle.

Джеймсу Лиру (James Lear), члену команды разработки Discoverer в Oracle.

Дэвиду Мартленду (David Martland), техническому архитектору Britannia Building Society в Великобритании, имеющему многолетний опыт в использовании базы данных Oracle.

Джуди Мюллер (Judy Muller), помощнику директора Business Service в University of New Hampshire.

Джеку Рейто (Jack Raitto), инженеру консультанту и ведущему специалисту проекта Summary Management в Oracle.

Питу Смиту (Pete Smith), независимому консультанту по системам хранения информации Oracle. Пит проживает в Великобритании и имеет многолетний опыт работы над различными проектами баз данных Oracle.

Джиму Стейнеру (Jim Steiner), главному директору по управлению выпуском продуктов Oracle Corporation.

В рецензировании предыдущего издания этой книги также принимали участие Джей Фринан (Jay Feenan), Лори Молески (Lory Molesky), Симон Оксбери (Simon Oxbury), Майкл Шмитц (Michael Schmitz), Дэннис Мюррей (Dennis Murray), и Майк Рубино (Mike Rubino).

Мы также должны сказать большое спасибо:

Стиву Хэгену (Steve Hagan), вице президенту New England Developer Center, за поощрение и поддержку написания этой книги.

Максу Хиллсону (Max Hillson), чей сайт электронной коммерции coolstuffcheap.com оказал влияние на многие примеры.

Чаку Розвоту (Chuck Rozwat), исполнительному вице-президенту Oracle Server Technologies за поддержку при написании этой книги.

И наконец, эта книга посвящается всем разработчикам программного обеспечения Oracle во всем мире, которые написали программы, сделавшие эту книгу возможной.

Мы также благодарим за создание этой книги Терона Шрива (Theron Shreve) и Пэм Честер (Pam Chester) из Reed Elsevier и Алана Роуза (Alan Rose) и его команду.

1

Системы хранения информации

1.1. Введение

С тех пор как мы опубликовали первое издание этой книга, посвященное Oracle 8i, мы стали свидетелями взлета и падения доткомов, за которыми следовали рост и падение индекса NASDAQ. Мы видели, как URL становятся частью нашего повседневного словаря, и мы обмениваемся электронными письмами практически со всеми, кого мы знаем. Мы изменили способы ведения бизнеса и можем покупать и продавать практически все через Интернет. Мы можем найти наших старых друзей, ближайший магазин домашних животных, узнать, какие фильмы идут в кинотеатрах, или найти хороший ресторан. Нас всегда можно отыскать по сотовому телефону, и мы вносим заметки в карманные компьютеры. Мы никогда не потеряемся, если нам поможет ориентироваться глобальная система навигации (GPS). Мы используем приложения, обеспечивающие самообслуживание, как в работе, так и дома. Мы можем перевести деньга из одного места в другое одним щелчком мыши, и нам не нужно беспокоиться о том, чтобы сделать это в часы работы банка, и волноваться за свою безопасность, выходя ночью к банкомату.

Что же стоит за всей этой технологией? Почти всегда существует база данных, которая организует и хранит информацию, и часто имеется хранилище информации, используемое в процессе принятия стратегических решений в бизнесе.

1.2. Что такое хранилище информации?

Хранилище информации — это база данных, содержащая данные, поступающие от многих рабочих систем, которые интегрируются, собираются и структурируются таким образом, чтобы их можно было использовать в анализе и в процессе принятия решений в бизнесе.

1.2.1. Зачем нужно хранилище информации?

Является ли информация, необходимая вам для ведения бизнеса, доступной тогда, когда это нужно, в той форме, которая вам требуется, с достаточной детализацией и достаточной точностью, чтобы по ней можно было принимать фундаментальные решения? Одни говорят, что объем продаж за март составил 500 миллионов долларов, другие называют цифру 524 миллиона. Проведя большую аналитическую работу, вы выясняете, что для подсчета объема продаж в этих отчетах использовались разные данные, и вы тратите значительное время на то, чтобы выяснить, почему это так, и на то, чтобы исправить проблему.

Есть ли в вашей компании несколько систем, выполняющих одну и ту же функцию, например старая система учета и новая, на создание которой вы только что потратили миллионы долларов? Вам нужно взять данные и из той, и из другой и объединить их для создания отчетов? Насколько хорошо это получается? Нужно ли пользователям знать разницу между двумя системами, чтобы обращаться к ним в режиме он-лайн? Если для этого требуется программирование, то есть ли для этого время и ресурсы?

Есть ли у вас достаточно исторических сведений для анализа? Предысторию за сколько месяцев вы можете хранить? Достаточна ли при этом степень детализации? Сохраняете ли вы все данные предыстории? Можете ли вы проанализировать объем продаж каждого продукта в каждом географическом регионе до и после крупной реорганизации структуры отчетов продавцов? Хранилища данных созданы для того, чтобы помочь решить проблемы такого типа.

1.3. Историческая перспектива

В 70-х годах были созданы первые коммерческие приложения для компьютеризации повседневных деловых операций. Эти системы создавались для компьютеров-мэйнфреймов, которые стоили чрезвычайно дорого. Только крупные компании могли позволить себе аппаратное обеспечение, программистов, для того чтобы программировать его, и обслуживающий персонал, чтобы поддерживать его в рабочем состоянии. Эти системы были ориентированы на введение новых данных и последовательное их считывание при помощи магнитных лент.

После изобретения хранения данных на дисках доступ к данным мог осуществляться напрямую. Это привело к созданию первых систем управления базами данных, которые организовывали данные либо иерархически, либо в сети. Такие системы были очень сложны. Программистам было необходимо знать, как данные хранятся на диске, и осуществлять навигацию по данным для создания отчетов.

Прикладные программисты для создания специализированных отчетов использовали COBOL. Написание программы для каждого нового отчета занимало несколько дней и даже недель. Отчеты печатались на бумаге и распространялись среди пользователей вручную. Программистов всегда не хватало, и всегда не хватало времени и ресурсов на программирование. Как только стал возможным прямой доступ к данным, были созданы первые системы оперативной обработки транзакций (online transaction processing, OLTP).

В конце 70-х и начале 80-х появление миникомпьютеров, таких, как PDP-11 и VAX 11/780 от Digital, снизило стоимость аппаратного обеспечения. Данные часто хранили в сложной базе данных CODAYSL, которую было крайне сложно изменять и тяжело понять и спроектировать. Все изменилось с появлением реляционных баз данных. В 1979 году база данных от Oracle стала первой коммерческой реляционной системой.

В реляционной модели данные организованы в виде таблиц, состоящих из столбцов и строк. Вместо использования указателей (pointers) для создания связи между данными здесь используется уникальное значение, например номер клиента, который хранится во многих таблицах и идентифицирует строку. Реляционную модель намного легче понять, а языку SQL, который используется для доступа к базам данных, не требуется знать, каким образом данные физически хранятся. Стало намного легче создавать приложения, что привело к широкому распространению систем управления базами данных. После первого появления реляционных систем многие компании начали разрабатывать продукты для доступа к реляционным базам данных, использующие нерегламентированные запросы, отчеты и средства анализа.

С появлением персональных компьютеров вычисления перешли с мэйнфреймов на системы архитектуры клиент/сервер. Приложения от Oracle появились в конце 80-х. Компаниям больше не нужно было разрабатывать собственные приложения, они могли теперь приобрести программное обеспечение, обеспечивающее базовые функции, от таких производителей, как Oracle, PeopleSoft и SAP.

Когда реляционные базы данных достигли зрелости, были созданы OLTP-системы, использующие реляционные системы и автоматизирующие рабочие аспекты бизнеса. Сюда входили системы обработки заказов, ввода заказов, учета, ведения гроссбухов, а также бухгалтерские системы. OLTP-системы автоматизируют процессы и представляют состояние системы на текущий момент времени. В приложениях для учета существуют операции для добавления новых элементов в перечень, удаления их после продажи, обновления сведений об имеющемся количестве, что всегда позволяет получить имеющийся баланс. Сохраняется ограниченный объем сведений предыстории. Легко определить, например,

какое количество продукта «111-45-222» имеется в наличии или когда был отгружен заказ «45321». В это время производители реляционных баз данных ориентировались на повышение производительности приложений OLTP и конкурировали друг с другом, используя оценки промышленного стандарта TPC-C.

1.3.1. Появление хранилищ данных

Как только были созданы OLTP-системы для эффективного сбора данных, возник вопрос о том, как лучше всего интерпретировать эти данные. В конце 80-х и начале 90-х в ходе попыток получить более широкий взгляд на бизнес в целом возникли хранилища данных (системы хранения информации) (data warehouses) (термин введен Биллом Инмоном (Bill Inmon), «отцом» хранилищ данных). Данные от многих рабочих систем, используемых в повседневной работе, собирались вместе и обеспечивали общекорпоративную перспективу.

Системы хранения информации создавались для того, чтобы видеть бизнес во времени и выявлять тенденции. Принятие многих решений требует способности отрешиться от повседневных операций и шире взглянуть на бизнес. К типичным запросам к хранилищам относятся составление отчетов по продажам продукта за последние два года или рассмотрение влияния крупного снежного бурана на розничную продажу и продажу через Интернет. Запросы включают определение того, как эти значения менялись со временем, что изменилось еще, а также выявление возможной связи.

Чтобы выполнять анализ такого типа, данные в хранилище должны сохраняться в течение долгого периода времени, часто пять–десять лет.

Система хранения находит информацию, стоящую за данными

В хранилище данных главная операция — это запрос или чтение данных. Единственная операция модификации выполняется при загрузке новых данных. Системы принятия решений (Decision Support Systems, DSS), такие, как Oracle Discoverer, обеспечивают интерактивную посылку запросов, составление диаграмм, графиков и отчетов. Oracle добавила для повышения производительности запросов новые типы структур доступа, такие, как индексы на основе битовых карт (bitmap indexes). В Oracle 8i для повышения производительности сбора и суммирования данных были введены материализованные представления (materialized views).

Oracle Express Server стал аналитическим интерфейсом хранилища данных, обеспечивающим аналитическую обработку информации в режиме он-лайн

(OLAP). Программное обеспечение OLAP используется для анализа бизнес-данных иерархическим образом сверху вниз. Оно предполагает, что запросы будут поступать итеративно, т. е. результаты ответа на один вопрос будут порождать много новых вопросов.

Недостаточно знать только прибыль, полученную за этот год. Аналитики должны знать изменение прибыли во времени для каждого продукта и в каждом географическом регионе. Это называется трехмерный запрос. Измерениями являются продукт, время и географический регион. Аналитику может понадобиться сравнить объем продаж в этом месяце с объемом продаж в том же месяце прошлого года по каждому магазину и с Интернет-сайтом. Он может перейти к более детализованному уровню иерархии и получить объемы продаж по отдельным магазинам и определить, какой из них оказался наиболее прибыльным, а какой — убыточным.

Системы хранения информации требуют иного построения базы данных

Хранилища информации разработаны для быстрого извлечения данных, когда путь доступа к ним неизвестен заранее. Источниками информации часто служат другие данные, составление сводок по данным, действия по выяснению подробностей или поиск тенденций и закономерностей.

В системах OLTP при проектировании схемы базы данных используется техника диаграмм связи сущностей (entity relationship, E-R). Каждая сущность становится таблицей, ее атрибуты становятся столбцами, а связи отражаются путем объединения столбцов первичного и внешнего ключей в ходе выполнения.

Нормализованное построение обеспечивает оптимальную производительность систем OLTP, что способствует большему объему операций и частому обновлению данных. Нормализация обеспечивает корректное формирование таблиц, собирая связанные данные в одну таблицу и устраняя избыточность. Наличие единственной копии данных позволяет избежать аномалий при обновлении. Когда данные обновляются только в одном месте, сохраняется их связность. После нормализации данных для повышения производительности возможно внесение некоторой избыточности в столбцы, обновление которых не проводилось.

Чтобы оптимизировать производительность хранилища, в котором главной операцией является запрос к данным, необходима новая модель данных. Ральф Кимбелл (Ralph Kimball), главный пропагандист пространственного моделирования и автор книги «The Data Warehouse Toolkit», предложил схему звезды, новый способ проектирования базы данных для облегчения обработки OLAP. Для

оптимизации производительности системы хранения информации используется техника пространственного моделирования.

При пространственном подходе данные организуются в таблицы фактов (fact table) и измерений (dimension table). Данные представляются в понятном для пользователя виде. Пользователи часто запрашивают отчеты поквартально с секционированием по магазинам и географическому региону. Число продаж — это факты. Магазин, регион и квартал — это измерения, по которым анализируются данные и которые используются для их организации. При пространственном моделировании вносится денормализация и избыточность. В главе 2 мы увидим, как можно спроектировать базу для компании Easy Shopping Inc. Этот пример будет затем использоваться на всем протяжении книги.

Логическое построение переводится в физическое, с максимальной оптимизацией производительности и управляемости. Определяются таблицы, ограничения, индексы и секционирование.

Oracle добавила несколько свойств, поддерживающих пространственное построение. Тот, кто проводит оптимизацию, может распознать схему звезды и выполнить специальную оптимизацию, что позволит избежать создания более дорогого продукта с полными декартовыми координатами. Помимо создания таблиц и столбцов, вы также можете определять измерения.

1.3.2. Хранилища данных эволюционируют как отдельные системы

Вы, должно быть, уже хотите узнать, почему вы не можете использовать ваши работающие системы в качестве хранилищ данных. У вас уже имеются базы данных, доступ к которым осуществляется через корпоративную сеть. Так почему же нельзя просто использовать их для получения необходимой информации для более эффективного управления бизнесом? Зачем нужно копировать данные с одной системы в другую, чтобы создать хранилище? Поскольку рабочие системы могут не содержать данных о предыстории, информация не будет доступна для анализа. Кроме того, схема базы не предназначена, а данные не структурированы для выполнения «умных» бизнес-запросов.

Кроме того, запросы к хранилищу, как правило, обращены к большим объемам данных, что требует больших объемов памяти, мощности процессора и ресурсов ввода/вывода. Выполнение запросов, связанных с принятием решений, которое требует больших объемов вычислительных мощностей для сортировки и суммирования миллионов строк, может повлиять на производительность ваших рабочих систем.

И что даже еще сложнее, многие различные рабочие системы часто функционируют на разных платформах, в разных операционных системах и системах управления базами данных. Некоторые приложения могут быть приобретены, а другие — разработаны самостоятельно. В некоторых случаях исходный код может оказаться недоступным для внесения изменений, а некоторые системы могут иметь возраст в несколько лет.

Когда-то считалось, что распределенные базы данных позволят пользователю отправить запрос, а глобальные оптимизаторы запросов явным образом обнаружат данные и вернут их пользователю настолько быстро, что он даже и не узнает, что данные располагаются на машине в другом географическом регионе. Однако системы такого типа так и не были реализованы.

В итоге, данные необходимо переносить из работающих систем в отдельное хранилище данных, где они сохраняются для анализа в общем формате на другой машине.

Хранилище данных строится на основе действующих систем

Создание хранилища происходит путем извлечения данных из рабочих систем, комбинирования их иногда с дополнительной информацией, поступающей от третьей стороны, переведения их в однотипный формат и загрузки в базу данных.

После того как данные введены в хранилище, они почти никогда не меняются, поскольку данные отражают факты какого-то события или состояния, которое присутствовало в некоторый момент времени, как, например, конкретная продажа продукта, случившаяся 23 декабря 1998 года. Если имелась другая продажа такого продукта 24 декабря 1998 года, она будет зафиксирована как отдельное событие.

Часто до 80 % работы по созданию хранилища посвящается процессу извлечения/преобразования/загрузки (extraction/transformation/load, ETL): обнаружить данные, написать программу их извлечения, отфильтровать и очистить данные, перевести их в общую схему кодировки и загрузить в хранилище.

Рабочие данные необходимо извлечь из рабочих систем и скопировать в промежуточную область — временное местоположение, где происходит очистка данных, их преобразование и подготовка к введению в хранилище. Иногда вы имеете полный доступ к исходным системам. Однако часто доступ серьезно ограничен, вы можете только получать извлеченные файлы данных. Рабочие системы часто должны функционировать в режиме 24/7/52, и производительность не должна нарушаться ни в коем случае.

Данные, полученные из многих систем, должны быть переведены в общий формат для последующего использования в хранилище. Необходимо понимать смысл данных в рабочих системах, включая, например, следующее.

- Каждая рабочая система может ссылаться на один и тот же элемент по разным именам или ключам. Например, компания по производству инструментов, находящаяся в Соединенных Штатах, может называть продукт с номером 1234 «wrench», а компания, находящаяся в другой стране, назовет тот же продукт 1234 «spanner».
- Каждая система может использовать свою схему кодирования. Идентификатор продукта в одной системе может представлять собой три символа, разделенные тире (xxx-xx-xxx), а в другой системе разделяться пробелами (xxx xx xxx). Данные для хранилища должны быть преобразованы в общую схему кодирования.
- Атрибуты таблицы могут иметь различные имена. Одна система может называть столбец в таблице, посвященной заказчику, «gender» и показывать его значениями 0 и 1, а другая система может назвать этот же столбец «sex» и показывать его буквами «M» и «F».
- Разные системы могут использовать разные единицы измерения. Объем продаж в Соединенных Штатах будет измеряться в долларах, а в Европе — в евро. Данные в хранилище должны быть преобразованы в общие единицы.

При разработке процесса трансформации устанавливается соответствие между этими разными названиями столбцов и общим названием, выбранным для хранилища, и производится трансформация в общую схему кодировки.

Как только данные преобразованы, они готовы к загрузке в хранилище. Часто область, в которой производится преобразование, располагается не на той машине, на которой находится хранилище, и следовательно, данные нужно перенести на ту машину, где хранилище расположено.

Если преобразованные данные сохранены в сплошном файле (flat file), их можно транспортировать с помощью FTP, а потом загрузить с помощью утилиты SQL*Loader или внешних таблиц 9i. Если данные преобразованы в базу данных Oracle, то для перемещения табличных пространств из одной базы в другую можно использовать транспортируемые табличные пространства (transportable tablespaces).

Oracle Warehouse Builder, средство из пакета Oracle Internet Developer Suite, помогает автоматизировать некоторые аспекты извлечения, преобразования, транспорта и загрузки. Мы еще рассмотрим процессы извлечения и преобразования в главе 5.

Новые данные, как правило, добавляются в хранилище периодически. Они обычно загружаются пакетами по вечерам, или в другое время, когда хранилище не очень интенсивно используется аналитиками.

Помимо ваших собственных данных, вы можете закупать данные у внешних поставщиков и добавлять их в ваше хранилище. Например, можно приобрести данные о погоде, демографическую и социоэкономическую информацию.

К примерам использования можно отнести следующее.

- Добавление данных, в которых ежедневно отражаются изменения погоды. Таким образом вы можете определить, продажи каких продуктов повышаются или понижаются в снежный буран.
- Добавление демографической информации о покупателях. Можно осуществлять селективный маркетинг, выявляя тех покупателей, которые с большей вероятностью ответят на стимулирование сбыта.
- Зная, какие типы покупателей какие типы продуктов приобретают, вы можете предвидеть спрос и повысить прибыльность. Демографическую информацию можно использовать для определения нового места для магазина розничной торговли.
- Добавление данных от компании «Dun and Bradstreet», которые содержат информацию о компаниях и продуктах.

1.3.3. Тематические хранилища данных

Создание хранилища данных может быть делом очень сложным и часто требует от 18 месяцев до трех лет. Поскольку хранилище содержит множество предметных областей и пересекается со многими организациями, его создание может иметь серьезный политический аспект. Многие ранние проекты по созданию хранилищ данных окончились неудачей.

Было обнаружено, что те же самые многочисленные преимущества хранилищ данных можно свести к подразделению или отрасли бизнеса и решению конкретной деловой задачи. Хранилища данных содержат множество тем, что обеспечивает общекорпоративный взгляд на все отрасли бизнеса. Тематические хранилища данных (data marts) являются предметно-специфичными или специфичными для конкретного применения хранилищами данных и содержат данные только по одной отрасли бизнеса, например, по продажам или маркетингу. Главным отличие тематических хранилищ от данных являются размеры области хранимой информации. Поскольку область информации тематического хранилища данных значительно уже, данные получаются из небольшого числа источников и типичное время создания тематического хранилища меньше.

Независимые тематические хранилища

Тематические хранилища, в зависимости от источника информации, могут быть зависимыми или независимыми. Источником информации для зависимого тематического хранилища является существующее хранилище информации

(warehouse). Тематическое хранилище считается независимым, если хранилище отсутствует и данные извлекаются напрямую из рабочих систем.

Поскольку независимые тематические хранилища можно создавать очень быстро, они стали очень популярны в середине и конце 90-х годов. Каждое подразделение в компании создавало свое собственное тематическое хранилище для своих нужд. К сожалению, после создания нескольких тематических хранилищ начинают возникать проблемы. Каждое тематическое хранилище само по себе является «островком информации». Очевидно, что возникают проблемы, когда приходят два разных отчета на один и тот же запрос.

Одной из причин возможности столь быстрого внедрения тематических хранилищ является то, что они оттягивают принятие критических решений, которые становятся необходимы, когда число таких хранилищ данных возрастает. Понимать и идентифицировать нужно только данные отдельных подразделений. Полное понимание всех данных компании не является необходимым. Создание независимых тематических хранилищ позволяет не затрагивать политические вопросы, связанные с общей системой имен и схемы кодирования.

Другие проблемы появлялись в силу того факта, что отдельные тематические хранилища часто создавались независимо друг от друга различными автономными группами. Эти группы часто выбирают разное аппаратное и программное обеспечение и используемые инструменты.

Каждое независимое тематическое хранилище получает данные из рабочей системы. Если у компании есть пять разных тематических хранилища и каждому из них требуется информация о покупателе, то будут работать пять отдельных программ для извлечения данных из таблицы рабочей системы, содержащей данные о покупателе. Вполне вероятно, что уже сейчас вам не хватит времени, чтобы выполнить пять программ извлечения данных в пакетном режиме, и вы, определенно, не сможете запускать большее число программ в будущем, когда вы введете новые тематические хранилища.

Каждый проводит очистку и преобразование своих данных, и вероятно, несколько различными способами. Это очень легко приводит к несовместимости данных. Избыточность и несовместимость данных приводит к разным ответам на запросы, что затрудняет принятие решений. Представьте себе попытку объединить эти разные взгляды в общем хранилище данных.

Склады рабочих данных консолидируют создание отчетов по текущей информации

Как уже говорилось ранее, есть большая разница между данными в рабочих системах и данными в хранилище. Рабочие данные относятся к текущему состоя-

нию компании и используются для управления повседневными операциями. Данные в хранилище являются информационными и содержат исторические записи о прошлых событиях.

Если существует потребность в предоставлении информации о текущем состоянии бизнеса для принятия тактических решений и обслуживания повседневных операций, можно создать склад рабочих данных (operational data store, ODS), как часть архитектуры управления информацией. Склады рабочих данных содержат предметно-ориентированные интегрированные данные при возможности их оценки и очистки. Их используют для обработки рабочих запросов и отчетов. Примером может служить организация службы работы с покупателями, которой требуется доступ к текущим бухгалтерским балансам и информации о счетах.

Обновление ODS можно проводить в реальном времени, чтобы они вовремя отражали текущее состояние. ODS могут служить источниками информации для хранилищ и лавок данных.

Поэтапное создание хранилища данных с помощью зависимых лавок данных

Для решения этих проблем при обеспечении своевременной окупаемости инвестиций вместо создания хранилища, соответствующего всей корпорации, люди начали создавать хранилища данных по одной функциональной области за раз с помощью фазового подхода.

На рис. 1.1 показана наиболее распространенная в настоящее время архитектура. Данные извлекаются из систем OLTP и внешних источников, загружаются в склады рабочих данных и производственные хранилища данных, а затем загружаются в зависимые тематические хранилища данных.

Создание хранилища информации очень похоже на создание программного обеспечения. Вы не сможете сделать все в одной версии, и вы никогда не сможете предвидеть всех возможных способов применения. Лучше всего разработать общую архитектуру и создавать основу на основе компонентов, что позволит строить хранилище пофазно. Ограничьте область задачи и планируйте вводить усовершенствования каждые три–шесть месяцев.

Средства создания отчетов, обработки запросов и проведения анализа

В 1995 году Лари Эллисон (Larry Ellison), основатель и главный администратор Oracle, впервые предложил свою концепцию сетевого компьютера — небольшого,

недорогого устройства, которое обеспечивает запуск приложений, обращающихся к информации через Интернет. Хотя сетевой компьютер так и не завоевал большой доли на рынке, концепция бизнес-расчетов, ориентированных на Интернет, ускорила снижение цен на персональные компьютеры, что удовлетворяет запросам на более дешевые и простые настольные вычислительные системы.

В 1998 году Oracle выпустила полный комплект приложений для использования в Web и остановила разработку приложений для использования в системах архитектуры клиент/сервер. Oracle 8i, база данных, основанная на Интернет, имеет много возможностей для работы с Интернет-операциями и приложениями.

Возможность публиковать отчеты в Web делает информацию доступной практически каждому. Это дает служащим, партнерам и покупателям доступ к критической информации в режиме реального времени. Вам больше не нужно быть в офисе, чтобы увидеть отчет. Просто загляните в Интернет-кафе или соединитесь с Интернет из комнаты в отеле. Размещение информации в Web (в вашей внутренней web-сети или во Всемирной паутине) означает, что ваш офис поистине может быть в любом месте.

1.4. Oracle 9*i*

Начиная с конца 2000 года и с первым появлением Oracle 9i Release 2 в 2002 году компания Oracle выпустила серию базовых технологических продуктов Oracle 9i, к которым относятся Oracle 9i Database, Oracle 9i Application Server, 9iAS и Oracle 9i Developer Suite. Компоненты, которые ранее выпускались по отдельности, теперь объединены в комплекты.

Oracle серьезно переработала свой сервер приложений Oracle 9iAS для версии Oracle 9i и включила в него технологии создания порталов, анализа web-трафика и создания и запуска приложений для беспроводной связи. Рабочие книги Oracle 9iAS Discoverer и отчеты Oracle 9iAS могут публиковаться в папках (portlets). Информация по передвижению по Web-узлу Oracle 9iAS, записываемая в протоколах сервера, отражает эффективность работы web-сайта.

Кроме достижений в технологиях сервера приложений, появились усовершенствования средств разработки приложений. Объединенные теперь в пакете Oracle 9i Developer Suite, эти инструменты ускоряют создание ориентированных на Интернет BI-приложений (Business Intelligence, бизнес-информация). Мастера Jdeveloper формируют среду для быстрого создания перекрестных табличных данных, графиков и других элементов для анализа и представления данных.

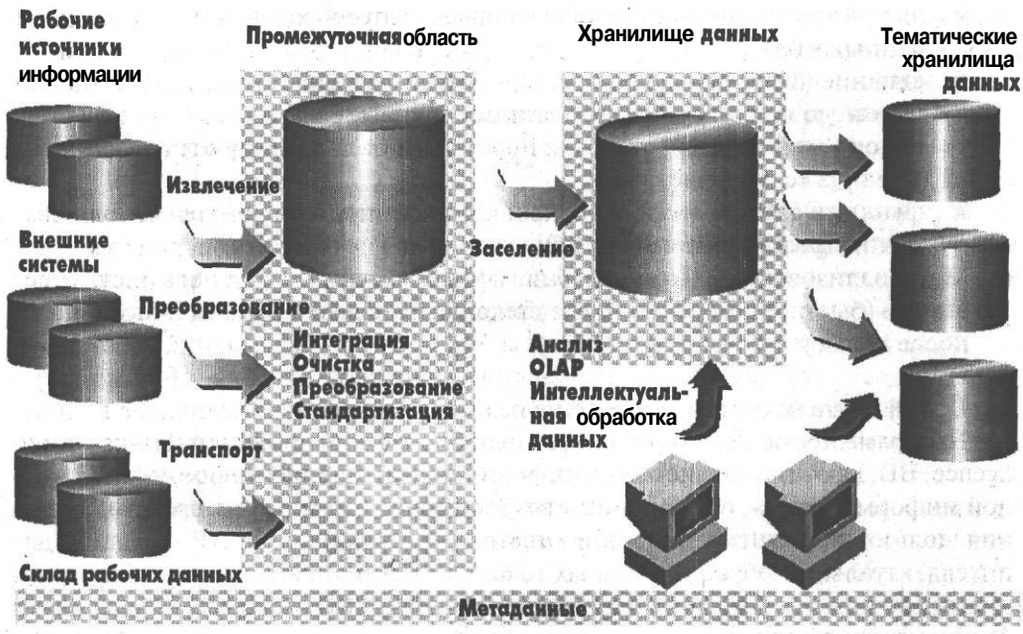


Рис. 1.1. *Производственное хранилище информации с зависимыми тематическими хранилищами*

1.4.1. Возможности хранения информации в 9i Database

Для улучшения производительности, управляемости и масштабируемости хранилища данных в Oracle *9i* Database было добавлено много возможностей. К ним относятся следующие.

- Соединительные индексы на основе битовых карт (bitmap join indexes), повышающие производительность соединения таблиц.
- Новый метод секционирования на разделы, позволяющий администратору базы данных перечислить те данные, которые будут храниться в каждом разделе.
- Улучшения в обработке ETL (extraction/transformation/load, извлечение/преобразование/загрузка):
  - изменение механизма фиксации данных позволяет выявлять измененные строки в базе данных Oracle;

- внешние таблицы позволяют проводить преобразования при загрузке данных в базу;
- слияние (Merge) — новая команда SQL, выполняющая подгрузку (upsert), полезную при загрузке многомерных данных. Если строка уже существует, она модифицируется. Если строка в базе данных отсутствует, эта команда ее вставляет;
- функции таблиц позволяют проводить канальное параллельное выполнение преобразований PL/SQL;
- Материализованные представления (Materialized views) теперь можно поэтапно (быстро) обновлять после введения отдельных команд DML, а также после прямой загрузки путей.

В этой книге мы будем рассматривать эти и многие другие вещи.

Программное обеспечение, связанное с бизнес-информацией (business intelligence, BI), дают специалистам по информационным технологиям доступ к важной информации для создания отчетов и анализа. Сюда входят средства создания пользовательских запросов и отчетов, инструменты OLAP, средства для интеллектуальной обработки даных (data mining) и управленческие информационные системы (Executive Information Systems, EIS), которые обеспечивают детализацию, навигацию и сообщения об исключительных ситуациях. В Oracle 9i многие функции бизнес-информации включены в базу данных.

1.4.2. Объединение Oracle 9i Database и технологии экспресс-сервер

OLAP была впервые предложена доктором Коддом (Dr. E.F. Codd), «отцом» реляционных баз данных. Он утверждал, что реляционные базы данных изначально не предназначены для проведения анализа, синтеза и консолидации данных-функций, определяемых в многомерном анализе. В течение многих лет для обеспечения этих возможностей, недоступных в реляционных базах данных, применялись отдельные аналитические базы, такие, как Oracle Express.

Oracle добавила множество функций, облегчающих обработку запросов OLAP, а с появлением Oracle 9i Release 2 стало возможным напрямую использовать сервер Oracle для OLAP. Язык SQL был расширен с целью обеспечения аналитических функций, таких, как классификация, перемещение оконных агрегатов, попериодное сравнение, соотношение к отчету, статистические функции, обратные процентили, гипотетическая классификация и распределение, гистограммы и агрегаты

типа первый/последний. Множественные уровни агрегации можно рассчитать с помощью куба, свертки и группировки. Большинство вычислений выполняются напрямую сервером. Эти функции позволяют выразить запросы OLAP без сложных самообъединений и подзапросов, что позволяет специалисту по оптимизации лучше составить план выполнения запроса.

1.4.3. В Oracle 9i Database встроена функция интеллектуальной обработки данных

Oracle 9i Data Mining, новая возможность версии Enterprise Edition, обеспечивает встраивание в базу данных функции интеллектуальной обработки данных для выполнения классификации, составления прогнозов и определения связей.

Интеллектуальная обработка данных является частью системы поиска информации. С помощью статистических методов можно преобразовать огромные объемы данных в полезную информацию. Данные служат чем-то вроде руды, извлекаемой из обычного рудника. После превращения в информацию они становятся драгоценным металлом.

Интеллектуальная обработка данных извлекает из данных новую информацию. Это позволяет бизнесу получить из хранилища ранее неизвестные сведения и использовать их при принятии важных деловых решений.

Процесс поиска, как правило, начинается без всякого предварительного представления о том, что будет найдено. Прочитываются большие объемы данных и ищутся сходства, которые можно сгруппировать и по которым можно выявить схемы и тенденции.

Средства OLAP и DSS просматривают предписанные взаимоотношения, связанные со структурой данных. Они отражаются в ограничениях (constraints) и измерениях (dimensions). Интеллектуальная обработка данных выявляет взаимоотношения, связанные с содержанием данных, которые еще не определены. Например, какие продукты с наибольшей вероятностью приобретаются вместе. Это называется анализом потребительской корзины. При анализе данных за определенный промежуток времени можно выявить неожиданные схемы поведения. Иногда можно определить вероятность выполнения определенных действий после осуществления некоторых других действий. Типичным применением интеллектуальной обработки данных является удержание покупателя, выявление мошенничества и определение общей схемы совершения покупок покупателями. Интеллектуальную обработку данных можно применять для поиска новых рыночных возможностей.

Средства OLAP можно применять для ответов на такие вопросы, как: «Возрос ли объем продаж светильников в ноябре по сравнению с предыдущим годом?». Инст-

рументы интеллектуальной обработки данных помогают отвечать на такие вопросы, как: «Какие факторы определяют объем продаж светильников?».

При применении средств OLAP аналитики начинают с формулирования вопроса или гипотезы и посылают запрос к хранилищу данных для доказательства или опровержения своей теории. При применении средств интеллектуальной обработки данных эта работа больше смещается от аналитика к компьютеру. Средства интеллектуальной обработки данных решают множество различных проблем с помощью разнообразных методов.

Oracle 9iAS Personalization, приложение, дающее рекомендации в реальном времени, использует встроенные функции Oracle 9i Database и динамически выдает персонализированные рекомендации как анонимным посетителям web-сайта, так и зарегистрированным покупателям, обеспечивая маркетинг по схеме 1:1 для систем электронной коммерции. Oracle 9iAS Personalization отвечает на такие вопросы, как «Какие из этих товаров человеку скорее понравятся или будут им приобретены и с какой вероятностью?», «Возможно ли, что люди, купившие данный товар, купят еще какой-нибудь другой?». Такую персонализацию можно наблюдать на Amazon.com, и она становится обычной и для многих других web-сайтов.

1.5. Создание хранилища данных ставит много проблем

Хотя системы хранения информации как технология уже созрели за последние несколько лет, разработчики хранилищ данных сталкиваются с все новыми проблемами по мере того, как меняется бизнес и технологическое окружение. Базы данных должны строится с учетом определенной производительности и надежности и должны иметь возможность расширяться по мере роста бизнеса.

1.5.1. Управление хранилищем

Когда хранилища данных увеличиваются в размерах до нескольких терабайт, а требования к доступности данных все повышаются, обеспечение хорошей производительности для большого числа пользователей, широко разбросанных географически, становится критическим фактором. Нужно определить процедуры резервного копирования и восстановления и нужно управлять как содержанием данных в хранилище, так и их использованием.

Рабочие операции, связанные с принятием решений, весьма вариабельны. В системах OLTP приложения настроены на как можно более быструю обработку многих идентичных операций обновления. В хранилище данных необходимо Настроить производительность так, чтобы обрабатывалось как можно больше разнообразных запросов.

Схемы использования создают основу для наилучшей настройки производительности хранилища. Кто какие данные использует? Какие уровни обобщения просматриваются? Какие данные не используются? Эффективно ли структурированы данные? Проиндексированы ли они по верным столбцам? Можно ли для большинства запросов применить суммарные таблицы? Если во многих запросах данные одной таблицы объединяются с другой, может быть лучше денормализировать данные, объединив их? Информация о рабочих операциях помогает определить, где следует ввести индексы, где таблицы следует объединить, а где нужно создать сводки (summary).

В конечном итоге, может оказаться не нужным и непрактичным обеспечивать немедленный доступ ко всем подробностям данных в режиме он-лайн. Данные можно очищать без сохранения копии. Или же данные можно заархивировать и перенести на какой-нибудь дешевый носитель, например на магнитную ленту или на CD-ROM, откуда их можно извлекать по мере необходимости.

1.5.2. Роль метаданных

Метаданные — это данные, описывающие другие данные или операции над этими данными. Метаданные можно использовать для технических или бизнес-целей. Когда данные переходят из рабочих систем в хранилище, они извлекаются, преобразуются и суммируются. Технические метаданные необходимы для описания этого процесса и важны для перехода к более подробным уровням детализации. Бизнес-метаданные позволяют конечным пользователям определить, какие данные в общем или тематическом хранилище доступны и как можно осуществить доступ к ним. Метаданные обеспечивают интеграцию и единообразие данных по всей корпорации. Именно здесь разные подразделения описывают свое использование термина «продукт». Метаданные хранятся в репозитории, который, как правило, представляет собой набор таблиц в базе данных Oracle. Возможно совместное использование его любыми пользователями и инструментами.

В 2000 году Object Management Group (OMG) опубликовала спецификацию «Common Warehouse Metamodel (CWM) Specification», в которой определяется формат метаданных для всех хранилищ данных и продуктов, связанных с бизнес-

информацией. Эта спецификация была разработана совместно несколькими компаниями, включая Oracle и IBM Corporation. С момента опубликования этой спецификации многие системы хранения информации были приведены в соответствие с этим стандартом.

1.5.3. Увеличение объемов данных

Одной из самых больших технологических проблем, с которой сейчас сталкивается промышленность, является взрывообразное увеличение объемов данных, ожидаемое в течение нескольких следующих лет. Хранилища — это «ну очень» большие базы данных. Размеры баз данных увеличиваются сейчас гораздо быстрее, чем когда-либо. Это объясняется многими причинами, к которым можно отнести следующие.

- По мере совершенствования аппаратной базы и продолжающегося падения цен на хранение информации становится экономически приемлемо содержать все более и более подробные исторические сведения. Вы теперь можете хранить запись по каждому продукту, который покупатель приобрел в супермаркете, а не только о том факте, что было приобретено пять продуктов на общую сумму 75.25$.
- Бизнес хранит все больше и больше данных за все более долгий промежуток времени.
- Данные сохраняются по нескольку раз для разных целей. Создаются индексы и материализованные представления, улучшающие производительность, но эти структуры доступа требуют дополнительного пространства для хранения, что еще больше увеличивает размер базы данных.
- Неструктурированные данные можно объединять с традиционными приложениями для бизнес-информации. Хранение мультимедийных данных увеличивает размер базы данных. Для хранения 1 часа видеозаписи требуется около 1 Гб свободного места. Для хранения 1 минуты аудиозаписи требуется чуть менее 1 Мб. Размеры изображений изменяются от 20 Кб до 60 Мб, в зависимости от типа и качества.
- В документы можно ввести теги метаданных на основе XML и сохранить в Oracle 9i. Информация по перемещению пользователя по web-сайту, как ожидается, добавит в хранилище терабайты данных.

Однако помните, что хранилище данных не нужно просматривать как репозиторий архивных данных, оно не предназначено для этой цели.

1.5.4. Увеличенная доступность

Обеспечение доступности хранилища данных становится все более и более критичным для достижения цели во многих предприятиях. По мере того как хранилища становятся все более функциональными по своей природе, направляя информацию в системы OLTP в предприятиях, функционирующих в глобальном масштабе, пользователям становится необходим доступ к хранилищу в режиме 24 часа в сутки, 7 дней в неделю.

Система Oracle 9i Database разработана с целью устранения запланированных простоев и приобретения возможности выдержать любую аварию: сбой системы, ошибку хранения, нарушение работы сайта или человеческую ошибку. Если сервер выходит из строя, ваши приложения продолжают работать. Кластеры реальных приложений (Real Application Clusters, RAC) делают приложения расширяемыми и имеющими высокую доступность. Одну базу данных можно запустить на группе серверов, объединенных в кластер. При добавлении к кластеру дополнительных серверов приложения могут масштабироваться и повышать пропускную способность без всяких модификаций.

Систему Data Guard можно использовать для обслуживания транзакционно-совместимой копии хранилища данных, что гарантирует выполнение операций с минимальным вмешательством человека в случае аварии, человеческой ошибки или повреждения данных.

1.5.5. Больше пользователей/ лучше производительность

Возможность публиковать информацию в Web делает ее доступной гораздо большему числу людей. По мере того как данные в хранилищах и системах бизнес-информации становятся все более доступны, число конечных пользователей продолжает расти. Запросы на большую производительность более важны, чем когда-либо ранее. Кроме того, возрастает сложность типов запросов.

Поскольку средний пользователь находится на одной web-странице в течение пяти секунд, информация должна обновляться и быть доступной мгновенно. В большинстве случаев ее необходимо просчитывать заранее и сохранять в кэше.

1.5.6. Новые типы приложений

Системы хранения информации используются для поддержки новых инициатив в области электронной коммерции, таких, как Управление связью с покупателями (Customer Relationship Management, CRM) и Управление цепью поставки (Supply Chain Management). Система CRM помогает привлекать новых покупателей и сформировать их приверженность, что важно для удержания клиентов. Хранилище данных содержит информацию о клиентах данной компании и часто служит местом интеграции приложений, связанных с объемами продаж, маркетингом и заботой о покупателях.

1.6. Будущее систем хранения информации

В каком направлении мы будем двигаться дальше? Хотя будущее предсказать не может никто, некоторые тенденции, кажется, проявляются.

1.6.1. Хранилища данных в режиме реального времени

Системы хранения информации будут эволюционировать в направлении обеспечения анализа и принятия решений в режиме реального времени. Вместо периодического пакетного обновления хранилища, когда операция осуществляется в системе OLTP, станет возможным проводить ее в самом хранилище, что даст возможность принятия решений в реальном времени.

Это позволит применять хранилище данных для принятия как тактических, так и стратегических решений. Это даст возможность компании, связанной с кредитными картами, выявлять и останавливать мошенничества в момент их совершения, транспортной компании — перестроить маршруты своих машин сразу после аварии, а он-лайновому розничному продавцу — выдвигать специальные предложения на основе перемещений покупателя по Web.

1.6.2. Исчезновение отдельных хранилищ данных

Однажды мы сможем использовать одну базу данных и для OLTP, и для хранения информации. Oracle создает в базе данных возможности, которые позволяют смешивать операционные и аналитические функции. Многие из этих возможностей доступны в Oracle 9i. При таком подходе больше не будут нужны отдельные базы данных для OLTP, ODS, хранилищ и лавок данных. Это устранит необходимость перемещений огромных объемов информации — извлечения, преобразования, загрузки и копирования данных в этих четырех базах и снизит стоимость и сложность интегрирования и управления несколькими базами.

Уже в Oracle Applications и OLTP, и принятие решений, и создание отчетов выполняется в одном и том же экземпляре базы Oracle с помощью RAC. При использовании Oracle 9i больше не нужно иметь отдельные реляционные системы, OLAP, системы интеллектуальной обработки данных и ETL. Это значит тельно упрощает функционирование и управление инфраструктурой хранилища данных.

Конечно, все еще есть необходимость в интеграции данных из многих источников, если они не хранятся все в одной базе данных Oracle.

1.6.3. Заключение

Мы прошли от мейнфреймов 70-х годов к миникомпьютерам 80-х и к системам клиент/сервер 90-х. В настоящее время наиболее значительным изменением являются вычисления в Интернет, поскольку они снизят цены и усовершенствуют предоставляемые услуги для любого бизнеса.

Сейчас *с* помощью Интернет стало возможным внедрять приложения, связанные с бизнес-информацией, в крупные, географически распределенные популяции пользователей, как внутренним, так и внешним способом по отношению к поставщикам и клиентам.

В этой главе мы рассмотрели то, что способствовало подъему хранилищ и лавок данных, показали главные черты Oracle 9i, охарактеризовали некоторые из множества проблем, с которыми сталкиваются разработчики систем хранения информации, и рассказали о том, что мы думаем о будущем. Теперь пора рассмотреть, как можно использовать всю эту технологию для построения нашего хранилища данных и доступа к нему.

2

Создание хранилища

2.1. Введение

Читателей этой главы, скорее всего, можно разделить на три категории.

1. Те, кто никогда ранее не проектировал базы данных
2. Те, кто разрабатывал базу данных для систем, связанных с обработкой транзакций.
3. Те, кто уже создавал систему хранения информации.

В последнем случае читатель может опустить этот текст или просто освежить свои познания, особенно если последней созданной базой данных была база Oracle. Таким образом, эта глава предназначена для тех, кто подпадает под категории номер один или два. Это может вызвать удивление у тех, кому ранее приходилось разрабатывать базы, не связанные с обработкой данных. Почему? Потому что навыки и методы, используемые при создании базы данных для хранилища, будут отличаться от тех, которые применяются в системах, связанных с обработкой транзакций. Следовательно, хотя вы и будете иметь некоторое преимущество, поскольку некоторые методы в этих двух случаях одинаковы, очень важно, чтобы вы сказали себе: «Я разрабатываю базу данных совершенно другого типа».

Так чем же отличается создание базы данных для хранилища? В системах, связанных с обработкой транзакций, целью разработчика является чрезвычайно быстрое выполнение транзакции. Кроме того, разработчик имеет то преимущество, что он знает, какие вопросы задает бизнес и как он использует данные. Хранилище данных отличается тем, что, хотя запросы должны исполняться как можно быстрее, их обработка, тем не менее, может занимать часы.

Еще одна важная проблема состоит в определении той информации, которую нужно держать в хранилище, а также степени ее детализованности. В этой

книге мы не будем обсуждать то, как определять, какую информацию нужно поместить в хранилище, и каким образом можно собирать эти данные. Есть множество книг, в которых эта тема обсуждается развернуто.

Тем не менее, трудно переоценить важность отбора информации для включения в хранилище. Это так важно потому, что может не пройти и года эксплуатации хранилища, как вы внезапно обнаружите, что информация либо недоступна, либо сохранялся не должный ее уровень. Например, телефонная компания решает не сохранять в базе каждый звонок, а хранить только общую величину затрат клиента за день. Потом кто-то в компании решает предложить покупателям скидку при вызове определенных номеров. Теперь, если бы хранилище содержало все телефонные звонки клиентов, компания смогла бы в точности выяснить, во сколько обошлось бы применение этой схемы за 12 месяцев. Однако данные недоступны, и приходится гадать, какова может быть цена, или же откладывать ввод запланированной схемы до тех пор, пока не поступит достаточно данных, чтобы можно было точно определить, во что это обойдется компании.

Одним из трудных решений, которые должен принять проектировщик базы, является определение уровня детализации данных, хранимых в хранилище. Часто сохранение каждой транзакции, как в нашем примере с телефонной компанией, может показаться несколько чрезмерным, а поскольку это легко увеличивает размер хранилища до многих терабайт, существует тенденция объединять данные. Управление терабайтным хранилищем требует точных и строго контролируемых процедур. Чем больше становится база данных, тем труднее управлять ею и обрабатывать запросы. Поэтому существует склонность к объединению информации.

Поскольку объединение — это важное проектное решение, проектировщику перед внедрением данной стратегии следует получить одобрение со стороны пользователей хранилища. Нужно четко объяснить пользователям те ограничения, которые могут появиться из-за объединения данных. При такой дешевизне дисков мы надеемся, что большинство компаний будет сохранять всю необходимую информацию.

2.1.1. Не используйте моделирование связи сущностей

Типичный подход, используемый в системах обработки транзакций, — это конструирование диаграмм связи сущностей (entity-relationship, E-R) для данного бизнеса. В конечном счете, они используются в качестве основы для создания

физического проекта базы данных, поскольку многие сущности модели становятся таблицами базы. Если вам никогда не приходилось проектировать хранилище данных, но у вас имеется опыт по созданию систем обработки транзакций, вы, вероятно, подумаете, что хранилище не отличается от любых других баз, в которых вы использовали этот подход.

К сожалению, это не так и разработчики хранилищ данных быстро обнаруживают, что модель связи сущностей, в действительности, не подходит для разработки хранилища. Ведущие авторитеты в данной области, например Ральф Кимбелл (Ralph Kimball), выступают за использование пространственной модели (dimensional model), и мы считаем, что этот подход идеален для создания хранилища.

Диаграмма связи сущностей может показать нам, и весьма подробно, взаимоотношения многочисленных сущностей в нашей системе, убирая везде, где это возможно, избыточность. Результатом будет очень плоское изображение предприятия, где описываются сотни сущностей в их взаимодействии с другими сущностями. Хотя этот подход отлично соответствует миру обработки транзакций, где нам необходим именно такой уровень детализации, он слишком сложен для хранилища данных. Если вы спросите администратора базы данных о том, есть ли у него диаграмма связи сущностей, он, скорее всего, ответит, что когда-то, когда система была только что разработана, она у него была. Но из-за ее размеров, а также потому что в системе за время ее существования произошли многочисленные изменения, диаграмма связи сущностей не обновлялась, и сейчас она правильна лишь частично.

Если мы будем использовать для хранилища другой подход, который даст нам гораздо более простое изображение, это изображение будет гораздо проще поддерживать в соответствии с состоянием системы, а также знакомить с ним конечных пользователей, чтобы помочь им понять хранилище. Еще один фактор, который нужно принять во внимание, состоит в том, что диаграммы связи сущностей как правило порождают нормализованную схему базы данных, тогда как в хранилищах часто используется денормализованная схема.

2.1.2. Пространственное моделирование

Альтернативой модели связи сущностей является пространственная модель, которая отличается от предыдущей тем, что смотрит на данные с другой точки зрения. Вместо рассмотрения сущности, которая представляет собой некоторую вещь, такую как продукт или место, и связей между такими сущностями, пространственная модель описывает данные с помощью измерений (dimensions) и фактов (facts), которые и становятся конкретными таблицами базы данных.

Пространственная модель, как показано на рис. 2.1, несмотря на то, что она иногда выглядит очень простой, обеспечивает очень эффективный способ хранения текущих и исторических данных в такой форме, которая делает эти данные доступными пользователям и дает Им возможность принимать правильные бизнес-решения. Пространственно организованное хранилище данных можно считать содержащим:

- проверенные данные (никаких неверных кодов продуктов).
- исторические данные (за последние 36 месяцев).
- интегрированные данные (один и тот же ключ используется всеми системами).
- легкодоступные данные.

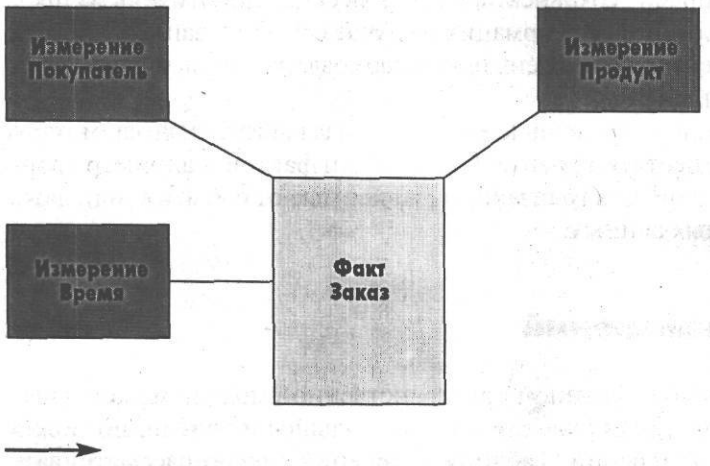


Рис. 2.1. *Пространственное моделирование*

2.1.3. Таблица фактов

Таблица фактов (таблиц может быть более одной) содержит фактическую информацию, является обычно самой большой таблицей в хранилище и, как правило, быстро растет. Именно в таблицах фактов, как правило, сохраняются подробные данные, которые вы хотите держать в хранилище, например все телефонные звонки, сделанные клиентом, или, как показано на рис. 2.1, заказы, сделанные покупателем.

Следовательно, если клиент совершил 20 телефонных звонков, то, скорее всего, в таблице фактов, относящейся к данному клиенту, будет сохранено 20 строк.

В результате таблицы фактов, безусловно, будут самыми крупными таблицами базы данных, и в больших хранилищах они могут содержать сотни миллионов строк. Если вы не уверены в том, что данные являются фактическими, то можно сказать, что они часто являются числовыми, а иногда представляют собой значение, которое можно подсчитать, например стоимость заказа или число купленных товаров.

Информация, содержащаяся в таблице фактов, не обязательно представляет собой наиболее подробный ее уровень. Это могут быть соединенные данные, как, например, телефонные звонки, сделанные клиентом за сегодняшний день. Степень подробности данных, хранимых в таблице фактов, называется глубиной детализации (granularity) и является одним из важных проектных решений, которое должен принять разработчик хранилища данных. В описанном здесь примере разница в количестве записей, сохраненных за 24 месяца, будет очень велика. Сравните объем регистрируемой информации в случае создания записи для всех телефонных звонков клиента за один день и в случае создания записи для каждого телефонного звонка клиента.

При разработке хранилища вы, в зависимости от рода вашего бизнеса, можете столкнуться с тем, что существуют разные виды таблиц фактов, например уровня транзакций, уровня элементов транзакций, зависимые от событий, или даже таблицы для объединенных данных.

2.1.4. Таблица измерений

При проектировании с использованием пространственной модели может существовать только одна таблица фактов или лишь небольшое их число, но может быть множество таблиц измерений. Таблицу измерений можно рассматривать как справочную таблицу по отношению к таблице фактов, где хранятся описания и более статическая информация об элементе данных. Например, «продукт» считается измерением потому, что в данной таблице сохраняется вся информация о продукте, в частности его полное название, поставщики и габаритные размеры. В таблице фактов будет содержаться столбец под названием «product_key», используемый для извлечения из таблицы измерения всей информации о продукте.

Если вы не знаете, являются ли некоторые данные измерением или фактом, задайте себе такой вопрос: «Являются ли эти данные относительно статичными?». Как правило, измерения, например идентификатор продукта, меняются нечасто, в то время как таблица фактов будет содержать сведения о продажах продуктов. Обычно число строк в таблице фактов на порядок больше числа строк в таблице измерения.

Таблица фактов содержит миллионы строк, тогда как таблицы измерений — только несколько (например, измерение Время может содержать 52 строки данных, которые сохранялись понедельно в течение года). Или измерение Регион может содержать только 15 строк, если в стране имеется только 15 регионов. Измерения не обязательно должны быть маленькими. Вы можете продавать 50000 продуктов или иметь измерение Покупатели с 5 миллионами строк. Все это — примеры допустимых измерений.

Трудно сказать, сколько измерений потребуется для вашего случая, но как правило, бывает не более 20 измерений и не менее 4. Следовательно, наше хранилище будет состоять всего из нескольких таблиц, но у него будут большие запросы на объемы записанной информации из-за количества строк в таблице фактов.

2.1.5. Ключи хранилища

Данные, скорее всего, будут поступать в хранилище из многих источников, и код продукта в одной системе может отличаться от другой. Еще одна проблема состоит в том, что, когда данные хранятся в течение определенного периода времени, ключи, применяемые в рабочей системе, могут использоваться повторно. Следовательно, разработчик должен серьезно подумать о реализации ключей-заменителей (surrogate keys), которые имели бы полный контроль над идентификацией данных в хранилище. Преобразование ключа рабочей системы в ключ хранилища обрабатывается в ходе загрузки данных при помощи ETL и потребляет незначительный объем ресурсов. Преобразовать в ключ-заменитель можно любой ключ, включая временные ключи (time keys). Ключи-заменители не должны быть слишком сложны и могут просто начинаться с единицы, а далее последовательно увеличиваться при помощи последовательностей Oracle. Реализовав ключи-заменители, можно сэкономить место для хранения информации.

2.1.6. Нормализация хранилища

Когда доходит до решения вопроса о том, должны ли данные в хранилище быть нормализованы, не все выбирают одно и то же. Некоторые эксперты считают, что хранилище нужно нормализовать, тогда как другие уверены, что более подходящей является нормальная пространственная форма.

Нормальная пространственная форма — вещь довольно интересная, поскольку она выглядит как комбинация нормализации и денормализации. На рис. 2.2 мы видим разницу между двумя подходами к хранению измерения «Магазин».



Рис. 2.2. *Нормализованная и нормальная пространственная формы*

Нормализованную форму также называют снежинкой (snowflaking). Хотя Oracle 9i допускает использование нормализованных измерений, будьте осторожны с этим подходом. Одним из его недостатков является то, что он может повлиять на производительность, поскольку в запросах понадобится большее число соединений, что потребует времени. Форма снежинки является хорошим примером техники, используемой в системах обработки транзакций, но она не всегда подходит для хранилища данных.

На первый взгляд, кажется, что нормальная пространственная форма дублирует данные. Хотя можно подумать, что это непроизводительная трата места, на самом деле, число строк в измерениях, как правило, очень мало по сравнению с размерами таблицы фактов. Поэтому вы, возможно, удивитесь, узнав, что хранение этих дополнительных данных обойдется вам всего в несколько десятков мегабайт. Преимущество этого подхода в том, что для доступа к информации в данной модели требуется теперь только два навигационных уровня, что облегчает создание запросов, быстро возвращающих ответы.

2.1.7. Хранилище или тематическое хранилище?

Альтернативой созданию больших хранилищ является создание лавок данных, где каждое тематическое хранилище содержит часть данных большого хранилища.

Преимущество тематических хранилищ — в акценте на одной области бизнеса, так что в них можно хранить данные по региону или департаменту. Однако при использовании лавок данных следует быть осторожным. Хотя несколькими тематическими хранилищами (в отличие от одного источника) легче управлять и они идеальны для создания отчетов, интегрировать данные может быть чрезвычайно сложно. В результате можно получить несколько лавок, содержащих дублирующиеся данные, которые нельзя связать друг с другом. Тем не менее, довольно распространенной является практика создания сначала нескольких лавок данных, которые затем используются в качестве основы для создания хранилища масштаба всего предприятия.

2.1.8. Пример. Компания Easy Shopping Inc.

На всем протяжении этой книги мы будем использовать пример с придуманной нами компанией под названием Easy Shopping Inc. Это организация, которая не имеет розничных торговых точек и осуществляет продажи через Интернет-сайт или через систему спутникового или кабельного телевидения. На рис. 2.3 показана пространственная модель для Easy Shopping Inc.

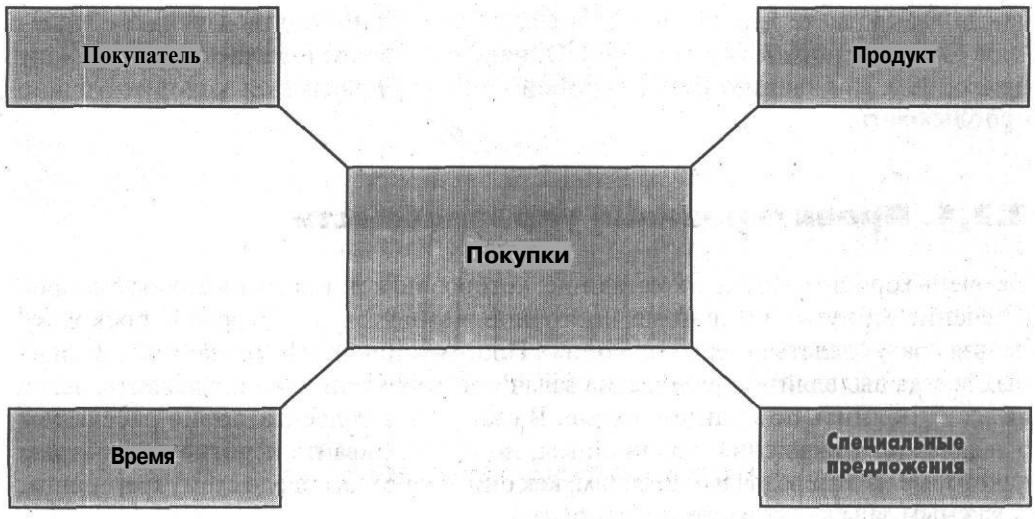


Рис. 2.3. *Пространственная модель для компании Easy Shopping Inc.*

В данном примере у нас имеется таблица фактов, называющаяся «Покупки», где мы записываем все купленные покупателями товары. Определено четыре измерения, в которых хранятся сведения о покупателях, продуктах, времени и деталях наших ежедневных специальных предложений. Хотя это простой пример, взятый для иллюстрации данной книги, однако то, что вы будете создавать сами, будет ненамного сложнее. У вас будет больше измерений и намного больше столбцов в таблице фактов.

Схемы хранилищ иногда называют звездообразными схемами, и рис. 2.3 — пример такой схемы. Центральная точка — это таблица фактов, а измерения располагаются вокруг таблицы фактов, подобно лучам звезды. Как и в диаграмме связи сущностей, нарисовав пространственную модель, вы можете легко перевести ее в физическую схему базы данных, поскольку каждый квадрат представляет таблицу базы. Хотя в этой книге мы говорим о таблицах фактов и таблицах измерений, внутри базы данных Oracle все они — просто таблицы и рассматриваются именно в этом качестве. Однако перед тем как приступить к работе и создавать физическую базу данных по ее пространственной модели, нужно принять еще несколько решений перед тем как схема будет готова.

2.2. Прочие вопросы дизайна

Когда вы создаете хранилище данных, легко можно задаться целью обеспечить быструю обработку запросов. Однако слепое следование этому подходу приводит к появлению базы, которой сложно управлять и которую трудно использовать.

2.2.1. Проектирование управляемости

Не очень хорошо создавать хранилище, которое может ответить на любой вопрос в течение минуты, но данные постоянно находятся под угрозой, поскольку нельзя сразу сделать резервную копию. Поэтому при проектировании базы данных всегда выявляйте критические задачи по управлению и определяйте, легко ли их выполнить при данной схеме. В главе 7 мы более подробно расскажем о задачах по управлению хранилищем, но сейчас давайте коротко рассмотрим некоторые из этих задач и определим, как они влияют на общую схему хранилища. К важным задачам управления относится:

- создание резервных копий;
- загрузка новых данных;

- объединение новых данных;
- задачи по обслуживанию, такие как индексирование и архивация.

Во всех базах данных нужно регулярно проводить резервное копирование. В Oracle 9i есть утилита RMAN, которая позволяет создавать резервные копии в режиме он-лайн, а также проводить инкрементное резервное копирование измененных данных. На первый взгляд, резервное копирование может показаться тривиальной задачей, однако создание резервной копии хранилища размером в терабайт требует времени, даже если оно проводится в режиме он-лайн. Следовательно, разработчик должен тщательно продумать табличные пространства (объясняются в главе 3), в которых хранятся данные, чтобы сделать резервные копии табличных пространств, предназначенных только для чтения, или использовать секционирование (partitioning) (описывается в главе 3), чтобы можно было сделать полную резервную копию, запустив несколько параллельных задач. Вы можете даже использовать для резервного копирования хранилища систему Data Guard.

В очень больших хранилищах полное резервное копирование, скорее всего, будет роскошью. Следовательно, нужно разработать хранилище так, чтобы можно было проводить инкрементное резервное копирование, захватывающее только изменения, вносимые в данные хранилища. Из-за огромных объемов данных в хранилище любая операция, которую можно выполнять параллельно, значительно увеличивает время, требуемое для выполнения задачи, особенно если одновременно запущены несколько параллельных процессов. Большинство новых данных будет сохраняться в таблице фактов, следовательно, на этапе проектирования разработчик должен определить, когда и сколько данных будет загружаться. Затем подсчитывается ожидаемое время загрузки, и, если ввести все данные за отведенное время не получается, следует подумать об использовании таких методов, как секционирование, чтобы данные можно было вводить параллельно.

Когда группа разработчиков тестирует хранилище, нужно концентрироваться не только на тестах производительности. Это позволит определить, сколько времени будет занимать загрузка, и обсудить с отделом эксплуатации то, каким образом будет проводиться резервное копирование и сколько времени для этого необходимо.

В главе 4 мы увидим, как при обслуживании объединенных данных можно использовать управление суммированием (summary management), которое было предложено в Oracle 8i. Одним из методов повышения производительности, широко применяемых в хранилищах, является создание суммарных таблиц (summary tables) данных перед объединением, которые известны в Oracle 9i как материализованные представления (materialized views). Теперь при запросе, вместо чтения всех данных, можно явным образом прочитать материализованное

представление. Выигрыш в производительности может быть огромен и будет зависеть от того, насколько число строк в материализованном представлении меньше, чем в основных данных.

К сожалению, бесплатно в этом мире ничего не бывает и материализованные представления нужно обслуживать. Это довольно значительно задействует каналы ввода-вывода, и нагрузка зависит от того, сколько добавляется новых записей и создается ли материализованное представление с нуля или к нему просто добавляются данные. Если определено много материализованных представлений и все они обновляются одновременно, то нужно подумать о том, чтобы разместить их в разных табличных пространствах или на разных дисках. Если этого не сделать, все каналы ввода-вывода для данного диска будут перегружены, что значительно замедлит процесс обновления. Это особенно важно, если операции обновления выполняются параллельно.

2.2.2. Проектирование производительности

Как мы увидим в главах 3 и 4, существуют различные методы улучшения производительности обработки запросов, которые может применять разработчик. Некоторые из них используют способы составления запросов, но многие относятся, собственно, к схеме базы данных. Например, любая база данных выиграет от индексирования, и хранилище данных — не исключение. Поэтому не забудьте определить тип индекса, его местоположение и объем места для него: В случае хранилища разработчику не нужно беспокоиться о множестве пользователей, которые вносят новые элементы в индекс, и обо всех проблемах, которые этим порождаются. Сейчас разработчика заботит время, необходимое для обслуживания или создания индекса. Например, повторное создание индекса для таблицы фактов в 100 миллионов строк потребует более нескольких минут!

В Oracle 9i предлагаются разные типы индексов, и разработчик должен выбрать один наиболее подходящий. Это может означать, что в схему могут входить много индексов на основе битовых карт (bitmapped indexes). Разные типы запросов к хранилищу будут использовать разные методы доступа к данным, которые могут пользоваться преимуществами разных типов индексов (например, преобразование по типу звезды использует индекс на основе битовых карт).

Еще один важный вопрос — физическое размещение данных, особенно если используется секционирование и параллельные операции. Если данные физически расположены на разных дисках, тогда задачи и запросы не будут перенасыщать каналы ввода-вывода конкретного диска.

Одним из важных улучшений производительности является использование материализованных представлений, описываемых в главе 4. Поскольку их нужно создавать и обслуживать, для этих данных нужно зарезервировать место и нужно сохранять равновесие между улучшениями времени обработки запроса и временем, необходимым для обслуживания этих данных.

Новые данные в хранилище часто поступают пакетами. Есть надежда, что они будут загружены в базу, когда она не будет использоваться, но гарантировать это нельзя. Поэтому, если при исследовании предлагаемой системы вы обнаружили, что данные будут загружаться в ходе использования хранилища, изучите методы, которые позволяют вставлять данные в область, не задействованную пользователями (например, секционирование).

Еще один вопрос — возможность обновления таблицы фактов. При таком количестве записей это может весьма существенно повлиять на производительность. Поэтому необходимо установить процедуры, которые будут препятствовать несанкционированным обновлениям таблицы фактов.

2.3. Реализация схемы

Когда схема базы данных кажется нам удовлетворительной, пора приступать к физическому созданию нашей базы. Сначала нужно создать маломасштабную версию базы и протестировать все идеи на ней, перед тем как строить полномасштабную рабочую систему. Существуют разнообразные средства, которые могут помочь разработчику создать хранилище. Их мы обсудим в других главах. Здесь же мы увидим, как можно создать нашу базу, не применяя ничего более изощренного, чем SQL.

Разработчики баз данных часто предпочитают создать файл скрипта, содержащий команды SQL по созданию базы, и это совершенно приемлемый подход. Альтернативный путь — использование средств с графическим пользовательским интерфейсом (Graphical User Interface, GUI), входящих в Oracle Enterprise Manager, применение которых будет нами проиллюстрировано.

Совет. Если вы считаете SQL сложным, используйте для создания команд SQL средства с графическим интерфейсом, а затем вставьте код SQL в свой текстовый файл.

В данном разделе мы пройдемся по различным стадиям создания хранилища данных. Во-первых, мы увидим, как создать саму базу данных, а потом узнаем,

как создавать табличные пространства и файлы данных, в которых, собственно, и хранятся данные. Далее последуют пояснения по созданию таблиц и краткое описание создания индексов и разделов. И наконец, мы обсудим создание материализованных представлений и то, как обезопасить объекты в только что созданном хранилище.

2.3.1. Одна база данных или много?

Было время, когда хранилище создавалось в отдельной базе данных. Но времена меняются, и теперь некоторые компании предпочитают иметь одну базу данных, содержащую в себе все системы.

Каждый подход имеет свои «за» и «против», и выбор между одной или несколькими базами данных зависит от требований вашего бизнеса. Создание новой базы данных — дело нетрудное, и лучше всего использовать для этого средство с графическим интерфейсом — Oracle Database Configuration Assistant. Как только база создана, вы можете добавлять в нее свои собственные файлы данных и табличные пространства с помощью Oracle Enterprise Manager Console или скриптов SQL.

Базу данных можно создать прямо с помощью SQL, но, если вы используете этот подход, нужно делать это осторожно, поскольку вам придется запустить множество файлов скриптов, необходимых для Oracle 9i. Если вы используете инструменты с графическим интерфейсом, все это производится автоматически.

2.3.2. Соглашения об именах

Перед тем как сделать что-либо с базой данных, нужно изучить соглашения об именах, используемых для всех объектов базы, таких, как файлы данных, табличные пространства, названия таблиц и столбцов. Некоторые средства, находящиеся в распоряжении конечных пользователей, могут позволять им увидеть названия таблиц и столбцов. Поэтому при наличии непонятных имен, пользователи, которые, как правило, не очень знают компьютер, могут быть введены в заблуждение.

В нашем примере с компанией Easy Shopping Inc. в таблице фактов есть столбец, названый «time_key» (ключ_времени). Для тех, кто знаком с базами данных, содержимое этого поля очевидно, но для конечных пользователей базы это не говорит ничего. Поэтому в данном случае лучше назвать столбец

«дата_и_время_покупки». Разработчик хранилища должен также помнить, что если хранилище будут использовать конечные пользователи, нужно давать имена столбцов более похожие на естественный язык.

Пока что мы не затрагивали тему метаданных, но в хранилище метаданные играют очень важную роль. Должно быть одно определение элемента данных, а элемент данных в идеальных условиях должен иметь только один набор значений. Например, предполагается, что код региона представляет собой значение, состоящее из трех буквенно-цифровых символов, и это соблюдается во всех системах, кроме одной, где код региона определен как цифра.

Есть несколько проблем и у команды, отвечающей за загрузку данных в хранилище. Они обсуждаются ниже, в главе 5, где мы применим все необходимые преобразования данных таким образом, чтобы обеспечить единообразие данных, когда они будут находится в хранилище.

2.3.3. Oracle Database Configuration Assistant

В Oracle 9i входит несколько средств с графическим интерфейсом, которые помогут управлять базой данных. Очень полезным инструментом является Oracle Database Configuration Assistant, с помощью которого вы можете создавать, удалять и модифицировать базу данных или управлять шаблонами, используемыми для создания базы. Данный инструмент запускается в одиночку (найдите файл DBCA в директории %ORACLE_HOME%/bin) и не требует запущенного Oracle Enterprise Manager Server. В Oracle 9i он поставляется с тремя уже сконфигурированными базами данных, и выбор одной из них значительно уменьшает время, необходимое для создания базы. Вы также можете создать новую базу данных.

После того как вы приняли решение о том, какой тип базы данных вам требуется, ей надо дать имя. Как показано на рис. 2.4, мы назвали нашу базу данных EASYDW.

Дальнейшие шаги будут различаться в зависимости от того, создается уже сконфигурированная или новая база данных.

Использование сконфигурированной базы данных

С Oracle 9i поставляется три типа сконфигурированных баз.

1. Хранилище данных (warehouse).
2. База данных общего назначения.
3. Система обработки транзакций (OLTP).

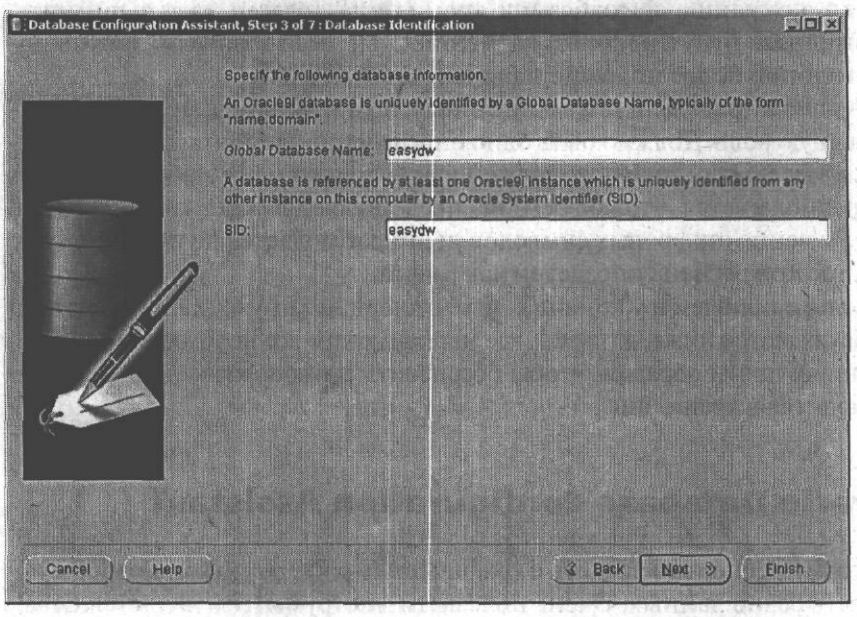


Рис. 2.4. *Oracle Database Configuration Assistant — наименование базы данных*

Если вы не уверены в том, какой из этих типов подходит для вашей ситуации, физические атрибуты базы данных и установленные компоненты можно посмотреть, щелкнув по кнопке Show Details (Подробнее). Будет выведен используемый шаблон.

При использовании сконфигурированной базы данных ввод информации требуется всего в нескольких окнах. В одном из первых определяются некоторые параметры, используемые для конфигурирования базы. В базе данных Oracle 9i есть несколько конфигурируемых компонентов. На рис. 2.5 мы видим пятый из семи шагов в Oracle Database Configuration Assistant, в котором мы можем определить параметры соответствующих областей, щелкнув по соответствующим закладкам:

- Memory (Память);
- Character sets (Наборы символов);
- Memory used during sort operation (Память, используемая при сортировке);
- File locations for initialization and trace files (Местоположения файлов инициализации и трассировки);
- Archiving (Архивация).

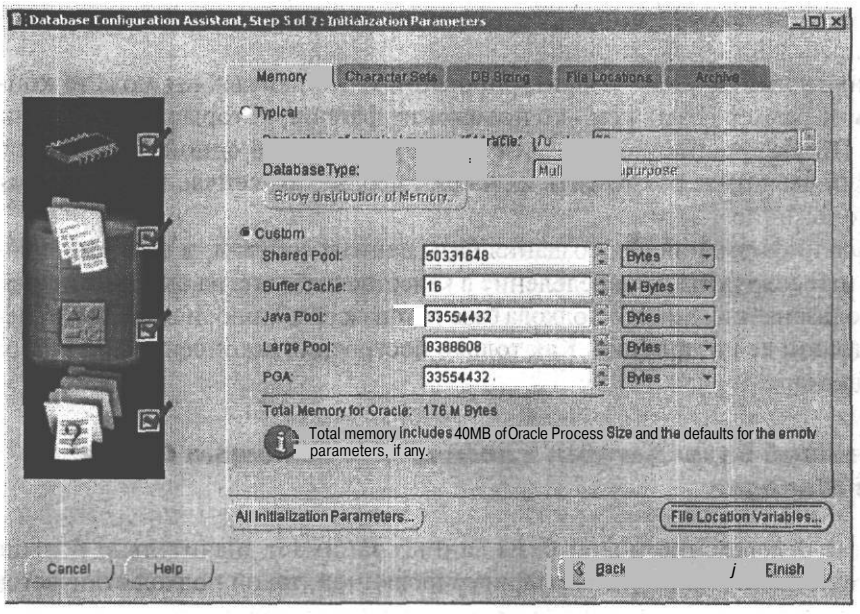


Рис. 2.5. *Oracle Database Configuration Assistant — параметры инициализации*

Параметры, установленные по умолчанию, могут подходить для вашей ситуации. Однако если вы создаете небольшую базу для проверки некоторых свойств и доступная память у вас ограничена, щелкните по кнопке Custom (Выбрать) и внесите изменения в объем выделяемой памяти. Вы можете сразу увидеть результат своих изменений, посмотрев значение Total Memory for Oracle (Всего памяти для Oracle).

Совет: Если выбранное значение параметра слишком низко, вы получите предупреждение и возможность его увеличить.

Одна из кнопок на рис. 2.4 — File Location Variables (Переменные местоположения файлов). Когда база данных создается, положение всех файлов, составляющих базу, т. е. файлов данных, управляющих файлов, файлов инициализации, файлов буфера отмены и архивированных файлов, задается в соответствии со значениями этих параметров, которые вы можете сконфигурировать в базе, как показано на рис. 2.6.

Также в этот момент вы можете определить свои собственные переменные и использовать их в определении базы данных.

При использовании уже сконфигурированных баз данных вы можете контролировать не размер базы, а только положение файлов, которые перечислены на рис. 2.6. По умолчанию все эти файлы расположены в одной директории. Поэтому, если вы хотите разместить их на разных дисках, сейчас как раз время указать их положение.

Теперь вся информация для создания базы данных собрана, и вы можете создавать базу или сохранить ее определение в качестве шаблона на следующий раз.

При использовании данного подхода база готова к созданию, и вы можете следить за процессом ее построения. Как только построение закончено, база готова к использованию.

Создание новой базы данных с помощью с помощью Oracle Enterprise Manager

Хотя создание совершенно новой базы данных занимает значительно больше времени, чем использование уже сконфигурированной, такой подход отличается

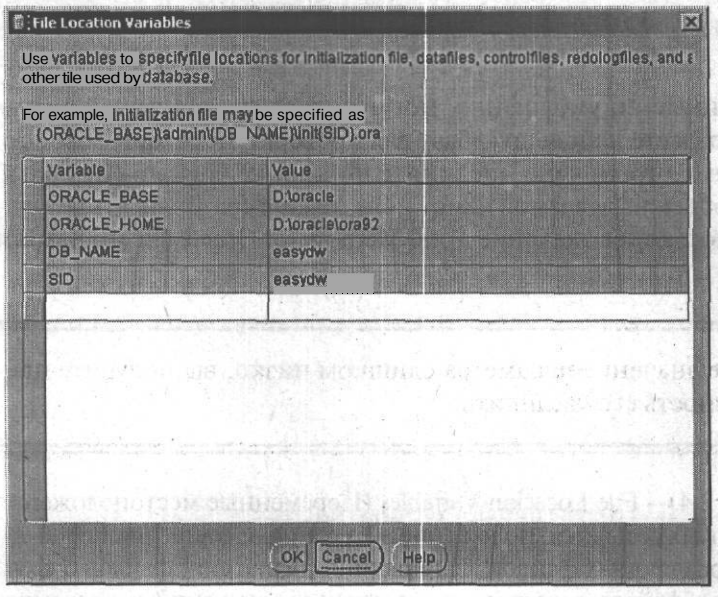


Рис. 2.6. *Oracle Database Configuration Assistant — переменные местоположения файлов*

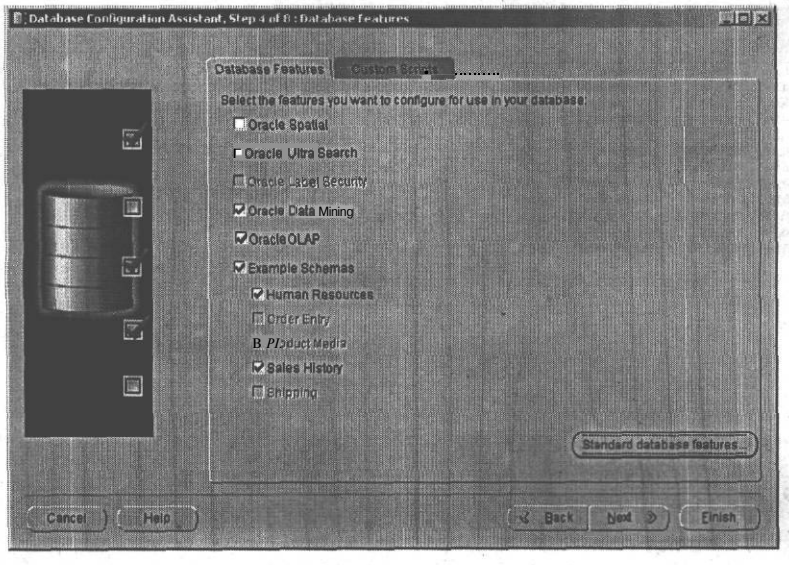
большей гибкостью и, соответственно, шаги, которые необходимы для выполнения этой задачи, будут другими.

Когда база данных получила имя, как показано на рис. 2.4, вам будет предложено указать, какие свойства вы желаете видеть в своей настраиваемой базе. На рис. 2.7 можно видеть, что есть несколько свойств, которые мы не хотим устанавливать. Например, мы решили не включать Oracle Spatial, но нам нужно Oracle Data Mining.

Совет. Время создания базы данных можно значительно уменьшить, если устанавливать только необходимые свойства.

Наряду с этими свойствами можно инсталлировать несколько шаблонных схем, в которых даются примеры, поясняющие функциональность данных свойств.

Затем нужно будет указать параметры, показанные на рис. 2.5. Можно оставит значения, определяемые по умолчанию, или же, при необходимости, их можно заменить. При щелчке по закладке DB Sizing (Размер базы данных) появляется возможность указать размер блока базы, определяющий самый низший уровень



Рте. 2.7. *Oracle Database Configuration Assistant — свойства базы данных*

детализации и контроля над выделением места в хранилище. Рекомендуется создавать хранилище с большим размером блока, например 16384 байта, что даст вам возможность хранить больше записей одного типа в одном блоке, а это в свою очередь снизит нагрузку на каналы ввода-вывода.

В следующем окне, показанном на рис. 2.8, можно создавать, удалять или изменять любые файлы, составляющие базу данных.

С помощью этого окна можно добавлять табличные пространства, щелкая по кнопке Add (Добавить). В данном случае мы видим, что было добавлено новое табличное пространство с названием SUMMARY. Некоторые табличные пространства создаются автоматически, и, раскрывая пункты навигационного окна слева и выбирая в нем элементы, вы можете изменять размеры и характеристики этих табличных пространств.

На этом этапе, нажимая на кнопку Add (Добавить), также можно определять файлы данных, сегменты буфера отката или групп протоколов возврата (Redo Log Groups).

Подготовка к созданию базы данных практически завершена. Теперь вы можете создать базу данных, записать данную конфигурацию в качестве шаблона для последующего использования и, по желанию, потребовать создать скрипт SQL, чтобы базу данных можно было создать позже.

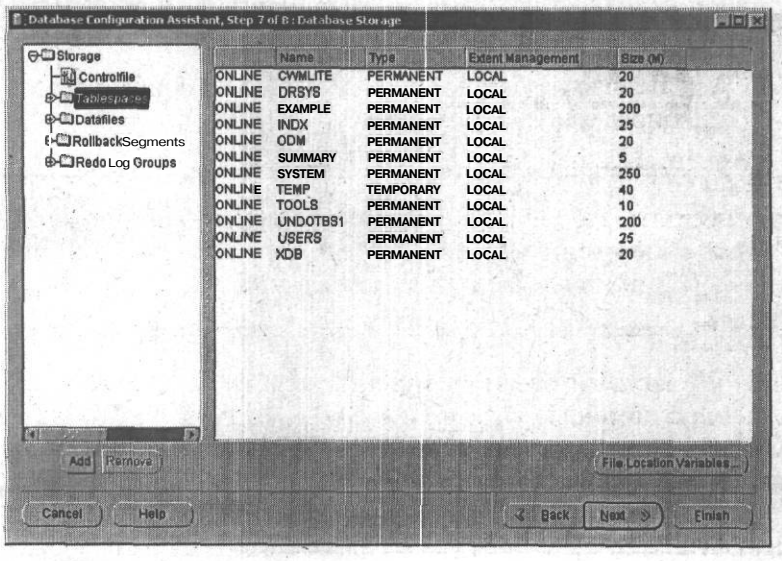


Рис. 2.8. *Oracle Database Configuration Assistant — хранение базы данных*

Если вы запросите создание скрипта SQL, несколько SQL-файлов будет записано в указанное вами место, и там будет один главный скрипт, с названием, соответствующим имени базы данных, указанному, как показано на рис. 2.4, который будет запускать все остальные скрипты.

Если вы решите создать базу данных из средства с графическим интерфейсом, вы сможете понаблюдать за процессом ее создания. Как уже говорилось ранее, в ходе создания базы запускаются несколько скриптов, обеспечивающие определенные функции. Создание базы данных занимает много минут, так что, запустив его, сделайте небольшой перерыв или займитесь чем-нибудь еще.

2.3.4. Какая схема?

Теперь, когда у нас есть база данных, все объекты, определяемые в базе Oracle, должны располагаться внутри схемы (schema), логической структуры, описывающей коллекции объектов. Поэтому, перед тем как создавать таблицы, вам нужно решить, сколько схем вам необходимо. Поскольку хранилище может содержать лишь несколько таблиц, вероятно, неплохо держать их все в одной схеме.

Однако вы можете предпочесть создать несколько схем по тематическим областям, но такой подход также повысит необходимые затраты на управление базой, чтобы сделать доступной информацию, содержащуюся в таблицах. Это объясняется тем, что появятся дополнительные задачи, такие как предоставление доступа к таблицам и определение синонимов.

Совет. Очень важно принять это решение в самом начале проектирования, поскольку схема играет существенную роль в соглашении об именах, которое используется при извлечении информации из базы данных.

Объект схема создается каждый раз при определении пользователя базы. Пользователь создается с помощью выражения SQL CREATE USER или из Oracle Enterprise Manager путем выбора пункта User (Пользователь) из списка объектов и заполнения окна, показанного на рис. 2.9.

В нашем примере с компанией Easy Shopping Inc. мы решили создать пользователя с именем EASYDW. Здесь мы можем указать, как этот пользователь будет аутентифицироваться, и мы выбрали режим, указанный по умолчанию, — By Password (По паролю). Это означает, что должен быть указан пароль, который

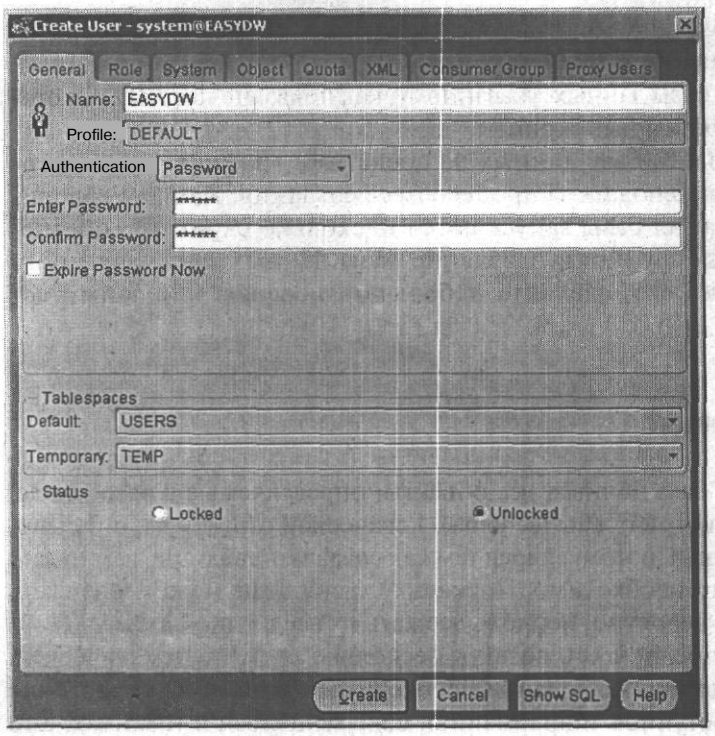


Рис. 2.9. *Oracle Enterprise Manager — создание* ***пользователя***

будет использоваться для всех последующих операций доступа к базе данных. Если этот пароль в будущем понадобится изменить, это можно будет сделать из Oracle Enterprise Manager, в окне, показанном на рис. 2.9, или с помощью SQL.

Мы также должны указать область по умолчанию, где будут располагаться все объекты, созданные этим пользователем, например таблицы и индексы. В хорошо спроектированной базе это не проблема, поскольку для каждого объекта будет явным образом указано, в каком табличном пространстве его следует сохранить. Есть также и другие опции, которые можно указать для пользователя, например привилегии. Установить их важно. В противном случае вы не сможете извлекать данные. По ходу этой книги мы будем информировать вас о том, какие привилегии необходимы для обсуждаемой темы.

При создании пользователя автоматически создается схема с тем же именем. Однако вы не сможете увидеть эту схему, пока не будет определен первый объект для данного пользователя, например таблица.

Имя схемы играет очень важную роль, поскольку используется для полной идентификации объекта в базе данных. Например, у нас может быть таблица с именем TIME в схеме EASYDW, и еще одна — в схеме ORDERS. Чтобы проинформировать оптимизатора, из какой таблицы вы желаете извлечь данные, вы указываете имя таблицы следующим образом:

```
имя схемы. имя таблицы
```

Следовательно, чтобы извлечь все записи из нашего измерения Время, мы должны однозначно указать имя таблицы следующим образом:

```
SELECT * FROM easydw. time;
```

Когда Oracle Enterprise Manager запускается в первый раз, в левой части отображается стандартное окошко навигации. Если вы щелкнете по пункту Shema (Схема), будет выведен список всех пользователей, определенных в базе данных, как это показано на рис. 2.10.

Вы можете создать столько пользователей базы данных, сколько вам нужно, но рекомендуется, чтобы только один из них использовался для создания объектов, таких как таблицы и индексы. Следовательно, когда администратор базы данных соединяется с системой от имени пользователя EASYDW, все таблицы и индексы будут располагаться здесь, как показано на рис. 2.10. В разделе 2.3.11 мы обсудим указание привилегий для пользователя.

2.3.5. Файлы данных и табличные пространства

Когда база данных создана, вы можете добавлять в нее свои собственные файлы данных и табличные пространства (tablespaces). По умолчанию будет существовать несколько файлов данных. В системах Windows NT их можно найти в директории

```
\<Oracle_Home>\Oradata\<имя базы данных>\
```

Сюда входят два управляющих файла, область индексов, системная область, файл отката и пользовательская область.

Именно внутри этих табличных пространств и хранятся объекты базы данных Oracle, и табличное пространство может иметь один или более файлов данных, связанных с ним.

Следовательно, табличное пространство представляет собой логическое имя, используемое внутри схемы базы данных для указания места, в котором должны

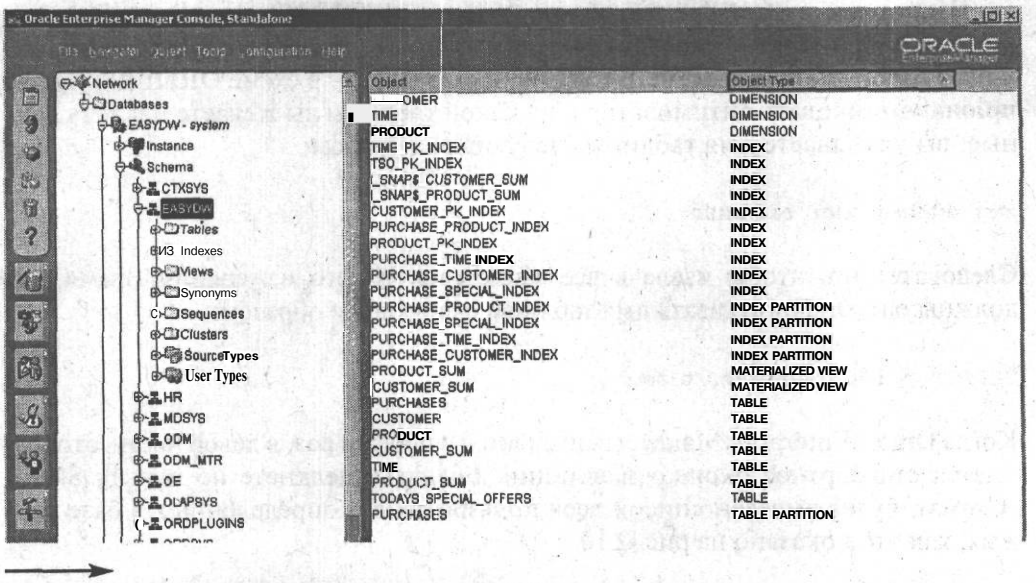


Рис. 2.10. *Консоль Oracle Enterprise Manager — схемы*

находиться объекты. Затем объект фактически сохраняется в одном или более файлах данных, и частью определения табличного пространства будут физическое местоположение и размер этих файлов.

В нашем примере с компанией Easy Shopping Inc. мы решили реализовать следующие табличные пространства.

- Измерения (Dimensions) — для всех данных из измерений.
- Область по умолчанию (Default area), которая выделяется пользователю по умолчанию.
- Суммарные данные (Summary) — для создаваемых нами материализованных представлений.
- Год и месяц покупки(Purchases_month_year) — для таблицы фактов, в которой применяется секционирование.
- Индексное табличное пространство (Index tablespaces) — для индексов.
- Временная область (Temp area) — в качестве временного места хранения.

В данном примере мы создадим для каждого табличного пространства только один файл данных, но вы, конечно, можете при необходимости создать и больше. Кроме того, показанные здесь файлы будут очень малы, а в реальной ситуации они могут оказаться чрезвычайно большими.

Данные табличные пространства и связанные с ними файлы данных можно создать либо напрямую, с помощью SQL, либо используя такое средство, как Oracle Enterprise Manager Console, являющееся частью Oracle Enterprise Manager и показанное здесь на рис. 2.11.

Если вы управляете базой данных с помощью Oracle Enterprise Manager, это значит, что вам больше не надо запрашивать метаданные, чтобы получить информацию о состоянии и сведения об объектах базы данных, поскольку эта информация уже доступна.

На рис. 2.11 показана база данных EASYDW. Мы выбрали базу EASYDW, потом пункт storage (хранение), потом — tablespaces (табличные пространства), и теперь в правом окне мы видим список табличных пространств, их размеры, текущее состояние и используемое место.

Если вы предпочитаете создавать табличные пространства напрямую, через SQL, вы можете использовать утилиту SQL*Plus. Если вы не уверены в том, какой код SQL требуется для решения необходимой задачи, нажмите в большинстве средств GUI кнопку Show SQL (Показать SQL), и в нижней части экрана появится окно, содержащее код SQL. На рис. 2.12 можно видеть код, создающий табличное пространство для измерений.

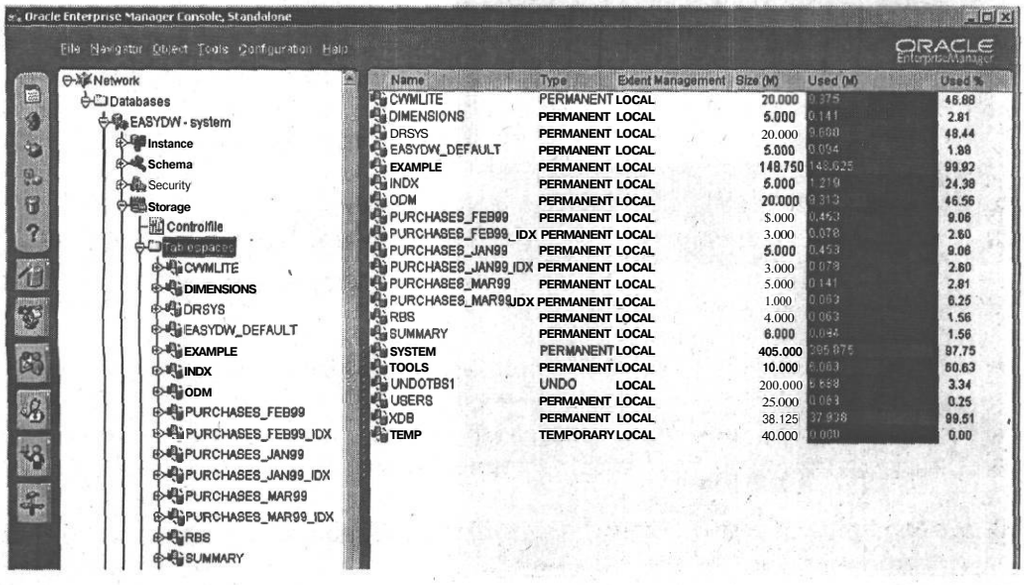


Рис. 2.11. *Oracle Enterprise Manager Console — табличные пространства*

Создавать табличные пространства при помощи Oracle Enterprise Manager очень просто. Просто выберите пункт database (база данных), потом Storage (Хранение), затем Tablespaces (Табличные пространства) и наконец щелкните правой кнопкой мыши и выберите пункт Create (Создать). Появится окно, показанное на рис. 2.12, в котором вы можете указать атрибуты данного табличного пространства.

Первый этап — это ввести имя табличного пространства. Лучше всего выбирать осмысленные имена, поскольку вы будете постоянно использовать их по всей схеме, и очень помогает, если имя имеет для вас какой-то смысл. Например, табличное пространство, названное DIMENSIONS, будет использоваться для хранения таблиц измерений, а покупки, сделанные в январе, будут храниться в табличном пространстве с именем PURCHASES_JAN2002. Еще один предпочтительный подход — использовать суффикс «D» для всех данных и «X» — для всех индексов, чтобы идентифицировать тип данных, хранящихся в табличном пространстве.

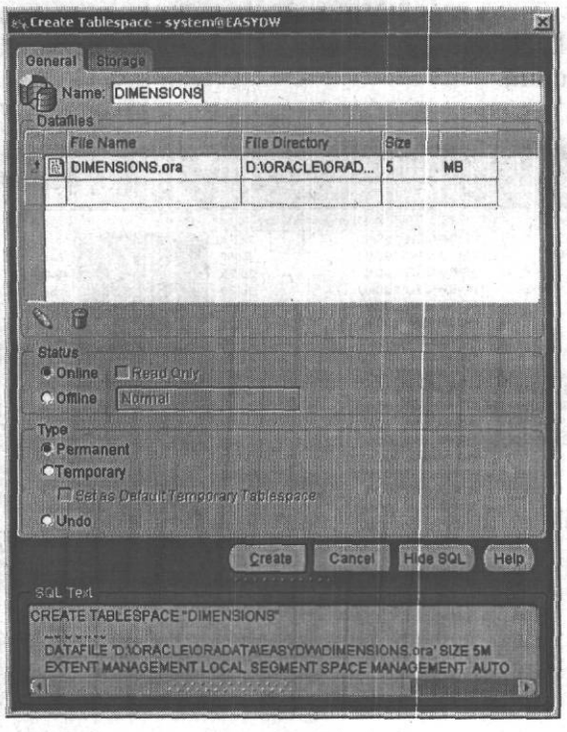


Рис. 2.12. *Oracle Enterprise Manager Console — измерения*

Чтобы указать положение файла данных, щелкните по столбцу File Directory (Директория для файлов) и измените положение, указанное по умолчанию. Тот же самый подход используется для изменения имени файла и его размера. Щелкните по закладке Storage (Хранение). Появится страница, в которой вы можете увидеть файлы, для которых будет проводиться автоматическое выделение памяти. Некоторым разработчикам не нравится возможность автоматического расширения файла, но многие это делают, поскольку это гарантирует, что выполнение задачи не сорвется из-за исчерпания места в базе данных. Файл данных будет автоматически расширен, и вы можете контролировать степень этого расширения. Конечно, если диск будет заполнен, автоматическое расширение не сработает.

2.3.6. Создание таблиц фактов и измерений

Теперь, когда у нас есть база данных и в ней определены пользователи и табличные пространства, мы готовы к созданию таблиц фактов и измерений. Таблицы фактов и измерений создаются так, как если бы они были обычными таблицами базы данных. Следовательно, все опции, которые можно указать для таблицы, например первоначальный и последующий размер, можно указать и здесь. Хотя мы называем эти таблицы таблицами фактов и таблицами измерений, они не отличаются от любой другой таблицы базы данных.

При определении таблицы фактов осторожно выбирайте типы данных для столбцов, поскольку выбор того типа, который занимает излишне много места, притом что таблица фактов содержит миллионы строк, будет способствовать значительной и бесполезной трате места на диске.

Таблицы создаются при помощи команды SQL CREATE TABLE, но мы сейчас увидим, как можно быстро создавать таблицы с помощью Oracle Enterprise Manager. На какой-то момент таблица измерения определяется так, как если бы она была единственной таблицей базы. В главе 4 мы увидим, как определить объект-измерение, который будет основан на созданной здесь нами таблице. Фактически таблица измерения, созданная на этой стадии, является предпосылкой создания объекта-измерения.

Если после запуска Oracle Enterprise Manager Console вы выберете Object (Объект), Create (Создать), Table (Таблица), Create (Создать), запустится Мастер создания таблиц (Table Wizard). Первое его окно показано на рис. 2.13.

В показанном на рис. 2.13 окне вы можете указать имя таблицы, схему, в которой она будет располагаться, и табличное пространство для этой таблицы. После щелчка по кнопке Next (Далее) появится окно, изображенное на рис. 2.14, в котором вы определяете столбцы, которые должны быть в таблице.

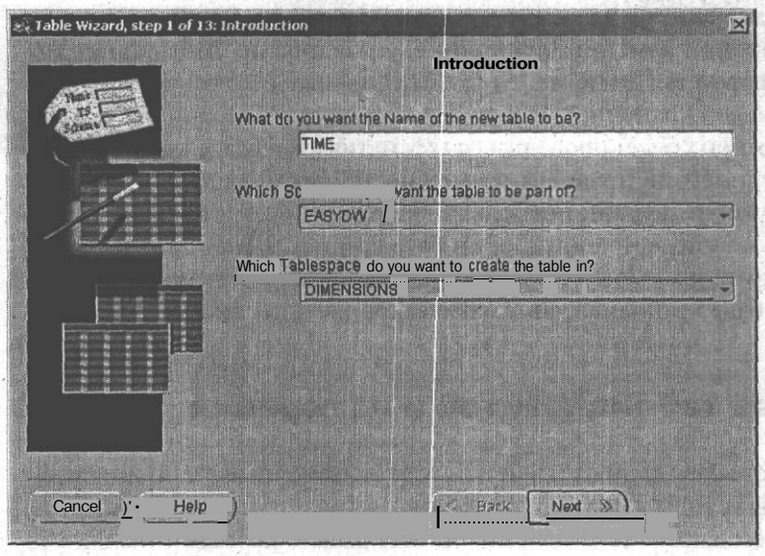


Рис. 2.13. *Oracle Enterprise Manager Console — Мастер создания таблиц*

Это действительно очень приятное средство для создания таблицы. Все, что вам нужно сделать, — это ввести имя столбца, выбрать тип данных и их размер, а затем нажать кнопку Add (Добавить), и столбец появится в списке левой части окна. Используя данный подход, определите все столбцы, и не щелкайте по кнопке Next (Далее), пока не определите всех столбцов таблицы.

Если вы нажали на кнопку Next по ошибке, вы можете вернуться к этому окну, нажав кнопку Back (Назад). Когда все столбцы таблицы определены, следующие окна дадут вам возможность установить ограничения.

2.3.7. Ограничения

Функции ограничений — проверка соответствия данных определенным правилам, например проверка принадлежности значения к определенному диапазону с помощью ограничения CHECK. Определение первичного ключа (primary key) гарантирует уникальность значения. Внешний ключ (foreign key) обеспечивает соответствие всех значений одному из первичных ключей в другой таблице.

Скажите об ограничениях разработчикам, и они, скорее всего, ответят, что они считают их присутствие в базе данных нежелательным, поскольку они создают

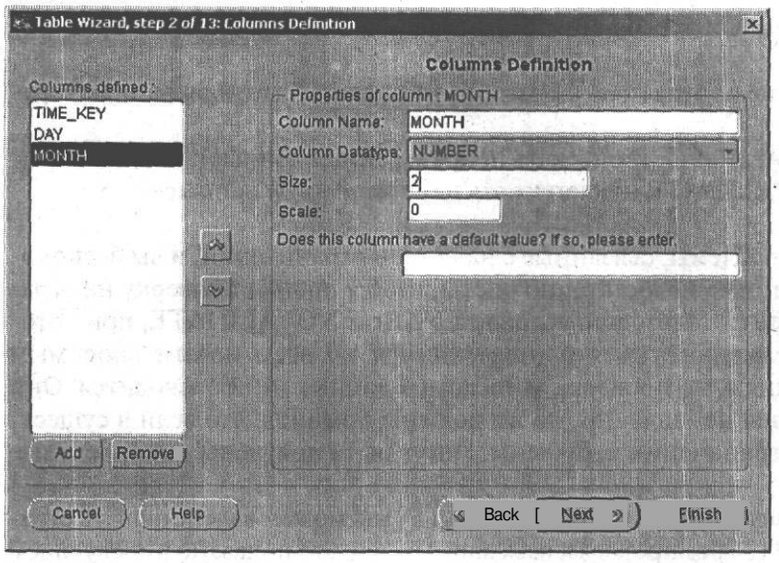


Рис. 2.14. *Oracle Enterprise Manager Console — столбцы таблицы*

перегрузку, особенно при загрузке новых данных. Мы настоятельно рекомендуем реализовывать хотя бы такие ограничения, как первичные и внешние ключи, особенно если вы хотите Использовать возможности Summary Management (Управления суммарными данными), описанные в главе 4. Одним из интересных аспектов Summary Management является возможность переписать запрос для использования материализованного представления (materialized view). Если вы определили в вашей базе данных ограничения, будет возможность осуществлять некоторые сложные формы переписывания запросов.

В случае хранилищ данных часто существует некоторая озабоченность, что, поскольку данные поступают из многих источников, они могут быть не такими «чистыми», как нормальные данные, и, следовательно, ограничения могут не сработать. Хотя это замечание и справедливо, в хранилище нужно всегда хранить чистые данные, чтобы гарантировать точность получаемых результатов.

Еще одним аргументом против использования ограничений является то, что проверка каждой записи при первом введении ее в базу создает определенную нагрузку на процесс ввода данных. Поэтому загрузка данных осуществляется медленнее, чем при стандартной ситуации, и если временное «окно» для ввода данных мало, единственным способом уменьшить время будет отказ от ограничений. Однако можно также использовать данную стадию ETL для программ-

ной проверки возможных ограничений, отбрасывая не прошедшие проверку данные.

В Oracle 9i многие из этих проблем можно преодолеть благодаря опциям ограничений:

- ENABLE NOVALIDATE (допускать непроверенные данные);
- DISABLE NOVALIDATE (не допускать непроверенные данные).

Если вас заботят проблемы, связанные с наличием ограничений, и вы беспокоитесь о том, что данные недостаточно чисты, чтобы пройти проверку на ограничения, вы можете использовать условие ENABLE NOVALIDATE, при котором ограничения задействуются и применяются ко всем новым вносимым данным и изменениям, но проверка записанных данных не производится. Они становятся доступны немедленно, но вы должны помнить, что если в существующих строках ограничения нарушены, могут быть получены некорректные результаты.

Используя условие ENABLE NOVALIDATE, мы можем задействовать ограничение SYS_C001136 без проверки всех данных, как это показано в следующем фрагменте кода.

```
SQL> ALTER TABLE todays_special_offers
    ENABLE NOVALIDATE CONSTRAINT SYS_C001136;
```

Если мы знаем, или хотим предположить, что данные чисты, мы можем включить ограничение немедленно, не вызывая при этом никаких перегрузок. При использовании этого подхода база данных не тратит время на проверку ограничения во всех строках таблицы, но это подразумевает, что разработчику нужно быть уверенным в чистоте данных.

Совет. Используйте осмысленные названия ограничений, например customer_pkey, это гораздо понятнее, чем SYS_C001136.

Кроме того, перед вводом данных ограничения можно быстро отключить при помощи условия DISABLE, как показано в следующем фрагменте кода:

```
SQL> ALTER TABLE todays_special_offers
       DISABLE CONSTRAINT SYS_C001136;
```

На момент написания этой книги, Oracle *Enterprise* Manager не поддерживает условие ENABLE/DISABLE NOVALIDATE, следовательно, вам придется вводить их вручную на SQL, Но условие ENABLE/DISABLE можно применять и в Oracle Enterprise Manager.

Существует также дополнительное условие, называющееся RELY (Доверять), которое требуется для функционирования Summary Management. Это условие говорит оптимизатору, что можно положиться на правильность ограничения. В следующем фрагменте кода дается пример использования условия RELY в ограничении к таблице «todays_special_offers».

```
SQL> ALTER TABLE purchases MODIFY CONSTRAINT special_offer RELY;
```

Вы можете проверить определяемые в базе данных ограничения с помощью Менеджера схем (Schema Manager). Выберите интересующую вас таблицу и щелкните закладку Constraints (Ограничения), как это показано на рис. 2.15.

На рис. 2.16 видно, как можно создать первичный ключ при помощи Мастера таблиц (Table Wizard), щелкнув по столбцу ключей. Не каждая таблица нашей базы данных будет иметь первичный ключ, но может потребоваться, чтобы значение в некоторых столбцах было ненулевым.

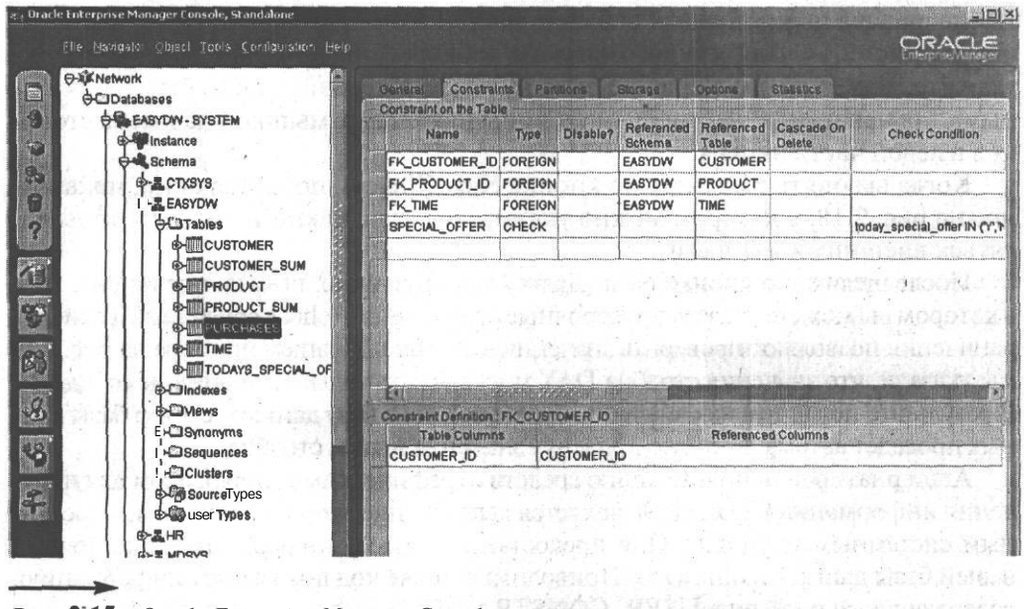


Рис. 2.15. *Oracle Enterprise Manager Console — ограничения*

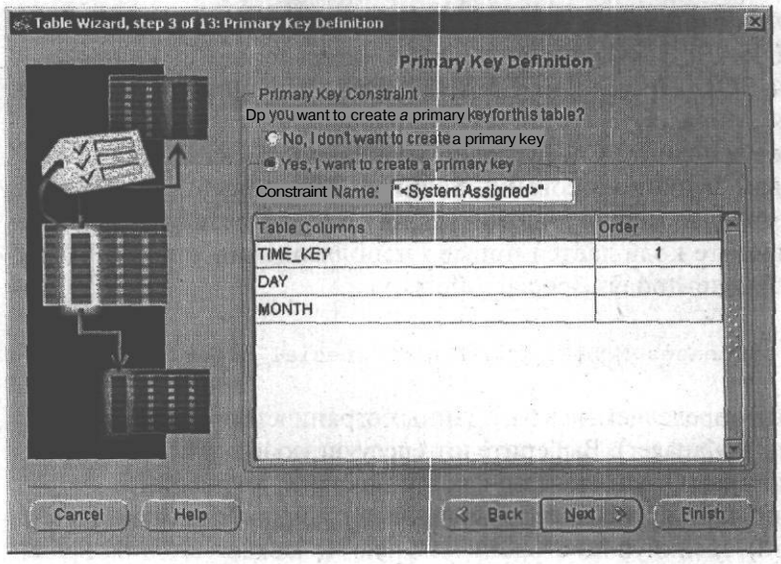


Рис. 2.16. *Мастер таблиц — указание первичного ключа*

После щелчка по кнопке Next (Далее) мастер таблиц отобразит окно, показанное на рис. 2.17. Для каждого столбца таблицы при помощи переключателя укажите, может ли его значение быть нулевым и должно ли оно быть уникальным. Для навигации по столбцам таблицы нажимайте мышкой на имена столбцов в левой части экрана.

Когда вы опять щелкнете по кнопке Next (Далее), появится окно, показанное на рис. 2.18, в котором можно указать, будут ли какие-нибудь столбцы являться внешними ключами.

После щелчка по кнопке Next (Далее) появится окно, показанное на рис. 2.19, в котором вы можете указать проверочные ограничения (check constrains). Такие ограничения позволяют проверить значения в столбце. В нашем примере на рис. 2.19 мы указали, что значения столбца DAY могут лежать только в диапазоне от 1 до 31. В результате перед тем как значение будет введено в базу данных, сервер базы данных проведет автоматическую проверку значения в этом столбце.

Альтернативой использованию средств с графическим интерфейсом для получения информации из Oracle 9i является выполнение запросов ко многим доступным системным таблицам. Они предоставляют массу информации о состоянии вашей базы данных и объектов. Приводимый ниже код показывает информацию, содержащуюся в таблице USER_CONSTRAINTS.

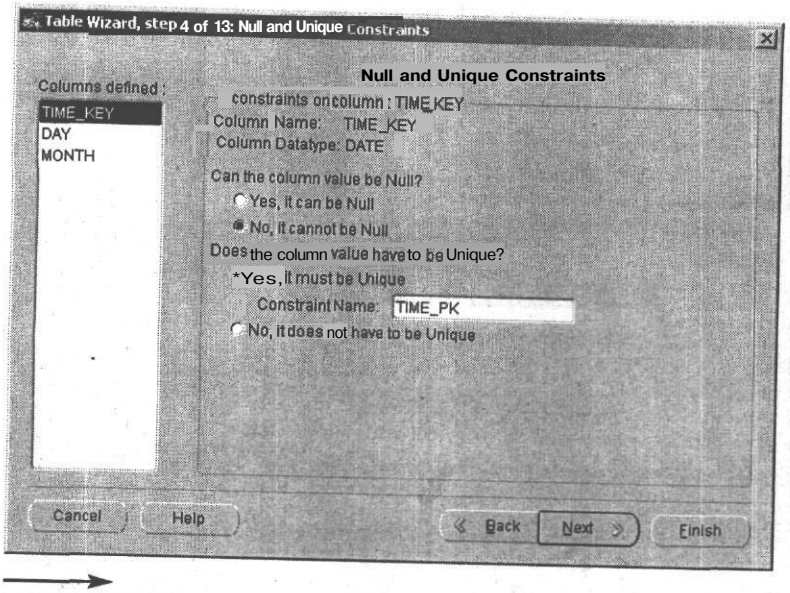


Рис. 2.17. *Oracle Enterprise Manager — ограничение на нулевое значение*

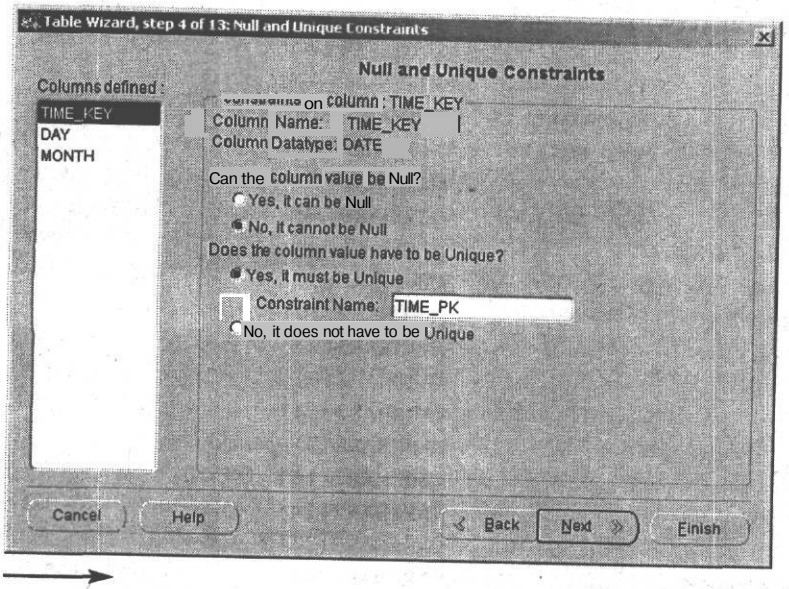


Рис. 2.18. *Oracle Enterprise Manager — внешний ключ*

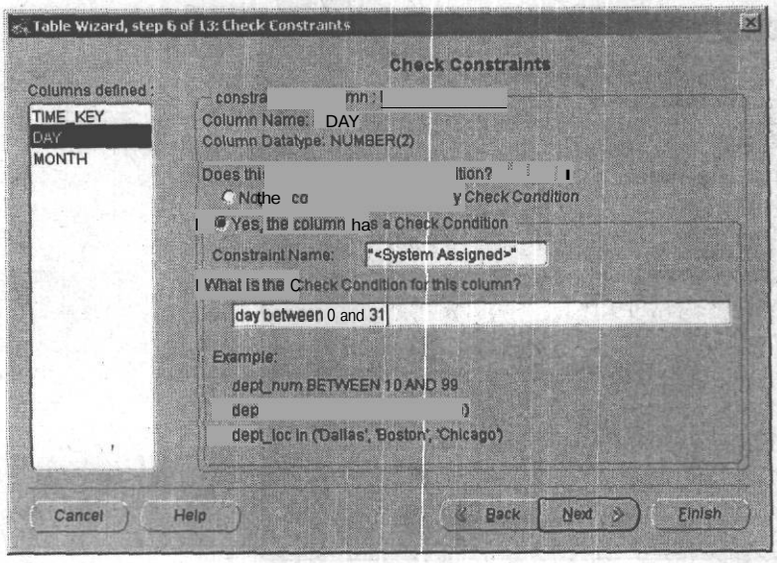


Рис. 2.19. *Oracle Enterprise Manager — проверочные ограничения*

```
SQL> DESCRIBE user_constraints
Name                          Null?                Type
----------------------------- -------------------- ------------
OWNER                         NOT NULL             VARCHAR2(30)
CONSTRAINT_NAME               NOT NULL             VARCHAR2(30)
CONSTRAINT_TYPE                                    VARCHAR2(1)
TABLE_NAME                    NOT NULL             VARCHAR2(30)
SEARCH_CONDITION                                   LONG
R_OWNER                                            VARCHAR2(30)
R_CONSTRAINT_NAME                                  VARCHAR2(30)
DELETE_RULE                                        VARCHAR2(9)
STATUS                                             VARCHAR2(8)
DEFERRABLE                                         VARCHAR2(14)
DEFERRED                                           VARCHAR2(9)
VALIDATED                                          VARCHAR2(13)
GENERATED                                          VARCHAR2(14)
BAD                                                VARCHAR2(3)
RELY                                               VARCHAR2(4)
LAST_CHANGE                                        DATE
```

```
INDEX_OWNER                    VARCHAR2(30)
INDEX_NAME                     VARCHAR2(30)
INVALID                        VARCHAR2(7)
VIEW_RELATED                   VARCHAR2(14)
```

Каждое ограничение, определяемое для данной схемы, будет записываться в эту таблицу. Все системные таблицы, предоставляемые Oracle, имеют префикс ALL_, USER_ или DBA_.

Чтобы увидеть, для каких ограничений вы применяете условие RELY, используйте запрос, приведенный ниже:

```
SQL> SELECT constraint_name, table_name, rely FROM
     all_constraints
    WHERE OWNER = 'EASYDW';

CONSTRAINT_NAME            TABLE_NAME           RELY
---------------            ----------------     ----------
COST_PRICE_NOT_NULL        PRODUCT
FK_CUSTOMER_ID             PURCHASES
FK_PRODUCT_ID              PURCHASES
FK_TIME                    PURCHASES
NOT_NULL_CUSTOMER_ID       PURCHASES.
NOT_NULL_PRODUCT_ID        PURCHASES
NOT_NULL_TIME              PURCHASES
PK_CUSTOMER                CUSTOMER
PK_PRODUCT                 PRODUCT
PK_SPECIALS                TODAYS_SPECIAL_OFFERS
PK_TIME                    TIME
PUBLIC_HOLIDAY             TIME
SELL_PRICE_NOT_NULL        PRODUCT
SHIPPING_CHARGE_NOT_NULL   PRODUCT
SPECIAL_OFFER              PURCHASES                 RELY
```

Можно видеть, что ограничение SPECIAL_OFFER (специальное предложение) имеет условие RELY, а ограничение SHIP-PING_CHARGE_NOT_NULL (ненулевые расходы по доставке) — нет. На рис. 2.20 видны ограничения, определенные для таблицы PURCHASES (покупки).

На момент написания этой книги в Oracle Enterprise Manager вы можете посмотреть, используется ли ограничение или нет, но не сможете увидеть, проверя-

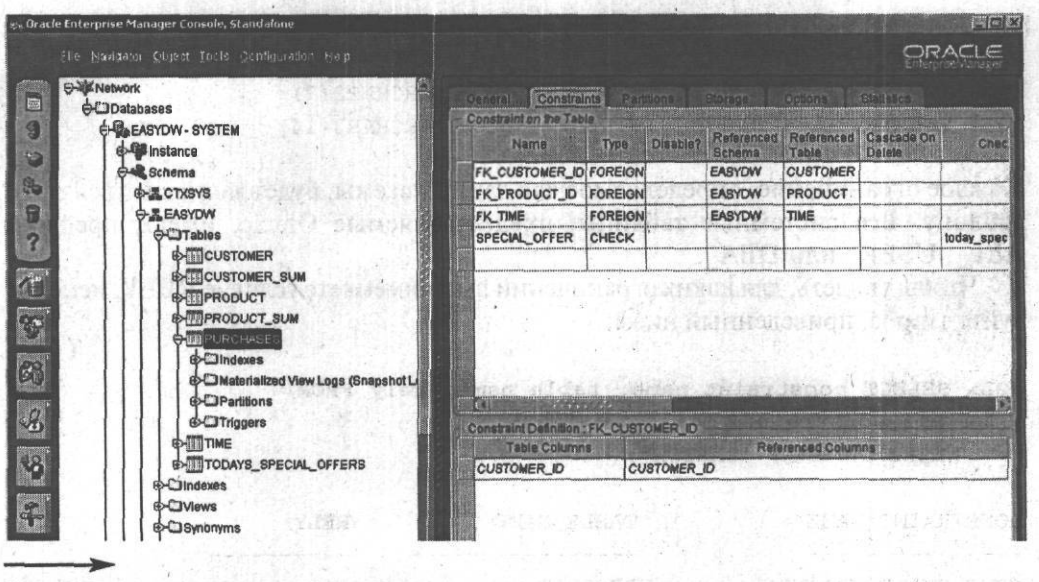


Рис. 2.20. *Oracle enterprise Manager — просмотр ограничений*

ется ли оно. Эту информацию можно получить, если послать запрос к одной из системных таблиц, посвященных ограничениям, например к таблице USER_CONSTRAINTS. Чтобы увидеть задействованные ограничения с условием NOVALIDATE, используйте следующий код запроса.

```
SQL> SELECT constraint_name, validated FROM
     user_constraints
     WHERE OWNER = 'EASYDW';

CONSTRAINT_NAME              VALIDATED
-------------------------    ---------
PK_CUSTOMER                  VALIDATED
COST_PRICE_NOT_NULL          VALIDATED
SELL_PRICE_NOT_NULL          VALIDATED
SHIPPING_CHARGE_NOT_NULL     VALIDATED
```

Процесс определения таблицы почти завершен. Мы указали необходимые столбцы и ограничения, и мы можем нажать кнопку Finish (Готово). Однако есть еще две категории информации, которые требует Мастер таблиц: информация о хранении и секционировании, которая будет рассмотрена позже.

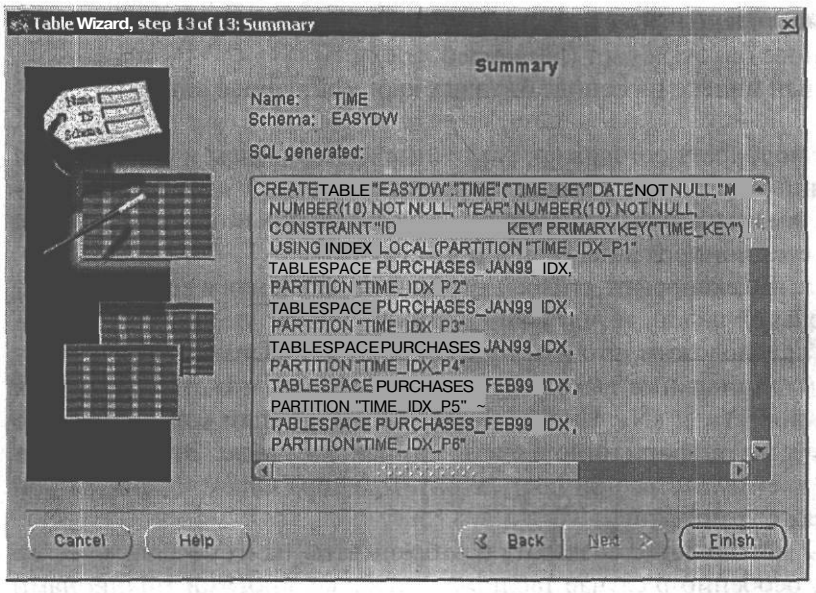


Рис. 2.21. *Мастер таблиц — завершающее окно*

Завершающее окно Мастера создания таблиц показано на рис. 2.21, в котором можно посмотреть на код SQL, который и создаст нашу таблицу. Если вы удовлетворены таблицей, нажмите кнопку Finish (Готово). В противном случае нажмите кнопку Back (Назад) и внесите соответствующие изменения.

Когда вы нажмете на кнопку Finish, таблица будет создана и все будет готово к созданию следующей таблицы. Надеемся, вы согласитесь, что этот путь создания таблиц очень прост, а поскольку хранилище данных, скорее всего, содержит лишь несколько таблиц, вы, наверное, предпочтете этот дружественный подход написанию команд SQL, в которых вы можете допустить синтаксические ошибки, которые потом придется исправлять.

2.3.8. Индексы

Хранилище данных, вероятно, будет содержать несколько индексов, и, каки в любой другой базе данных, разработчик должен выбрать наиболее подходящие. Oracle предлагает несколько разных типов индексов, но разработчику наиболее интересны следующие:

- иерархический индекс (b*tree index);
- индекс на основе битовых карт (bitmapped index);
- соединительный индекс на основе битовых карт (bitmapped join index).

Индексы следует подбирать осторожно, различные альтернативы и обоснования некоторых решений будут описаны более подробно в главе 3. Пожалуйста, обратитесь к этой главе, там вы узнаете, выбирать вам локальный или глобальный индекс и как сделать его секционирование, если это необходимо.

Если вы хотите индексировать столбец хранилища, значения в котором могут быть лишь нескольких видов, то идеально подходит индекс на основе битовых карт. Например, предположим, что нам нужно проиндексировать столбец PUBLIC_HOLIDAY (официальные выходные дни), в котором может содержаться только два значения: «Y» и «N». Индекс, который мы опишем более подробно в главе 3, сохранит эту информацию в очень компактном виде. Это также дает дополнительные преимущества при доступе оптимизатора Oracle к данным при типичных запросах к хранилищу.

Хотя индексы можно легко создавать и отбрасывать, на их создание все же требуется время, особенно в случае таблицы фактов со многими миллионами строк. Индексы необходимо тщательно планировать в начале проекта, чтобы гарантировать, что их создание не займет слишком много времени. Индекс можно создать либо с помощью команды SQL CREATE INDEX, либо с использованием Oracle Enterprise Manager (рис. 2.22).

2.3.9. Секционирование

Секционирование данных — это проектировочный метод, являющийся очень важным для хранилища, поскольку он дает возможность управлять большими объемами данных и контролировать их размещение на дисках. Вместо того чтобы помещать все данные таблицы в одно табличное пространство, мы можем с помощью секционирования разместить их во многих табличных пространствах. Для определения того, в каком табличном пространстве хранятся данные, выбирается ключ секционирования, например, как показано на рис. 2.23, ключ времени (time_key).

В данном примере мы проводим секционирование по месяцам, так что данные января попадают в один раздел, данные февраля — в другой и т.д. Ключ секционирования нужно выбирать осторожно, хотя обычным является секционирование по времени. Следовательно, вы можете секционировать данные по месяцам, и каждый месяц будет располагаться в своем табличном пространстве.

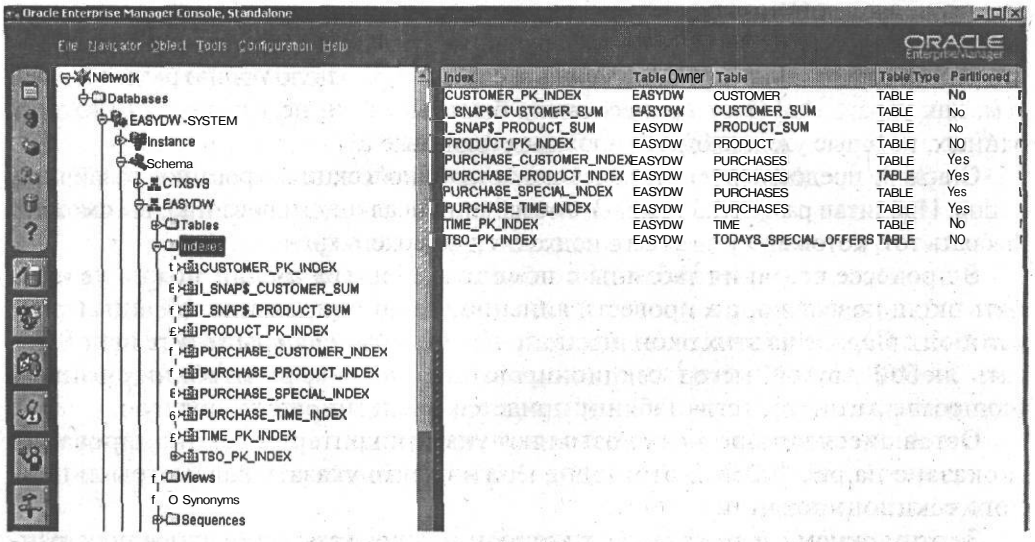


Рис. *2.22. Oracle Enterprise Manager — индексы*



Рис. 2.23. *Секционирование*

Это позволяет получить более управляемое секционирование, а также то преимущество, что, если вам понадобится заархивировать данные, достаточно будет просто убрать раздел. И конечно, не забудьте сделать резервную копию раздела перед тем, как убрать его! Этот процесс очень быстрый, и он не уничтожает никаких данных, которые уже находятся в таблице фактов.

Oracle 9i предоставляет вам несколько методов секционирования различных типов. Прочитав раздел 3.3 главы 3, посвященный секционированию, вы сможете выбрать тот, который лучше всего подходит для вашего хранилища.

В процессе создания таблицы с помощью Мастера таблиц можно увидеть пять окон, позволяющих провести линейное секционирование таблицы (range partition). Первое из этих окон показано на рис. 2.24, Если вы хотите использовать любой другой метод секционирования, например, комбинированный (composite partition), тогда таблицу придется секционировать вручную.

Оставшиеся четыре окна позволяют указать критерии секционирования (показано на рис. 2.25). В этом окне можно легко указать параметры линейного секционирования.

За этим окном следует другое, в котором можно указать табличное пространство, используемое для каждого из разделов, столбцы, используемые в индексном секционировании и табличные пространства для каждого из индексных разделов.

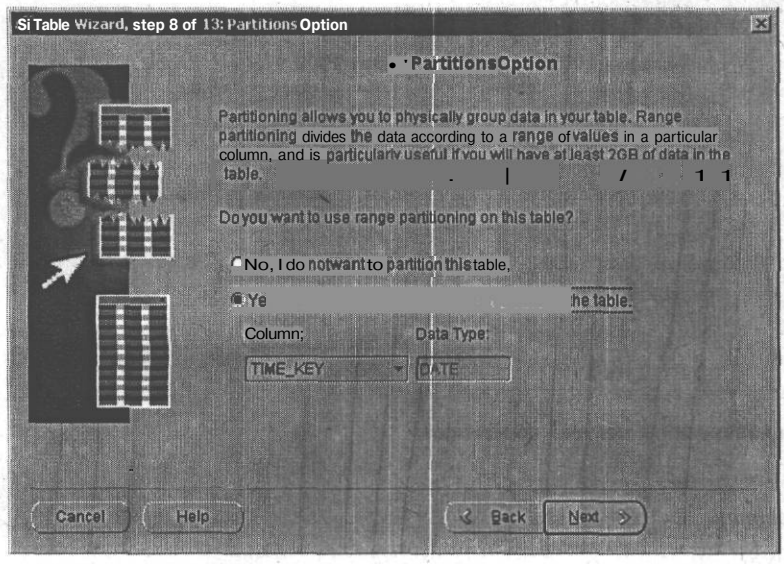


Рис. 2.24. *Мастер таблиц — окно секционирования*

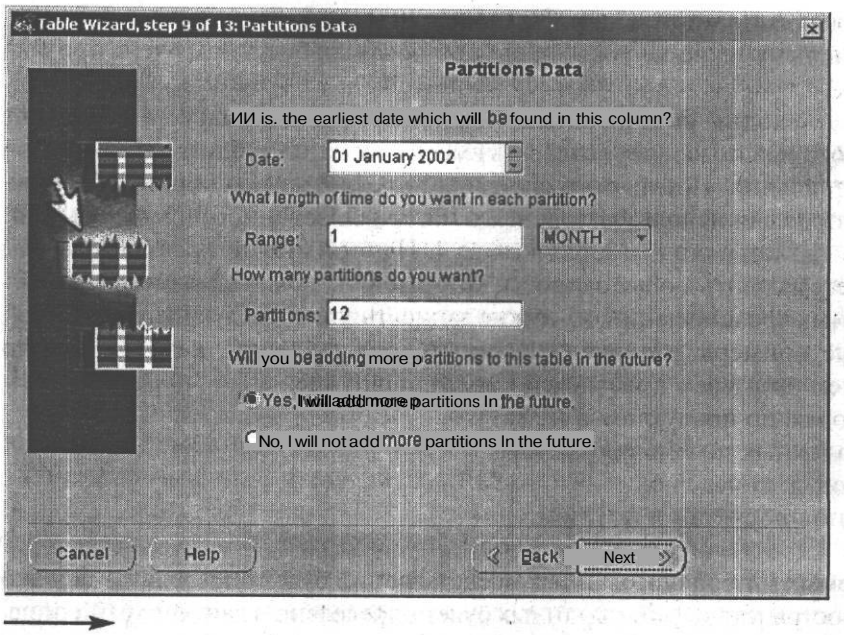


Рис. 2.25. *Мастер таблиц — критерии секционирования*

2.3.10. Материализованные представления

Мы уже видели, что хранилище или тематические хранилища могут хранить в таблице фактов огромные количества записей. Даже если бы у нас была самая быстрая в мире машина и мы могли бы кэшировать часть данных хранилища в памяти, время ответа на запросы могло бы составлять дни, и наверняка займет минуты или часы.

Для преодоления этой проблемы, разработчики хранилищ используют метод создания сводок (summary), где под сводкой имеется в виду заранее созданная таблица результатов, которая в Oracle называется материализованным представлением (materialized view). Например, предположим, что вы всегда запрашиваете число покупок за день по сегодняшнему специальному предложению. Вместо того чтобы каждый раз подсчитывать эти результаты, создается материализованное представление, содержащее требуемую информацию. Затем, когда вы будете делать такой запрос, он будет направлен к материализованному представлению, а не к таблице фактов. Хотя это частично нарушает концепцию хранилища, в которой вы можете посылать базе данных заранее неизвестные вопросы, нужно честно

признать, что довольно многие запросы хорошо известны. Если мы сможем улучшить время ответа на такие запросы, наши пользователи будут нам очень признательны.

В Oracle 9i входит специальный компонент — Summary Management Component, который позволяет создавать вместо обычных таблиц материализованные представления. После этого оптимизатор будет явным образом переписывать ваш запрос для использования этого материализованного представления. Подробно это свойство описывается в главе 4. На этой стадии проектирования, если вы сможете выявить любые запросы, которые могут быть сделаны в виде материализованных представлений, их нужно записать для последующего создания. Вот некоторые примеры материализованных представлений, которые можно создать в нашем примере с компанией Easy Shopping Inc.

- Сумма продаж по продуктам и дням.
- Число проданных за день продуктов.
- Сумма продаж за неделю.
- Прибыль по продуктам и по дням.

Число ожидаемых материализованных представлений будет определять, сколько табличных пространств и файлов данных будет определено в данном хранилище.

Совет. Не обязательно создавать материализованное представление для каждой возможной комбинации. Объяснения приводятся в главе 4.

2.3.11. Безопасность

Нельзя забывать, что некоторые данные в хранилище могут быть очень критичными, и, следовательно, по разнообразным причинам, вы можете не желать, чтобы весь ваш персонал имел доступ к этим данным. В Oracle 9i содержатся различные типы средств безопасности, которые не дают пользователям изменять объекты базы и осуществлять доступ к данным.

Привилегии доступа к объектам

Привилегии доступа можно установить для следующих типов объектов:

- модули PL/SQL;
- очереди;

- последовательности;
- синонимы;
- таблицы;
- типы;
- представления.

Скорее всего, вы будете устанавливать привилегии для доступа к таблицам и представлениям путем объявлений того, может ли пользователь выбирать, вставлять и обновлять данные, а также многих других опций. Если вы решили создать несколько пользователей, обеспечьте достаточно привилегий для того, чтобы все они могли читать данные. Это можно сделать либо с помощью оператора SQL, команды GRANT SELECT ON, команды GRANT SELECT ANY TABLE или же их можно указать в Oracle Enterprise Manager, в разделе, посвященном безопасности. На рис. 2.26 мы даем пользователю EASYDW права изменять (ALTER), вставлять (INSERT), выбирать (SELECT) и обновлять (UPDATE) таблицу CUSTOMER в схеме EASYDW. Просто повторите этот процесс для всех пользователей во всех таблицах, к которым им разрешается доступ.

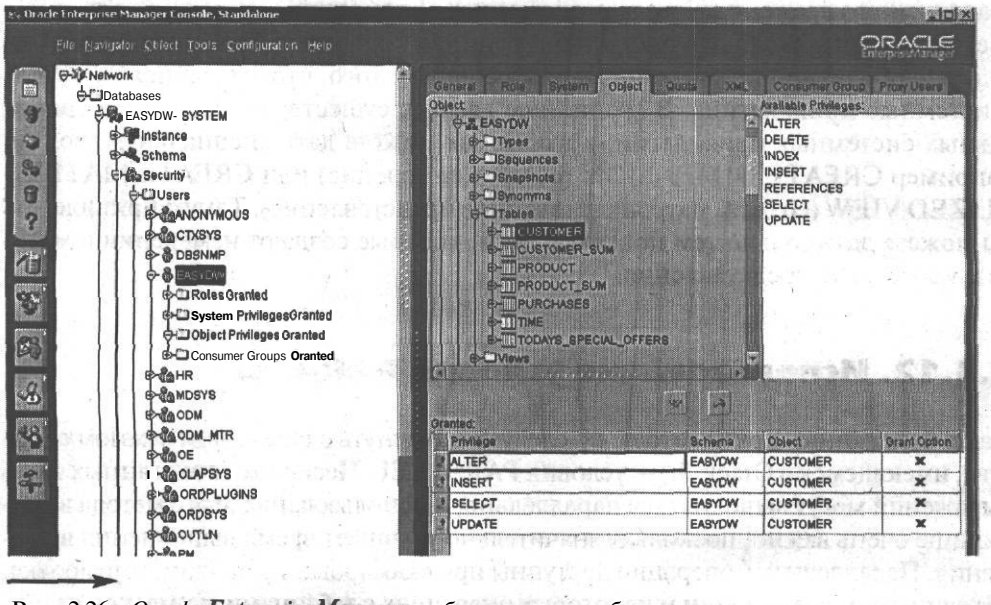


Рис. 2.26. *Oracle Enterprise Manager — безопасность объекта*

Роль

В качестве альтернативы вы можете создать роль. Затем вы назначаете этой роли все привилегии, и эта роль прикрепляется к пользователю. Такой подход является предпочтительным, особенно если пользователей много, поскольку вы сможете создавать роли для разных уровней работы и каждому пользователю выдается одна из этих ролей и привилегии не придется назначать индивидуально. Применяя ролевой подход, вы снижаете вероятность случайного получения пользователем доступа к тем данным, доступ к которым ему не положен. Кроме того, это быстрее, если нужно определить много пользователей. Вы можете либо использовать существующие роли, либо создать свою собственную, используя выражение SQL CREATE ROLE.

Системные привилегии

Большая часть времени будет уходить на раздачу пользователям привилегий доступа к объектам и, возможно, на создание ролей. Однако некоторым пользователям потребуются системные привилегии. Системная привилегия дает право вышолнения высокоуровневых задач, таких, как создание или удаление таблицы. Поскольку вам не нужно давать такое право многим пользователям, вам и не нужно тратить много времени на раздачу системных привилегий.
Метод, используемый для определения системной привилегии, аналогичен методу для объектной привилегии, за исключением того, что вы выбираете раздел «системные привилегии». В хранилище данных существуют несколько специальных системных привилегий, которые вы можете дать специальным ролям, например CREATE DIMENSION (создать измерение) или CREATE MATERIALIZED VIEW (создать материализованное представление). Такую привилегию вы можете дать только тем пользователям, которые создают измерения и материализованные представления.

2.3.12. Использование условия PARALLEL

Наша база данных почти готова, но следует упомянуть о еще одном важном свойстве, имеющемся в Oracle 9i, — условии PARALLEL. Несколько показанных здесь выражений могут выполняться параллельно, и использование этого метода в хранилище очень важно, поскольку значительно улучшает время вышолнения выражения. Параллельные операции доступны при сканировании таблиц, сортировке, объединении, соединении и некоторых операциях с таблицами и индексами.

Если вы используете симметричную мультипроцессорную систему, или систему с высокой степенью параллелизма, то нужно серьезно подумать о возможности использования условия PARALLEL. Если указано это условие, выражение, если оно подходит для параллельной обработки, разбивается на несколько потоков и Oracle 9i выполняет работу в виде параллельных задач и координирует их выполнение и результаты. Следовательно, все, что нужно сделать пользователю, — включить это условие, а Oracle 9i сделает остальное.

В условии PARALLEL вы должны указать число используемых параллельных процессов. Выбирайте это значение осторожно. Может потребоваться провести тестирование, чтобы подобрать оптимальное значение. Например, приведенное условие можно добавить к выражению CREATE TABLE для нашей таблицы фактов, посвященной покупкам:

```
PARALLEL (DEGREE 2)
```

Это будет означать, что любые операции над этой таблицей нужно проводить с использованием двух серверных процессов, если возможно их параллельное выполнение.

Данные также можно загружать параллельно. Утилита SQL*Loader, которую мы обсудим позже, позволяет задавать параллельность операций. Очевидно, что это может значительно улучшить время сохранения данных. Однако вы должны помнить о том, что возможно появление фрагментации данных. Поэтому мы предлагаем при параллельной загрузке данных указать число параллельных операций равным числу доступных для данного табличного пространства файлов данных. Обращаясь к нашему примеру с компанией Easy Shopping Inc., если мы укажем значение

```
PARALLEL (DEGREE 3)
```

в нашей таблице фактов, то для каждого табличного пространства должны быть определены три файла данных.

2.4. Тестирование проектной модели

Никто из нас не стал бы выпускать написанное приложение, не протестировав его сперва. Однако поразительно, сколько проектов баз данных создается, а потом обрушивается на ничего не подозревающих пользователей. Ваше хранилище не исключение, особенно если бизнес полагается на него в получении важной инфор-

мации. Поэтому крайне важно, чтобы все аспекты проекта и процессы перед выпуском в эксплуатацию были тщательно протестированы.

Предполагается, что поначалу вы загружаете в хранилище небольшой процент данных, а затем тестируете следующие области:

- время, затрачиваемое на загрузку данных;
- очистка и преобразование данных;
- время ответа на запрос;
- необходимость в сводках по данным;
- время, необходимое для задач по управлению.

Если вы создаете терабайтное хранилище, рекомендуется повторить этот процесс еще раз, при даже еще большем количестве данных в хранилище, на случай появления неожиданных проблем, связанных с такими объемами данных.

Намного легче исправить проблемы, выявленные при тестировании, чем решать их на ходу. Еще один способ тестирования хранилища — использование поэтапной реализации пользовательской базы, в том случае если вы не хотите ждать, пока будет завершено все тестирование.

Очень важный момент — помнить, что из-за размеров базы данных не только загрузка данных занимает больше времени, обработка запросов также вряд ли будет проводиться быстро. Поэтому полный процесс тестирования займет намного больше времени, чем, скажем, в традиционных OLTP-базах.

2.5. Схема для компании Easy Shopping Inc.

В этой главе мы увидели, как создать базу данных с помощью средств с графическим интерфейсом, но многие читатели могут предпочесть создать базу данных напрямую, с помощью SQL. Код SQL для этого приведен ниже, но он предполагает, что база данных уже создана. Приведенный здесь пример сделан для системы Windows 2000. Все, что нужно для переноса на другую платформу, — отредактировать спецификации файла. Заметим также, что размеры приводимых файлов очень малы по сравнению с теми, которые вы имели бы в рабочей системе.

2.5.1. Создание табличных пространств и файлов данных

Первый шаг — соединиться с базой данных с помощью имени пользователя с администраторскими привилегиями.

```
connect system/manager ;
```

Следующий шаг — создание табличных пространств, в которых будут располагаться данные, и связанных с ними файлов данных.

```
-- Создаем табличное пространство для буфера отмены
CREATE TABLESPACE easy_undo
 datafile 'D:\EASYDW\easy_undo.f'
 size 8m reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);

-- Временное табличное пространство
CREATE TEMPORARY TABLESPACE easy_temp
 tempfile 'D:\EASYDW\easy_temp.f'
 size 10m reuse autoextend on next 16k ;

-- Табличное пространство для хранения материализованных представлений
CREATE TABLESPACE mview
 datafile 'D:\EASYDW\easy_mview.f'
 size 6m reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);

— Табличное пространство для измерений
CREATE TABLESPACE easy_dim
 datafile 'D:\EASYDW\dimensions.f' size 5m
 reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);

-- Табличное пространство для индексов
CREATE TABLESPACE easy_idx
 datafile 'D:\EASYDW\index.f'
 size 5m reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);

-- Табличное пространство по умолчанию
CREATE TABLESPACE easydw_default
 datafile 'D:\EASYDW\easydw_default.f'
 size 5m reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);
```

Как только созданы табличные пространства для измерений, мы можем создавать табличные пространства для таблицы фактов (purchases, покупки). Поскольку мы

будем делать секционирование данных, мы должны создать табличное пространство для каждого раздела. Будут помесячные разделы для данных и еще один раздел для индексов. Здесь мы создадим разделы только для данных января месяца и индекса. Для создания других разделов просто повторяйте эти операции.

```
— создать табличное пространство на три месяца для разделов таблицы
фактов
CREATE TABLESPACE purchases_jan2002
 datafile 'D:\EASYDW\PURCHASESJAN2002.f' size 5m
 reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);

-- создать табличное пространство на три месяца для индексов таблицы
фактов
CREATE TABLESPACE purchases_jan2002_idx
 datafile 'D:\EASYDW\PURCHASESJAN2002_IDX.f' size 3M
 reuse autoextend on default storage
  (initial 16k next 16k pctincrease 0 maxextents unlimited);
```

2.5.2. Создание таблиц, ограничений и индексов

Когда табличные пространства определены, можно создавать пользователя EASYDW, что образует схему, в которой будут сохраняться данные.

```
-- создать пользователя с именем EASYDW
-- это будет схема, в которой будут размещаться объекты

connect system/manager;

CREATE USER easydw identified by easydw
 default tablespace easydw_default
 temporary tablespace temp
 profile default account unlock;

GRANT unlimited tablespace to easydw ;
GRANT dba to easydw ;
GRANT create session TO easydw;
```

Пользователям даются привилегии администратора базы, чтобы они могли создавать таблицы и схемы и управлять ими.

Совет. Не забудьте войти в систему как пользователь EASYDW перед созданием таблиц и индексов, или таблицы и индексы попадут не в ту схему.

```
-- теперь создаем базу данных
CONNECT easydw/easydw;
-- измерение CUSTOMER
CREATE TABLE easydw.customer
  (customer_id                                    varchar2(10) ,
 town                                              varchar2(10),
 county                                            varchar2(10),
 postal_code                                       varchar2(10),
 dob date,
 country                                           varchar2(20),
 occupation                                        varchar2(10))
 pctfree 0 pctused 99 tablespace easy_dim
 storage (initial 16k next 16k pctincrease 0) ;

CREATE UNIQUE INDEX easydw.customer_pk_index ON customer
  (customer_id)
 pctfree 5
 tablespace easy_idx
 storage (initial 16k next 16k pctincrease 0) ;

ALTER TABLE customer
ADD CONSTRAINT pk_customer PRIMARY KEY (customer_id);
```

Здесь мы определили ограничение, добавив его внизу. Его также можно создать внутри определения CREATE TABLE, но, если использовать такой подход, вы не сможете контролировать имя индекса (с целью быстрой проверки первичного ключа), которое в этом случае создается по умолчанию. Следовательно, мы сначала создаем индекс, а когда определено ограничение, оно использует этот индекс, а не создает свой собственный.

```
— измерение PRODUCT
CREATE TABLE easydw.product
  (product_id                                     varchar2(8),
 product_name                                     varchar2(30),
 category                                         varchar2(4),
```

```
 cost_price number (6,2)
 constraint cost_price_not_null NOT NULL,
 sell_price                                    number (6,2)
 constraint sell_price_not_null NOT NULL,
 weight                                        number (6,2),
 shipping_charge                               number (5,2)
 constraint shipping_charge_not_null NOT NULL,
 manufacturer                                  varchar2(20),
 supplier                                      varchar2(10))
 pctfree 0 pctused 99
 tablespace easy_dim
 storage (initial 16k next 16k pctincrease 0) ;

CREATE UNIQUE INDEX easydw.product_pk_index ON product
  (product_id)
 pctfree 5 tablespace easy_idx
 storage (initial 16k next 16k pctincrease 0) ;

ALTER TABLE product
ADD CONSTRAINT pk_product PRIMARY KEY (product_id);
```

Затем создается таблица TIME. В эту таблицу мы включили проверочное (CHECK) ограничение, обеспечивающее наличие в столбце «public_holiday» только значений «Y» или «N».

```
-- измерение TIME
CREATE TABLE easydw.time
 (time_key                                     date,
 day                                           number (2,0),
 month                                         number (2,0),
 quarter                                       number (2,0),
 year                                          number (4,0),
 day_number                                    number (3,0),
 day_of_the_week                               varchar2(8),
 week_number                                   number (2,0),
 public_holiday                                varchar2(1)
  constraint public_holiday
  CHECK (public_holiday IN ('Y','N')) )
pctfree 0 pctused 99
tablespace easy_dim
```

```
storage (initial 16k next 16k pctincrease 0) ;
CREATE UNIQUE INDEX easydw.time_pk_index ON time
 (time_key)
 pctfree 5 tablespace easy_idx
 storage (initial 16k next 16k pctincrease 0) ;
ALTER TABLE time
ADD CONSTRAINT pk_time PRIMARY KEY (time_key);
```

Для таблицы TODAYS_SPECIAL_OFFER мы определили первичный ключ, включив два столбца, а не один.

```
-- измерение TODAYS_SPECIAL_OFFERS
CREATE TABLE easydw.todays_special_offers
 (product_id                                varchar2(8),
 offer_date                                 date,
 special_price                              number (6,2),
 offer_price                                number (6,2))
 pctfree 0 pctused 99
 tablespace easy_dim
 storage (initial 16k next 16k pctincrease 0) ;

CREATE UNIQUE INDEX easydw.tso_pk_index ON todays_special_offers
 (offer_date, product_id)
 pctfree 5 tablespace easy_idx
 storage (initial 16k next 16k pctincrease 0) ;

ALTER TABLE todays_special_offers
 ADD CONSTRAINT pk_specials PRIMARY KEY
     (offer_date,product_id );
```

Теперь мы переходим к созданию наиболее важной таблицы — таблицы фактов, которую мы назвали PURCHASES (Покупки). Определение этой таблицы весьма сложно, поскольку в нее входят несколько внешних ключей (FOREIGN KEYS) к нескольким таблицам, идентифицируемым по условию REFERENCES (ссылки). В таблице определяются два ограничения для столбца product_id: ограничение NOT NULL и ограничение foreign-key. Если каждому из ограничений дать уникальное имя, тогда их можно применить к столбцу таблицы. Здесь мы приводим пример того, как секционировать таблицу. Более подробно это обсуждается в Главе 3.

```
-- Таблица фактов PURCHASES
CREATE TABLE easydw.purchases
(product_id                                    varchar2(8)
 constraint not_null_product_id NOT NULL
 constraint  fk_product_id
     REFERENCES product(product_id),
 time_key                                      date
 constraint not_null_time NOT NULL
 constraint  fk_time
     REFERENCES time(time_key),
 customer_id                                   varchar2(10)
 constraint not_null_customer_id NOT NULL
 constraint  fk_customer_id
     REFERENCES customer(customer_id),
 purchase_date                                 date,
 purchase_time                                 number(4,0),
 purchase_price                                number(6,2),
 shipping_charge                          ...  number (5,2),
 today_special_offer                           varchar2(1)
 constraint special_offer
 CHECK (today_special_offer IN ('Y','N')) )
PARTITION by RANGE (time_key )
(
    partition purchases_jan2002
     values less than (TO_DATE('01-02-2002', 'DD-MM-YYYY'))
     pctfree 0 pctused 99
     storage (initial 64k next 16k pctincrease 0)
     tablespace purchases_jan2002,
    partition purchases_feb2002
     values less than (TO_DATE('01-03-2002', 'DD-MM-YYYY'))
     pctfree 0 pctused 99
     storage (initial 64k next 16k pctincrease 0)
     tablespace purchases_feb2002,
    partition purchases_mar2002
     values less than (TO_DATE('01-04-2002', 'DD-MM-YYYY'))
     pctfree 0 pctused 99
     storage (initial 64k next 16k pctincrease 0)
     tablespace purchases_mar2002 );
```

В этом примере создали индексы сразу после определения таблицы. В реальном хранилище данных, число индексов, создаваемых до загрузки данных, сводится к абсолютному минимуму, чтобы время загрузки было как можно меньше. В результате, индексы служат только для проверки ограничений.

```
-- Теперь создаем индексы
-- Секционирование по ключу времени. Индекс локальный, с префиксом.
CREATE BITMAP INDEX easydw.purchase_time_index
  ON purchases (time_key )
local
(partition indexJan2002 tablespace purchases_jan2002_idx,
partition indexFeb2002 tablespace purchases_feb2002_idx,
partition indexMar2002 tablespace purchases_mar2002_idx );

CREATE BITMAP INDEX easydw.purchase_product_index
  ON purchases (product_id )
local
pctfree 5 tablespace indx
storage (initial 64k next 64k pctincrease 0) ;

CREATE INDEX easydw.purchase_customer_index
  ON purchases (customer_id )
local
pctfree 5 tablespace indx
storage (initial 64k next 64k pctincrease 0) ;

CREATE BITMAP INDEX easydw.purchase_special_index
  ON purchases (today_special_offer )
local
pctfree 5 tablespace indx
storage (initial 64k next 64k pctincrease 0) ;
```

2.5.3. Определение безопасности

Следующий шаг — выделить некоторые привилегии нашему пользователю EASYDW. Начнем с тех, которые позволят нам использовать Summary Management (Управление сводками). По ходу книги мы обсудим и другие привилегии, которые нужно давать пользователям.

```
connect  system/manager;
— Добавляем привилегии
GRANT SELECT ANY TABLE TO easydw;
GRANT EXECUTE ANY PROCEDURE TO easydw;

-- Добавляем привилегии для summary management
GRANT CREATE ANY DIMENSION TO easydw;
GRANT ALTER ANY DIMENSION TO easydw;
GRANT DROP ANY DIMENSION TO easydw;
GRANT CREATE ANY MATERIALIZED VIEW TO easydw;
GRANT ALTER ANY MATERIALIZED VIEW TO easydw;
GRANT DROP ANY MATERIALIZED VIEW TO easydw;
GRANT QUERY REWRITE TO easydw;
GRANT GLOBAL QUERY REWRITE TO easydw;
```

Нужно повторить эти шаги для каждого пользователя, и, конечно, привилегии, даваемые разным пользователям, будут различаться.

2.5.4. Заключительные шаги

Последний этап завершается в действительности не сейчас, а после загрузки данных. Однако мы включили его сюда, чтобы вы не забыли об этом важном шаге анализа таблиц и индексов. Статистические данные собираются с помощью пакета DBMS_STATS. Собранную пакетом статистику использует оптимизатор. Без этой статистики такие возможности, как Summary Management (Управление сводками), окажутся недоступны. Команду DBMS_STATS мы объясним в главе 3.

```
-- Теперь — анализ таблиц и индексов
EXECUTE dbms_stats.gather_table_stats('EASYDW','CUSTOMER');
EXECUTE dbms_stats.gather_table_stats('EASYDW','TODAYS_SPE-
CIAL_OFFERS');
EXECUTE dbms_stats.gather_table_stats('EASYDW','product');
EXECUTE dbms_stats.gather_index_stats('EASYDW', 'purchase_
customer_index');
```

Теперь наша база данных готова. У нас есть главная основа, и в следующей главе мы узнаем, как можно расширить эту основу, включая в нее и используя возможности, имеющиеся в Oracle 9i и недоступные в более ранних версиях.

3

Свойства хранилища данных

3.1. Оптимизация хранилища данных

Хранилища используются, в основном, для организации данных, чтобы на запросы, необходимые для анализа и принятия решений, мог быть дан быстрый ответ. По мере того как хранилище увеличивается в размерах, весьма часто размер таблицы фактов начинает составлять несколько гигабайт или даже терабайт. Для быстрой обработки запроса крайне важно, чтобы извлечение и обработка данных базы производились эффективно. Процесс оптимизации производительности в OLTP-системах с акцентом на обновлениях хорошо известен. Однако в хранилищах данных, акцент в которых делается на обработке запросов, оптимизация имеет совершенно иные требования. В этой главе мы рассмотрим некоторые свойства базы данных Oracle, которые специально разработаны для улучшения обработки запросов к хранилищу.

Хорошую производительность хранилища в высокой степени определяет физическая схема базы данных. Мы рассмотрим несколько методов, которые подходят для хранилищ, включая секционирование, сжатие сегментов данных и индексирование на основе битовых карт. Также мы обсудим оптимизацию запросов, в частности отсечение разделов (partition pruning), соединение с учетом разделов (partition-wise join) и преобразование по типу звезды (star transformation).

Приложениям, ориентированным на помощь в принятии решений, часто требуется выполнять такие вычисления, как сравнение по периодам и кумулятивная агрегация (cumulative aggregation). Такие запросы традиционно требуют создания очень сложного и длинного кода SQL. Oracle и некоторые другие производители баз данных начали работу по выработке стандартизованных расширений языка SQL, которые позволят упростить составление таких запросов и эффективно их обрабатывать. Мы рассмотрим некоторые из этих новых функций SQL.

3.2. Индексирование

Индексирование всегда являлось важным методом эффективной обработки запросов в базах данных. В отличие от OLTP-систем, в которых основными операциями являются операции обновления, в хранилищах производится чтение огромных объемов данных с целью ответа на запросы. Поэтому важно понять требования хранилищ к индексированию.

Решение о том, какие индексы создавать, является важной частью физической разработки базы данных. Индексы можно создавать по столбцам, которые часто появляются в критериях выбора в запросах. Столбцы, на которые часто ссылаются в условии SQL WHERE, — хорошие кандидаты на индексирование. Большей части запросов, связанных с принятием решений, требуются определенные строки из таблицы измерений, так что таблицы измерений должны быть хорошо проиндексированы. Например, если пользователю нужно расположение всех магазинов в северо-восточном регионе, как в приведенном ниже запросе, то для обнаружения этих строк в северо-восточном регионе можно использовать индекс созданный по столбцу «region», а не читать все строки таблицы.

```
SELECT store, location FROM stores WHERE region = 'Northeast';
```

В типичном хранилище немного операций обновления, отличных от загрузки новых данных. Поэтому хранилище может иметь намного больше индексов, чем OLTP-система. В хранилище свободного места для индексов часто должно быть значительно больше, чем для хранения данных, особенно это касается таблицы фактов. Поэтому вы можете захотеть свести индексирование таблицы фактов к минимуму. Как правило, для таблицы фактов можно иметь один-два соединенных индекса. Кроме того, может быть, стоит подумать о индексах на основе битовых карт для столбцов внешних ключей, чтобы ускорить соединения по типу звезды. С другой стороны, таблицы измерений намного меньше, по сравнению с таблицей фактов, и их можно проиндексировать намного шире. Любой столбец таблицы измерений, который часто используется в операциях выбора или является уровнем объекта измерения, описанного в главе 4, — это хороший выбор для индексирования.

Индекс можно создать по одному или более столбцу таблицы. Эти столбцы называются ключами индекса. Для каждого значения ключа индекс содержит указатель на местоположение строк с данным значением ключа. Когда данные в таблице изменяются, индекс автоматически обновляется, отражая изменения данных.

Oracle предлагает несколько типов индексов. В этой книге мы обсудим три из них:

- иерархический индекс (b*tree index);
- индекс на основе битовых карт (bitmapped index);
- соединительный индекс на основе битовых карт (bitmapped join index).

3.2.1. Иерархические индексы

Иерархические (b*tree) индексы представляют собой иерархические структуры, которые позволяют проводить быстрый поиск и получение адреса строки таблицы, содержащей определенное значение. Иерархические индексы хранят указатели на строки таблицы с использованием идентификаторов строк (rowids). Существуют две разновидности иерархических индексов:

- уникальные;
- не уникальные.

Уникальный индекс гарантирует, что для каждой строки имеется отдельное значение ключа. Совпадения не допускаются. Уникальный индекс создается автоматически после того, как в таблице определяется ограничение PRIMARY KEY или UNIQUE. Иерархические индексы также могут быть неуникальными. Неуникальные индексы повышают производительность обработки запросов при выборе небольшого числа строк.

Когда в условии WHERE используются одновременно несколько столбцов, вы можете создать индекс по этой группе столбцов. Индексы, созданные по нескольким столбцам, называются сложными, многоключевыми или объединенными индексами. Например, чтобы отличить город Портленд, штат Мэн от города Портленд, штат Орегон, требуются одновременно и название города, и название штата. Столбец, который наиболее часто используется сам по себе, нужно указать как первый столбец индекса. В данном примере название штата будет ведущим столбцом, поскольку ожидается поступление запросов как по одному штату, так и по штату и городу.

Когда вставляется или удаляется строка или когда обновляются столбцы ключей, индекс автоматически обновляется, отражая произведенные изменения. Иерархические индексы разработаны таким образом, что время, которое требуется для поиска любого конкретного значения в индексе, почти постоянно. Такая схема называется сбалансированным индексом (balanced index). Когда создаются новые узлы индекса, иерархия автоматически балансируется заново. Следовательно, обслуживание индекса создает дополнительную нагрузку на выполнение

оператора DML. При загрузке большого числа строк иногда быстрее будет пометить индекс как неиспользуемый в ходе операций вставки и перестроить индекс, как это показано в следующем сегменте кода.

```
ALTER INDEX customer_gender_index UNUSABLE;

INSERT /*+ APPEND */ INTO customers ...
COMMIT;

ALTER INDEX customer_gender_index REBUILD;
```

Иерархический индекс полезен в тех случаях, когда ключ индекса имеет множество возможных значений, каждое из которых ведет к небольшому числу строк таблицы. Например, в первичном ключе предполагается, что каждое значение ключа соответствует одной строке таблицы. Однако если столбец ключей индекса имеет только несколько возможных значений, тогда каждое значение приведет к извлечению большого числа строк таблицы. Это если и дает прирост производительности, то он невелик. Рассмотрим, например, такой запрос: сколько женщин, проживающих в Калифорнии, покупают палатки? В столбце «пол» может содержаться только два значения, следовательно, поиск всех строк, соответствующих женщинам, может вызвать прочитывание всех блоков данных в таблице, поскольку в одном блоке, как правило, хранится много записей. Более эффективным может оказаться сканирование всей таблицы. Кроме того, для больших таблиц требования места на диске для иерархических индексов могут оказаться чрезмерными.

Для решения этих проблем разработаны индексы на основе битовых карт, и они используются в хранилищах более часто.

3.2.2. Индексы на основе битовых карт

Индексы на основе битовых карт (bitmapped indexes) разработаны для ответов на запросы, которые затрагивают столбцы с небольшим числом возможных значений, каждое из которых потенциально соответствует большому числу строк. Число возможных значений для столбца называется количеством элементов (cardinality). В отличие от иерархических индексов, хранящих указатели на строки, используя идентификаторы строк (rowids), индекс на основе битовых карт хранит для каждого возможного значения столбца битовую карту.

Таблица 3.1. *Индекс на основе битовых карт «пол»*

Строка	Пол	Битовая карта «Male»	Битовая карта «Female»
1	M	1	0
2	F	0	1
3	M	1	0
4	M	1	0
5	M	1	0
6	F	0	1
7	M	1	0
8	M	1	0
9	F	0	1
10	M	1	0

Битовая карта для каждого из возможных значений имеет столько битов, сколько строк есть в таблице. Бит устанавливается, если соответствующая строка имеет данное значение. В табл. 3.1 показан индекс на основе битовых карт для столбца «пол» (количество элементов = 2, возможные значения — «M» и «F»). Создаются две битовые карты: одна для значения «M», другая для значения «F». В карте для «M» все строки, соответствующие покупателям-мужчинам, имеют биты, установленные в 1, а покупателям-женщинам — биты, установленные в 0.

Индексы на основе битовых карт по двум столбцам таблицы можно эффективно комбинировать, используя булевы операторы AND и OR. Это позволяет использовать их в запросах с множественными условиями, затрагивающими индексированные данным индексом столбцы таблицы. Это дает то преимущество, что вы можете отвечать на очень разнообразные запросы с помощью всего нескольких одностолбцовых индексов на основе битовых карт. Например, предположим, что у нас есть таблица покупателей, которая содержит имя покупателя, пол и профессию (учитель, инженер, лицо свободной профессии, домохозяйка, врач и т. п.). Мы рассматриваем новое предложение, нацеленное на женщин

свободных профессий, и хотели бы найти города для этого предложения. Это можно выразить в форме такого запроса:

```
SELECT customer_name, town
FROM customer
WHERE sex = 'F' AND occupation = 'Self-Employed';
```

В табл. 3.2 показана битовая карта для столбца «профессия».

Таблица 3.2 *Индекс на основе битовых карт «профессия»*

Строка	Профессия	Учитель	Инженер	Свободная профессия	Домохозяйка
1	Учитель	1	0	0	0
2	Инженер	0	1	0	0
3	Учитель	1	0	0	0
4	Учитель	1	0	0	0
5	Инженер	0	0	0	0
6	Свободная профессия	0	0	1	0
7	Свободная профессия	0	1	1	0
8	Домохозяйка	0	0	0	1
9	Инженер	0	1	0	0
10	Домохозяйка	0	0	0	1

Битовую карту «пол», показанную в табл. 3.1, можно объединить с картой «профессия», показанной в табл. 3.2 и быстро определить те строки таблицы покупателей, которые удовлетворяют запросу. Результат показан в табл. 3.3.

Таблица 3.3. *Использование индексана основе битовых карт — сколько покупателей-женщин свободной профессии?*

Строка	Битовая карта «female»	Свободная профессия	Результат
1	0	0	0 ;
2	1	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	1	1	1 (женщина свободной профессии)
7	0	1	0
8	0	0	0
9	1	0	0
10	0	0	0

Оператор SQL, используемый для создания индекса на основе битовых карт по столбцу customer.sex, выглядит следующим образом:

```
CREATE BITMAP INDEX easydw.customer_gender_index
 ON customer(sex)
pctfree 5 tablespace indx
storage (initial 64k next 64k pctincrease 0);
```

В качестве альтернативы, вы можете создать индекс в Oracle Enterprise Manager, как показано на рис. 3.1. Чтобы создать индекс на основе битовых карт, а не иерархический индекс, отметьте галочкой слово bitmap (на основе битовых карт).

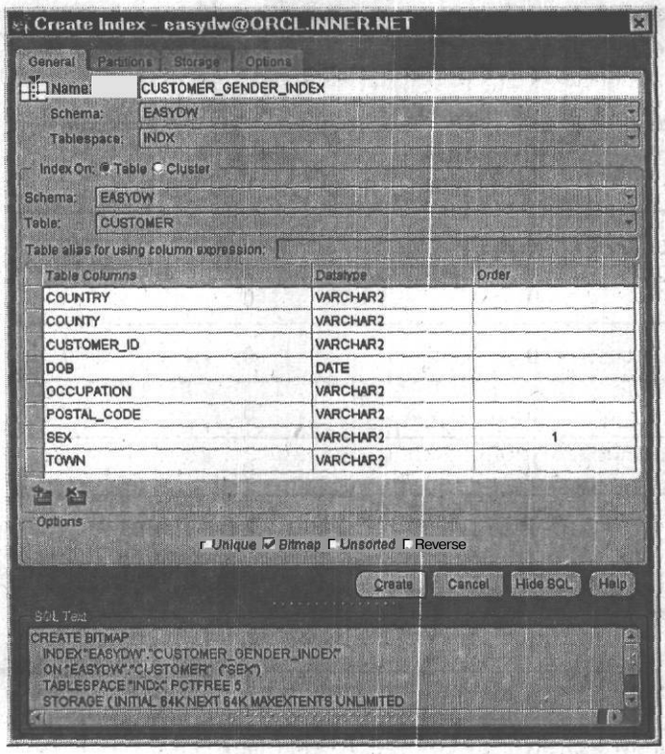


Рис. 3.1. *Создание индекса на основе битовых карт в Oracle Enterprise Manager*

Вы можете указать параметры сохранения и увидеть код SQL, генерируемый для создания индекса.

Индексы на основе битовых карт можно сжимать, и они, следовательно, потребуют значительно меньше места, чем соответствующие иерархические индексы. Экономия места может оказаться весьма значительной, если число возможных значений мало в сравнении с числом строк в таблице.

Недостаток индексов на основе битовых карт состоит в том, что их сложно обслуживать, когда данные в таблице изменяются. Это объясняется тем, что битовые карты должны быть разжаты, сжаты вновь и, возможно, перестроены. К счастью, в хранилищах отдельные строки обновляются очень редко. Обслуживание индексов на основе битовых карт оптимизировано для распространенного случая загрузки новых данных в хранилище сразу «оптом». В данном случае имеющиеся битовые карты не нужно менять. Они просто расширяются, включая в себя новые

строки. Кроме того, в иерархических индексах при обновлении строки недоступным становится только эта строка, а в индексах на основе битовых карт может быть заблокирована значительная часть карты. Таким образом, индексы на основе битовых карт снижают параллелизм системы и поэтому не подходят для систем с большим числом параллельных операций обновления.

3.2.3. Соединительный индекс на основе битовых карт

В Oracle 9i предлагается разновидность индекса на основе битовых карт, называющаяся «соединительный индекс на основе битовых карт» (bitmap join index). Данный индекс можно использовать для индексирования результата соединения таблицы фактов с одной или более таблицами измерений. Для иллюстрации Концепции соединительных индексов на основе битовых карт рассмотрим такой запрос: сколько тратят в нашем магазине женщины?

```
SELECT sum(p.purchase_price)
FROM purchases p, customer c .
WHERE p.customer_id = c.customer_id
  AND c.sex = 'F'
```

Мы могли бы создать обычный индекс на основе битовых карт по столбцу customer.sex, как это обсуждалось в предыдущем разделе, и быстро определить, какие покупатели являются женщинами. Однако нам, чтобы определить покупки, сделанные женщинами, окажется еще нужно подсчитать соединение между Покупками (purchases) и Покупателями (customers).

Если вместо этого мы сначала получим результат соединения, а потом создадим битовую карту, выявляющую строки, соответствующие покупкам, сделанным мужчинами и женщинами, мы получим соединительный индекс на основе битовых карт. В табл. 3.4 приводится результат выполненного соединения таблиц покупок и покупателей (показаны только первые десять строк).

Таблица 3.4 . *Соединение таблиц покупок и покупателей*

Строка	Идентификатор покупателя (customer_id)	Пол	Стоимость покупки
1	AB123456	F	28.01
2	AB123457	F	28.01
3	AB123457	F	28.01
4	AB123457	F	28.01
5	AB123456	F	67.23
6	AB123457	F	67.23
7	AB123458	M	67.23
8	AB123459	M	67.23
9	AB123460	F	67.23
10	AB123461	M	50.71
...	...	...	...

Соединительный индекс на основе битовых карт, как показано в табл. 3.5, имеет две битовые карты: одну — для значения male, а другую — для значения female. Каждая строка в битовой карте соответствует одной строке таблицы покупок. Таким образом, мы можем сразу определить строки, соответствующие покупкам, сделанным женщинами, по битовой карте для значения female.

Соединительный индекс на основе битовых карт по Покупкам, соединенным с Покупателями, создается с помощью следующего оператора SQL. Соединяемые таблицы и условия их соединения указаны в условиях FROM и WHERE так, как это делалось в операторе SELECT. Таблица, по которой строится индекс, указывается в условии ON, как и в случае любого другого индекса.

```
CREATE BITMAP INDEX easydw.purchase_cust_index
ON
purchases (customer.sex)
FROM purchases, customer
WHERE purchases.customer_id = customer.customer_id;
```

Таблица 3.5. *Соединительный индекс на основе битовых карт для таблиц Покупок и Покупателей по столбцу* ***customer.sex***

Строка	Пол	Битовая карта **«male»**	Битовая карта «female»
1	F	0	1
2	F	0	1
3	F	0	1
4	F	0	1
5	F	0	1
6	F	0	1
7	M	1	0
8	M	1	0
9	F	0	1
10	M	1	0
...	...	...	...

Можно сразу заметить некоторые отличия между простым и соединительным индексами на основе битовых карт. Обычный индекс создается по столбцам одной таблицы, а соединительный индекс — по нескольким, т. е. сам индекс создается по таблице фактов **(«покупки»),** но столбцы индекса берутся из таблицы измерений **(«покупатели»).**

Как и в случае обычного индекса, столбцы, включаемые в соединительный индекс, должны иметь небольшое количество элементов (cardinality).

Чтобы создание соединительного индекса имело смысл, число строк в результате соединения должно быть равно числу строк в таблице фактов, по которой оно создается (в нашем примере — числу строк в таблице покупок). Чтобы добиться этого, столбец таблицы измерений, который соединяет ее с таблицей фактов (в нашем примере — столбец cutomer.customer_id), должен иметь ограничение, обеспечивающее уникальность его значений.

Соединительные индексы на основе битовых карт накладывают некоторые ограничения на параллельность операций DML над таблицами, задействованными в индексе, однако для большинства приложений, связанных с хранилищами, это не является крупной проблемой.

Альтернативой соединительным индексам на основе битовых карт являются представления с соединениями, которые мы обсудим в главе 4.

3.3. Секционирование данных

3.3.1. Преимущества секционирования

Когда какая-то задача кажется неподъемной, ее выполнение облегчится, если разбить ее на несколько меньших задач. Представьте, что вы упаковываете вещи, собираясь переезжать. Будет легче, если разделить процесс по комнатам. Если каждый член семьи будет упаковывать вещи из одной комнаты, все вещи будут упакованы быстрее. Такова идея, лежащая в основе секционирования и параллелизма.

Очень большие таблицы и индексы можно разделить на меньшие, более управляемые разделы. Секционирование данных позволяет производить операции, связанные с управлением, на уровне разделов. Многие операции, такие, как загрузка данных, построение локальных индексов, ввод ограничений, сбор статистики работы оптимизатора, очистка данных, резервное копирование и восстановление, можно осуществлять над разделами.

Секционирование данных также позволяет использовать преимущества функций параллельной обработки Oracle при выполнении запросов к таблицам. Кроме того, операции DML над таблицами могут осуществляться параллельно, что дает существенный прирост производительности.

Если таблица секционирована, оптимизатор запросов может определить, может ли ответ на данный вопрос быть дан после чтения только некоторых конкретных разделов. Так можно избежать затратного сканирования всей таблицы. Это свойство известно как отбрасывание разделов (partition elimination) или динамическое отсечение разделов (dynamic partition pruning). Если хранилище секционировано по месяцам, а запрос относится к продажам декабря 1998 года,

оптимизатор будет знать, в каком разделе хранятся данные, и отбросит при поиске разделы, содержащие другие месяцы.

3.3.2. Как секционировать данные

Таблицу можно секционировать, используя любой столбец этой таблицы. Этот столбец называется ключом секционирования (partition key). Важно проводить секционирование по столбцу, данные в котором не меняются, например, по дате покупки. Если секционирование проводится по идентификатору продуктов, а предприятие изменило схему их кодирования, то обновление столбца секционирования потребует переноса данных в другой раздел, что может занять много времени.

Для хранилищ данных обычным является секционирование по времени. Это позволяет выполнять операции по типу «движущейся рамки». Если наше хранилище хранит данные за один год, то в конце апреля 2002 года мы можем добавить новый раздел с данными этого месяца и удалить данные апреля 2001 года. В главе 5 мы рассмотрим загрузку данных в новый раздел.

Секционирование повышает доступность данных хранилища. Если разместить каждый раздел на своем диске, то при аварии этого диска недоступными окажутся данные только этого раздела, а не вся таблица. Операции по обслуживанию можно проводить над одним разделом, в то время как пользователь будет продолжать иметь доступ к прочим разделам таблицы.

Для максимальной производительности каждый раздел следует хранить в отдельном табличном пространстве, а каждое табличное пространство — на одном или более отдельном устройстве, согласно ситуации. В больших системах пути ввода-вывода должны также быть распределены по разным контролерам и каналам. Распределив данные по нескольким физическим дискам, вы сбалансируете ввод-вывод и улучшите производительность обработки запросов, доступность и управляемость.

Существует несколько путей секционирования данных, включая следующие:

- по диапазону значений данных (линейное секционирование);
- по хэш-функции (хэш-секционирование);
- с использованием дискретных значений (списочное секционирование);
- с использованием комбинации методов (комбинированное секционирование).

Линейное секционирование

При линейном секционировании (range partitioning) данные секционируются на неперекрывающиеся диапазоны значений При вставке строки в таблицу она

помещается в соответствующий раздел на основе значения в столбце ключа секционирования. Линейное секционирование особенно полезно, когда ключом секционирования является время. Это позволяет оптимизатору выполнять отсечение разделов для запросов, содержащих конкретное значение или диапазон значений ключа секционирования.

В схеме EASYDW покупки каждого месяца сохраняются в отдельном разделе. Таблица фактов «покупки» секционируется линейно, столбец time_key (ключ времени) служит ключом секционирования. Каждый раздел сохраняется в отдельном табличном пространстве, распределенном по нескольким дискам, как это показано на рис. 3.2.

Код SQL, который создает такую таблицу с секционированием, показан ниже. Условие VALUES LESS THAN показывает верхнюю границу значений ключа секционирования в данном разделе. Нижняя граница (если есть) указана в условии VALUES LESS THAN предыдущего раздела. В нашем примере раздел purchases_feb2002 содержит данные со значениями ключа времени в диапазоне ?01-02-2002 и <01-03-2002.

```
CREATE TABLE easydw.purchases
(product_id varchar2(8),
 time_key date,
 customer_id varchar2(10),
 purchase_date date,
 purchase_time number(4,0),
```



Рис. 3.2. *Линейное секционирование*

```
 purchase_price number(6,2),
 shipping_charge number(5,2),
 today_special_offer varchar2(1))
PARTITION by RANGE (time_key)
 (partition purchases_jan2002
   values less than (TO_DATE('01-02-2002', 'DD-MM-YYYY'))
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_jan2002,
 partition purchases_feb2002
   values less than (TO_DATE(' 01-03-2002', 'DD-MM-YYYY'))
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_feb2002,
 partition purchases_mar2002
   values less than (TO_DATE('01-04-2002', 'DD-MM-YYYY'))
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_mar2002,
 partition purchase_catchall
   values less than (MAXVALUE)
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_maxvalue);
```

Отметьте, что последний раздел таблицы покупок имеет особую границу, называемую MAXVALUE. Этот раздел — необязательный раздел-ловушка, в который собираются все строки, не соответствующие ни одному из диапазонов значений.

Хэш-секционирование

При линейном секционировании возможна ситуация, когда данные не распределяются равномерно по разделам. Некоторые разделы могут оказаться очень большими, а другие — маленькими. Если данные были секционированны по месяцам, а в некоторые месяцы объем продаж был выше, чем в прочие, это может привести к очень большим различиям в размерах разделов. Когда наблюдается такая неравномерность, формируется «горячая точка», в которой происходит конфликт за ресурсы.

Хэш-секционирование (hash partitioning) уменьшает подобного рода неравномерность путем применения к ключу секционирования **хэш-функции.** Получившееся значение определяет, в каком разделе нужно сохранить данную строку. Так что если вместо секционирования по месяцам мы разобьем таблицу покупок **хэш-секционированием** по идентификатору продукта **(product_id),** то строки, относящиеся к одному месяцу, окажутся разбросаны по разным разделам, как показано на рис. 3.3. Заметим, что все строки, относящиеся к одному продукту, попадут в один раздел. Однако нельзя указать, в каком разделе какой продукт будет содержаться.

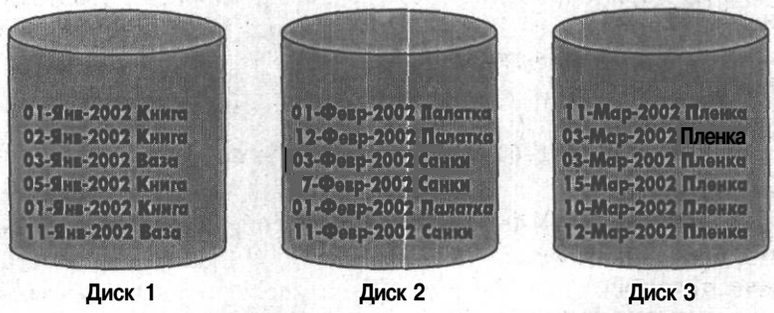


Рис. 3.3. *Хэш-секционирование*

В приведенном примере показан код SQL, создающий таблицу с хэш-секционированием.

```
CREATE TABLE easydw.purchases
  (product_id varchar2(8),
  time_key date,
  customer_id varchar2(10),
  purchase_date date,
  purchase_time number(4,0),
  purchase_price number(6,2),
  shipping_charge number(5,2),
  today_special_offer varchar2(1))
PARTITION BY HASH(product_id)
PARTITIONS 3;
```

Хэш-секционирование хорошо применять при поиске строк, соответствующих одному значению, но в данном случае оптимизатор не может выполнять отсечение разделов при обработке запросов на диапазон значений. Если у нас есть запрос по конкретному продукту, например: «Сколько палаток мы продаем ежемесячно?», — оптимизатор сможет определить, какой раздел проверять. Однако если у нас будет запрос: «Какие продукты мы продали в январе?», — нам придется проводить поиск во всех разделах.

При хэш-секционировании пользователь не знает и не может управлять тем, какие продукты в какой раздел попадут. Вы можете лишь указать, сколько разделов вы хотели бы иметь. Это осложняет операции обслуживания, требующие знания данных, например архивирование старых данных таблицы. Поэтому хэш-секционирование само по себе используется редко, однако применяется в связке с линейным секционированием.

Списочное секционирование

В некоторых случаях может оказаться неудобным организовывать данные по диапазонам значений. Данные могут не иметь естественного ключа секционирования, такого, как время. В интересах бизнеса может понадобиться свести вместе значения, лежащие далеко друг от друга. Например, если у нас есть данные о продажах по разным штатам США, поместить в один раздел данные по всем штатам Новой Англии будет непросто. В Oracle 9i для решения этой проблемы предложен новый тип секционирования. Этот метод, известный как списочное секционирование (list partition), позволяет распределить данные по отдельным значениям столбцов. На рис. 3.4 показан пример таблицы со списочным секционированием.

В следующем выражении SQL списочное секционирование используется для организации данных об объемах продаж по регионам. Последний раздел — раздел-ловушка, в который попадают строки, не соответствующие ни одному из прочих разделов.

```
CREATE TABLE easydw.regional_sales
 (state varchar2(2),
   store_number number,
   dept_number number,
   dept_name varchar2(10),
   sales_amount number  (6,2)
)
```



Рис. 3.4. *Списочное секционирование*

```
PARTITION BY LIST(state)
(
  PARTITION northeast VALUES ('NH', 'VT', 'MA', 'RI', 'CT'),
  PARTITION southeast VALUES ('NC', 'GA', 'FL'),
  PARTITION northwest VALUES ('WA', 'OR'),
  PARTITION midwest VALUES ('IL', 'WI', 'OH'),
  PARTITION west VALUES ('CA', 'NV', 'AZ'),
  PARTITION otherstates VALUES (DEFAULT));
```

Списочное секционирование позволяет оптимизатору запросов выполнять отсечение разделов при обработке запросов, связанных с конкретными значениями ключа секционирования. Например, запрос данных по штатам Массачусетс или Нью-Хэмпшир требует доступа только к разделу Северо-восточного региона.

Комбинированное секционирование

Oracle предоставляет двухуровневую схему, называющуюся комбинированное секционирование (composite partitioning) и объединяющую преимущества двух

методов. При комбинированном секционировании данные делятся на разделы с помощью одной схемы секционирования, а потом секционируются на подразделы с применением другой схемы. В настоящее время Oracle поддерживает два вида схем комбинированного секционирования:

- линейное-хэш;
- линейно-списочное.

Ранее мы говорили, что **хэш-секционирование** не позволяет пользователю контролировать распределение данных. С другой стороны, линейное **секционирование** страдает от потенциальной проблемы неравномерности данных. Комбинированное линейное-хэш секционирование объединяет преимущества линейного и **хэш-секционирования.** Сначала данные секционируются по диапазонам, а затем делятся на подразделы с помощью **хэш-функции.** При секционировании по дате последний раздел часто является «горячей точкой». При дальнейшем секционировании при помощи хэш-функции снижается нагрузка на каналы ввода-вывода, поскольку доступ распределяется среди подразделов.

Все разделы таблицы или индекса имеют одинаковые логические атрибуты, но могут иметь разные физические атрибуты. Например, все разделы таблицы имеют одинаковые определения столбцов и ограничений, но могут иметь разные атрибуты хранения и физическое размещение. Подразделы одного раздела могут размещаться *в* разных табличных пространствах. Это позволяет распределять данные по многим физическим устройствам, снижая нагрузку на каналы ввода-вывода.

На рис. 3.5 данные сначала секционируются по месяцам, а дальнейшее секционирование производится по продукту. Каждый раздел имеет три подраздела. Каждый подраздел сохраняется в отдельном табличном пространстве. Код SQL, создающий таблицу с комбинированным секционированием, **показан** ниже. Условие STORE IN позволяет задать имя табличного пространства, в котором будет сохраняться каждый из подразделов.

```
CREATE TABLE easydw.purchases
(product_id varchar2(8),
 time_key date,
 customer_id varchar2(10),
 purchase_date date,
 purchase_time  number(4,0),
 purchase_price number(6,2),
 shipping_charge  number(5,2),
 today_special_offer varchar2(1))
```

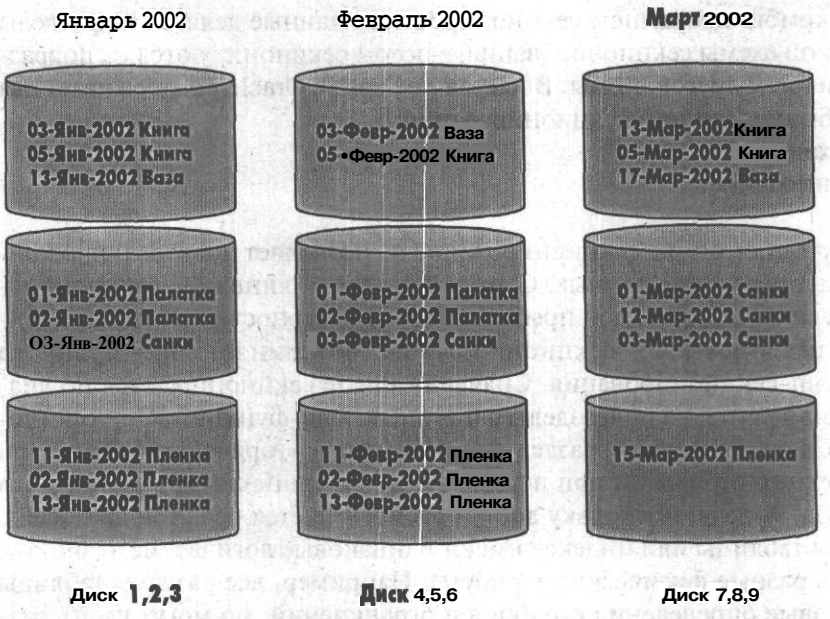


Рис. 3.5. *Комбинированное линиейное-хэш секционирование*

```
PARTITION by RANGE (time_key)
 SUBPARTITION BY HASH(product_id)
 SUBPARTITIONS 3
(partition purchases_jan2002
 values less than (TO_DATE('01-02-2002', 'DD-MM-YYYY'))
 pctfree 0 pctused 99
 storage (initial 64k next 16k pctincrease 0)
 STORE IN (purchases_jan2002,
     purchases_jan2002_2,
     purchases_jan2002_3) ,      ,
partition purchases_feb2002
 values less than (TO_DATE('01-03-2002', 'DD-MM-YYYY'))
 pctfree 0 pctused 99
 storage (initial 64k next 16k pctincrease 0)
 STORE IN (purchases_feb2002,
```

```
     purchases_feb2002_2,
     purchases_feb2002_3),
partition purchases_mar2002
 values less than (TO_DATE('01-04-2002', 'DD-MM-YYYY'))
 pctfree 0 pctused 99
 storage (initial 64k next 16k pctincrease 0)
 STORE IN (purchases_mar2002,
     purchases_mar2002_2,
     purchases_mar2002_3));
```

Метод линейно-списочного комбинированного секционирования сначала делит таблицу на основе непрерывного ключа, например ключа времени, а затем делит каждый раздел на подразделы, используя дискретные значения. В приводимом ниже примере таблица линейно секционируется по месяцам, а затем секционируется списочным секционированием по штатам. Если таблица продаж секционирована с помощью линейно-списочного секционирования, может быть эффективно обработан запрос объемов продаж для дат января месяца в штатах Новой Англии.

```
CREATE TABLE sales
(state varchar2(2),
 store_number number,
 dept_number number,
 dept_name varchar2(10) ,
 sales_amount number (6,2),
 sale_date date,
 item_number number (10)
)
PARTITION BY RANGE (sale_date)
 SUBPARTITION BY LIST(state)
 SUBPARTITION TEMPLATE
  (SUBPARTITION «NorthEast» VALUES ('NH','VT','MA','RI','CT')
                            TABLESPACE sales_ne,
 SUBPARTITION «SouthEast» VALUES ('NC','GA','FL')
                            TABLESPACE sales_se,
 SUBPARTITION «Northwest» VALUES ('WA','OR')
                                  TABLESPACE sales_nw,
 SUBPARTITION «Midwest» VALUES ('IL','WI','OH')
                            TABLESPACE sales_mw,
```

```
 SUBPARTITION «West» VALUES ('CA', 'NV', 'AZ')
                          TABLESPACE sales_w)
(
PARTITION sales_jan_2002
  VALUES LESS THAN (TO_DATE('01-Feb-2002', 'DD-Mon-YYYY')),
PARTITION sales_feb_2002
  VALUES LESS THAN (TO_DATE('01-Mar-2002', 'DD-Mon-YYYY')),
PARTITION sales_mar_2002
  VALUES LESS THAN (TO_DATE('01-Apr-2002', 'DD-Mon-YYYY'))
);
```

В этом примере мы показываем использование условия SUBPARTITION TEMPLATE (шаблон подраздела). Спецификация линейно-списочного секционирования может получиться очень длинной, поскольку нужно указывать подробные условия деления на подразделы каждого раздела. Часто нужно указать одинаковое списочное секционирование всех разделов. Условие SUBPARTITION TEMPLATE делает удобным определение одинаковой информации о секционировании на подразделы для всех линейных разделов таблицы. Oracle сгенерирует имена подразделов, используя комбинацию имени раздела и имени, указанного в шаблоне. Затем, в соответствии с определениями в шаблоне, будут созданы подразделы. Информацию о секционировании таблицы на разделы и подразделы можно получить в представлениях словаря (dictionary views) user_tab_partition и user_tab_subpartition. В нашем примере имена разделов и подразделов будут следующие.

```
SELECT partition_name, subpartition_name
FROM user_tab_subpartitions
WHERE table_name = 'SALES';

PARTITION_NAME       SUBPARTITION_NAME
---------------      ------------------------------
SALES_JAN_2002       SALES_JAN_2002_NorthEast
SALES_JAN_2002       SALES_JAN_2002_SouthEast
SALES_JAN_2002       SALES_JAN_2002_NorthWest
SALES_JAN_2002       SALES_JAN_2002_MidWest
SALES_JAN_2002       SALES_JAN_2002_West
SALES_FEB_2002       SALES_FEB_2002_NorthEast
SALES_FEB_2002       SALES_FEB_2002_SouthEast
SALES_FEB_2002       SALES_FEB_2002_NorthWest
```

```
SALES_FEB_2002        SALES_FEB_2002_MidWest
SALES_FEB_2002        SALES_FEB_2002_West
...
```

Вы можете не использовать условие SUBPARTITION TEMPLATE, а указать значения для списочных подразделов и их имена в каждом линейном разделе.

В следующем примере запрашивается общий объем продаж по штатам NH (Нью-Хемпшир), MA (Массачусетс) и CT (Коннектикут) в некотором диапазоне дат. Оптимизатор определяет, что ответ на этот запрос может быть быстро дан после чтения первого подраздела разделов за февраль и март. Значения Pstart и Pstop на выходе утилиты EXPLAIN PLAN показывают диапазон разделов, используемых при ответе на запрос, в данном случае — разделы 2–3.

```
EXPLAIN PLAN FOR
SELECT store_number, dept_number, SUM(sales_amount) as ql_sales
FROM sales
WHERE sale_date between TO_DATE('15-Feb-2002', 'DD-Mon-YYYY')
  and TO_DATE('15-Mar-2002', 'DD-Mon-YYYY')
  and state in ('NH', 'MA', 'CT')
GROUP BY store_number, dept_number;
```

```
-------------------------------------------------------------------
|Id| Operation                |Name |Rows |Cost | Pstart| Pstop  |
-------------------------------------------------------------------
|0 | SELECT STATEMENT         |     | 1   | 16  |       |        |
|1 | SORT GROUP B Y           |     | 1   | 16  |       |        |
|2 | PARTITION RANGE ITERATOR |     |     |     | 2     | 3      |
|3 | PARTITION LIST INLIST    |     |     |     |KEY(I) |KEY(I)  |
|4 | TABLE ACCESS FULL        |SALES| 1   | 14  | KEY   | KEY    |
-------------------------------------------------------------------
```

Совет. Линейно-списочное секционирование особенно хорошо в качестве схемы секционирования в представлениях, которые, как правило, содержат итоговые данные по многим измерениям, одним из которых является время.

Многостолбцовые ключи секционирования

Ключ секционирования, который применяется в линейном секционировании, может содержать несколько столбцов. Следующий фрагмент кода SQL создает таблицу с линейным секционированием и использованием многостолбцового ключа.

```
CREATE TABLE easydw.purchases
 (product_id varchar2(8),
  time_key date,
  customer_id varchar2(10),
  purchase_date  date,
  purchase_time number(4,0),
  purchase_price number(6,2),
  shipping_charge  number(5,2),
  today_special_offer varchar2(1))
PARTITION by RANGE (time_key, product_id)
  (partition purchases_jan2002_100
   values less than (TO_DATE('31-01-2002','DD-MM-YYYY'), 100)
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_jan2002_100,
  partition purchases_jan2002_200
   values less than (TO_DATE('31-01-2002','DD-MM-YYYY'), 200)
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_jan2002_200,
  partition purchases_feb2002_all
   values less than (TO_DATE('28-02-2002','DD-MM-YYYY'), 100)
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_feb2002,
  partition purchases_mar2002_all
   values less than (TO_DATE('31-03-2002','DD-MM-YYYY'), 100)
   pctfree 0 pctused 99
   storage (initial 64k next 64k pctincrease 0)
   tablespace purchases_mar2002
);
```

Чтобы понять, каким разделам соответствуют данные, давайте вставим несколько значений в эту таблицу и посмотрим, в какие разделы они попадут.

```
insert into purchases (product_id, time_key)
   values (1, TO_DATE('15-01-2002', 'DD-MM-YYYY'));
insert into purchases (product_id, time_key)
   values (150, TO_DATE('15-01-2002', 'DD-MM-YYYY'));
insert into purchases (product_id, time_key)
   values (101, TO_DATE('31-01-2002', 'DD-MM-YYYY'));
insert into purchases (product_id, time_key)
   values (170, TO_DATE('31-01-2002', 'DD-MM-YYYY'));
insert into purchases (product_id, time_key)
   values (200, TO_DATE('31-01-2002', 'DD-MM-YYYY'));
insert into purchases (product_id, time_key)
   values (1, TO_DATE('28-02-2002', 'DD-MM-YYYY'));

SELECT product_id, time_key
FROM purchases partition(purchases_jan2002_100);

PRODUCT_ID    TIME_KEY
----------    -----------
1             15-JAN-2002
150           15-JAN-2002

SELECT product_id, time_key
FROM purchases partition(purchases_jan2002_200);

PRODUCT_ID    TIME_KEY
----------    -----------
101           31-JAN-2002
170           31-JAN-2002

SELECT product_id, time_key
FROM purchases partition(purchases_feb2002_all);

PRODUCT_ID    TIME_KEY
----------    -----------
200           31-JAN-2002
1             28-FEB-2002
```

Заметьте, что строка от 15 января 2002 для идентификатора продукта = 150 попадает в первый раздел, а не во второй. Это, вероятно, не то, что вы ожидаете. В данном случае проверяется условие, что в первый раздел попадают значения (time_key < '31-01-2002') или (time_key = '31-01-2002' и product_id < 100). Если значения time_key и product_id не соответствуют первому разделу, проверяется условие второго раздела: (time_key < '31-01-2002') или (time_key = '31-01-2002' и product_id < 200). Секционирование с несколькими ключами может быть полезно, если у вас на одно значение ключа времени приходится много значений идентификаторов продуктов (например, в канун Рождества может быть сделано несколько покупок). В таком случае вы можете провести секционирование по данному значению и распределить данные по нескольким разделам.

Схему линейного секционирования с многостолбцовыми ключами не следует путать с комбинированным линейно-линейным секционированием. На момент написания этой книги Oracle не поддерживает линейно-линейное секционирование. Если бы его можно было использовать, то строка со значением time_key = 15-Jan-2002 и product_id = 150 в предыдущем примере попала бы во второй раздел.

Если обобщить все сказанное, линейное секционирование следует использовать, когда в таблице имеется непрерывный ключ, например время. Списочное секционирование идеально подходит для таблиц, в которых нужно поместить в один раздел строки с конкретными дискретными значениями. Хэш-секционирование равномерно распределяет данные по всем разделам и может применяться в комбинации с линейным секционированием во избежание появления «горячих точек» в таблице. Наконец, линейно-списочное секционирование можно использовать, когда таблица хранит данные из многих измерений, как непрерывных (например, время), так и дискретных (например, продукт или географический регион). Использование секционирований в хранилище дает массу преимуществ, повышая производительность обработки запросов, управляемость и доступность данных.

В следующем разделе мы рассмотрим, как можно применить секционирование к индексу.

3.3.3. Секционирование индексов

И иерархические индексы, и индексы на основе битовых карт можно секционировать с помощью любого доступного метода секционирования. Их можно секционировать так же, как таблицу, или же по другому критерию. Иерархический

индекс можно секционировать, даже если соответствующая таблица не секционирована. Индексы с секционированием могут быть двух типов:

- глобальные;
- локальные.

Глобальные индексы

В глобальном индексе ключи секционирования индекса могут не соответствовать никаким разделам или подразделам таблицы. Глобальный индекс может быть вообще не секционирован или же может иметь ключ секционирования, совершенно отличающийся от ключа секционирования таблицы. Однако в секционированом глобальном индексе ключ секционирования должен быть первым столбцом ключа индекса. Индексы на основе битовых карт для секционированных таблиц глобальными быть не могут.

На рис. 3.6 показан глобальный индекс purchases_product_index для таблицы покупок. В данном случае таблица покупок секционирована по ключу времени (time_key), а индекс секционирован по идентификатору продукта (product_id). Первым столбцом индекса также является идентификатор продукта.

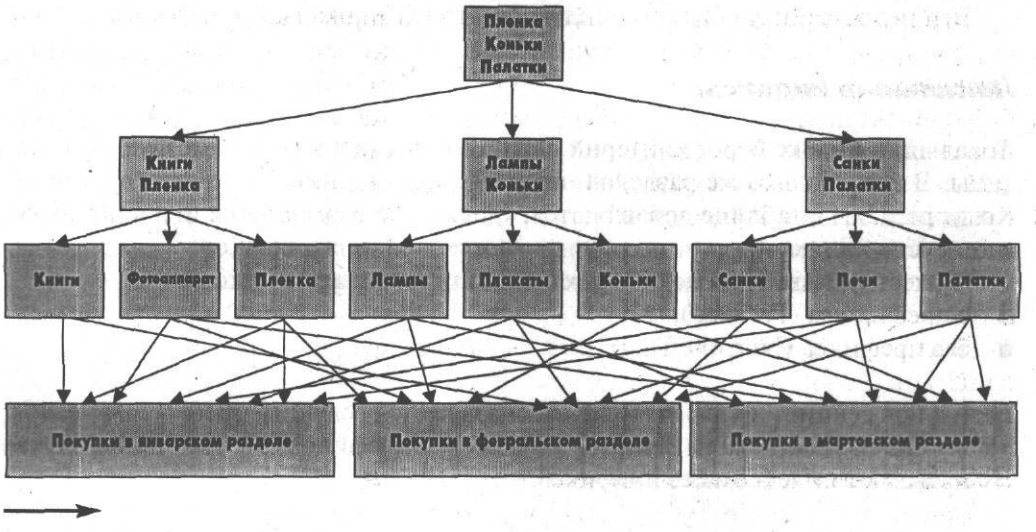


Рис. 3.6. *Глобальный индекс*

В следующем фрагменте показан код SQL, создающий такой индекс.

```
CREATE INDEX easydw.global_product_index on purchases
 (product_id)
 global
 partition by range (product_id)
 (partition spl000 values less than ('SP1000'),
  partition sp2000 values less than ('SP2000'),
  partition other values less than (maxvalue) );
```

Рис. 3.6 иллюстрирует то, почему глобальные индексы не слишком эффективны для обработки запросов. Поскольку между разделами индекса и таблицы отсутствует соответствие, доступ к значениям раздела таблицы может потребовать доступа к нескольким или сразу всем разделам индекса.

Кроме того, когда данные в соответствующем разделе таблицы перемещаются или удаляются, все разделы глобального индекса повреждаются и индекс помечается как UNUSABLE (неиспользуемый). Для перестройки индекса нужно дать команду ALTER INDEX REBUILD. (Заметим, что в Oracle 9i release 2 существует возможность перестроить поврежденный глобальный индекс, указав условие UPDATE GLOBAL INDEXES в операции секционирования).

Эти недостатки глобальных индексов учтены в локальных индексах.

Локальные индексы

Локальный индекс берет критерий секционирования из соответствующей таблицы. В нем столько же разделов, подразделов и границ, сколько и в таблице. Когда разделы в таблице добавляются, удаляются, разделяются или сливаются, соответствующим образом модифицируются и разделы индекса.

Существует два типа локальных секционированных индексов:

- с префиксом (Prefixed);
- без префикса (Non-prefixed).

Если ключ секционирования индекса оказывается ведущим префиксом (leading prefix) индекса, индекс называется индексом с префиксом, в противном случае его называют индексом без префикса.

Локальные индексы с *префиксом*

В нашем примере таблица покупок секционируется по ключу времени. На рис. 3.7



Рис. 3.7. *Локальный индекс с префиксом*

для этой таблицы приводится локальный индекс с префиксом, purchase_time_index. Ключом секционирования является ключ времени (time_key), а столбцами индекса являются ключ времени и идентификатор покупателя (time_key, customer_id). Ключ секционирования, time_key, является префиксом ключа индекса.
Следующий код SQL создает такой индекс. Схема секционирования локального индекса та же, что и для таблицы, но вы можете задать имена разделов, а также табличных пространств, в которых располагается каждый раздел.

```
CREATE INDEX easydw.purchase_time_index
 ON purchases (time_key,, customer_id)
local
  (partition indexJan2002 tablespace purchases_jan2002_idx,
   partition indexFeb2002 tablespace purchases_feb2002_idx,
   partition indexMar2002 tablespace purchases_mar2002_idx );
```

При таком индексе, если мы захотим узнать, какие покупки сделал конкретный покупатель в январе, нам нужно будет только поискать в разделе indexJan2002.

Локальные индексы без префикса

Локальные индексы, в которые не входит в качестве ведущего столбца индекса ключ секционирования, называются локальными индексами без префикса. Такие индексы полезны, если мы хотим провести секционирование по одному столбцу для удобства обслуживания, а для извлечения данных проиндексировать другие столбцы. Например, может оказаться нужным секционировать

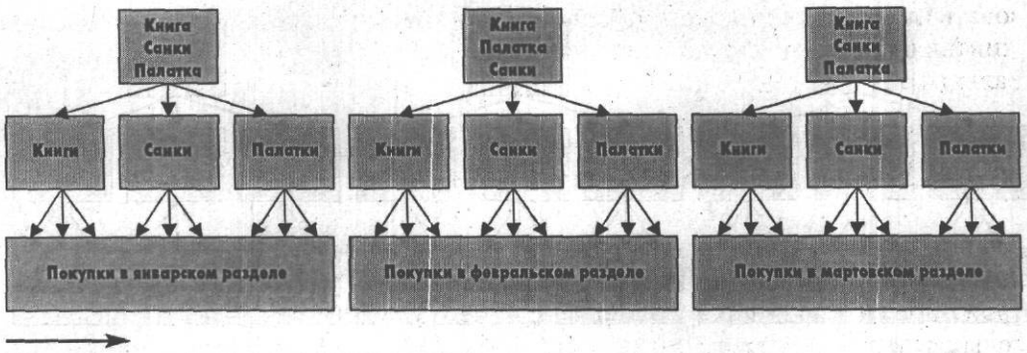


Рис. 3.8 *Локальные индексы без префикса*

индексы и таблицы по ключу времени, чтобы легче было добавлять данные новых месяцев, но чтобы получить хорошую производительность обработки запросов объемов продаж по продуктам, нам будет нужен индекс по идентификаторам продуктов.

На рис. 3.8 показан локальный индекс без префикса для таблицы покупок по столбцу идентификатора продуктов product_id. Схема секционирования та же, что и для таблицы покупок (т. е., по столбцу time_key). Приведенный пример показывает, как можно создать такой индекс.

```
CREATE BITMAP INDEX easydw.purchase_product_index
ON purchases  (product_id)
local
  (partition indexJan2002 tablespace purchases_jan2002_idx,
   partition indexFeb2002 tablespace purchases_feb2002_idx,
   partition indexMar2002 tablespace purchases_mar2002_idx);
```

При таком индексе, если производить поиск объема продаж определенного продукта, например палаток, оптимизатор не сможет выполнять операцию отсечения разделов и должен будет проводить поиск данных по всем месяцам. Однако при запросе «Сколько палаток продано январе?» — оптимизатор сможет отбросить разделы индекса, посвященные февралю и марту.

Если обобщить сказанное, локальные индексы с префиксом являются наиболее эффективным типом индекса с точки зрения производительности обработки запросов, поскольку оптимизатор сможет использовать отсечение разделов и не заглядывать в ненужные. Локальные индексы также облегчают операции по типу «сдвигающейся рамки». Если возможно, избегайте исполь-

зовать глобальные индексы, поскольку они требуют дополнительного обслуживания и потому что индексы на основе битовых карт не могут быть глобальными.

3.3.4. Технология RAID

Альтернативным методом распределения запросов на ввод-вывод по нескольким устройствам является технология хранения RAID — матрица недорогих устройств с избыточностью (Redundant Array of Inexpensive Disks). Производительность и надежность можно улучшить путем использования множества дисков одновременно. Вместо того чтобы хранить все данные в одном файле на одном диске, данные распределяются по многим дискам, что снижает конкуренцию за доступ к диску. Технологию RAID можно использовать во всех функциях хранилищ, о которых говорится в этой книге.

3.4. Сжатие данных

Oracle предлагает разнообразные пути сжатия данных для уменьшения места, необходимого для хранения таблиц и индексов. Преимущества сжатия максимальны, когда в данных содержится множество повторяющихся значений (например, таблицы с внешними ключами) и данные обновляются нечасто.

В иерархических индексах Oracle выполняет сжатие ключей, сохраняя общий префикс значений ключа индекса в блоке индекса только один раз. Число столбцов, используемых в префиксе, равно числу столбцов-ключей -1 для уникальных индексов и равно числу столбцов-ключей для неуникальных индексов. Индексно-ключевое сжатие можно задействовать с помощью ключевого слова COMPRESS в операторе CREATE INDEX.

В индексах на основе битовых карт битовую карту каждого значения данных Oracle сжимает автоматически. В тех столбцах, количество элементов в которых намного меньше числа строк в таблице, битовые карты, как правило, плотные (т. е. имеют много единиц) и могут быть сжаты весьма значительно. Таким образом, индекс на основе битовых карт может занимать намного меньше места, чем соответствующий ему иерархический индекс.

Еще один очень полезный тип сжатия — это сжатие сегмента данных, появившееся в Oracle 9i Release 2. Сжатие сегмента определяет в блоке данных повторяющиеся значения и помещает их в справочную таблицу в начале блока. Теперь строки данных указывают на значение в этой справочной таблице. Это

может значительно уменьшить занимаемое место, если в данных содержится много повторяющихся значений, например имен. Особенно это полезно для представлений с агрегатами (aggregations), поскольку значения в группируемых столбцах часто повторяются.

Чтобы использовать сжатие сегмента данных, вы должны указать ключевое слово COMPRESS при создании табличного пространства, таблицы или отдельных разделов таблицы с линейным или списочным секционированием. Это свойство недоступно при хэш-секционировании и в таблицах, организованных по типу индексов. В следующем примере показана таблица PURCHASES (покупки) с линейным секционированием, где первые три раздела сжаты, а последний оставлен несжатым.

```
CREATE TABLE easydw.purchases
(product_id varchar2(8),
 time_key date,
 customer_id varchar2(10),
 purchase_date date,
 purchase_time number(4,0),
 purchase_price number(6,2),
 shipping_charge number(5,2),
 today_special_offer varchar2(1))
PARTITION by RANGE (time_key)
 (partition purchases_jan2002
  values less than (TO_DATE('01-02-2002', 'DD-MM-YYYY'))
  pctfree 0 pctused 99
  storage (initial 64k next 64k pctincrease 0)
  tablespace purchases_jan2002 COMPRESS,
 partition purchases_feb2002
  values less than (TO_DATE('01-03-2002', 'DD-MM-YYYY'))
  pctfree 0 pctused 99
  storage (initial 64k next 64k pctincrease 0)
  tablespace purchases_feb2002 COMPRESS,
 partition purchases_mar2002
  values less than (TO_DATE('01-04-2002', 'DD-MM-YYYY'))
  pctfree 0 pctused 99
  storage (initial 64k next 64k pctincrease 0)
  tablespace purchases_mar2002 COMPRESS,
 partition purchase_catchall
  values less than (MAXVALUE)
```

```
  pctfree 0 pctused 99
  storage (initial 64k next 64k pctincrease 0)
  tablespace purchases_maxvalue NOCOMPRESS);
```

Если вы впервые вводите раздел со сжатием сегмента в ранее несжатую секционированную таблицу, вам будет нужно перестроить все индексы на основе битовых карт этой таблицы.

Совет. Чтобы добиться наилучшего сжатия, при добавлении данных в новый раздел сначала отсортируйте их, чтобы повторяющиеся значения как можно чаще попадали в один блок.

Изменять параметры сжатия можно с помощью команды ALTER TABLE MOVE. Например, чтобы сжать раздел purchase_catchall таблицы purchases, пишется такая команда:

```
ALTER TABLE purchases MOVE PARTITION purchases_catchall COMPRESS;
```

Если в сжатую таблицу добавляются новые строки с помощью традиционного оператора INSERT, эти новые данные не могут быть сжаты эффективно. Также не способствуют эффективности сжатия команды DELETE и UPDATE. Однако если данные вводятся с помощью DIRECT PATH (через SQL Loader или выражения INSERT /*+APPEND */), новые данные будут вставлены в новые блоки, которые могут быть сжаты. Поэтому сжатие сегментов хорошо подходит для хранилищ, в которых применяется пакетная загрузка данных, или для разделов, данные в которых по большей части доступны только для чтения.

Совет. Вам может понадобиться слить и сжать разделы, соответствующие старым данным, поскольку, скорее всего, они не будут очень часто изменяться. Это позволит держать больше данных в режиме он-лайн.

3.5. Техника оптимизации запросов

Существует несколько различных путей извлечения любой данной строки из базы. Для определения местоположения строки может быть использован индекс, если же таблица невелика, то может оказаться быстрее выполнить полное ее сканирование, последовательно читая каждую строку. Если имеется несколько индексов, оптимизатор определяет, какой лучше использовать. Если существует представление (materialized view), то для его использования запрос может быть переписан. Цель оптимизатора — выбрать стратегию для как можно более быстрой обработки запроса.

Существует два подхода к оптимизации запросов:

- оптимизация на основе правил;
- оптимизация на основе стоимости.

Оптимизатор, работающий на основе правил, использует набор правил, ранжирующих все имеющиеся пути доступа в соответствии со скоростью прохождения каждого пути. Оптимизатор выбирает путь с наивысшим рангом. Такой оптимизатор не принимает в расчет размер таблицы и распределение данных. Поэтому он всегда выберет использование индекса, даже когда полное сканирование может быть более эффективным.

Оптимизатор, работающий на основе стоимости, использует для определения «стоимости» пути статистическую информацию, такую, как количество элементов (cardinality) таблицы, число возможных значений в столбце и распределение данных. Затем он выбирает для обработки запроса путь с наименьшей стоимостью. Стоимость — это мера того, насколько велики затраты путей ввода-вывода, времени процессора и объемов памяти для обработки запроса. Чтобы эффективно использовать оптимизатор, работающий на основе стоимости, для каждой таблицы, индекса и представления нужно собрать статистику о количестве элементов и распределении данных. В главе 7 мы опишем использование пакета для сбора статистики DBMS_STATS.

Заметим, что работа над оптимизатором на основе правил в компании Oracle более не ведется. Многие новые свойства, такие как секционирование, индексы на основе битовых карт, оптимизация по типу звезды и переписывание запросов, применимы только при использовании оптимизатора на основе стоимости. Поэтому, чтобы воспользоваться всеми этими свойствами, вам следует запланировать использование в своем хранилище именно оптимизатора на основе стоимости.

Сейчас мы рассмотрим несколько оптимизаций обработки запросов, которые используются в оптимизаторе на основе стоимости. Но перед этим давайте коротко рассмотрим такую функцию, как EXPLAIN PLAN.

3.5.1. EXPLAIN PLAN

Oracle предоставляет несколько средств, позволяющих отобразить план обработки запроса. Вы можете для отображения плана использовать опцию autotrace утилиты SQL*Plus при посылке любого запроса. Кроме того, чтобы просто получить план обработки без самой обработки запроса вы можете использовать функцию EXPLAIN PLAN. Выходные данные этой функции помещаются в таблицу под названием PLAN_TABLE в вашей схеме. Таблицу PLAN_TABLE вы должны создать в своей схеме, используя скрипт $ORACLE_HOME/rdbms/admin/utlxplan.sql. Вы также можете указать функции EXPLAIN PLAN использовать какую-нибудь другую таблицу, но в ней должны содержаться те же столбцы, что и в PLAN_TABLE. Отобразить план можно с помощью скриптов $ORACLE_HOME /rdbms/admin/utlxpls.sql (для планов обычных запросов) и utlxplp.sql (для планов параллельных запросов).

Давайте рассмотрим выходные данные команды EXPLAIN PLAN. Эти данные состоят из пути доступа к каждой таблице, стоимости этого пути и числа извлеченных строк. Данные выведены с отступами, и соединенные таблицы показаны на одном уровне. Операции, выполненные раньше, располагаются по отступам глубже.

```
EXPLAIN PLAN FOR
SELECT t.month, t.year, p.product_id,
     SUM (purchase_price) as sum_of_sales,
     COUNT (purchase_price) as total_sales,
     COUNT(*) as cstar
FROM time t, product p, purchases f
WHERE, t.time_key = f.time_key AND
     f.product_id = p.product_id
GROUP BY t.month, t.year, p.product_id;

@utlxpls;

PLAN_TABLE_OUTPUT
```

```
-------------------------------------------------------------------------
Id|Operation              |Name       |Rows  | Bytes |Cost  |Pstart |Pstop
-------------------------------------------------------------------------
0 | SELECT STATEMENT      |           |  567 |23814  | 415  |       |
1 |  SORT GROUP BY        |           |  567 |23814  | 415  |       |
2 |   NESTED LOOPS        |           |70585 | 2895K |  43  |       |
*3|    HASH JOIN          |           |70585 | 2412K |  43  |       |
4 |     TABLE ACCESS FULL |TIME       |   15 |  225  |   2  |       |
5 |     PARTITION RANGE ALL|          |      |       |   1  |   7   |
6 |      TABLE ACCESS FULL| PURCHASES |94113 | 1838K |  40  |   1   |   7
*7|    INDEX RANGE SCAN   |PRODUCT_   |    1 |    7  |      |       |
  |                       | PK_INDEX  |      |       |      |       |
-------------------------------------------------------------------------

Predicate Information (identified by operationid):
---------------------------------------------------
   3 - access("T"."TIME_KEY"="F"."TIME_KEY")
   7 - access("F"."PRODUCT_ID"="P"."PRODUCT_ID")

Note: cpu costing is off
21 rows selected.
```

В этом примере самой «глубокой» операцией является хэш-соединение (hash join) между таблицами PURCHASES и TIME (строки 3–6). Результат этой операции соединяется с таблицей PURCHASES с помощью соединения вложенных контуров (nested loops join) (строки 2–7). Заметим, что для доступа к таблице PRODUCT используется индекс PRODUCT_PK_INDEX (строка 7). Последняя операция — сортировка с целью выполнения операции GROUP BY. В Oracle 9i Release 2 скрипт utlxpls.sql был усовершенствован и показывает применяемые предикаты для каждой операции. В приведенном примере применяемые предикаты — предикаты соединения. Условие PARTITION_RANGE (строка 5) показывает первый и последний просканированный раздел. Ключевое слово ALL означает, что отсечение разделов не проводится. Поэтому pstart =1 и pstop = 7.

Теперь рассмотрим различные методы, которые оптимизатор может применять для выдачи данных в ответ на запрос.

3.5.2. Оптимизация соединений

Одной из наиболее распространенных операций в запросах является соединение двух таблиц. Операция соединения комбинирует данные двух или более таблиц на основе условия (называющегося предикатом соединения), в котором упоминаются столбцы таблиц. Существует три главных метода соединения, используемые оптимизатором запросов Oracle:

- соединение Сортировка-Слияние;
- соединение вложенных контуров;
- хэш-соединение.

Соединение Сортировка-Слияние (Sort-Merge join)

При соединении Сортировка-Слияние (Sort-Merge join) данные всех таблиц сортируются по значению столбцов, входящих в условие соединения. Затем отсортированные таблицы сливаются, так что пары строк с соответствующими столбцами соединяются. Инициализационный параметр SORT_AREA_SIZE указывает объем доступной памяти для сортировки. Как только этот объем превышается, промежуточные данные записываются во временные сегменты во временное табличное пространство на диске. Если в любой из таблиц оказывается индекс для столбца соединения, оптимизатор может использовать его и извлекать данные в сортированном виде, избежав сортировки этой таблицы.

Соединение вложенных контуров (nested loops join)

При соединении вложенных контуров одна таблица выбирается как внешняя, а другая — как внутренняя. Для каждой строки внешней таблицы обнаруживаются все строки внутренней таблицы, удовлетворяющие условию соединения. Вложенные контуры могут быть чрезвычайно эффективны, если внутренняя таблица имеет индекс на столбец соединения, а во внешней таблице строк немного.

Хэш-соединение (hash join)

При хэш-соединении меньшая таблица загружается в память, к столбцам соединения применяется хэш-функция и строится хэш-таблица. После этого сканируется вторая таблица. Ее столбцы соединения хэшируются и сравниваются с хэш-таблицей в памяти на предмет совпадающих строк. Если памяти для хранения меньшей таблицы недостаточно, таблицы будут секционированы на меньшие разделы. Хэш-соединение можно использовать только для равнозначных соединений

(соединений типа таблица1.строка1 = таблица2.строка2). Параметр HASH_AREA_SIZE показывает объем памяти для построения хэш-таблицы, по исчерпанию которого придется писать на диск.

Хэш-соединения особенно полезны, когда для соединяемых столбцов нет индексов. Учитывая специализированность запросов, связанных с принятием решений, администратор базы не всегда может проиндексировать все столбцы, которые пользователи могут соединить. При использовании базы Oracle в качестве основы для преобразований данных данные могут перемещаться из одной временной таблицы в другую. Использование хэш-соединений может оказаться более эффективным, чем построение, с затратами времени, индексов на таблицы. Чтобы задействовать хэш-соединение, нужно установить инициализационный параметр HASH_JOIN_ENABLED в TRUE. Если вы используете средство Database Configuration Assistant для создания базы данных для хранилища, этот параметр будет установлен автоматически.

Теперь рассмотрим некоторые оптимизации соединений, которые чрезвычайно полезны для хранилища.

Оптимизация *соединения с учетом разделов (partition-wise join)*

Когда соединяемые таблицы секционированны, оптимизатор может осуществить соединение с учетом разделов. Вместо выполнения крупномасштабного соединения таблиц, производится его секционирование на несколько небольших операций соединения разделов и подразделов. Заметим, что при соединении с учетом разделов может использоваться любой из ранее описанных методов: Сортировка-Слияние, хэш-соединение или соединение вложенных контуров.

Полное соединение с учетом разделов можно выполнить, когда соединяемые таблицы имеют равнозначное секционирование по ключу соединения, указанному в запросе. Равнозначность секционирования означает наличие соответствия между разделами (или подразделами) одной таблицы и разделами (или подразделами) другой. Поэтому каждый раздел первой таблицы может быть соединен только с соответствующим ему разделом второй таблицы. При обработке параллельных запросов каждое соединение можно выполнять на отдельном процессоре.

Если только одна из таблиц секционирована по ключу соединения, при параллельной обработке запроса можно выполнить так называемое частичное соединение с учетом раздела. Таблица, не секционированая по ключу соединения, динамически секционируется в соответствии с критерием секционирования уже секционированой таблицы, и каждый сервер запросов соединяет пары раз-

делов из двух таблиц, как это делается при полном секционировании с учетом разделов.

Решение об использовании соединения с учетом разделов или любого из его вариантов принимается оптимизатором на основе стоимости из плана обработки запроса.

Заметим, что если запрос требует доступа лишь к некоторым из разделов любой таблицы, то только эти разделы будут задействованы в соединении с учетом разделов.

Преобразование по типу звезды (Star Transformation)

Звездообразный запрос (star query) — это типичный запрос, обрабатываемый в хранилище, построенном по звездообразной схеме. Каждая таблица измерений соединена с таблицей фактов с помощью отношения первичный ключ/внешний ключ. Если в таблице фактов присутствует индекс на основе битовых карт по столбцу внешних ключей, база Oracle может оптимизировать производительность звездообразных запросов с помощью алгоритма, известного под названием «преобразование по типу звезды» (star transformation). Такое преобразование основывается на комбинировании индексов на основе битовых карт для тех столбцов таблицы фактов, которые соответствуют таблицам измерений запроса. Сначала для извлечения необходимых строк из таблицы фактов используются индексы на основе битовых карт. Затем результат соединяется с таблицами измерений для получения тех столбцов, которые требуются для окончательного ответа на запрос. Вместо комбинации индексов на основе битовых карт может также применяться индекс на основе битовых карт с учетом разделов. Проведение преобразования по типу звезды также возможно, если лишь некоторые из столбцов внешних ключей имеют индексы на основе битовых карт.

Если ваше хранилище построено по схеме звезды, вам следует задействовать этот вид преобразования. Это делается путем установки инициализационного параметра STAR_TRANSFORMATION_ENABLED в TRUE. В качестве альтернативы вы можете использовать указание STAR_TRANSFORMATION в запросах, которые соединяют таблицу фактов и таблицы измерений. Заметим, что преобразование по типу звезды возможно только при работе с оптимизатором на основе стоимости. Оптимизатор будет оценивать планы обработки запросов с преобразованием и без него и выберет наиболее эффективный.

Предположим, что для каждого столбца внешнего ключа таблицы покупок (т. е. purchases.customer_id, purchases.time_key и purchases.product_id) существуют индексы на основе битовых карт. В приведенном примере показан запрос, пользующийся преимуществами преобразования по типу звезды. Этот запрос

Соединяет таблицу фактов (purchases) и таблицы измерений (pruduct, time, customer) в звездообразную схему и делает выбор из каждой таблицы измерений небольшой части данных (month = 1, county = 'Hants' и category = 'HDRW').

```
SELECT c.town, t.quarter, p.product_name,
       SUM(f.purchase_price) sales
FROM purchases f, time t, customer c, product p
       WHERE f.time_key = t.time_key and
       f.customer_id = c.customer_id and
       f.product_id = p.product_id and
       t.month = 1. and c.county = 'Hants' and
       p.category = 'HDRW'
GROUP BY c.town, t.quarter, p.product_name;
```

План обработки запроса показан ниже.

```
-------------------------------------------------------------------------
|ID | Operation                      | Name          |Rows |Bytes|Cost |
-------------------------------------------------------------------------
|  0|SELECT STATEMENT                |               |   54| 3834| 715 1
|  1| SORT GROUP BY                  |               |   54| 3834| 715 |
|* 2|  HASH JOIN                     |               |  328|23288| 712 |
|* 3|   TABLE ACCESS FULL            |PRODUCT        |   54| 11881   2 |
|* 4|    HASH JOIN                   |               |  984|48216| 709 |
|  5|     MERGE JOIN CARTESIAN       |               |    3|   78|   4 |
|* 6|      TABLE ACCESS FULL         |CUSTOMER       |    1|   12|   2 |
|  7|       BUFFER SORT              |               |    1|  141|   2 |
|* 8|        TABLE ACCESS FULL -     |TIME           |    1|   14|   2 |
|  9|     TABLE ACCESS BY INDEX ROWID |PURCHASES     | 7030| 157KI | 703 |
| 10|      BITMAP CONVERSION TO ROWIDS|              |     1|     |     |
|  HI       BITMAP AND               |               |     |     |     |
| 12|         BITMAP MERGE           |               |     |     |     |
| 13|          BITMAP KEY ITERATION  |               |     |     |     |
|*14|           TABLE ACCESS FULL    |CUSTOMER       |    1|   12|   2 |
|*15|           BITMAP INDEX RANGE SCAN|CUST_IDX     |     |     |     |
| 16|         BITMAP MERGE           |               |     |     |     |
| 17|          BITMAP KEY ITERATION  |               |     |     |     |
|*18|           TABLE ACCESS FULL    |ITIME          |   21|   28|   2 |
|*19|           BITMAP INDEX RANGE SCAN|TIME_IDX     |     |     |     |
```

```
| 20|          BITMAP MERGE                 I              |     |      |      |
|  21|           BITMAP KEY ITERATION       '  |           |     |      |      |
|*22|             TABLE ACCESS FULL        I PRODUCT      |   54|  1188|    2 |
|*23 |            BITMAP INDEX RANGE SCAN|PROD_IDX          |     |      |      |
-------------------------------------------------------------------------------
```

Давайте попробуем понять, как работает преобразование по типу звезды. Заметьте, что строки, соответствующие category = 'HDRW', можно извлечь с помощью подзапроса.

```
SELECT *
FROM purchases
WHERE product_id IN  (SELECT product_id
      FROM product
      WHERE category =  'HDRW');
```

Такой запрос можно обработать с помощью индекса на основе битовых карт prod_idx, если извлечь и слить все битовые карты, соответствующие всем идентификаторам продуктов категории HDRW. Похожие подзапросы создаются для всех остальных таблиц измерений, и, наконец, получившиеся битовые карты объединяются с помощью побитовой операции AND в одну карту. В плане обработки запроса это показано в строках с 10 по 23. Эта битовая карта используется для извлечения строк из таблицы покупок. И, наконец, производится соединение с таблицами измерений (показано в строках 0–9 плана) для получения значений прочих столбцов (town, quarter и product_name). Такой подход эффективен, поскольку в соединении теперь участвует только небольшая часть таблицы покупок.

Совет. Создание одностолбцового индекса на основе битовых карт по каждому столбцу внешних ключей в таблице фактов позволит оптимизатору рассмотреть возможность использования преобразования по типу звезды для звездообразных запросов.

3.6. Автоматическое управление памятью

Запросы, которые используются в приложениях, связанных с бизнес-информацией, как правило, требуют больших объемов памяти для сортировки, хэш-

соединений, операций с картами и т. п. Существуют различные инициализационные параметры, такие как SORT_AREA_SIZE, HASH_AREA_SIZE, CREATE_BITMAP_AREA_SIZE и BITMAP_MERGE_AREA_SIZE, которые можно настраивать для получения наилучшего быстродействия. Оптимальные значения этих параметров меняются от запроса к запросу и могут зависеть от нагрузки на систему. Точная настройка этих параметров может отнять у администратора базы данных очень много времени.

В Oracle 9i, для облегчения этой работы, предлагается возможность автоматического управления памятью. При использовании этой функции администратору нужно только указать общий объем доступной памяти, например, установив параметр PGA_AGGREGATE_TARGET. Оптимизатор определит наилучший путь использования памяти для различных операций SQL для всех серверных процессов базы, запущенных в данный момент. Чтобы включить автоматическое управление памятью, нужно установить параметр WORKAREA_SIZE_POLICY в AUTO (если установить для него значение MANUAL, будет Вновь включено управление памятью вручную).

Объем памяти, выделенной и используемой каждым процессом, можно увидеть в представлении V$PROCESS.

```
select spiel, pga_used_mem, pga_alloc_mem, pga_max_mem
from v$process;
```

SPID	PGA_USED_MEM	PGA_ALLOC_MEM	PGA_MAX_MEM
340	132552	198880	198880
341	135056	3371624	3371624
343	4349880	7570468	7570468

Размеры рабочей области, используемые операторами SQL, можно увидеть с помощью представления (view) под названием V$SQL_WORKAREA. Вы можете соединиться с V$SQL и получить sql_text для используемых операторов. (Мы отредактировали приводимые выходные данные и приводим sql_text лишь частично из-за недостатка места).

```
SELECT sql_text, operation_type,
     estimated_optimal_size estsize, last_memory_used
FROM v$sql_workarea w, v$sql s
WHERE w.address = s.address
AND parsing_schema_id = USERENV('SCHEMAID') ;
```

```
SQL_TEXT              OPERATION_TYPE       ESTSIZE   LAST_MEMORY_USED
--------------        --------------       -------   ----------------
SELECT t.month,       GROUP BY (SORT)        56320              49152
SELECT t.month,       HASH-JOIN •           874496              62,8736
...
```

Размеры используемой при обработке запроса рабочей области можно отслеживать с помощью представления V$SQL_WORKAREA_ACTIVE.

```
-- показывает текущее использование рабочей области во время обработки запроса
SELECT operation_type, work_area_size, expected_size,
    actual_mem_used
FROM v$sql_workarea_active;

OPERATION_TYPE   WORK_AREA_SIZE   EXPECTED_SIZE   ACTUAL_MEM_USED
--------------   --------------   -------------   ---------------
HASH-JOIN              1087488         1086464            628736
```

3.7. Функции SQL

В этом разделе мы рассмотрим различные SQL-функции в Oracle.

3.7.1. Функции для агрегирования

Приложения, использующие OLAP, часто используют вычисление итоговых значений — агрегатов (aggregate) — по данным из нескольких измерений. Для облегчения этих операций Oracle предлагает несколько возможностей. Помимо традиционного условия SQL GROUP BY, Oracle предлагает функции CUBE, ROLLUP и GROUPING SETS.

Операторы CUBE и ROLLUP появились, начиная с версии Oracle 8i, а оператор GROUPING SETS предложен в Oracle 9i. Оператор ROLLUP используется для подсчета промежуточных результатов (subtotals) в измерении, а оператор CUBE рассчитывает все возможные результаты для набора столбцов в условии GROUP BY. Оператор GROUPING SETS позволяет пользователю рассчитать

множественную независимую группировку для одного запроса. В прошлом такие операции выполнялись с помощью средств создания отчетов. При выполнении их внутри базы данных появилась возможность выполнять их параллельно и пользоваться различными ранее описанными преимуществами и оптимизацией запросов.

Операторы CUBE, ROLLUP и GROUPING SETS можно использовать со всеми поддерживаемыми Oracle типами агрегатов, такими, как COUNT, AVG, MIN, MAX, STDDEV и VARIANCE, а также с аналитическими функциями, описанными в разделе 3.7.2.

Теперь мы подробно рассмотрим эти операторы.

CUBE

Оператор CUBE проводит вычисления агрегатов для всех возможных комбинаций столбцов в условии GROUP BY. В приведенном примере мы подсчитываем общий объем продаж продуктов по категориям продуктов и дате покупки.

```
SELECT p.category, t.time_key, SUM(f.purchase_price)
FROM product p, purchases f, time t
WHERE p.product_id = f.product_id AND
   t.time_key = f.time_key
GROUP BY CUBE (p.category, t.time_key);
```

Ответ на данный запрос (показанный ниже) включает результат операции SUM(f.purchase_price) сгруппированный по (category, time_key), по (category), по (time_key) и итоговый результат. Стрелками мы указали первую строку каждой группы.

```
CATEGORY   TIME_KEY          SALES
---------- ---------------   ----------
ELEC       01-JAN-02         1574137.45 <- (category, time_key)
ELEC       02-JAN-02           56.02
ELEC       01-FEB-02           56.02
ELEC       02-FEB-02         1574137.45
HDRW       01-JAN-02           16704.22
HDRW       02-FEB-02           16704.22
MUSC       01-JAN-02           17989.1
MUSC       02-FEB-02           17989.16
ELEC                         3148386.94 <- (category
```

```
HDRW                          33408.44
MUSC                          35978.32
01-JAN-02                   1608830.83 <-  (time_key)
02-JAN-02                        56.02
01-FEB-02                        56.02
02-FEB-02                   1608830.83
                            3217773.7  <- итог
```

ROLLUP

Функция ROLLUP создает промежуточные результаты (subtotals), двигаясь по данным от низкого уровня к высокому. При этом также подсчитывается общий итоговый результат (grand total). Оператор ROLLUP полезен при обсчете данных по иерархическому измерению, например времени.

В следующем примере операция ROLLUP подсчитывает общий объем продаж по группам (category, time_key) и (category) и общий итоговый результат. Стрелками мы выделили все группы.

```
SELECT p.category, t.time_key, SUM(f.purchase_price)
FROM product p, purchases f, time t
WHERE p.product_id = f.product_id AND
      t.time_key = f.time_key
GROUP BY ROLLUP (p.category, t.time_key);
```

```
CATEGORY             TIME_KEY             SALES
----------           ---------------      ----------
ELEC                 01-JAN-02             1574137.45 <- (category, time_key)

ELEC                 02-JAN-02                56.02
ELEC                 01-FEB-02                56.02
ELEC                 02-FEB-02             1574137.45
HDRW                 Ol-JAN-02               16704.22
HDRW                 02-FEB-02               16704.22
MUSC                 Ol-JAN-02               17989.16
MUSC                 02-FEB-02               17989.16
HDRW                                         33408.44 <- (category)
ELEC                                       3148386.94
MUSC                                         35978.32
                                           3217773.7  <- итог
```

При операции ROLLUP группы определяются путем повторных расчетов агрегатов по крайнему правому столбцу предыдущей группы. В предыдущем примере мы сначала проводим группировку по (category, time_key), а потом по (category). Чтобы правильно перемещаться по иерархии, мы должны упорядочить столбцы от высшего к низшему уровню слева направо. Например, если двигаться по временной иерархии, то порядок столбцов будет (год, месяц, день).

Если мы сравним выходные данные инструкций CUBE и ROLLUP, мы заметим, что ROLLUP подсчитывает лишь некоторые из возможных комбинаций группировки, имеющихся в CUBE. В предыдущем примере группировка (time_key) присутствует в выходе CUBE, но не ROLLUP. Выходные данные CUBE всегда включают в себя выходные данные ROLLUP.

С увеличением числа столбцов оператор CUBE может потреблять массу времени и дискового пространства. Число группировок для двух столбцов в операторе CUBE равно четырем, для трех — восемь и т. д. Оператор ROLLUP намного проще и, следовательно, эффективнее: для двух столбцов ROLLUP создает три группы, для трех — четыре и т. д.

GROUPING SETS

Для хранения рассчитанных агрегатов часто используются суммарные таблицы и представления (materialized views). Если бы нам понадобилось сохранить результат выполнения оператора CUBE, произведя подсчет по всем возможным комбинациям группировки столбцов, потребовалось бы слишком много дискового пространства. Не очень-то необычной является ситуация, когда объем выходных данных CUBE в несколько раз превосходит размер самих таблиц! Эта проблема решается путем использования оператора GROUPING SETS, который дает возможность избирательно подсчитать только интересующие комбинации группировок, не выводя всю информацию, как CUBE.

В следующем примере мы подсчитываем объем продаж по группам (category, time_key), (category, state_id), (time_key, region) и общий объем продаж, обозначенный как (). Мы решили не просчитывать другие комбинации, такие, как продажи для каждой категории, для каждого момента времени и штата.

```
SELECT p.category as cat, t.time_key, g.region, g.state_id as st,
     SUM(f.purchase_price) sales
FROM product p, purchases f, time t, geography g
WHERE p.product_id = f.product_id AND
     t.time_key = f.time_key AND
     g.state_id = f.state_id
```

```
GROUP BY GROUPING SETS ((p.category, t.time_key),
   , (p.category, g.state_id), (t.time_key, g.region), ());

CATEGORY   TIME_KEY    REGION    ST      SALES
---------- ----------- --------- ----- ---------
ELEC       01-JAN-02                  1574137.45 <- (category, time_key)
ELEC       02-JAN-02                       56.02
ELEC       01-FEB-02                       56.02
ELEC       02-FEB-02                  1574137.45
HDRW       01-JAN-02                    16704.22
HDRW       02-FEB-02                    16704.22
MUSC       01-JAN-02                    17989.16
MUSC       02-FEB-02                    17989.16

ELEC                                CA  1574193.47 <- (category, state_id)
ELEC                                MA  1484355.05
ELEC                                NH    89838.42
HDRW                                CA    13115.79
HDRW                                NH    16704.22
HDRW                                WI     3588.43
MUSC                                CA      2977.3
MUSC                                NH     7192.53
MUSC                                OH     7819.33
MUSC                                WI     17989.16

           01-JAN-02  West             2977.3  <- (time_key, region)
           01-JAN-02  MidWest          7819.33
           01-JAN-02  NorthEast     1598034.2
           02-JAN-02  NorthEast          56.02
           01-FEB-02  West               56.02
           02-FEB-02  West          1587253.24
           02-FEB-02  MidWest         21577.59
                                    3217773.7  <- итог
```

Заметьте, что ROLLUP является частным случаем GROUPING SET. ROLLUP (category, time_key) эквивалентно GROUPING SETS ((category, time_key), (category), ()).

В запросе можно указать несколько операторов GROUPING SETS. Это позволяет точно указать область пересечения группировок многих измерений.

Предположим, вам нужно подсчитать объемы продаж по категориям продуктов для географического измерения — столбцов (state, city) и для временного измерения (year, quarter). Вместо того чтобы указывать все комбинации этих четырех столбцов, мы можем просто использовать следующий запрос (это называется составной оператор GROUPING SETS).

```
SELECT p.category as cat, t.quarter as quart, t.year,
     g.state_id as st, g.region, SUM(f.purchase_price) sales
FROM purchases f, time t, geography g, product .p
WHERE p.product_id = f.product_id AND
     t.time_key = f.time_key AND
     g.state_id = f.state_id
GROUP BY p.category,
       GROUPING SETS (g.state_id, g.region),
       GROUPING SETS (t.quarter, t.year);
```

```
CAT    QUART   YEAR   ST   REGION     SALES
----   -------  -----  --   ---------  ----------
ELEC      1           CA              1574193.47 <- (quarter, state)
ELEC      1           MA              1484355.05
ELEC      1           NH                89838.42
...
MUSC      1           WI                17989.16

ELEC      1                 West        1574193.47 <- (quarter, region)
ELEC      1.                NorthEast   1574193.47
...
MUSC      1                 NorthEast      7192.53

ELEC            2002    CA              1574193.47 <-(year, state)
ELEC            2002    MA              1484355.05
ELEC            2002    NH                89838.42
...
MUSC            2002    WI                17989.16

ELEC            2002          West      1574193.47 <-(year, region)
ELEC            2002          NorthEast 1574193.47
...
MUSC            2002          NorthEast    7192.53
```

Столбец p.category, который находится за пределами оператора GROUPING SETS, входит во все группы. Таким образом, приведенный запрос производит расчет по четырем группировкам — (category, quarter, state), (category, quarter, region), (category, year, state) и (category, year, region).

Функции GROUPING и GROUPING_ID

Операторы CUBE, ROLLUP и GROUPING SETS проводят расчеты по нескольким уровням агрегации в одном запросе. Вашему приложению может понадобиться выявить ту часть выходных данных, которая соответствует определенному уровню агрегации. Функции SQL GROUPING() и GROUPING_ID() предоставляют механизм, позволяющий определить, какие строки ответа на запрос какому уровню агрегации соответствуют.

Приводимый ниже запрос иллюстрирует поведение функции GROUPING().

```
SELECT p.category as cat, t.time_key,
SUM(f.purchase_price) sales,
     GROUPING(p.category) grp_c, GROUPING(t.time_key) grp_t
     FROM product p, purchases f, time t
WHERE p.product_id = f.product_id AND
    t.time_key = f.time_key
GROUP BY ROLLUP (p.category, t.time_key);
```

```
CAT     TIME_KEY        SALES    GRP_C    GRP_T
----    ---------   ----------   -----    -----
ELEC    01-JAN-02   1574137.45       0        0   <-(category, time_key)
ELEC    02-JAN-02        56.02       0        0
ELEC    01-FEB-02        56.02       0        0
ELEC    02-FEB-02   1574137.45       0        0
HDRW    Ol-JAN-02     16704.22       0        0
HDRW    02-FEB-02     16704.22       0        0
MUSC    Ol-JAN-02     17989.16       0        0
MUSC    02-FEB-02     17989.16       0        0
HDRW                   4455.58       0        0 *
ELEC                3148386.94       0        1   <-(category)
HDRW                  37864.02       0        1 *
MUSC                   5978.32       0        1
                    3222229.28       1        1   <-итог
```

Для каждой группы оператор GROUPING(category) возвращает нуль, если в группе есть столбец category, в противном случае возвращается единица. Точно так же оператор GROUPING(time_key) возвращает нуль, если в группе есть столбец time_key, в противном случае возвращается единица. Таким образом, по результату функции GROUPING() можно определить все уровни агрегации. Для группы (category, time_key) значения функции GROUPING() будут (0,0), для группы (category) эти значения составят (0,1). Заметим, что общий итог легко определяется как строка, в которой во всех столбцах функции будет значение 1.

В выходных данных операторов CUBE и ROLLUP строки, соответствующие более высокому уровню агрегации, имеют значение NULL в тех столбцах, которые прошли агрегацию. Еще одним применением функции GROUPING() является различение таких значений NULL от реальных, которые содержатся в самих данных. Для примера, внимательно посмотрите в предыдущем примере на строки, помеченные звездочкой. Обе эти строки имеют в столбце time_key значение NULL. Первый соответствует строкам таблицы покупок, относящимся к категории HDRW, в которых ключ времени недоступен (NULL). Во втором случае мы проагрегировали значения time_key, и следовательно, эта строка соответствует агрегации на уровне категории. Функция GROUPING() позволяет различить эти две похожих строки. В первом случае операция GROUPING(time_key) дает значение 0, а во втором — единицу.

Вместо использования для каждого столбца функции GROUPING(), вы можете применить ко всем ним функцию GROUPING_ID() следующим образом:

```
SELECT p.category, t.time_key, SUM(f.purchase_price) ,
     GROUPING_ID(p.category,t.time_key) gid
FROM product p, purchases f, time t
WHERE p.product_id = f.product_id AND
   t.time_key = f.time_key
GROUP BY ROLLUP (p.category, t.time_key) ;

CATEGORY   TIME_KEY          SALES      GID
--------   ---------      ----------   -----
ELEC       01-JAN-02       1574137.45       0 <-(category, time_key)
ELEC       02-JAN-02            56.02       0
ELEC       01-FEB-02            56.02       0
ELEC       02-FEB-02       1574137.45       0
HDRW       01-JAN-02         16704.22       0
HDRW       02-FEB-02         16704.22       0
MUSC       01-JAN-02         17989.16       0
```

```
MUSC        02-FEB-02        17989.16        0
HDRW                          4455.58        0
ELEC                       3148386.94        1 <-(category)
HDRW                         37864.02        1
MUSC                         35978.32        1
                           3222229.28        3 <- итог
```

Если представить все отдельные функции GROUPING() вместе в виде двоичного числа, то функция GROUPING_ID() возвращает соответствующее десятичное число. Так, если в предыдущем примере GROUPING(p.category) = 0, а GROUPING(r.time_key) = 1, то GROUPING_ID(p.category, t.time_key) будет двоичным числом 01, что соответствует десятичному числу 1. Для строки итога GROUPING(p.category) = 1 и GROUPING(r.time_key) = 1, и следовательно, GROUPING_ID(p.category, t.time_key) равно двоичному числу 11, что соответствует десятичному 3. Функция GROUPING_ID(), таким образом, является гораздо более компактным представлением, чем отдельные функции, но интерпретация отдельных функций для столбцов не столь очевидна.

3.7.2. Функции SQL для аналитических вычислений

Приложения, связанные с принятием решений, часто имеют дело с вопросами типа «Каковы были десять наиболее часто продаваемых продуктов в 2001 году?», или «Каковы объемы продаж в январе этого года по сравнению с прошлым?», или «Каковы совокупные объемы продаж по каждому месяцу этого года?».

Хотя эти вопросы кажутся очень простыми, соответствующие традиционные запросы SQL могут оказаться чрезвычайно сложными. Такие запросы очень трудно оптимизировать, и они могут потребовать многократного сканирования данных. Производительность их обработки может быть очень низкой или же потребовать обработки, относящейся к уровню приложений. Потому многие производители баз данных выдвинули инициативу введения в SQL расширений для точного представления запросов такого типа. Сейчас эти расширения являются частью стандарта SQL99. При использовании аналитических функций вычисления такого рода могут использовать преимущества параллелизма и прочих методов оптимизации, имеющихся в базе данных.

Аналитические функции в Oracle обеспечивают множество полезных возможностей, к которым относятся следующие.

- Функции ранжирования (ranking functions) можно использовать при ответе на запросы о наиболее часто продаваемых товарах. Также их можно применять для систематизации данных по сегментам (buckets). Примерами ранжирующих функций являются RANK(), DENSE_RANK() и NTILE(). Вы также можете определить ранг величины, как если бы она гипотетически была вставлена в данные.
- Функции сдвигающейся рамки (moving window) можно использовать для подсчета таких величин, как совокупная сумма или скользящее среднее. Вычисления по типу сдвигающейся рамки полезны при работе с данными за определенный период времени. Например, если вы хотите знать средний объем продаж продукта за последние семь дней, шесть месяцев, пять лет и т. п.
- Отчетные агрегаты (reporting aggregates) можно использовать для того, чтобы увидеть суммарное значение рядом с теми строками, которые его определяют. Например, если вы хотите сравнить объемы продаж каждого продукта со средним объемом продаж всех продуктов.
- Функции упреждения (lead functions) и запаздывания (lag functions) можно применять для сравнения по периодам, например, сравнить продажи текущего года с предыдущим.
- Статистические функции, такие, как линейная регрессия и расширенные функции суммации, такие, как ковариантность, можно использовать для анализа тенденций.
- Теперь мы рассмотрим эти функции подробно, с примерами.

Ранжирующие функции

Ранжирующие функции выставляют строкам в наборе данных ранг в соответствии с определенным критерием. Следующий пример кода SQL отвечает на такой запрос: «Каковы десять наиболее продаваемых продуктов?». В данном случае критерием являются общие объемы продаж продукта, указанные в условии ORDER BY.

```
SELECT *
FROM
 (SELECT p.product_id, SUM(f.purchase_price) as sales,
      RANK() over (ORDER BY SUM(f.purchase_price)) as rank
  FROM purchases f, product p
  WHERE f.product_id = p.product_id
  GROUP BY p.product_id)
WHERE rank <= 10;
```

```
PRODUCT_ID      SALES          RANK
----------      ---------      ----
SP1222          38832.31          1
SP1244          39145.54          2
SP1256          39354.24          3
SP1263          39458.78          4
SP1236          39699.14          5
SP1220          39893.6           6
SP1231          40080.61          7
SP1242          40208.33          8
SP1248          40312.98          9
SP1254          40418.02         10
```

Этот запрос состоит из двух частей: внутреннего запроса, который подсчитывает функцию RANK(), и внешнего запроса, который выбирает первые десять рангов. Внутренний запрос подсчитывает суммирующую функцию SUM(f.purchase_price) по каждому идентификатору продукта. Затем он располагает результат в соответствии с объемами продаж и применяет функцию RANK(). Эта функция присваивает ранги от 1 до N, пропуская один ранг в случае «узла». Поэтому если два продукта имеют одинаковые объемы продаж и ранг 3, то следующим рангом будет 5. Вариант функции RANK(), функция DENSE_RANK() присваивает смежные ранги без учета узлов. Например, если два продукта имеют один ранг 3, функция DENSE_RANK() следующему продукту присвоит ранг 4.

Далее, в условии ORDER BY определяется порядок по возрастанию или по убыванию и то, следует ли считать NULL первым или последним в этом ряду. По умолчанию установлен порядок по возрастанию. NULL LAST задается как значение по умолчанию для порядка по возрастанию, а NULL FIRST — по убыванию. В приведенном ниже запросе в условии ORDER BY задан порядок по убыванию, а NULL считается последним в этом ряду.

```
SELECT p.product_id, SUM(f.purchase_price) as sales,
     RANK() over (ORDER BY SUM(f.purchase_price) DESC NULLS LAST)
     as rank
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.product_id;
```

```
PRODUCT_   SALES          RANK
--------   ----------  ------
SP1046     1115634.20       1
SP1056     1109091.08       2
SP1036     1107147.52       3
SP1060     1095253.99       4
SP1042     1002747.91       5
SP1051      979279.86       6
SP1041      941985.49       7
SP1054      919424.74       8
SP1048      919051.12       9
SP1052      853124.58      10
PRODUCT_         SALES    RANK
--------   ----------  ------
SP1222        81385.09     161
SP1039                     162  <- NULL в конце
```

Важно отметить, что аналитические функции применяются после условий WHERE, GROUP BY и HAVING обрабатываемого запроса, поэтому в качестве критериев расстановки в ранжирующих функциях можно использовать суммирующие функции, такие, как SUM(f.purchase_price).

Условие PARTITION BY

Полезное свойство аналитических функций — то, что данные можно разделить на группы, а затем применить функцию к каждой группе. Группа указывается с помощью условия PARTITION BY. Например, если вы хотите ранжировать продукты в каждой из категорий продуктов, можно использовать следующий запрос. (За недостатком места показано только пять рангов для каждой категории).

```
SELECT p.category, p.product_id, SUM(f.purchase_price) as sales,
     RANK() over (PARTITION BY p.category
          ORDER BY SUM(f.purchase_price)) as rank
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category, p.product_id;
```

```
CATEGORY    PRODUCT_ID SALES       RANK
----------  ---------- --------    -----
ELEC        SP1080     40495.3       1
ELEC        SP1073     41440.34      2
ELEC        SP1086     41599.05      3
ELEC        SP1065     41939.38      4
ELEC        SP1084     42253.72      5
...
HDRW        SP1222     38832.31      1
HDRW        SP1236     39699.14      2
CATEGORY    PRODUCT_ID SALES       RANK
----------  ---------- --------    -----
HDRW        SP1220     39893.6       3
HDRW        SP1231     40080.61      4
HDRW        SP1214     40930.96      5
...
```

На рис. 3.9 показано, как условие PARTITION BY влияет на вычисление, проводимое функцией RANK(). Как и ранее, сначала при обработке запроса просчитываются объемы продаж по каждому идентификатору продукта и категории. Затем условие PARTITION BY секционирует полученный результат по категории. Функция RANK() для каждой категории присваивает идентификаторам продуктов ранг в соответствии с их объемами продаж.

Вы можете в одном и том же запросе выполнять ранжирование, используя разные операторы PARTITION и ORDER BY. Условие PARTITION BY в аналитических функциях совершенно не связано с секционированием данных в базах Oracle, которое обсуждалось в разделе 3.3.

Группирующие функции

Аналитическим приложениям часто требуется возможность систематизировать данные по группам или определить, как данный продукт соотносится со всем диапазоном продуктов. Эту возможность предоставляют ранжирующие функции CUME_DIST(), PERCENT_RANK() и NTILE().

Функция CUME_DIST() рассчитывает положение значения относительно набора значений, возвращая число в диапазоне от 0 до 1. Функция PERCENT_RANK() возвращает ранг строки относительно всех строк в наборе данных. Возвращаемые значения и той, и другой функции лежат в диапазоне от 0 до 1.

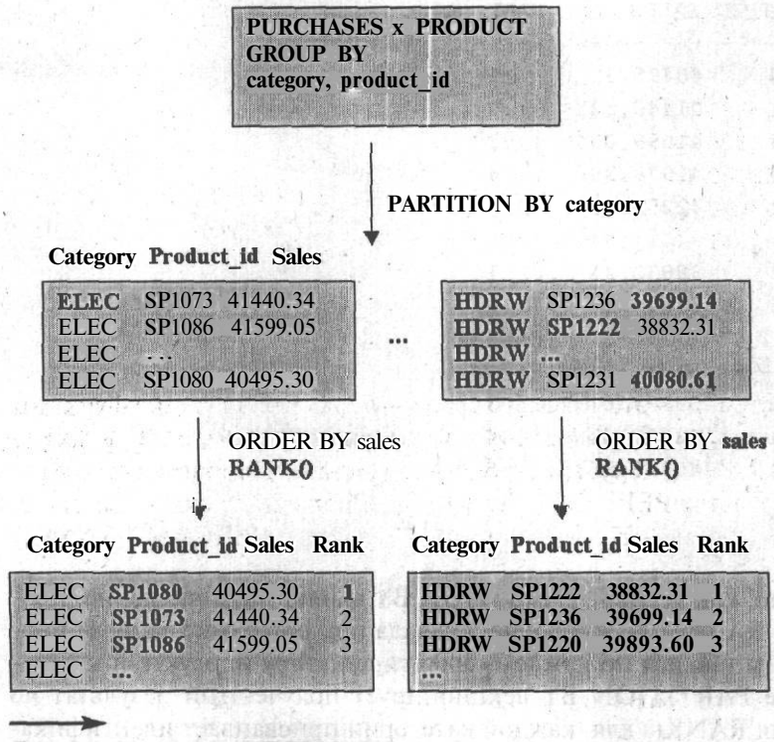


Рис. 3.9. *Условие PARTITION BY*

В следующем примере показано различие между функциями CUME_DIST() и PERCENT_RANK(). (Показаны только строки, относящиеся к категории HDRW).

```
SELECT p.category, p.product_id, SUM(f.purchase_price) as sales,
     CUME_DIST() over (PARTITION BY p.category
        ORDER BY SUM(f.purchase_price)) as cume_dist
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category, p.product_id;
HAVING SUM(f.purchase_price) < 100000 ;
```

```
CATE PRODUCT_       SALES    CUME_DIST
---- --------  ----------  ----------
...
HDRW SP1222       81385.09   .142857143
HDRW SP1236       82520.97   .285714286
HDRW SP1220       82622      .428571429
HDRW SP1231       83073.57   .571428571
HDRW SP1214       83841.66   .714285714
HDRW SP1212       99663.51   .857142857
HDRW SP1232       99863.42           1
...
```

Положение первой строки — это 1, а всего строк — 7, так что значение функции CUME_DIST() равно 1/7 или 0,142857143 и т. д. Функция PERCENT_RANK() рассчитывается как RANK() текущей строки −1, деленное на общее число рангов −1. Поэтому результат PERCENT_RANK() для первой строки — всегда 0. RANK() для второй строки — 2, а всего строк — 7, так что PERCENT_RANK() = 1/6 или 0,166666667.

```
SELECT p.category, p.product_id, SUM(f.purchase_price) as sales,
     PERCENT_RANK() over (PARTITION BY p.category
        ORDER BY SUM(f.purchase_price))
as pct_rank
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category, p.product_id
HAVING SUM(f.purchase_price) < 100000;
```

```
CATE PRODUCT_       SALES    CUME_DIST
---- --------  ----------  ----------
...
HDRW SP1222       81385 .09          0
HDRW SP1236       82520 .97  .166666667
HDRW SP1220       82622      .333333333
HDRW SP1231       83073 .57  .5
HDRW SP1214       83841 .66  .666666667
HDRW SP1212       99663 .51  .833333333
HDRW SP1232       99863 .42          1
...
```

Функция NTILE() полезна для проведения статистических измерений, таких, как квартиль и процентиль. Функция NTILE(N) располагает данные в соответствии с определенным критерием, а затем секционирует результат на N сегментов (buckets), присваивая номер сегмента всем строкам сегмента. Если N=100, то это называется процентилем, а при N=4 — квартилем.

В следующем запросе подсчитывается NTILE(4) для каждой категории продуктов, с использованием объемов продаж для каждого продукта категории. Функция присваивает каждой строке номер от единицы до четырех.

```
SELECT, p.category, p.product_id, SUM(f.purchase_price) as sales,
 NTILE(4) over (PARTITION BY p.category
        ORDER BY SUM(f.purchase_price)) as quartile
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category, p.product_id;

CATE PRODUCT_      SALES    CUME_DIST
---- --------  ----------  ----------
...
HDRW SP1222       81385.09           1
HDRW SP1236       82520.97           1
HDRW SP1220       82622              2
HDRW SP1231       83073.57           2
HDRW SP1214       83841.66           3
HDRW SP1212       99663.51           3
HDRW SP1232       99863.42           4
...
```

Все сегменты, сгенерированные функцией NTILE(4), имеют приблизительно равное число строк, однако диапазон между наибольшим и наименьшим значением в сегментах может различаться. Это называется «гистограмма при равной высоте» (equi-height histogramm). С другой стороны, можно с помощью функции WIDTH_BUCKET() построить «гистограммы при равной ширине» (equi-width histogramm), когда данные в сегментах распределены таким образом, что в каждом сегменте содержится примерно равный диапазон значений, но число строк в сегменте может различаться.

```
SELECT p.product_id, SUM(f.purchase_price) as sales,
     WIDTH_BUCKET(SUM(f.purchase_price), 80000, 120000, 4)
       as width_bucket
FROM purchases f, product p
WHERE f.product_id = p.product_id and p.category = 'HDRW'
GROUP BY p.product_id
ORDER BY sales;
```

PRODUCT_	SALES	WIDTH_BUCKET
SP1222	81385.09	1
SP1236	82520.97	1
SP1220	82622.00	1
SP1231	83073.57	1
SP1214	83841.66	1
SP1212	99663.51	2
SP1232	99863.42	2
SP1224	100825.37	3
SP1234	109433.07	3
SP1219	111809.95	4
SP1239	112749.64	4
SP1213	112749.90	4
...		
SP1233	148895.43	5
SP1221	167743.34	5

Входные данные функции WIDTH_BUCKET — конечные точки рассматриваемых диапазонов и количество промежуточных точек. Затем функция секционирует диапазон объемов продаж на равные по ширине (equi-width) сегменты, равномерно размещая точки между конечными точками, и присваивает каждой строке сегмент в соответствии с тем, в какой диапазон она попадает. В наших примерах диапазон от 80000 до 120000 разделяется на пять приблизительно равных диапазонов (или сегментов). Можно видеть, что в каждый сегмент попадает разное число строк.

Рамочные суммирующие функции

Рамочные суммирующие функции позволяют выполнить такую функцию, как SUM над конкретной «рамкой» строк от некоторого текущего. Эти функции

позволяют выполнять вычисления по типу «сдвигающейся рамки», такие, как совокупная сумма или скользящее среднее.

Следующее выражение SQL отвечает на такой запрос: «Каково на данный момент совокупное значение объемов продаж по каждому месяцу в текущем году?». Сначала для каждого месяца вычисляется SUM(f.purchase_price). Выражение SUM(SUM(f.purchase_price)) показывает, что мы проводим дальнейшее суммирование объемов продаж, в данном случае для всех месяцев, предшествующих текущему. Выражение ROWS UNBOUNDED PRECEDING показывает, что в рамку входят все предыдущие строки, включая текущий. Рамка, определенная с помощью указания строк, предшествующих или следующих за текущим, называется физической рамкой (phisical window).

```
SELECT t.month, SUM(f.purchase_price) as sales,
       SUM(SUM(f.purchase_price))
       OVER (ORDER BY t.month ROWS UNBOUNDED PRECEDING)
         as cumulative_sales
FROM purchases f, time t
WHERE f.time_key = t.time_key
GROUP BY t.month;
MONTH        SALES    CUMULATIVE_SALES
------  ----------    ----------------
1        8989096.34         8989096.34
2       17213413.50        26202509.90
3         429035.78        26631545.70
4        4520301.94        31151847.60
5        1268317.48        32420165.10
6        7092585.46        39512750.50
...
```

В следующем примере мы вычисляем по каждому месяцу скользящее среднее объемов продаж данного месяца и двух предыдущих. Это задается условием ROWS 2 PRECEDING.

```
SELECT t.month, SUM(f.purchase_price) as sales,
     AVG(SUM(f.purchase_price))
   . OVER (ORDER BY t.month ROWS 2 PRECEDING) as moving_average
FROM purchases f, time t
WHERE f.time_key = t.time_key
GROUP BY t.month;
```

```
MONTH       SALES          MOVING_AVERAGE
------  ----------         --------------
1        8989096.34          8989096.34
2       17213413.5          13101254.90
3         429035.78          8877181.89
4        4520301.94          7387583.75
5        1268317.48          2072551.73
6        7092585.46          4293734.96
...
```

Если бы мы захотели, чтобы рамка включала в себя текущий месяц и два последующих, соответствующее выражение нужно было бы просто изменить на ROWS 2 FOLLOWING. Физическая рамка хороша только в тех случаях, когда данные имеют высокую плотность. Если, например, в ваших датах есть пробелы, использование физической рамки не будет иметь особого смысла.

Вместо указания смещения в строках можно создать логическую рамку (logical window), задав интервал относительно текущего значения. Например, если выражение имеет тип DATE, можно указать логическую рамку с помощью константы или таких выражений, как INTERVAL DAY, MONTH или YEAR. Логические рамки допустимы только для данных числового типа, единиц времени или интервалов. В приводимом ниже примере показывается вычисление объемов продаж за пятидневный интервал (включая два дня до и два дня после текущего). В нашем примере этот интервал, как видно в ответе на запрос, для 2 апреля 2002 года состоит из дат с 31 марта по 4 апреля 2002 года, для 7 июля — из дат с 5 по 8 июля.

```
SELECT t.time_key, SUM(f.purchase_price) as sales,
     SUM(SUM(f.purchase_price)) OVER (ORDER BY t.time_key
     RANGE BETWEEN INTERVAL '2' DAY PRECEDING AND
          INTERVAL '2' DAY FOLLOWING) as sales_5_day
FROM purchases f, time t
WHERE f.time_key = t.time_key
GROUP BY t.time_key;
```

```
TIME_KEY        SALES        SALES_5_DAY
---------  ----------        -----------
01-JAN-02   8989040.32         8989096.34
02-JAN-02        56.02         8989096.34
01-FEB-02   3401896.62        17213525.6
```

```
02-FEB-02    13811629       17213525.6
02-MAR-02     429035.78      429035.78
30-MAR-02        448.19     1958695.44                    _.
31-MAR-02        448.19     1960040.01                     |
01-APR-02    1957799.06     1960936.39                     |
02-APR-02       1344.57     1962280.96 <- текущая строка  | логический
03-APR-02        896.38     1961832.77                    | интервал
04-APR-02       1792.76        4033.71                   _|
08-APR-02    2562502.88     2562502.88
06-MAY-02     839281.7      1268317.48
07-MAY-02     429035.78     1268317.48
01-JUN-02    6192443.48     7092585.46
03-JUN-02     900141.98     7092585.46
02-JUL-02        56.02          56.02                    _
05-JUL-02      2240.95        2268.96                     |
07-JUL-02        28.01        2296.97 <- текущая строка  | логический
08-JUL-02        28.01          56.02                   _| интервал
```

В приведенном примере для указания верхней и нижней границы интервала использовано условие BETWEEN. Его также можно использовать для указания физического сдвига (например, ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING).

Отчетные агрегаты

Предположим, нам нужно ответить на такой вопрос: «Продукты какой категории лучше всего продаются в каждом из регионов?». Если мы попробуем написать такой SQL-запрос, мы быстро поймем, что код SQL, который потребуется для получения ответа на такой простой вопрос, весьма сложен. Например, одним из способов может быть такой: сначала создать таблицу, с именем, скажем, total_sales, содержащую общие объемы продаж по категориям продуктов для каждого региона. Затем провести дальнейшую агрегацию по этой таблице для выясненя максимальных объемов продаж по каждому региону, сохранив их в таблице с именем max_sales. После этого провести соединение таблиц total_sales и max_sales и выявить в таблице max_sales те строки, которые соответствуют максимальным объемам продаж для каждого региона, и возвратить соответствующие категории продуктов. Также вы можете вписать все эти шаги в один большой SQL-запрос, включающий несколько подзапросов, но все это будет чрезвычайно громоздко.

Причиной сложности подобных запросов является то, что при запросе на SQL простого агрегата, такого, как SUM или MAX, вы теряете конкретные строки, задействованные в агрегате. Поэтому, когда мы считаем максимальные объемы продаж в таблице max_sales, мы теряем информацию о категориях продуктов, в которых содержится данное значение объема продаж.

Отчетные агрегаты (reporting aggregates) решают эту проблему, выдавая отчет о вычисленных значениях агрегата вместе со всеми строками, которые при этом использовались. Как и при использовании остальных аналитических функций, вы можете применить условие PARTITION BY и секционировать данные перед вычислением отчетного агрегата.

В следующем запросе выясняются максимальные объемы продаж по каждому региону с применением отчетного агрегата. Условие PARTITION BY просто секционирует данные по регионам. Отчетный агрегат MAX(SUM(f.purchase_price)) вычисляет максимальные объемы продаж по каждому региону и выдает отчет для каждой категории продуктов. Таким образом, значение max_sales региона повторяется для каждой категории каждого региона. То, что данный агрегат является отчетным, а не обыкновенным, определяет условие OVER.

```
SELECT p.category, g.region,
     SUM(f.purchase_price) as regional_prod_sales,
     MAX(SUM(f.purchase_price) ) OVER (PARTITION BY g.region)
        as max_sales
FROM product p, geography g, purchases f
WHERE f.product_id = p.product_id
   AND f.state_id = g.state_id
GROUP BY p.category, g.region;
```

CATEGORY	REGION	REGIONAL_PROD_SALES	MAX_SALES	
HDRW	MidWest	8044.01	25808.49	
MUSC	MidWest	25808.49	25808.49	*
ELEC	NorthEast	1574193.47	1574193.47	*
HDRW	NorthEast	16704.22	1574193.47	
MUSC	NorthEast	7192.53	1574193.47	
ELEC	West	1574193.47	1574193.47	*
HDRW	West	13115.79	1574193.47	
MUSC	West	2977.3	1574193.47	

Теперь вернемся к нашему исходному вопросу: «Продукты какой категории лучше всего продаются в каждом из регионов?». Все, что нам нужно, — это выбрать из предыдущего запроса те строки, в которых значения regional_prod_sales равны max_sales. Эти строки в приведенных данных помечены звездочками.

```
SELECT category, region, regional_prod_sales
FROM
(
 SELECT p.category, g.region,
      SUM(f.purchase_price) as regional_prod_sales,
      MAX(SUM(f.purchase_price))
          OVER (PARTITION BY g.region) as max_sales
FROM product p, geography g, purchases f
WHERE f.product_id = p.product_id
AND f.state_id = g.state_id
GROUP BY p.category, g.region
)
WHERE regional_prod_sales = max_sales;
```

CATEGORY	REGION	REGIONAL_PROD_SALES	MAX_SALES
MUSC	MidWest	25808.49	25808.49
ELEC	NorthEast	1574193.47	1574193.47
ELEC	West	1574193.47	1574193.47

Еще один пример, где могут пригодиться отчетные агрегаты, — это составление отчетов по зарплате каждого работника вместе со средней зарплатой по департаменту.

```
SELECT employee, department, salary
      AVG(salary)
        OVER (PARTITION by department) as avg_dept_salary
FROM payroll;
```

Особый случай отчетного агрегата — это функция RATIO_TO_REPORT, которая подсчитывает отношение значения в наборе данных к сумме всех значений в этом наборе. В приведенном примере находится отношение объемов продаж в каждой категории продуктов к общему объему для всех категорий. В выражении

SUM(SUM(f.purchase_price)) вычисляется общий объем продаж каждого продукта категории. RATIO_TO_REPORT, по сути, представляет собой более краткую форму деления reg_sales/ reg_sales_total.

```
SELECT category, SUM(f.purchase_price) reg_sales,
  SUM(SUM(f.purchase_price)) OVER () as reg_sales_total,
  RATIO_TO_REPORT(SUM(f.purchase_price)) OVER () as ratio_to_report
FROM product p, purchases f
WHERE f.product_id = p.product_id
GROUP BY p.category;
```

```
CATE    REG_SALES     REG_SALES_TOTAL   RATIO_TO_REPORT
----    -----------   ---------------   ---------------
ELEC    33334833.10         39520369.7     .843484848
HDRW    2962332.61          39520369.7     .074957108
MUSC    3223204.06          39520369.7     .081558044
```

Интересный аспект предыдущего примера — использование пустого условия OVER(). Это просто означает, что функция проводит вычисления без секционирования по условиям (иными словами, по всем строкам).

Функции упреждения (Lead) и запаздывания (Lag)

Функции упреждения и запаздывания обеспечивают доступ к значениям столбца с известным смещением от текущей строки. Например, функция Lag(sales, 1) возвращает значение предыдущей строки столбца sales. Функция Lead(sales, 1) возвращает значение следующей строки столбца sales. Таким образом, запросы, которые иначе потребовали бы соединять таблицу саму с собой, могут быть обработаны без такой операции. Например, в следующем запросе выдаются объемы продаж по месяцам вместе с объемами за предыдущий и последующий месяцы 2002 года.

```
SELECT t.month, SUM(f.purchase_price) as sales,
     LAG(SUM(f.purchase_price),1)
        OVER (ORDER BY t.month) as sales_last_month,
     LEAD(SUM(f.purchase_price),1)
        OVER (ORDER BY t.month) as sales_next_month
PROM purchases f, time t
WHERE f.time_key = t.time_key AND t.year = 2002
GROUP BY t.month;
```

```
MONTH    SALES          SALES_LAST_MONTH  SALES_NEXT_MONTH
-----    -----------    ----------------  ----------------
1        8989096.34                       17213413.5
2        17213413.5       8989096.34        429035.78
3        429035.78       17213413.5        4520301.94
4        4520301.94        429035.78        1268317.48
5        1268317.48       4520301.94        7092585.46
6        7092585.46       1268317.48
...
```

Lag- и lead-функции можно использовать для проведения сравнений объемов продаж по периодам времени. Например, чтобы сравнить продажи за квартал с тем же кварталом год назад (т. е. четыре квартала назад), вы можете использовать выражение Lag (sales, 4).

Заметим, что если предыдущая (или следующая) строка отсутствует, функции Lag и Lead возвращают значение NULL.

Функции FIRST и LAST

В некоторых ситуациях может потребоваться отсортировать данные по некоторому количественному показателю и провести анализ на основе первого и последнего значения в этом ряду. Функции FIRST и LAST позволяют проводить подобные операции. Например, в следующем запросе выясняется количество покупок самого дорогого и самого дешёвого товара в каждой категории. Сначала продукты в каждой категории сортируются по цене (sell_price), а затем проводится агрегирование полученных строк с помощью выражения COUNT(*), что определяет число покупок для тех продуктов, которые определили функции FIRST и LAST. Заметим, что данный агрегат разрешает конфликты при наличии нескольких кандидатов на место первого и последнего, генерируя одну строку для всех.

```
SELECT p.category cat, SUM(f.purchase_price) total_sales,
     MIN(p.sell_price) cheap_prod,
     COUNT(*) KEEP (DENSE_RANK FIRST
          ORDER BY p.sell_price) cheap_sales,
     MAX(p.sell_price) costly_prod, .
     COUNT(*) KEEP (DENSE_RANK LAST
          ORDER BY p.sell_price) costly_sales
FROM purchases f, product p
```

```
WHERE f.product_id = p.product_id
GROUP BY p.category;

CAT  TOTAL_SALES CHEAP_PROD CHEAP_SALES COSTLY_PROD COSTLY_SALES
---- ----------- ---------- ----------- ----------- ------------
ELEC  33327213.9         24.99       16141      1267.89        15036
HDRW  2962332.61         15.67        9980           18         5001
MUSC  3223204.06         10           2305        15.67        13710
```

Функции FIRST и LAST можно использовать для выявления тенденций. Например, в нашем примере мы обнаружили, что в категории ELEC объем продаж для самых дорогих товаров практически тот же, что и для дешевых, тогда как в категории HDRW различие в $3 снижает объем продаж наполовину.

Функции FIRST и LAST можно использовать в качестве отчетных агрегатов.

Функции для статистического анализа

Часто на решения в бизнесе могут влиять взаимоотношения различных количественных показателей. Например, цена товара может повлиять на то, какое количество этих товаров будет продано. Метод линейной регрессии — это статистический метод, используемый для количественной оценки того, как один показатель может повлиять или определить значение другого. Идея в том, чтобы как можно точнее расположить данные этих двух количественных показателей на прямой линии. Эта линия называется линией регрессии. Некоторые интересные функции — это наклон этой линии, точка пересечения с осью Y (y-intercept) и коэффициент корреляции (т. е. насколько точно линия соответствует точкам). Oracle предлагает разнообразные диагностические функции, широко используемые для такого анализа, например для вычисления стандартной ошибки и регрессионной суммы квадратов.

Все функции линейной регрессии вычисляются одновременно, за один проход по данным. Их можно рассматривать как обычные или отчетные функции-агрегаты.

Приведенный пример иллюстрирует применение некоторых из этих функций. Здесь мы вычисляем уклон, точку пересечения и коэффициент корреляции линии регрессии для значений sell_price и purchase_price в каждой из категорий продуктов.

```
SELECT p.category cat,
     REGR_SLOPE(p.sell_price, f.purchase_price) slope,
     REGR_INTERCEPT(p.sell_price, f.purchase_price) intercept,
     REGR_R2(p.sell_price, f.purchase_price) coeff_determination
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category;

CAT   SLOPE           INTERCEPT    COEFF_DETERMINATION
----  ----------      ----------   -------------------
ELEC  .830878246      -87.405372             .831264004
HDRW  -.00003465      16.4546582             .000057048
MUSC  -.00005981      14.8659674             .000053216
```

По этому анализу мы можем видеть, что взаимоотношение между sell_price и purchase_price в категории ELEC можно приближенно смоделировать прямой линией. Однако в категориях HDRWи MUSC приближение этой линии невелико.

Для поддержки анализа линейной регрессии Oracle предлагает некоторые новые агрегатные функции для вычисления ковариантности популяции (COVAR_POP) или выборки (COVAR_SAMP) и корреляции (CORR) между переменными.

Обратный процентиль

Выше мы упоминали функцию CUME_DIST(), которая определяет положение каждой строки данных относительно всего набора. Также эта операция называется процентиль. Обратный процентиль делвает обратное преобразование, т. е. возвращает значение из набора строк, соответствующее данному значению процентиля.

В следующем запросе вычисляется функция CUME_DIST() для продаж в каждой категории. Строки сортируются функцией SUM(purchase_oprice), и каждой строке присваивается значение от 0 до 1. Мы покажем лишь несколько из этих строк.

```
SELECT p.category, p.product_id, SUM(f.purchase_price) as sales,
   CUME_DIST() over (PARTITION BY p.category
         ORDER BY SUM(f.purchase_price)) as cume_dist
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category, p.product_id;
CATE    PRODUCT        SALES       CUME_DIST
----  --------    ----------     ----------
...
ELEC     SP1100      150155.8       .495412844
ELEC     SP1093      150337.4       .504587156   <- PERCENTILE_DISC(0.5)
ELEC     SP1076      150717.89      .513761468
...
HDRW     SP1213      112749.9       .461538462
HDRW     SP1217      113019.02      .5           <- PERCENTILE_DISC(0.5)
HDRW     SP1122      113648.09      .538461538
...
MUSC     SP1251      114592.31      .481481481
MUSC     SP1253      114840.43      .518518519   <- PERCENTILE_DISC(0.5)
MUSC     SP1264      115221.38      .555555556
...
```

Существует два вида обратного процентиля. PERCENTILE_DISC() подразумевает, что значения являются дискретными, и возвращает значение данных, соответствующее ближайшему значению CUME_DIST(), превышающему указанный процентиль. Другой вид — PERCENTILE_CONT() — предполагает, что значения непрерывны, и возвращает интерполированное значение, соответствующее данному процентилю. В предыдущем примере значение объема продаж, соответствующее PERCENTILE_DISC(0.5), в категории HDRW равно 113019.02, а для категории MUSC равно 114840.43.

Следующий пример иллюстрирует использование функций обратного процентиля. Заметим, что поскольку функции PERCENTILE_DISC() и PERCENTILE_CONT() должны возвращать одно значение, необходимо указать критерий ORDER BY, состоящий из одного выражения.

```
SELECT p.category, p.product_id, SUM(f.purchase_price) as sales,
     PERCENTILE_DISC(0.5) WITHIN GROUP
          (ORDER BY SUM(f.purchase_price))
          OVER (PARTITION BY p.category) as pct_disc,
     PERCENTILE_CONT(0.5) WITHIN GROUP
          (ORDER BY SUM(f.purchase_price))
          OVER (PARTITION BY p.category) as pct_cont
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category, p.product_id;
```

CATE	PRODUCT_	SALES	PCT_DISC	PCT_CONT
...				
ELEC	SP1100	150155.8	150337.4	150337.4
ELEC	SP1093	150337.4	150337.4	150337.4
ELEC	SP1076	150717.89	150337.4	150337.4
...				
HDRW	SP1213	112749.9	113019.02	113333.555
HDRW	SP1217	113019.02	113019.02	113333.555
HDRW	SP1122	113648.09	113019.02	113333.555
...				
MUSC	SP1251	114592.31	114840.43	114840.43
MUSC	SP1253	114840.43	114840.43	114840.43
MUSC	SP1264	115221.38	114840.43	114840.43
...				

В данном примере функции PERCENTILE_DISC() и PERCENTILE_CONT() были использованы в качестве отчетных агрегатов.

Анализ возможных вариантов

Аналитические приложения часто используют анализ возможных вариантов (what-ifanalysis). Например, вам может понадобиться узнать, как будет вести себя новый продукт относительно других в своей категории при наличии ориентировочных цифр объемов продаж. Oracle предлагает для этой цели семейство функций гипотетического ранжирования и распределения. Используя эти функции, вы можете вычислить функции RANK(), PERCENT_RANK() и CUME_DIST() для данного значения, как если бы оно находилось в наборе значений.

Функции гипотетического ранжирования принимают условие сортировки и постоянное значение, которое должно быть вставлено в отсортированный набор данных. Для иллюстрации мы покажем объемы продаж разных продуктов, а затем присвоим ранг гипотетическому продукту.

В приводимый запрос выводим объемы продаж продуктов в категории HDWR и их соответствующие ранги.

```
SELECT p.product_id, SUM(f.purchase_price) sales,
     RANK() OVER (ORDER BY SUM(f.purchase_price)) as rank
FROM purchases f, product p
WHERE f.product_id = p.product_id and p.category = 'HDRW'
GROUP BY p.product_id;
    /
CATE    PRODUCT_      SALES      RANK
----    ----------  --------  -----
HDRW    SP1222       81385.09     1
HDRW    SP1236       82520.97     2
HDRW    SP1220       82622        3
HDRW    SP1231       83073.57     4
HDRW    SP1214       83841.66     5
                                     <- вставка предполагаемого значения 98000.00
HDRW    SP1212       99663.51     6
HDRW    SP1232       99863.42     7
HDRW    SP1224      100825.37     8
  ...
```

Теперь предположим, что нам необходимо определить предполагаемый ранг продукта с объемом продаж $98000. В приведенном выше фрагменте можно увидеть, что если мы вставим это значение в данные, оно получит ранг 6. Приводимый ниже запрос определяет гипотетический ранг.

```
SELECT RANK(98000.00) WITHIN GROUP
 (ORDER BY SUM(f.purchase_price)) as hrank,
FROM purchases f, product p
WHERE f.product_id = p.product_id and p.category = 'HDRW'
GROUP BY p.product_id;
HRANK
----------
6
```

Определить функцию гипотетического ранжирования в запросе можно по условию WITHIN GROUP и константному выражению в функции RANK().

3.7.3. Пользовательские агрегаты

В Oracle 9i введена возможность определения пользователем собственных агрегатных функций. Эти функции можно использовать как обычные агрегаты или как аналитические функции — отчетные или рамочные агрегаты. Пользовательские агрегаты можно применять в случаях сложных функций, которые нельзя написать на SQL, или сложных представлений данных, таких, как объекты или LOB. Они особенно полезны в приложениях, связанных с финансами.

Пользовательские агрегаты являются частью расширяемой основы Oracle. Чтобы определить пользовательский агрегат, нужно сначала определить тип, реализующий интерфейс ODCIAggregate. После этого вы сможете объявить функцию, использующую данный тип, и выполнить агрегирование. Реализацию агрегатных функций можно осуществить на любом процедурном языке, таком, как C, PL/SQL или Java.

Интерфейс ODCIAggregate состоит из следующих функций:

- ODCIAggregateInitialize() инициализирует значение агрегата перед началом обработки;
- ODCIAggregateIterate() обновляет значение агрегата для новой строки данных;
- ODCIAggregateTerminate() возвращает значение агрегата и завершает обработку;
- ODCIAggregateMerge() используется для поддержки параллельных вычислений агрегата. Агрегат вычисляется по разным частям данных и в конце вызывается функция ODCIAggregateMerge() для объединения результатов. Чтобы функция была доступна, нужно указать условие PARALLEL_ENABLE.

Предположим, например, что у вас есть собственный алгоритм прогнозирования объемов продаж, который принимает цифры имеющихся объемов и выдает прогноз на следующий год. Вы можете определить пользовательский агрегат следующим образом. Тип SalesForecastFunction реализует интерфейс ODCIAggregate. (Реализацию мы здесь опускаем из-за недостатка места). Функция SalesForecast() объявляется как агрегатная функция с помощью SalesForecastFunction.

```
CREATE OR REPLACE TYPE SalesForecastFunction AS OBJECT (
  data number,
  STATIC FUNCTION ODCIAggregateInitialize
         (ctx IN OUT SalesForecastFunction)
    RETURN number,
  MEMBER FUNCTION ODCIAggregateIterate
         (self IN OUT SalesForecastFunction,
         value IN number) RETURN number,
  MEMBER FUNCTION ODCIAggregateTerminate
         (self IN OUT SalesForecastFunction,
         returnValue OUT number,
         flags IN number) RETURN number,
  MEMBER FUNCTION ODCIAggregateMerge
         (self IN OUT SalesForecastFunction,
         ctx2 IN OUT SalesForecastFunction)
         RETURN number
);
/
CREATE OR REPLACE TYPE BODY SalesForecastFunction
IS
...
END;
/
CREATE or REPLACE FUNCTION SalesForecast (x number) RETURN number
PARALLEL_ENABLE AGGREGATE USING SalesForecastFunction;
```

Затем эту функцию можно использовать в SQL-запросе вместо любого другого агрегата следующим образом. Чтобы удалить повторяющееся значение столбца перед агрегированием, можно также использовать флаг DISTINCT.

```
SELECT p.category, SUM(f.purchase_price) sales,
     SalesForecast (f.purchase_price) as salesforecast
FROM purchases f, product p
WHERE f.product_id = p.product_id
GROUP BY p.category;
```

```
CATE          SALES        SALESFORECAST
----   -----------        -------------
ELEC    33327213.9          36659935.3
HDRW     2962332.61            3258565.87
MUSC     3223204.06            3545524.47
```

Совет. Перед тем как реализовывать пользовательский агрегат, проверьте, можно ли решить вашу задачу при помощи существующих SQL-агрегатов, поскольку быстродействие при этом будет выше. Обсуждаемая далее функция CASE может использоваться для самых разнообразных сложных вычислений.

3.7.4. Выражение CASE

Выражение CASE позволяет возвращать разные выражения в зависимости от условий. Простой оператор CASE аналогичен оператору DECODE, с помощью которого вы можете возвращать разные значения в зависимости от значения выражения. Поисковый оператор CASE, как показано в приводимом примере, дает большую гибкость. Предположим, вам нужно сделать новый анонс и вы хотите разделить продукты на группы по их общим объемам продаж. Вы можете использовать выражение CASE следующим образом.

```
SELECT f.product_id, SUM(f.purchase_price) as sales,
   CASE WHEN SUM(f.purchase_price) > 150000 THEN 'High'
     WHEN SUM(f.purchase_price)
        BETWEEN 100000 and 150000 THEN 'Medium'
     WHEN SUM(f.purchase_price)
        BETWEEN 50000 and 100000 THEN 'Low'
     ELSE 'Other' END as sales_value
FROM purchases f
GROUP BY f.product_id;

PRODUCT_        SALES    SALES_VALUE
--------    ----------   ------------
SP1000       147702.64         Medium
SP1001       106161.88         Medium
SP1010       122422.72         Medium
```

```
SP1011      159622.6          High
SP1012      143332.88         Medium
SP1013       89592.8           Low
SP1014      129582.53         Medium
SP1015      110505.91         Medium
SP1016      161189.64         High
SP1017       93159.68          Low
SP1018      174191.4          High
   ...
```

Вы можете использовать выражение CASE везде, где вы используете столбец или выражение, в том числе и внутри агрегатных функций. Комбинацию агрегатов и выражений CASE можно применять для проведения сложных вычислений. Например, предположим, что вам нужно увидеть средние значения объемов продаж при условии игнорирования значений менее $10, добавления 10-процентного налога на роскошь для всех товаров, превышающих $500 и закрытия магазина в Массачусетсе. Можно использовать следующее выражение.

```
SELECT AVG(f.purchase_price) as current_avg_sales,
   AVG  (CASE WHEN f.purchase_price < 10 THEN 0
     WHEN f.purchase_price > 500
        THEN 1.1 * f.purchase_price
     WHEN f.state_id = 'MA' THEN 0
     ELSE f.purchase_price END) projected_avg_sales
FROM purchases f;

CURRENT_AVG_SALES    PROJECTED_AVG_SALES
-----------------    -------------------
       272.493792             293.053466
```

3.7.5. Условие WITH

В аналитических приложениях часто используются запросы, содержащие сложные подзапросы. Фактически, один и тот же подзапрос может повторяться в запросе несколько раз. В Oracle 9i вводится условие WITH, которое позволяет дать подзапросу имя и затем использовать это имя в выражении вместо этого подзапроса. Если один подзапрос появляется в запросе несколько раз, база Oracle автоматически «материализует» этот подзапрос во временную таблицу, которая будет

повторно использоваться при выполнении запроса. Время существования этой таблицы ограничивается временем выполнения запроса, и она автоматически удаляется по окончанию выполнения.

Условия WITH может повысить удобочитаемость сложных запросов и улучшить производительность обработки тех запросов, которые требуют повторяющихся вычислений.

В приводимом примере показано, как можно использовать условие WITH для упрощения запроса. Допустим, мы ищем месяцы, в которые объемы продаж были максимальны, с помощью такого запроса:

```
SELECT s.category, s.month, s.monthly_prod_sales
FROM  (SELECT p.category, t.month,
          SUM(f.purchase_price) as monthly_prod_sales
       FROM product p, purchases f, time t
       WHERE  f.product_id = p.product_id
         AND f.time_key = t.time_key
       GROUP BY p.category, t.month) s
WHERE s.monthly_prod_sales
    IN  (SELECT MAX(v.monthly_sales)
       FROM  (SELECT p.category, t.month,
             SUM(f.purchase_price) as monthly_sales
          FROM product p, purchases f, time t
          WHERE  f.product_id = p.product_id
           AND f.time_key = t.time_key
          GROUP BY p.category, t.month) v
       GROUP BY v.month);
```

Можно видеть, что подзапросы с псевдонимами «s» и «v» идентичны, поэтому давайте перепишем этот запрос с помощью условия WITH. Сначала мы возьмем повторяющийся подзапрос и дадим ему имя, например, product_sales_by_month. Затем везде, где мы использовали этот подзапрос, мы будем использовать это имя. В результате получится такое выражение:

```
WITH product_sales_by_month                    <- даем запросу имя
AS
 (
 SELECT p.category, t.month,
      SUM(f.purchase_price) as monthly_prod_sales
 FROM product p, purchases f, time t
```

```
 WHERE f.product_id = p.product_id
   AND f.time_key = t.time_key
 GROUP BY p.category, t.month
)
SELECT s.category, s.month, s.monthly_prod_sales
FROM product_sales_by_month s
WHERE s.monthly_prod_sales                    <- здесь используем имя
    IN (SELECT MAX(v.monthly_prod_sales)
      FROM product_sales_by_month v       <- здесь используем имя
      GROUP BY v.month);
```

Обработка запроса происходит в этом случае более эффективно, поскольку база Oracle может принять решение о «материализации» результата подзапроса во временной таблице и повторном использовании его в обоих местах, что позволяет сэкономить на повторяющихся вычислениях. Кроме того, теперь запрос значительно легче читать. Заметим, что данный конкретный запрос также можно эффективно обработать с помощью ранее обсуждавшихся отчетных агрегатов.

В данном разделе мы обсудили несколько SQL-функций базы Oracle, которые позволяют составлять сложные аналитические запросы. Все возможности, которые мы пока описываем в этой главе, могут значительно выиграть от использования параллельных вычислений, которые мы обсудим в следующем разделе.

3.8. Параллелизм

Многие операции с хранилищем включают обработку больших объемов данных. Загрузка больших массивов, создание индексов и итогов, сортировка и соединение данных из многих таблиц — все это может занимать значительное время. Для уменьшения времени выполнения таких операций можно использовать параллельную обработку.

Параллелизм позволяет явно секционировать некоторые SQL-выражения на несколько параллельно выполняемых операций. Разделение работы на несколько процессов, выполняемых на разных процессорах, позволяет выполнить выражение быстрее, чем это было бы при одном процессе.

В базе Oracle параллельное выполнение осуществляется с помощью процесса-координатора параллельного выполнения и пула серверов параллельного выполнения. Процесс базы Oracle, который обрабатывает запрос пользователя, становится процессом-координатором для данного запроса. Процесс-координатор разделяет работу, которую нужно сделать, среди необходимого количества сер-

веров параллельного выполнения. Он обеспечивает балансировку нагрузки между процессами и перераспределяет работу, если какой-то процесс закончил свою часть до остальных. Координатор получает результаты от серверов параллельного выполнения и компонует из них окончательный результат.

Наибольший выигрыш в производительности от использования параллельной обработки имеют те процессы, которые обрабатывают большие объемы данных и возвращают результат небольшого размера, такие, как операции агрегации. Например, операция получения общего объема продаж за первые четыре месяца года показана на рис. 3.10.



Рис. 3.10. *Параллельный запрос*

В Oracle 9i многие операции могут выиграть от использования параллелизма, в том числе операторы DDL (такие, как CREATE TABLE, CREATE INDEX и CREATE MATERIALIZED VIEW), операции, связанные с секционированием (такие, как TRUNCATE, EXCHANGE, MERGE и SPLIT PARTITION), операторы SELECT и выражения DML. Можно выполнять параллельно такие операции, как соединение, агрегация, сортировка, выражение DISTINCT внутри оператора SELECT.

Производительность таких действий, как отсечение разделов (partition pruning) и соединение с учетом разделов (partition-wise join), которые обсуждались ранее, значительно выигрывает от использования параллельной обработки.

В Oracle 9i Release 2 теперь возможны параллельные операции DML как над секционированными, так и над несекционированными таблицами, однако параллельная операция INSERT поддерживается только для несекционированных таблиц.

Чтобы иметь возможность параллельно выполнять выражения DML, в сессии необходимо использовать выражение ALTER SESSION ENABLE PARALLEL DML. В некоторых случаях параллелизм возможен и для функций PL/SQL и табличных функций.

Выполняемую параллельно операцию можно разделить на части разными способами. Если в таблице или индексе применяется секционирование, как в предыдущем примере, координатор может назначить серверу параллельного выполнения работу над одним или более разделом. Как альтернатива, таблицу или индекс можно разделить на диапазоны физических блоков данных на диске. Все серверы параллельного выполнения работают параллельно над разными диапазонами блоков.

Можно устанавливать степень параллелизма для таблицы или индекса с помощью условия PARALLEL для операторов CREATE или ALTER TABLE, для выражения SQL — с помощью указаний оптимизатору и для сессии — с помощью выражения ALTER SESSION FORCE PARALLEL. Следующее выражение устанавливает уровень параллелизма для таблицы покупок равным четырем.

```
ALTER TABLE purchases PARALLEL 4;
```

Если не использовать эти методы, база Oracle определяет уровень параллелизма, учитывая количество процессоров или число разделов. Для хорошей производительности нужно, чтобы количество разделов было кратно уровню параллелизма. В противном случае некоторые серверы параллельного выполнения могут простаивать в ожидании, пока другие закончат операцию, перед тем как переходить к следующей. Оптимизатор запросов сделает попытку использовать указанный уровень параллелизма, но, если несколько запросов будут конкурировать за ресурсы, он произведет настройку на более низкое значение.

При запуске базы данных база Oracle запустит столько серверов параллельного выполнения, сколько их указано в инициализационном параметре PARALLEL_MIN_SERVERS. Эти серверы доступны для использования в любой параллельной операции. Координатор запроса берет необходимое число серверов параллельного выполнения из этого пула по мере необходимости. Когда обработка заканчивается, координатор возвращает сервера в пул.

Если параллельные операции использует большое число конкурирующих между собой пользователей, возможно создание дополнительных серверов параллельного выполнения. Инициализационный параметр PARALLEL_MAX_SERVERS определяет максимальное количество создаваемых серверных процессов. Если необходимость в серверах отпадает и они долго простаивают, их работа завершается. Пул серверов никогда не бывает меньше числа, указанного в параметре PARALLEL_MIN_SERVERS. Если сервер параллельного выполнения недоступен, операция выполняется последовательно. Инициализационный параметр PARALLEL_MIN_PERCENT используется для указания минимальной процентной доли серверов параллельного выполнения, необходимой для успешного выполнения операции. Если эта доля недостижима, база Oracle возвращает ошибку и вы можете попробовать выполнить эту операцию позже.

В Oracle 9i был добавлен новый инициализационный параметр, PARALLEL_AUTOMATIC_TUNING, устанавливающий значения по умолчанию для некоторых параметров настройки параллельных операций. Новое свойство автоматического управления памятью (см. раздел 3.6) устраняет необходимость ручной настройки размеров рабочей области для многих операций, таких, как хэш-соединение и сортировка. Это значительно облегчает использование возможностей параллельного выполнения.

3.8.1. Аппаратные требования для параллельной обработки

Требования к пропускной способности хранилища часто выполняются только при использовании параллельной обработки. Чтобы пользоваться преимуществами параллелизма, важно иметь правильные системные и аппаратные характеристики. Аппаратной базой для хранилища или тематического хранилища должна служить многопроцессорная машина, либо симметричная мультипроцессорная (symmatyric multiprocessor, SMP), либо кластерная, либо система с массовым параллелизмом (massively parallel processor, MPP). Однопроцессорные системы подходят только для небольших тематических хранилищ.

MPP-системы имеют несколько узлов, соединенных высокоскоростной связью. При архитектуре «без разделения» («shared nothing») каждый узел представляет собой процессор, память и локальные диски и работает под управлением своей собственной версии операционной системы. Чтобы работать в MPP-системе, нужно соответствующим образом секционировать данные по дискам. MPP-системы могут обеспечивать высокую производительность, но являются неустойчивыми к ошибкам. Потеря узла приводит к недоступности части данных.

Такими системами трудно управлять, поскольку добавление новых узлов может потребовать повторного секционирования данных. Кроме того, поскольку запросы могут не обращаться ко всем данным, использование узлов может быть неравномерным.

SMP-системы имеют несколько процессоров, которые разделяют память и диски посредством общей шины. Здесь работает только одна версия операционной системы, которая распределяет выполнение задач для процессоров. Системы с разделяемыми дисками требуют наличия менеджера распредленного блокирования (distributed lock manager), который координирует доступ к данным с разных узлов, что создает определенную нагрузку. Тем не менее, SMP-системы не страдают от проблем с обслуживанием и преодолением ошибок, как MPP-системы, и их хорошо выбирать для хранилища. Однако, SMP-системы поставляются в конфигурациях, имеющих не более некоторого числа процессоров, обычно 64.

Кластеры объединяют вместе несколько однопроцессорных узлов или SMP. Кластеры — это хороший метод масштабирования хранилища данных при возрастании потребностей в них. Так кластеры обеспечивают великолепную надежность, поскольку потеря узла не означает недоступности данных. В Oracle 9i вводится технология Real Application Clusters (RAC), которая обеспечивает усовершенствованную поддержку кластеров для базы Oracle. В кластерной системе каждый узел кластера работает со своим экземпляром сервера базы данных Oracle, которые все вместе осуществляют доступ к данным. Помимо общих данных на дисках, RAC обеспечивает разделение буферных кэшей экземпляров, что снижает нагрузку на дисковые каналы ввода-вывода и улучшает производительность. Если желательно параллельное выполнение, один SQL-запрос может быть распределен по подчиненным процессам, работающим на нескольких экземплярах базы данных. База Oracle определит в процессе выполнения, нужно ли запускать процесс на одном экземпляре или на нескольких, пытаясь свести общение между узлами к минимуму.

Параллелизм повышает производительность только в тех случаях, когда есть необходимые аппаратные ресурсы. Если ваши процессоры или диски уже сильно перегружены, обновите аппаратную конфигурацию перед тем, как задействовать параллелизм.

3.9. Заключение

В этой главе мы рассмотрели некоторые свойства базы данных Oracle, которые особенно подходят для хранилищ. Секционирование данных повышает доступность системы и делает ее управляемой. Параллельное выполнение улучшает

масштабируемость системы по мере роста объемов данных. Oracle предлагает несколько механизмов улучшения производительности обработки запросов, таких, как индексы на основе битовых карт, трансформация по типу звезды, соединение с учетом разделов и отсечение разделов. Потребности в сложном агрегировании и отчетах удовлетворяются при помощи новых агрегатных и аналитических функций. Автоматическое управление памятью облегчает настройку размеров рабочей области.

Хорошо спроектированное хранилище данных, в котором правильно используются эти возможности, может обеспечить отличную производительность при больших объемах данных. Однако это еще не все. В следующей главе мы обсудим еще один необходимый инструмент любого разработчика хранилищ: переписывание запросов при помощи таблицы итогов.

4

Управление сводными данными (Summary Management)

4.1. Сводные таблицы

Обычный метод, используемый в хранилищах, — это предварительный обсчет и сохранение результатов часто поступающих запросов. Это особенно выгодно, когда в запросах используется агрегация, поскольку ее результат, как правило, намного меньше по размерам, чем основные данные, используемые для его вычисления. Примером такого запроса является ежемесячный отчет по продажам в бизнесе. Поскольку многих пользователей интересует общий объем продаж по каждому продукту каждый месяц, эти данные нужно отобрать, соединить *с* другими, отсортировать, агрегировать, и так снова и снова для каждого пользователя. Чтобы не тратить ресурсы на повторное выполнение одного и того же запроса, его результат можно рассчитать заранее и сохранить в таблице. Такие предварительно вычисленные результаты часто называются сводками (summaries) или сводными таблицами (summary tables).

В табл. 4Л показана сводка, содержащая общее количество проданных за каждый месяц года товаров. Сводные таблицы обычно намного меньше таблиц, содержащих подробные данные. В данном примере 17 строк подробных данных из системы обработки транзакций сведены в 10 строк. В зависимости от данных, уменьшение размеров и, следовательно, выигрыш в производительности обработки запроса может быть весьма значительным.

Появившееся в Oracle 8i свойство Summary Management (Управление сводными данными) совершило прорыв в использовании сводных таблиц, позволив базе данных явным образом управлять сводками.

4.1.1. Зачем нужно Управление сводными данными?

Чтобы понять, зачем нужно Управление сводными данными, давайте сначала рассмотрим некоторые задачи, решаемые при управлении сводными таблицами.

Таблица 4.1. *Сводная таблица*

Подробные данные системы транзакций		
Дата	Покупатель	Продукт
1/5/02	Smith	Книга
1/6/02	Jones	Палатка
2/5/02	Smith	Книга
2/5/02	Mills	Книга
3/5/02	Smith	Пленка
3/7/02	Mills	Книга
3/5/02	Smith	Пленка
4/1/02	Smith	Книга
4/2/02	Jones	Книга
4/3/02	Smith	Палатка
4/3/02	Jones	Палатка
5/5/02	Peters	Книга
5/6/02	Smith	Книга
6/1/02	Jones	Книга
6/5/02	Smith	Палатка
6/6/02	Jones	Палатка
6/5/02	David	Палатка

Объемы продаж по месяцам		
Месяц	Продукт	Продано
1/02	Книга	1
1/02	Палатка	1
2/02	Книга	2
3/02	Книга	1
3/02	Пленка	2
4/02	Книга	2
4/02	Палатка	2
5/02	Книга	2
6/02	Книга	1
6/02	Палатка	3

Сводка, по сути, представляет собой заранее вычисленный результат SQL-запроса. Как только сводка создана, ответ на запрос может быть получен из нее. Сводку также можно использовать для ответа на другие, связанные запросы. Например, если нас интересуют объемы продаж каждого продукта за каждый

год, ответ можно получить, прибавляя друг к другу результаты по месяцам, поскольку месяцы складываются в год. Чтобы использовать сводку, запрос нужно несколько модифицировать, чтобы он ссылался на эту сводку. В некоторых ситуациях это означает, что необходимо изменить код SQL. В качестве альтернативы можно проинформировать пользователей о существовании сводных данных и обучить их использовать эти таблицы в конкретных запросах. Таким образом, хотя сводные таблицы и улучшают производительность обработки запросов, управление этими сводками может представлять собой сложную задачу:

Когда в хранилище загружаются новые данные, сводка оказывается не синхронизирована с таблицами данных. Когда такое происходит, говорят, что сводка устарела (stale). В табл. 4.2 показана устаревшая сводка. Чтобы сводные данные соответствовали подробным данным, устаревшую сводку нужно обновить (refresh). Сводку можно перестроить при загрузке в хранилище новых данных. Это называется полным обновлением (complete refresh). В некоторых случаях возможно провести инкрементное обновление (incremental refresh), касающееся только новых или изменившихся данных. До появления Управления сводными данными в Oracle 8i обновление сводок осуществлялось с помощью сложных, индивидуально настраиваемых процедур.

Таблица 4.2. *Устаревшая сводка*

Данные из системы управления транзакциями

Дата	Покупатель	Продукт
1/5/02	Smith	Книга
1/6/02	Jones	Палатка
2/5/02	Smith	Книга
2/5/02	Mills	Книга
3/5/02	Smith	Пленка
3/7/02	Mills	Книга
3/5/02.	Smith	Пленка
4/1/02	Smith	Книга
4/2/02	Jones	Книга
4/3/02	Smith	Палатка
4/3/02	Jones	Палатка

Объем продаж по продуктам и месяцам

Месяц	Продукт	Продано
1/02	Книга	1
1/02	Палатка	1
2/02	Книга	2
3/02	Книга	1
3/02	Пленка	2
4/02	Книга	2
4/02	Палатка	2
5/02	Книга	2
6/02	Книга	1
6/02	Палатка	3

Продолжение таблицы 4.2

Данные из системы управления транзакциями			
Дата	Покупатель	Продукт	
5/5/02	Peters	Книга	
5/6/02	Smith	Книга	
6/1/02	Jones	Книга	
6/5/02	Smith	Палатка	
6/6/02	Jones	Палатка	
6/5/02	David	Палатка	
7/1/02	Smith	Книга	Новые данные, не входящие в сводку
7/2/02	Jones	Книга	
7/1/02	David	Палатка	

Решение вопроса о том, какие создавать сводки, требует знания поля деятельности. Какие типы вопросов задают пользователи? Как часто требуется одна и та же информация? Число возможных сводных таблиц очень велико. Поскольку сводки занимают место на диске и на их обновление требуется время, важно выбрать несколько сводок, дающих максимальный выигрыш производительности.

В заключение, перед тем как использовать сводки, нужно ответить на следующие вопросы.

1. Какой набор сводок будет наилучшим?
2. Нужно ли пользователям знать о существовании сводок? Если нужно, то как пользователи узнают, какие имеются сводки и когда их можно использовать? Если позже выяснится, что сводка не столь уж полезна, и вы отбросите ее, пользователи должны также знать и об этом.
3. Как будет поддерживаться соответствие сводок и данных при изменении последних?

Управление сводными данными (Summary Management) от Oracle имеет ответы на все эти вопросы. С помощью этого средства сводки можно создавать прямо в базе данных Oracle. Предоставляются механизмы поддержания соответствия сводок и данных. Более того, база Oracle явным образом переписывает ваши запросы для использования сводок, поэтому пользователям не нужно ничего знать о сводках.

4.1.2. Управление сводными данными в базе Oracle

В функцию Управления сводными данными (Summary Management) в Oracle входят следующие компоненты:

- сущность, называющаяся «материализованное представление» (Materialized View), которая, по сути, представляет собой сводную таблицу;
- функция «переписывание запросов» (Query Rewrite), которая явным образом переписывает запросы SQL под использование материализованных представлений;
- механизм обновления сводок, использующий либо полное, либо инкрементное обновление;
- Summary Advisor (Консультант по сводкам), дающий рекомендации по созданию сводок;
- измерения (Dimensions), дающие возможность объявлять иерархические связи, такие, как вложения уровней данных (rollup), помогающие переписывать запросы.

С помощью Summary Advisor вы сможете легко определить, какие материализованные представления можно создать для данного набора запросов, чтобы уложиться в заданный объем дискового пространства. Как только сводки созданы и сделаны доступными для переписывания запросов, запросы автоматически начинают использовать эти сводки. Значительное преимущество такого подхода в том, что конечным пользователям базы и приложениям, использующим базу, больше не нужно знать о существовании сводок. Многие средства работы с запросами, такие, как DSS Agent от Microstrategy или Decision Suite от Advantage, также предоставляют некоторые возможности для переписывания запросов (известные также как агрегатная навигация (aggregate navigation)). Однако, в отличие от этих инструментов, переписывание запросов в Oracle имеет общий характер и не ограничивается звездообразной схемой или запросами, включающими агрегацию. Любое клиентское средство может воспользоваться этой функцией на сервере и обеспечить все возможности агрегатной навигации. Например, Oracle Discoverer использует Управление сводными данными для улучшения времени обработки запроса.

Управление сводными данными также предоставляет процедуры для полного и быстрого обновления сводок при загрузке новых данных в хранилище. Это устраняет необходимость писать сложные программы инкрементного обновления.

Теперь мы рассмотрим все аспекты Управления сводными данными более подробно.

4.2. Создание материализованного представления

В Oracle сводки создаются с помощью объекта, который называется материализованное представление (materialized view, MV). Материализованное представление заранее вычисляет и сохраняет результат SQL-запроса. Заметим, что сводка, как правило, представляет собой ответ на агрегирующий запрос. Однако материализованное представление можно создать для любого запроса. В этом отношении оно очень похоже на обыкновенное. Различие в том, что, материализуя представление, вы сохраняете результаты в базе данных. В оставшейся части главы мы будем использовать вместо термина «сводка» термин «материализованное представление».

Совет. Материализованные представления также можно использовать для репликации данных. До Oracle 8i такие материализованные представления назывались моментальными снимками (Snapshot). Материализованные представления включают в себя все свойства моментальных снимков и еще дополнительные возможности. В этой книге мы сконцентрируемся только на приложении материализованных представлений к хранилищам данных.

Материализованное представление создается с помощью выражения CREATE MATERIALIZED VIEW. В приведенном примере создается материализованное представление MONTHLY_SALES_MV, содержащее общие объемы продаж каждого продукта за каждый месяц.

```
CREATE MATERIALIZED VIEW MONTHLY_SALES_MV
PCTFREE 0 TABLESPACE mview                  <- параметры хранения
STORAGE  (initial  64k
   next  64k pctincrease  0)
BUILD IMMEDIATE                          <- когда нужно заполнять
REFRESH FORCE                         <- как проводить обновление
ON DEMAND                                  <- когда обновлять
ENABLE QUERY REWRITE <- использовать ли переписывание запросов
AS
SELECT t.month, t.year, p.product_id,  <- что будет содержать
     SUM (f.purchase_price) as sum_of_sales,
     COUNT (f.purchase_price) as total_sales,
     COUNT(*) as cstar
```

```
FROM time t, product p, purchases f
WHERE t.time_key = f.purchase_date AND
      f.product_id = p.product_id
GROUP BY t.month, t.year, p.product_id;
```

Определение материализованного представления содержит следующие части (указанные в примере стрелками).

- Описание того, как физически хранится материализованное представление (т. е. условие storage);
- Когда нужно заполнять материализованное представление сразу после создания или позже и существует ли сводная таблица, которую можно преобразовать в материализованное представление?
- Как проводить обновление, когда данные в соответствующих таблицах изменяются?
- Когда проводить обновление: в конце каждой транзакции или по явному требованию?
- Использовать представление для переписывания запросов или нет?
- Оператор SELECT, описывающий содержимое материализованного представления.

На рис. 4.1 показано материализованное представление, создаваемое в Oracle Enterprise Manager. Все опции выражения CREATE MATERIALIZED VIEW можно указать, используя графический интерфейс. Существует также возможность сбора статистических данных о материализованном представлении при его создании. Можно использовать кнопку Explain (Объяснить) для запуска утилиты EXPLAIN_MVIEW, которая помогает определить возможности материализованного представления. За информацией о EXPLAIN_MVIEW обращайтесь к разделу 4.4.

Теперь мы дадим объяснения по каждому условию выражения, создающего материализованное представление.

4.2.1. Физическое хранение материализованного представления

Материализованное представление имеет спецификацию хранения, в которой можно указать табличное пространство для хранения данных, первоначальный объем выделенного дискового пространства и максимум его расширения.

В нашем примере материализованное представление помещается в табличное пространство с именем «MVIEW», первый экстент 64kБ и все последующие экстенты тоже 64кБ

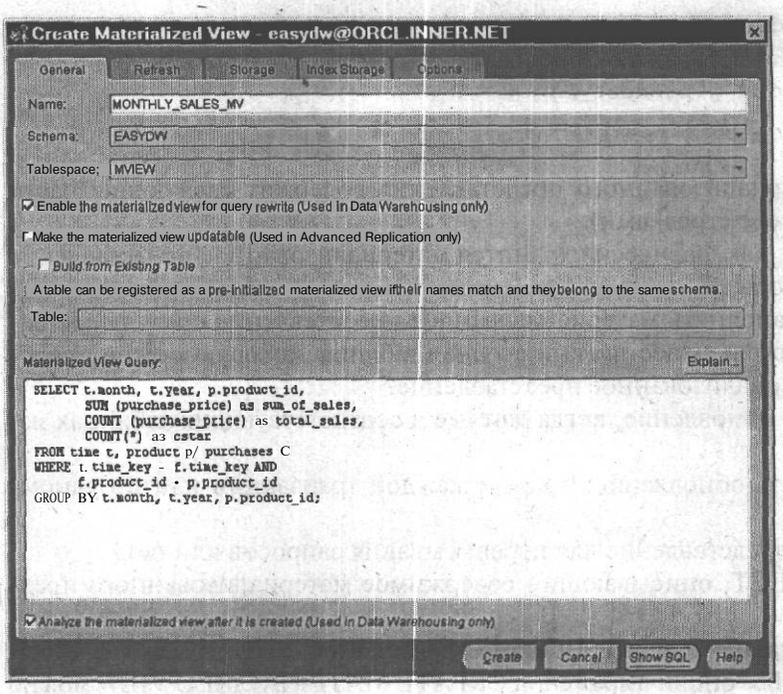


Рис. 4.1. *Создание материализованного представления в Oracle Enterprise Manager*

4.2.2. Когда материализованное представление следует заполнять данными?

В определении материализованного представления указывается, когда следует заполнять материализованное представление данными. Если вы укажете условие BUILD IMMEDIATE, как это сделано в приведенном фрагменте кода, материализованное представление будет заполняться сразу после создания. Если вы укажете BUILD DEFERRED, материализованное представление будет заполняться при выполнении операции REFRESH. Если у вас уже имеется сводная таблица, которой вы хотите управлять при помощи системы Oracle Summary Management, можно использовать условие ON PREBUILT TABLE. При этом существующая таблица будет преобразована в материализованное представление и сделана доступной для переписывания запросов (если это необходимо).

Совет. Материализованное представление не принимается в расчет системой переписывания запросов, пока оно не будет заполнено данными.

4.2.3. Как следует проводить обновление материализованного представления?

По мере загрузки в хранилище новых данных материализованные представления приходится обновлять, чтобы они отражали внесенные изменения. Имеется четыре варианта обновления:

- полное (Complete);
- быстрое (Fast);
- форсированное (Force);
- отсутствие обновления (Never).

Материализованное представление можно полностью перестроить, указав условие REFRESH COMPLETE. Или же можно провести инкрементное обновление, указав вариант REFRESH FAST. Быстрое обновление выполняется быстрее полного, однако быстрое обновление не всегда возможно.

В приведенном выше примере использовался вариант REFRESH FORCE, что означает, что база Oracle будет выполнять быстрое обновление, если это возможно, и полное обновление, если это необходимо. Материализованные представления, в которых используется вариант NEVER REFRESH, не будут обновляться ни одной из процедур, которые применяются в базе Oracle. Этот вариант полезен, если у вас имеются традиционные процедуры обновления, которые нельзя удалить, или если вы хотите хранить исторические сведения, которые нельзя обновлять.

4.2.4. Когда следует обновлять материализованные представления?

Когда основные данные в таблицах изменяются, базирующиеся на них материализованные представления устаревают и больше не отражают результата суммирования всех основных данных. Наиболее обычные способы обновления называются ON COMMIT и ON DEMAND.

Если ваш бизнес таков, что ему необходимо соответствие материализованного представления и данных на уровне транзакций, материализованное представление можно обновлять в конце каждой транзакции, указав условие ON COMMIT.

С другой стороны, при условии ON DEMAND вы должны обновлять материализованное представление вручную. Обновление с условием ON DEMAND позволяет контролировать момент обновления представления. Этот способ полезен в хранилищах, в которые данные загружаются пакетами, после чего необходимо провести обновление материализованного представления.

Также можно задать периодическое обновление материализованных представлений, указав начальную дату и последующие интервалы с помощью условий START WITH и NEXT.

Функции для обновления материализованных представлений обсуждаются в разделе 4.3.

4.2.5. Доступность материализованных представлений для переписывания запросов

В большинстве случаев нужно, чтобы запросы явным образом переписывались под использование материализованных представлений. При определении материализованного представления, если вы хотите сделать его доступным для переписывания запросов, используйте условие ENABLE QUERY REWRITE, как показано в приведенном примере. Если вы не хотите использовать материализованное представление в переписывании запросов, применяйте условие DISABLE QUERY REWRITE. Если у вас имеются материализованные представления, и вы хотите, чтобы база Oracle проводила их обновление, но не делала их доступными для функции переписывания запросов, укажите условие DISABLE QUERY REWRITE в определении материализованного представления.

Подробно использование переписывания запросов обсуждается в разделе 4.6.

4.2.6. Определение содержимого материализованного представления

В определение материализованного представления входит инструкция SELECT, описывающая его содержимое. Типичный запрос материализованного представления может включать в себя что-нибудь из нижеперечисленного.

- Условие WHERE, соединяющее таблицу фактов и одну или более таблиц измерений. В нашем примере таблица фактов PURCHASES соединяется с таблицами измерений TIME и PRODUCTS. Здесь можно сделать выборки, чтобы ограничить объем данных в материализованном представлении.
- Один или более оператор агрегирования. В нашем примере для получения общих объемов продаж мы используем оператор COUNT. Можно включать операторы SUM, MIN, MAX, AVG, COUNT(*), COUNT, COUNT(DISTINCT x), VARIANCE и STDDEV. В материализованном представлении также можно использовать аналитические функции и рамочные агрегаты, которые обсуждались в главе 3.
- Условие GROUP BY. В приведенном примере мы вычисляем общее число проданных товаров по столбцам года, месяца и идентификатора продукта.

Материализованное представление из предыдущего примера включает агрегацию и поэтому может быть названо материализованным агрегирующим представлением (materialized aggregate view).

Материализованное представление также можно использовать для соединения двух и более таблиц без всякой агрегации. Такой тип материализованного представления называется материализованным соединительным представлением (materialized join view, MJV) и используется для предварительного вычисления ресурсоемких соединений. Материализованное соединительное представление можно использовать вместо соединительного индекса на основе битовых карт. Такой индекс более приспособлен к звездообразным запросам и будет работать лучше, чем MJV, если вы хотите вычислить комбинации выборок по ключам индекса. С другой стороны, материализованные соединительные представления можно использовать при обработке более широкого класса запросов с применением переписывания запросов. Однако, как правило, MJV намного больше соединительных индексов на основе битовых карт.

В следующем примере показано материализованное соединительное представление, используемое для вычисления соединения таблиц PURCHASES и CUSTOMER.

```
CREATE MATERIALIZED VIEW customer_purchases_mv
BUILD IMMEDIATE
REFRESH COMPLETE
AS
SELECT c.sex, c.occupation, f.purchase_price
FROM purchases f, customer c
WHERE f.customer_id = c.customer_id
```

В примерах этой главы акцент будет делаться на этих двух типах материализованных представлений.

4.2.7. Использование управления сводными данными применительно к имеющимся сводным таблицам

Если у вас уже есть хранилище данных со сводками, хранящимися в пользовательских таблицах, и вы не хотите полностью пересоздавать их заново, вы можете использовать при управлении ими систему Summary Management и осуществлять переписывание запросов.

Чтобы зарегистрировать свою сводную таблицу в системе управления сводными данными, создайте материализованное представление, включив туда условие ON PREBUILT TABLE. При этом существующая таблица будет преобразована в материализованное представление. Если вы укажете условие ENABLE QUERY REWRITE, материализованное представление можно будет использовать при переписывании запросов. Например, у вас, как показано в приводимом запросе, может быть таблица с именем monthly_customer_sales, суммирующая все деньги, которые каждый покупатель тратит ежемесячно.

```
SELECT t.year, t.month, c.customer_id,
     SUM(f.purchase_price) as dollar_sales
FROM time t, purchases f, customer c
WHERE f.time_key = t.time_key AND
     f.customer_id = c.customer_id
GROUP BY t.year, t.month, c.customer_id;
Таблица monthly_customer_sales описывается следующим образом:
SQL> describe monthly_customer_sales;

Name                                      Null?        Type
----------------------------------------  --------     ----------
YEAR                                                   NUMBER(4)
MONTH                                                  NUMBER(2)
CUSTOMER_ID                                            VARCHAR2(10)
DOLLAR_SALES                                           NUMBER
```

Если вы хотите использовать эту таблицу в качестве материализованного представления и сделать ее доступной для функции переписывания запросов, можно сделать следующий запрос. Заметим, что при создании материализованного представления по имеющейся таблице материализованное представление должно иметь то же имя, что и таблица.

```
CREATE MATERIALIZED VIEW monthly_customer_sales
ON PREBUILT TABLE
ENABLE QUERY REWRITE
AS
SELECT t.year, t.month, c.customer_id,
     SUM(f.purchase_price) AS dollar_sales
FROM time t, purchases f, customer c
WHERE f.time_key = t.time_key AND
     f.customer_id = c.customer_id
GROUP BY t.year, t.month, c.customer_id;
```

Теперь это материализованное представление может использоваться при явном переписывании запросов и обновляться процедурами базы Oracle.

Перед тем как создавать материализованное представление с помощью условия PREBUILT, мы должны предупредить: база Oracle не проверяет соответствие имеющейся таблицы запросу, используемому для создания представления. Вы должны обеспечить корректное представление суммы подробных данных в содержимом вашей таблицы. В противном случае результаты переписывания запросов могут оказаться неправильными.

Совет. Чтобы функция переписывания запросов использовала материализованное представление с условием PREBUILT, нужно установить инициализационный параметр QUERY_REWRITE_INTEGRITY в значение TRUSTED или STALE_TOLERATED, как это объясняется в разделе 4.6.8.

4.2.8. Секционирование материализованного представления

Вы можете осуществлять секционирование материализованного представления, как и любой другой таблицы, с помощью любого имеющегося в базе

Oracle метода. В хранилищах типичным является секционировать материализованное представление так же, как таблицу фактов. Так же хорошо использовать для секционирования материализованных представлений линейно-списочное секционирование, когда представление содержит много измерений, таких, как время (линейное) и географическое положение (списочное).

В следующем примере материализованное представление секционируется с помощью линейного секционирования.

```
CREATE MATERIALIZED VIEW MONTHLY_SALES_MV
PARTITION by RANGE (time_key)
(  partition purchases_jan2002
    values less than (TO_DATE('01-02-2002', 'DD-MM-YYYY'))
    pctfree 0 pctused 99
    storage (initial 64k next 64k pctincrease 0)
    tablespace purchases_jan2002,
   partition purchases_feb2002
    values less than (TO_DATE ('01-03-2002', 'DD-MM-YYYY'))
    pctfree 0 pctused 99
    storage (initial 64k next 64k pctincrease 0)
    tablespace purchases_feb2002,
   partition purchases_mar2002
    values less than (TO_DATE('01-04-2002', 'DD-MM-YYYY'))
    pctfree 0 pctused 99
    storage (initial 64k next 64k pctincrease 0)
    tablespace purchases_mar2002
)
BUILD IMMEDIATE
REFRESH FORCE
ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT t.time_key, p.product_id, SUM (f.purchase_price) as
sum_of_sales
FROM time t, product p, purchases f
WHERE t.time_key = f.purchase_date AND
      f.product_id = p.product_id
GROUP BY t.time_key, p.product_id;
```

Секционирование материализованного представления может ускорить процесс обновления, если использовать параллельное выполнение команд DML. Это также позволяет использовать в запросах с материализованными представлениями преимущества оптимизации, такие, как отсечение разделов.

4.2.9. Индексирование материализованного представления

По материализованному представлению можно также создавать индексы, улучшая производительность выполнения запросов. Методы, используемые для индексирования материализованного представления, похожи на методы, применяемые для любой другой таблицы.

В следующем примере создается сцепленный индекс по группируемым столбцам материализованного представления MONTHLY_SALES_MV, показанного выше в этой главе. Также по группируемым столбцам создаются индексы на основе битовых карт. Эти индексы повысят производительность обработки запросов, переписанных под использование материализованного представления.

```
CREATE INDEX easydw.products_by_month_concat_index
ON MONTHLY_SALES_MV (month, year, product_id)
pctfree 5
tablespace indx
storage (initial 64k next 64k pctincrease 0) ;

CREATE BITMAP INDEX easydw.total_products_by_month_index
ON monthly_sales_mv (month, year);

CREATE BITMAP INDEX easydw.total_products_by_id_index
ON monthly_sales_mv (product_id);
```

База Oracle автоматически создает индекс по некоторым материализованным представлениям для ускорения операции быстрого обновления. Чтобы отменить создание этого индекса, используйте условие NO INDEX в выражении CREATE MATERIALIZED VIEW.

4.2.10. Безопасность материализованных представлений

Доступ к определенной информации в хранилище может быть ограничен, и важно обеспечить соответствующую политику безопасности и в отношении материализованных представлений. Может понадобиться разрешить пользователям доступ к материализованным представлениям, но не позволять им увидеть основные данные. Например, вы можете разрешить пользователю посмотреть среднюю зарплату по подразделению, но не разрешать видеть зарплаты отдельных служащих. С другой стороны, вам нужно сделать так, чтобы материализованные представления и переписывание запросов не использовались в качестве механизма обхода системы безопасности. Как и для других объектов базы, Oracle предлагает модель привилегий для создания и изменения материализованных представлений.

Чтобы пользователь мог создать материализованное представление в своей собственной схеме, он должен иметь привилегию CREATE MATERIALIZED VIEW. Чтобы создать материализованное представление в схеме, другого пользователя, нужно иметь привилегию CREATE ANY MATERIALIZED VIEW. Владелец схемы, в которой будет размещаться материализованное представление, должен иметь привилегии CREATE TABLE и CREATE INDEX и иметь возможность выполнить запрос, определяющий материализованное представление.

Совет. Полезно помнить, что наибольшие требования по привилегиям накладываются на владельца схемы, в которой размещается материализованное представление. Тому, кто создает представление нужно иметь только привилегию CREATE/ALTER MATERIALIZED VIEW (или CREATE/ALTER ANY). Кроме того, проверка привилегий производится при создании или изменении материализованного представления, но не при переписывании запросов.

Чтобы сделать материализованное представление доступным для переписывания запросов, владелец материализованного представления должен иметь объектную привилегию QUERY REWRITE для всех таблиц (упомянутых в материализованном представлении), не входящих в его собственную схему. В качестве альтернативы владелец должен иметь системную привилегию GLOBAL QUERY REWRITE (отметим, что системная привилегия QUERY REWRITE, которая была в Oracle 8i, теперь убрана).

Чтобы обновить материализованное представление в другой схеме, пользователь должен иметь привилегию ALTER ANY MATERIALIZED VIEW.

Чтобы создать материализованное представление с условием REFRESH ON COMMIT, владелец должен иметь объектные привилегии ON COMMIT для всех таблиц (упомянутых *в* материализованном представлении), не входящих в его схему, или же системную привилегию ON COMMIT.

В следующем примере пользователю EASYDW даются привилегии для создания материализованных представлений и измерений и для выдачи разрешения использовать их в переписывании запросов.

```
-- добавить привилегии
GRANT select any table to easydw;
GRANT execute any procedure to easydw;

-- добавить привилегии для управления сводными данными
GRANT create materialized view to easydw;
GRANT drop materialized view to easydw;
GRANT alter materialized view to easydw;
GRANT create any dimension to easydw;
GRANT alter any dimension to easydw;
GRANT drop any dimension to easydw;
GRANT global query rewrite to easydw;
```

4.3. Обновление

Когда в хранилище загружаются новые данные, все материализованные представления, основывающиеся на этих данных, требуется привести в соответствие с ними. Эта операция называется обновлением (refresh). Частота обновлений определяет то, насколько синхронизированы будут сводные и оригинальные данные. Операция обновления может занять очень много времени, в зависимости от объема данных. Вы должны аккуратно определять свою политику проведения обновлений, основываясь на имеющихся ресурсах и необходимости иметь самые последние данные. Если вашему приложению нужно, чтобы материализованное представление всегда было синхронизировано с основными данными, выберите условие ON COMMIT. При этом условии база Oracle автоматически обновит материализованное представление при совершении изменения основных данных. Если ваше хранилище получает новые данные один раз в день, в специально отведенное для этого время, правильным выбором будет отсроченное обновление

(deferring refresh). Отсроченное обновление или обновление по требованию (ON DEMAND) можно выполнить, используя пакет DBMS_MVIEW.

Политику проведения обновлений нужно определить при создании материализованного представления. Позже вы сможете изменить ее с помощью выражения ALTER MATERIALIZED VIEW.

4.3.1. Использование процедур обновления из пакета DBMS_MVIEW

Oracle предлагает несколько процедур для проведения обновлений по требованию (ON DEMAND), поставляемых в PL/SQL-пакете DBMS_MVIEW. Эти процедуры таковы:

- DBMS_MVIEW.REFRESH
- DBMS_MVIEW.REFRESH_DEPENDENT
- DBMS_MVIEW.REFRESH_ALL_MVIEWS\

Если вы хотите обновить все свои материализованные представления, что обычно осуществляется после загрузки массива новых данных, вы можете использовать процедуру DBMS_MVIEW.REFRESH_ALL_MVIEWS, как это показано в следующем примере. Здесь переменная связи :failures возвращает количество неудачных обновлений.

```
EXECUTE DBMS_MVIEW.REFRESH_ALL_MVIEWS(:failures);
```

Если у вас есть материализованные представления, обновляемые в разное время, например, одни — еженедельно, а другие — ежемесячно, вы можете указать список обновляемых с помощью процедуры DBMS_MVIEW.REFRESH материализованных представлений следующим образом:

```
EXECUTE DBMS_MVIEW.REFRESH('monthly_sales_mv, sales_by_city');
```

Вспомните, что при создании материализованного представления вы указываете метод обновления, используемый по умолчанию, — COMPLETE, FAST или FORCE. Вы можете переписывать этот метод, подавая команду обновления — C (complete), F (fast) или ? (force). Если иное не указано, будет использоваться метод, заданный в момент создания материализованного представления. В следующем примере материализованное представление MONTHLY_SALES_MV, для которого ранее определялось условие REFRESH FORCE, обновляется с помощью метода COMPLETE.

```
EXECUTE DBMS_MVIEW.REFRESH('monthly_sales_mv', 'C');
```

Если вы хотите обновить все материализованные представления, основывающиеся на конкретной таблице, содержимое которой изменилось, используйте процедуру DBMS_MVIEW.REFRESH_DEPENDENT, указав таблицу данных. Например, для форсированного (FORCE) обновления всех материализованных представлений, которые изменились после изменения таблицы customer, нужно выполнить следующее выражение:

```
EXECUTE DBMS_MVIEW.REFRESH_DEPENDENT(:failures, 'customer', '?')
```

Обновление нескольких материализованных представлений можно делать одновременно *с* помощью очереди заданий. Для этого нужно установить инициализационный параметр JOB_QUEUE_PROCESSES в ненулевое значение, а параметр atomic_refresh в процедуре обновления — в FALSE следующим образом:

```
ALTER SYSTEM SET job_queue_processes = 2;

EXECUTE DBMS_MVIEW.REFRESH('sales_by_month, sales_by_city', 'C',
        atomic_refresh=>FALSE);
```

В качестве альтернативы вы можете использовать для обновления сводок Oracle Enterprise Manager. Чтобы увидеть свойства всех материализованных представлений в системе, раскройте пункт Databases (Базы данных) в панели Navigator, потом — пункт Warehouse (Хранилище), потом — Summary Management (Управление сводными данными) и наконец — Materialized Views (Материализованные представления) (как показано на рис. 4.2).

Вы можете узнать, не является ли материализованное представление устаревшим (STALE), взглянув на пункт Refresh State (Состояние обновления) в разделе Status (Статус). Если материализованное представление устарело, вы можете обновить его, нажав кнопку Refresh (Обновить).

Совет. Чтобы вы получили точную картину состояния обновления, в поле состояния Compile State должно быть слово VALID. Если в нем указано NEEDS_COMPILE, сначала нажмите кнопку Compile (Собрать), чтобы проверить материализованное представление.

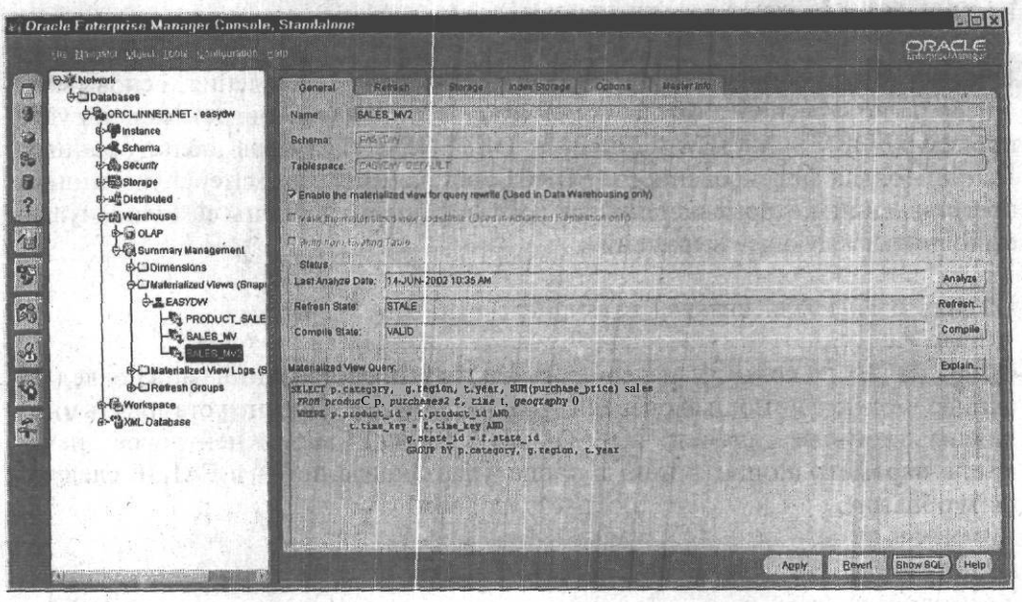


Рис. 4.2. *Обновление материализованного представления в Oracle Enterprise Manager*

После нажатия кнопки Refresh (Обновить) появится окно опций обновления (показанное на рис. 4.3), в котором вы можете выбрать тип обновления, выполнить обновление и, по желанию, обновить статистику по материализованному представлению.

После завершения обновления вы можете снова проверить состояние материализованного представления в разделе Status (Статус).

4.3.2. Быстрое обновление

По мере того как растет количество данных в хранилище, перестройка материализованного представления после каждой новой загрузки данных становится чрезмерно ресурсоемкой. Oracle предлагает возможность обновлять материализованное представление без полной перестройки. Это называется быстрым обновлением. Используемые в базе Oracle алгоритмы либо инкрементно вносят изменения в соответствии с содержанием основных данных, либо избирательно пересчитывают только те части материализованных представлений, которые

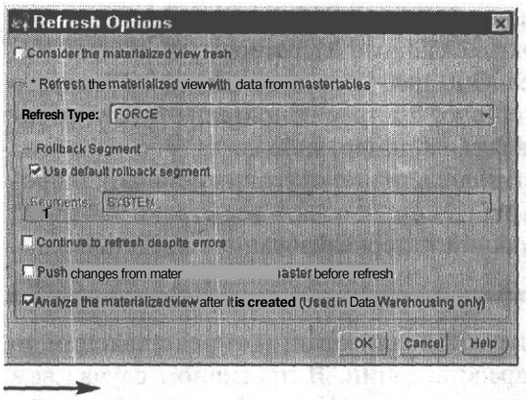


Рис. 4.3. *Обновление материализованного представления в Oracle Enterprise Manager — Опции*

были затронуты изменениями. В Oracle 8i поддерживалось быстрое обновление материализованных агрегирующих представлений после загрузки массивов данных. Обновление после выполнения команд DML поддерживалось только для ограниченного класса материализованных представлений. В Oracle 9i быстрое обновление поддерживает обычные операции DML (команды вставки, изменения и удаления), а также операции, связанные с обслуживанием разделов.

Чтобы материализованное представление можно было быстро приводить в соответствие с данными после выполнения команд DML, по каждой таблице нужно создать протокол материализованного представления (materialized view log). В этом протоколе фиксируются все изменения, которые вносятся в таблицы данных операциями вставки, изменения и удаления. Отметим, что изменения, вносимые при загрузке массивов (bulk load), например, командами INSERT /*+APPEND*/, или при использовании SQL*Loader, не записываются в протокол материализованного представления, а хранятся во внутреннем протоколе.

Чтобы для материализованного представления можно было применять быстрое обновление, оно должно соответствовать определенным правилам. В оператор SELECT материализованного представления должны входить некоторые агрегаты, поддерживающие используемый агрегат. Так, на каждый агрегат типа SUM(x) или AVG(x) должен приходиться агрегат COUNT(x). Если у вас есть агрегат STDDEV(x), должны присутствовать SUM(x), COUNT(x) и SUM(x*x).

В приведенном примере показана сводка с возможностью быстрого обновления, а также протоколы материализованного представления по ее таблицам данных. Отметьте, что для протокола материализованного представления необходимо указывать различные опции, такие, как ROWID, SEQUENCE и INCLUDING NEWVALUES. Опция ROWID вносит в протокол идентификаторы строк, которые

были изменены операторами DML. Условие INCLUDING NEWVALUES вносит в протокол старую и новую версию строки, созданную оператором изменения. Условие SEQUENCE появилось в Oracle 9i. Оно требуется для поддержки быстрого обновления. Если вы не включите эту опцию, быстрое обновление будет проводиться только при специфических условиях (например, либо когда была изменена только одна таблица, либо когда осуществлялись только операции вставки). В протокол материализованного представления нужно включать все строки таблицы, на которые ссылается запрос, формирующий материализованное представление.

Совет. Вы должны создавать протоколы материализованных представлений до создания самих материализованных представлений. В противном случае вам придется полностью обновлять материализованные представления перед тем, как можно будет выполнять быстрое обновление.

```
CREATE MATERIALIZED VIEW LOG on purchases
   WITH ROWID, SEQUENCE (purchase_date, product_id, purchase_price)
   INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG on product
 WITH ROWID, SEQUENCE (product_id)
INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW monthly_sales_mv
   PCTFREE 0 TABLESPACE summary
   STORAGE (initial 64k next 64k pctincrease 0)
   BUILD IMMEDIATE
   REFRESH FAST
   ON DEMAND
   ENABLE QUERY REWRITE
   AS
   SELECT t.month, t.year, p.product_id,
        SUM (ps.purchase_price) as sum_of_sales,
        COUNT (ps.purchase_price) as total_sales, COUNT(*)
   FROM time t, product p, purchases ps
   WHERE t.time_key = ps.purchase_date AND
        ps.product_id = p.product_id
   GROUP BY t.month, t.year, p.product_id;
```

Вы можете использовать процедуру EXPLAIN_MVIEW (раздел 4.4) и определить, можно ли использовать данный оператор SELECT для создания материализованного представления с возможностью быстрого обновления.

В следующем примере показано соединительное материализованное представление с возможностью быстрого обновления и условием ON COMMIT. Чтобы сделать материализованное представление такого типа доступным для быстрого обновления, нужно включить в материализованное представление столбец ROWID от каждой таблицы. В протоколе материализованного представления требуется только опция ROWID. (В этом примере мы предполагаем, что имеются в наличии созданные ранее протоколы материализованных представлений для таблиц product и purchases представления MONTHLY_SALES_MV).

```
CREATE MATERIALIZED VIEW LOG on customer WITH ROWID;

CREATE MATERIALIZED VIEW product_customer
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
AS
SELECT c.rowid r1, c.sex, p.rowid r2, p.product_id,
   f.rowid r3, f.purchase_price
FROM purchases f, product p, customer c
WHERE f.customer_id = c.customer_id
   AND f.product_id = p.product_id;
```

Совет. Для всех материализованных представлений, ссылающихся на данную таблицу, используется один и тот же протокол. Поэтому обязательно указывайте опции, удовлетворяющие всем материализованным представлениям.

При быстром обновлении база Oracle по протоколу материализованного представления определит изменения, произошедшие с момента последнего обновления, внесет их в материализованное представление. Как только будут обновлены все материализованные представления, соответствующие строки будут исключены из протокола. Никогда не следует модифицировать протокол материализованного представления вручную, поскольку тогда вам придется полностью перестраивать материализованное представление.

Чтобы добиться хорошей производительности при проведении обновлений, важно иметь точную статистическую информацию о данных. Для сбора

статистики оптимизатора по таблицам данных после каждой загрузки и перед созданием или обновлением материализованных представлений следует использовать утилиту DBMS_STATS. Также важно получить статистическую информацию после создания материализованных представлений или обновления их большими объемами данных. Это обеспечит оптимизатор точной статистикой при обновлениях и оптимизации запросов, переписываемых под использование материализованных представлений.

4.3.3. Быстрое обновление после операций над разделами

Секционирование таблиц — обычное свойство хранилищ. Операции по обслуживанию разделов используются для реорганизации данных в разделах. Сюда входят MERGE (слияние) и SPLIT (разделение) имеющихся разделов, TRUNCATE (усечение) или DROP (удаление) старого раздела и EXCHANGE (обмен между таблицей и разделом). Использование операций обслуживания разделов обсуждается в главе 7.

В Oracle 9i предлагается новое свойство, называющееся «слежение за изменениями разделов» (Partition Change Tracking, PCT). Это свойство позволяет проводить быстрое обновление материализованных представлений для секционированных таблиц после операций над разделами.

Когда выполняется операция над разделами, база Oracle следит за изменениями разделов. Затем она определяет те данные в материализованном представлении, которые соответствуют измененным разделам, и пересчитывает только эту часть представления. Такой подход называется PCT-обновление (PCT-refresh).

На рис. 4.4. показано концептуальное изображение того, как осуществляется отслеживание изменений данных в системе слежения за изменениями разделов. В данном примере таблица покупок секционирована по месяцам на разделы с данными января 2002, февраля 2002 и марта 2002. Материализованное представление MONTHLY_SALES_MVвключает в себя данные по продажам за каждый месяц. Соотношения между основными данными и данными материализованного представления показаны пунктирными линиями. Если изменяется раздел за февраль 2002, соответствующие данные в материализованном представлении оказываются устаревшими и более недоступными для переписывания запросов. Также, когда в таблицу покупок добавляется раздел с данными за апрель 2002, материализованное представление не будет содержать этот раздел и поэтому не может использоваться при обработке запросов к данным за апрель.

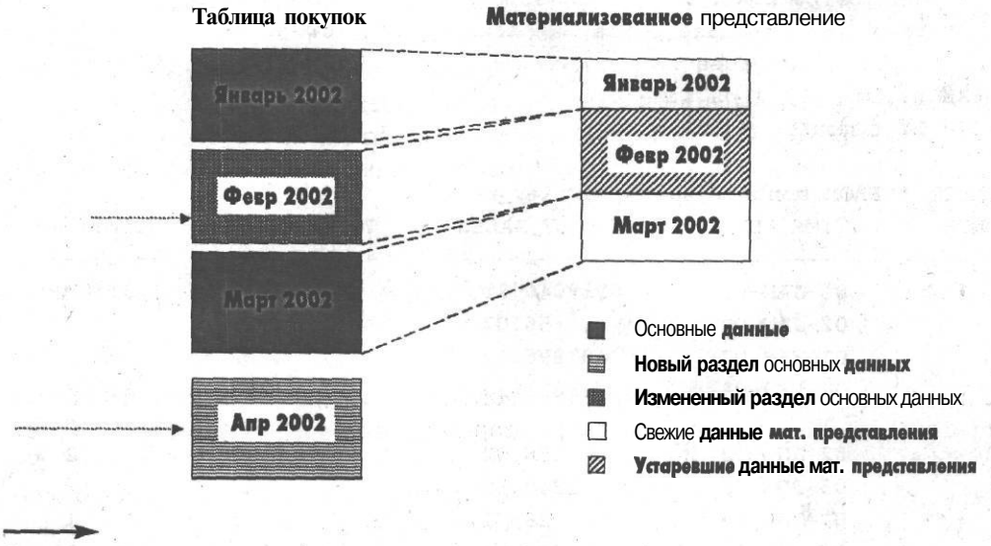


Рис. 4.4. *Слежение за изменениями разделов*

Чтобы материализованное представление было доступно для системы слежения за изменениями разделов, вы должны включить в представление ключ секционирования таблиц основных данных. Если это не представляется возможным, нужно включить в него специальный столбец, который называется маркер разделов (partition marker), для каждой секционированной таблицы с помощью функции DBMS_MVIEW.PMARKER(). Это позволит базе Oracle сопоставить разделы таблицы основных данных со строками материализованного представления. Чтобы пользоваться преимуществами PCT, само материализованное представление даже не нужно секционировать!

Следующий пример иллюстрирует использование ключа секционирования в материализованном представлении. В схеме EASYDW таблица покупок purchases секционирована по ключу времени time_key. Материализованное представление MONTHLY_SALES_PARTKEY_MV в операторе SELECT содержит столбец time_key и, следовательно, будет доступно для системы слежения за изменениями разделов, если таблица покупок окажется изменена.

```
CREATE MATERIALIZED VIEW monthly_sales_partkey_mv
REFRESH FAST
AS
SELECT t.month, ps.time_key,
```

```
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales, COUNT(*)
FROM time t, purchases ps
WHERE t.time_key = ps.time_key
GROUP BY t.month, ps.time_key;

SELECT * FROM monthly_sales_partkey_mv;
```

MONTH	TIME_KEY	SUM_OF_SALES	TOTAL_SALES	COUNT(*)
1	01-JAN-02	8989040.32	31709	31709
1	02-JAN-02	56.02	2	2
2	01-FEB-02	3401896.62	8902	8902
2	02-FEB-02	13811629	31203	31203
...				
7	02-JUL-02	56.02	2	2
7	05-JUL-02	2240.95	45	45
7	07-JUL-02	28.01	1	1
7	08-JUL-02	28.01	1	1

```
19 rows selected.
```

В этом материализованном представлении для каждого значения time_key существует своя строка, и поэтому его можно эффективно сохранять на ежедневной основе и оно может стать весьма большим. Чтобы уменьшить эту проблему, вы можете использовать вместо этого маркер разделов, как показывается в следующем примере.

```
CREATE MATERIALIZED VIEW monthly_sales_marker_mv
REFRESH FAST
AS
SELECT t.month, DBMS_MVIEW.PMARKER(ps.rowid) as pmark,
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales, COUNT(*)
FROM time t, purchases ps
WHERE t.time_key = ps.time_key
GROUP BY t.month, DBMS_MVIEW.PMARKER(ps.rowid) ;
SELECT * FROM monthly_sales_marker_mv;
```

```
MONTH    PMARK      SUM_OF_SALES    TOTAL_SALES    COUNT(*)
-----    ---------- --------------  -----------    ---------
1        29566       8989096.34          31711       31711
2        29567      17213525.6           40105       40105
3        29568       429483.97             952         952
4        29707      4524335.65           10126       10126
MONTH    PMARK      SUM_OF_SALES    TOTAL_SALES    COUNT(*)
-----    ---------- --------------  -----------    ---------
5        29708      1268317.48           3158        3158
6        29709      7092585.46           7995        7995
7        29843         2352.99             49          49
7 rows selected.
```

Мы видим, что маркер разделов создает одно значение для всех строк одного раздела. Это делает материализованное представление более управляемым в смысле размера.

Совет. Включайте в материализованное представление ключ секционирования, если в каждом разделе вариантов значений немного. Если вариантов значений для ключа секционирования, такого как ключ времени, имеется много, то использование маркера секционирования будет способствовать меньшему размеру материализованного представления.

Слежение за изменениями разделов в настоящее время поддерживается только для таблиц с линейным или линейным-хэш секционированием. Хотя PCT-обновление особенно полезно после выполнения операций над разделами, такой тип обновления можно также выполнять после обычных команд DML, и особенно большая выгода будет в том случае, если в данные вносится много изменений. База Oracle будет автоматически пытаться выполнить PCT-обновление с помощью протоколов материализованного представления, если не получается выполнить быстрое обновление.

Совет. Чтобы определить, может ли PCT-обновление применяться в случае вашего материализованного представления, используйте процедуру EXPLAIN_MVIEW, информация о которой приводится в следующем разделе.

Система слежения за изменениями разделов также повышает доступность устаревших материализованных представлений для функции переписывания запросов. Более подробно об этом — в разделе 4.6.

4.4. Утилита EXPLAIN_MVIEW

Как мы пока что видим, чтобы мы могли с максимальной пользой применять материализованные представления, они должны соответствовать определенным правилам. В частности, вам может потребоваться создать протоколы материализованных представлений, чтобы получить возможность быстрого обновления, или добавить маркер секционирования, чтобы пользоваться системой слежения за изменениями разделов. Интерфейс EXPLAIN_MVIEW разработан с целью помочь вам определить, какими будут эти правила. Перед тем как создавать материализованное представление, вы можете запустить формирующий его запрос с помощью утилиты EXPLAIN_MVIEW. Если такие функции, как быстрое обновление, недоступны, EXPLAIN_MVIEW укажет на ошибочную конструкцию или пропущенные столбцы. Вы получаете возможность исправить эти проблемы до того, как потратите драгоценные ресурсы на создание материализованного представления. Эта возможность является новинкой Oracle 9i. Утилита EXPLAIN_MVIEW может также подсказать вам, какие свойства доступны для вашего существующего материализованного представления.

Утилита EXPLAIN_MVIEW вызывается с помощью PL/SQL API DBMS_MVIEW.EXPLAIN_MVIEW и имеет две формы. Вы можете указать имя имеющегося материализованного представления и получить список его возможностей. Кроме того, вы можете подставить инструкцию SELECT, в которой определяется предполагаемое материализованное представление, и EXPLAIN_MVIEW предскажет его возможности. Результаты выполнения утилиты EXPLAIN_MVIEW помещаются в таблицу MV_CAPABILITIES_TABLE, которую нужно создать, запустив скрипт utlxmv.sql, находящийся в директории rdbms/admin.

В следующем примере мы произвольно внесли ошибки в протоколы материализованного представления для иллюстрации применения EXPLAIN_MVIEW. Мы убрали протокол материализованного представления по таблице продуктов (PURCHASES) и удалили из протокола таблицы времени (TIME) условие INCLUDING NEW VALUES.

```
ALTER MATERIALIZED VIEW LOG on TIME EXCLUDING NEW VALUES;

DROP MATERIALIZED VIEW LOG on PRODUCT;
```

```
BEGIN
dbms_mview.explain_mview (
'SELECT t.month, t.year, p.product_id,
    SUM (f.purchase_price) as sum_of_sales,
    COUNT (f.purchase_price) as total_sales,
    COUNT (*) as cstar
 FROM time t, product p, purchases f
 WHERE t.time_key = f.purchase_date AND
    f.product_id = p.product_id
 GROUP BY t.month, t.year, p.product_id' ) ;
END;
/
```

В выходных данных утилиты EXPLAIN_MVIEW перечислены все возможности материализованного представления и указано, доступны они (Y) или нет (N). Затем представлена подробная информация по каждой возможности и причина ее недоступности (если есть).

```
SELECT capability_name, possible p, related_text table,
     msgtxt explanation
FROM MV_CAPABILITIES_TABLE;

CAPABILITY_NAME                P     TABLE    EXPLANATION
------------------------------ -     -------  ---------------
PCT                            N
 REFRESH_COMPLETE               Y
 REFRESH_FAST                   N
 REWRITE                        Y
REWRITE_FULL_TEXT_MATCH         Y
 REWRITE_PARTIAL_TEXT_MATCH     Y
 REWRITE_GENERAL                Y
 PCT_TABLE                      N     TIME     relation is not
                                             a partitioned
                                             table
 PCT_TABLE                      N     PRODUCT  relation is not
                                             a partitioned
                                             table
```

```
CAPABILITY_NAME                P  TABLE      EXPLANATION
------------------------------ -  ---------- -------------------------------
PCT_TABLE                      N  PURCHASES  no partition key or PMARKER
                                             in select list
REFRESH_FAST_AFTER_INSERT      N  PRODUCT    the detail table does not
                                             have a materialized view log
REFRESH_FAST_AFTER_INSERT      N  TIME       mv log must have new values
REFRESH_FAST_AFTER_ONETAB_DML  N             see the reason why
                                             REFRESH_FAST_AFTER_INSERT
                                             is disabled
REFRESH_FAST_AFTER_ANY_DML     N             see the reason why
                                             REFRESH_FAST_AFTER_ONETAB_
                                             DML is disabled
REFRESH_FAST_PCT               N             PCT is not possible on any of
                                             the detail tables in the
                                             materialized view
REWRITE_PCT                    N             PCT is not possible on any of
                                             the detail tables in the
                                             materialized view
```

По этим выходным данным мы можем видеть, что возможность PCT_TABLE для таблицы PURCHASES недоступна, поскольку таблица секционирована, но ключ (или маркер) секционирования не включен в материализованное представление. Таблица TIME не секционирована, и поэтому PCT-обновление не будет доступно при ее изменении.

Быстрое обновление может быть недоступно в одних случаях и доступно в других, поэтому возможности быстрого обновления показаны для трех уровней, так что если недоступен первый, оставшиеся также недоступны и т. д.

- REFRESH_AFTER_INSERT (если в таблицы делались только вставки).
- REFRESH_AFTER_ONETABDML (если изменялась только одна таблица за раз).
- REFRESH_AFTER_ANY_DML (нет ограничений на тип операции DML).

Приведенные здесь выходные данные утилиты EXPLAIN_MVIEW показывают, что в протоколе материализованного представления для таблицы TIME пропущено условие INCLUDING NEW VALUES, и поэтому быстрое обновление с помощью протоколов материализованных представлений невозможно.

Утилита EXPLAIN_MVIEW также указывает, сделано ли доступным переписывание запросов и возможно ли оно в общем режиме или только в режиме совпадения текста (text match mode). •

Совет. Чтобы получить подробную информацию о причинах того, почему определенный запрос не переписывается с использованием определенного материализованного представления, применяйте утилиту EXPLAIN_REWRITE. Она будет обсуждаться ниже в этой главе.

Давайте теперь исправим проблемы в протоколах материализованных представлений и перезапустим утилиту. Теперь можно видеть, что быстрое обновление возможно (на это указывает запись REFRESH_FAST_AFTER_INSERTY).

```
CREATE MATERIALIZED VIEW LOG on product
   WITH ROWID, SEQUENCE (product_id)
INCLUDING NEW VALUES;

ALTER MATERIALIZED VIEW LOG on TIME including new values;

BEGIN
dbms_mview.explain_mview (
 'SELECT t.month, t.year, p.product_id,
     SUM (f.purchase_price) as sum_of_sales,
     COUNT (f.purchase_price) as total_sales,
     COUNT(*) as cstar
  FROM time t, product p, purchases f
  WHERE t.time_key = f.purchase_date AND
     f.product_id = p.product_id
```

```
    GROUP BY t.month, t.year, p.product_id');
END;
/
SELECT capability, possible
FROM MV_CAPABILITIES_TABLE
WHERE capability_name LIKE 'REFRESH%';

CAPABILITY_                    P
------------------------------ -
REFRESH_COMPLETE               Y
REFRESH_FAST                   Y
REFRESH_FAST_AFTER_INSERT      Y
REFRESH_FAST_AFTER_ONETAB_DML  Y
REFRESH_FAST_AFTER_ANY_DML     Y
REFRESH_FAST_PCT               N
```

Если бы вы хотели взглянуть на возможности существующего материализованного представления, например MONTHLY_SALES_MV, нужно было бы использовать утилиту EXPLAIN_MVIEW следующим образом:

```
EXECUTE DBMS_MVIEW.EXPLAIN_MVIEW('EASYDW.MONTHLY_SALES_MV');
```

Утилита EXPLAIN_MVIEW также имеет интерфейс для возврата результатов в виде массива PL/SQL, а не таблицы.

Окно CREATE MATERIALIZED VIEW в Oracle Enterprise Manager (рис. 4.1) имеет кнопку Explain (Объяснить), нажав которую, можно запустить утилиту EXPLAIN_MVIEW. Это позволяет немедленно выявлять и исправлять любые проблемы в материализованном представлении. На рис. 4.5 показано окно Oracle Enterprise Manager с выходными данными утилиты EXPLAIN_MVIEW.

Вы также можете запустить утилиту EXPLAIN_MVIEW с помощью кнопки Explain применительно к существующему материализованному представлению (как показано на рис. 4.2).

4.5. Измерения

В Oracle 8i, помимо определения взаимосвязей между таблицами по принципу первичный-внешний ключ, вы можете описьшать взаимоотношения столбцов с одной или более таблицами путем создания нового объекта схемы, называемого измере-

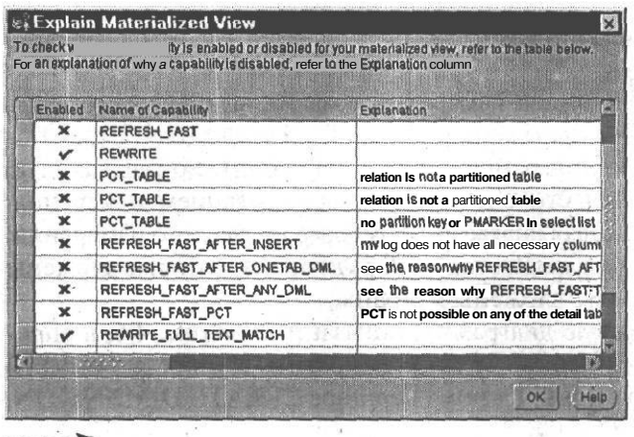


Рис. 4.5. *EXPLAIN_MVIEW в Oracle Enterprise Manager*

нием (dimension). Если указать в базе Oracle такие взаимоотношения данных, то консультант (advisor) сможет порекомендовать, какой набор сводок лучше создать, оптимизатор сможет переписывать больше запросов с использованием этих сводок, а средства OLAP смогут осуществлять операции вложения уровней данных (rollup) и развертывания уровней (drill-down). Мы познакомимся с возможностями измерений в разделе *4.6,* в котором речь будет идти о переписывании запросов.

Как правило, взаимоотношения между таблицами фактов и измерений указываются с помощью ограничений целостности ссылочных данных. Измерения используются при определении следующих типов взаимоотношений.

- Иерархические взаимоотношения данных внутри таблиц измерений. Это позволяет Oracle иметь информацию о том, какие данные можно вкладывать в другие, более высокие уровни иерархии для суммирования и как можно разворачивать уровни иерархии, чтобы увидеть более подробные данные. Иерархические взаимоотношения описываются с помощью условий HIERARCHY в выражении CREATE DIMENSION.
- Функциональные зависимости или связи «одно к одному» между двумя столбцами таблицы измерения. Это полезно при поиске одного значения по другому значению в таблице. Например, если нам дана дата, мы сможем определить день недели. Если дан номер месяца, мы сможем определить название месяца. Если есть почтовый код, мы сможем определить город. Если есть название продукта, мы узнаем его производителя. Взаимоотношения «одно к одному» определяются с помощью условия ATTRIBUTE в выражении CREATE DIMENSION.

4.5.1. Создание измерения

Как только таблица, содержащая данные измерения, создана, вы можете создавать объект DIMENSION с помощью выражения CREATE DIMENSION. Будет полезно нарисовать пузырьковую диаграмму (bubble diagramm), показывающую иерархические взаимоотношения столбцов перед формированием измерения.

На рис. 4.6 показана пузырьковая диаграмма для географического измерения. Индексы почтовых отделений вкладываются (rollup) в города, которые вложены в округа, которые, в свою очередь, вложены в страны.

Чтобы превратить пузырьковую диаграмму, показанную на рис. 4.6, в определение измерения, каждый «пузырек» нужно превратить в уровень (LEVEL) иерархии так, как это показано в приведенном операторе CREATE DIMENSION.

```
CREATE DIMENSION geography_dim
LEVEL postal_code IS customer.postal_code
LEVEL town IS customer.town
LEVEL county IS customer.county
LEVEL country IS customer.country
HIERARCHY loc_rollup (
     postal_code CHILD OF
     town CHILD OF
     county CHILD OF
     country
    );
```

Взаимоотношения между уровнями описываются в условии HIERARCHY. Те пузырьки на диаграмме, из которых начинаются стрелки, описываются с помощью условия CHILD OF. В нашем примере почтовый индекс вложен в город, следовательно, почтовый индекс (postal_code) является потомком (CHILD OF) города (town). Города вложены в округа, следовательно, город — это потомок округа (county). Округ вложен в страну (country), поэтому округ — потомок страны.

Любому значению в столбце-потомке иерархии должно соответствовать рдно и только одно значение столбца-предка. В таблице покупателей, использованной в приведенном примере, почтовый индекс 02134 относится к Бостону, который относится к штату Массачусетс, поскольку почтовый индекс уникален для любого данного города, он удовлетворяет этому правилу. Почтовый индекс 02134 относится только к адресам в Бостоне. Его нельзя использовать, чтобы ссылаться на адреса в Сан-Франциско.

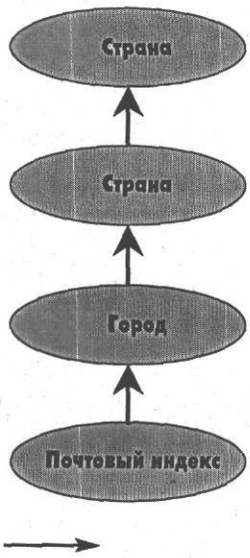


Рис. 4.6. *Пузырьковая диаграмма для географического измерения*

4.5.2. Формирование измерений с множественной иерархией

Иногда вложение одних и тех же данных может идти несколькими путями. В нашем примере схемы, EASYDW, как и во многих других предприятиях, мы используем обычный и финансовый календари. В обычном календаре дни складываются в месяцы, а месяцы — в годы. В финансовом календаре месяцы складываются в недели, а недели — в финансовые кварталы. Все это можно отразить, определив в измерении множественную иерархию.

На рис. 4.7 показана пузырьковая диаграмма измерения-времени. В ней есть две иерархии, описывающие два разных пути вложений ключа времени. Каждая стрелка, выходящая из «пузырька», описывается условием CHILD OF. Поскольку из «пузырька» time_key выходят две стрелки, имеется и два выражения CHILD OF — по одному на каждую иерархию. Для конкретной даты одна иерархия определяет, к какой неделе и финансовому кварталу она относится, а другая — к какому месяцу и году. В верхней части любой пузырьковой диаграммы есть особый уровень ALL, отражающий уровень общего суммарного итога для данного измерения. Все уровни иерархии вложены в ALL. Уровень ALL не определяется явным образом в выражении CREATE DIMENSION.

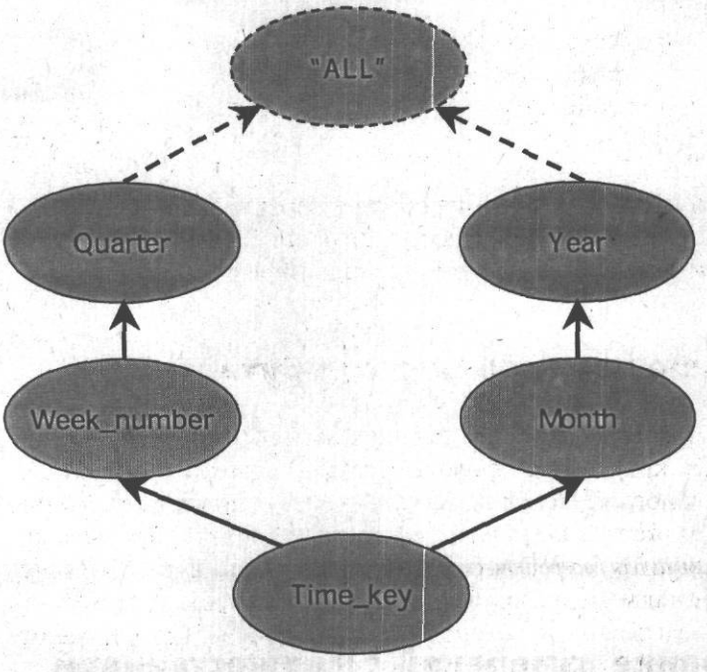


Рис. 4.7. *Пузырьковая диаграмма для измерения-времени*

Приводимый ниже фрагмент кода SQL представляет собой пример измерения со множественной иерархией.

```
...
CREATE DIMENSION time_dim
LEVEL time_key      IS time.time_key
LEVEL month  IS time.month
LEVEL quarter       IS time.quarter
LEVEL year          IS time.year
LEVEL week_number   IS time.week_number
HIERARCHY calendar_rollup (
                                time_key CHILD OF
                                month CHILD OF
                                Year
                                )
```

```
HIERARCHY fiscal_rollup (
                          time_key CHILD OF
                          week_number CHILD OF
                          quarter
                          );
```

И опять, каждому значению столбца-потомка соответствует только одно значение столбца-родителя. Любая данная дата, скажем, 21 марта 2002 года, относится к одному и только одному номеру финансовой недели конкретного квартала.

4.5.3. Определение измерения с атрибутами

Измерения можно использовать для определения взаимоотношений между столбцами одной таблицы, которые не являются частью иерархии. Часто пользователи не знают уникального идентификатора записи, но знают ее название. Например, пользователь может не знать идентификатор продукта, но знать название того, что он хочет купить. В теории реляционных баз данных эти отношения называются функциональными зависимостями, или отношениями «одно к одному». Функциональная зависимость столбцов A и B означает, что каждому значению A соответствует только одно значение B. Поэтому, если вы знаете значение A, вы можете определить значение B.

В определениях измерений для указания функциональных зависимостей используется условие ATTRIBUTE. Условие ATTRIBUTE следует применять только тогда, когда между двумя столбцами существует отношение «одно к одному».

Ниже мы видим определение измерения с условием ATTRIBUTE. Это условие говорит нам, что для данного идентификатора продукта (product_id) существует только одно название продукта (product_name) и один производитель (manufacturer). Однако это не означает, что по производителю мы можем определить идентификатор продукта.

```
CREATE DIMENSION product_dim
 LEVEL product_id IS product.product_id
 LEVEL category IS product.category
 HIERARCHY merchandise_rollup (
      product_id CHILD OF
      category
      )
ATTRIBUTE product_id DETERMINES (product_name, manufacturer);
```

4.5.4. Определение измерения с нормализованными таблицами

Если у вас есть схема «снежинки» (snowflake) или другая нормализованная схема, тогда ваша иерархия сможет ссылаться на столбцы в разных таблицах. Оператор создания измерения в Oracle позволяет объявлять и взаимоотношения такого рода.

На рис. 4.8 показано нормализованное измерение-время. «Пузырьки» и сплошные стрелки определяют отношения вложенности, как ранее. Пунктирные прямоугольники представляют собой таблицы, из которых получаются уровни. Пунктирные стрелки показывают, как эти таблицы соединяются друг с другом. В этом примере существуют отдельные таблицы для времени, недели, месяца, квартала и года. Таблица времени соединяется с таблицей недели с помощью предиката time.week=time.week_number и с таблицей месяца — предикатом time_month=month.month.

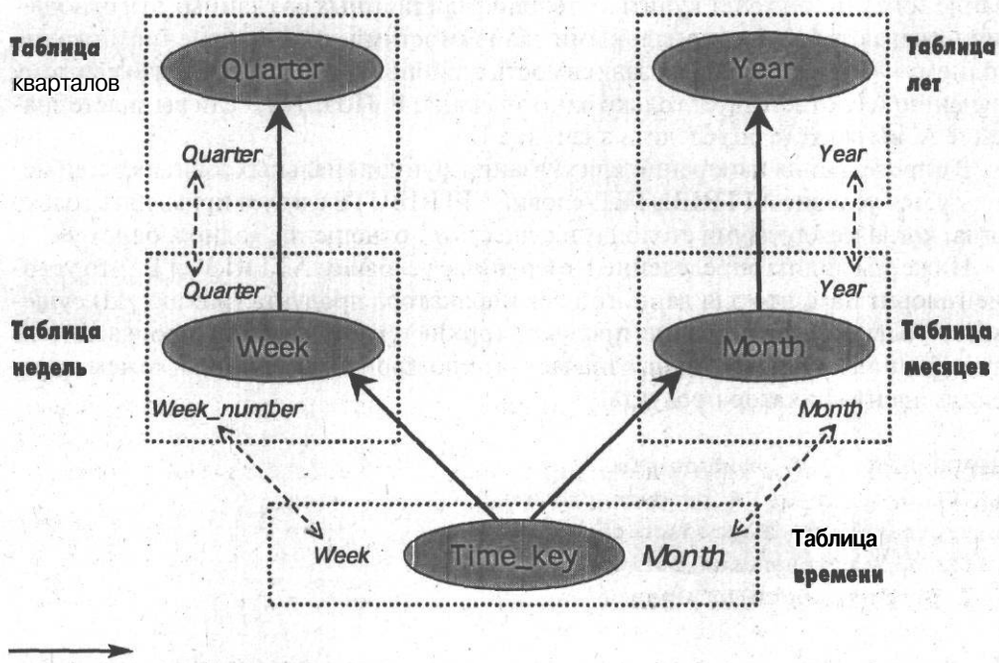


Рисунок 4.8 *Нормализованное измерение*

Чтобы получить из этой диаграммы выражение CREATE DIMENSION, нужно сначала для каждого «пузырька» определить уровень, указав для этого уровня как имя таблицы, так и название столбца. Например, уровень week — это week.week_number. Затем, используя имена уровней, как показывалось ранее, вы определяете иерархию. И наконец, вы указываете все соединения, которые нужно выполнить для перемещения по иерархии с помощью условия JOIN KEY. Таким образом, получается такое выражение:

```
CREATE DIMENSION time_dim
 LEVEL time_key IS time.time_key
 LEVEL month IS month.month
 LEVEL quarter IS quarter.quarter
 LEVEL year IS year.year
 LEVEL week IS week.week_number
 HIERARCHY calendar_rollup (
                                time_key CHILD OF
                                month CHILD OF
                                year
                                JOIN KEY time.month REFERENCES month
                                JOIN KEY month.year REFERENCES year
                                )
 HIERARCHY fiscal_rollup    (
                                time_key CHILD OF
                               week CHILD OF
                               quarter
                               JOIN KEY time.week REFERENCES week
                               JOIN KEY week.quarter REFERENCES quarter
                               );
```

В иерархии fiscal_rollup (финансовая иерархия) столбец условия JOIN KEY time.week используется для соединения таблицы времени с уровнем недель.

Как говорилось в главе 2, нормализованные таблицы измерений приводят к выполнению ресурсоемких дополнительных соединений в ходе обработки запроса, и поэтому их следует использовать с осторожностью.

4.5.5. Проверка измерения

Мы уже говорили, что взаимоотношения, объявленные в объекте-измерении, не проверяются автоматически базой Oracle. Чтобы результаты переписывания

запросов были точны, вы должны обеспечить соответствие этих связей данным. Чтобы облегчить решение этого вопроса, Oracle предлагает процедуру, называющуюся DBMS_OLAP.VALIDATE_DIMENSION. Вы можете запускать эту процедуру всякий раз после загрузки в таблицы измерений новых данных, что будет способствовать целостности данных.

Например, предположим, что у нас есть следующее измерение customer (покупатель).

```
CREATE DIMENSION customer_dim
  LEVEL customer IS customer.customer_id
  LEVEL town IS customer.town
  LEVEL region IS customer.county
  HIERARCHY customer_zone (
     customer CHILD OF
     town CHILD OF
     region
     )
  ATTRIBUTE town DETERMINES postal_code
  ATTRIBUTE customer DETERMINES (sex, occupation);
```

В этом измерении почтовый индекс (postal_code) определяется городом (town), а каждый город должен входить в один и только один округ (county). Давайте проверим, что это действительно так.

```
SELECT distinct town, county, postal_code FROM customer;

TOWN       - COUNTY       POSTAL_COD
---------- ---------- ----------
Chandlers    Hants        S053 1TZ
Eastleigh    Hants        S053 1LD
Liverpool    Mersey       LV1 1QT
London       London       W1 1QC
Soton        Hants        S06 1gh
Soton2       Hants        SO1 1TF
Soton3       Hants        SO1 1TF
Soton4       Hants        SO1 1TF
Soton5       Hants        SO1 1TF
Soton6       Hants        SO1 1TF
Soton7       Hants        SO1 1TF
Soton8       Hants        SO1 1TF '
Soton9       Hants        SO1 1TF
```

Теперь предположим, что в таблицу покупателей были вставлены неверные данные. Почтовый индекс Лондона был ошибочно введен как W1 2QC, а город Ливерпуль по ошибке был помещен в округ Hants.

```
INSERT INTO customer VALUES ('AB130000', 'London', 'London',
'W1 2QC', '7-JUL-74', 'UK', 'Doctor', 'F');

INSERT INTO customer VALUES ('AB130001', 'Liverpool', 'Hants',
'LV1 1QT', '18-JUN-73', 'UK', 'Doctor', 'M');

COMMIT;'
```

Если мы запустим процедуру VALIDATE_DIMENSION, она обнаружит несоответствие во взаимосвязях условий HIERARCHY и ATTRIBUTE, объявленных в измерении customer_dim.

```
variable run_id number;

execute dbms_olap.create_id(:run_id);
execute dbms_olap.validate_dimension('CUSTOMER_DIM', 'EASYDW',
        FALSE, TRUE, :run_id);
```

Процедура VALIDATE_DIMENSION проверит целостность данных и поместит все найденные исключения (exceptions) в таблицу MVIEW_EXCEPTIONS.

```
SELECT distinct owner, table_name, dimension_name, relationship
FROM system.MVIEW_EXCEPTIONS
WHERE runid = :run_id;

OWNER  TABLE_NAME          DIMENSION_NAME      RELATIONSHIP
------ ----------- --------------  ------------
EASYDW         CUSTOMER             CUSTOMER_DIM ATTRIBUTE
EASYDW         CUSTOMER             CUSTOMER_DIM CHILD OF
```

Столбец bad_rowid таблицы MVIEW_EXCEPTIONS содержит идентификаторы строк с несоответствиями (т. е. всех строк, соответствующих неверному взаимоотношению). В нашем примере будут возвращены все строки, соответствующие Лондону и Ливерпулю. Теперь, как показано в примере, мы можем посмотреть на сами данные, взглянув на таблицу заказчиков. Можно видеть,

что идентификаторы покупателей (customer_id) AB130000 и AB130001 соответствуют неверным данным о городе, почтовом индексе и округе относительно остальных данных.

```
SELECT customer_id, town, county, postal_code FROM customer
WHERE rowid IN (select bad_rowid FROM system.mview_exceptions);

CUSTOMER_I  TOWN        COUNTY      POSTAL_COD
----------  ----------  ----------  ----------
AB123477    London      London      W1 1QC
AB123478    London      London      W1 1QC
AB123480    London      London      W1 1QC
...
AB130000    London      London      W1 2QC <- неверные данные
AB123496    Liverpool   Mersey      LV1 1QT
AB123497    Liverpool   Mersey      LV1 1QT
AB123498    Liverpool   Mersey      LV1 1QT
...
AB130001    Liverpool   Hants       LV1 1QT <- неверные данные
```

В следующем разделе мы рассмотрим, как можно использовать переписывание запросов под использование материализованных представлений. Мы также увидим, как используются объекты-измерения.

4.6. Переписывание запросов

Наибольшим преимуществом при использовании Управления сводными данными в базе Oracle является автоматическое переписывание запросов. Традиционно при применении сводных таблиц запросы к основным таблицам нужно было изменять, чтобы можно было использовать эти сводные таблицы. При наличии системы переписывания запросов оптимизатор Oracle автоматически определяет, можно ли ответ на запрос получить из материализованного представления, и если да, то явным образом переписывает запрос под материализованное представление. Если оптимизатор решает, что сводки недостаточно для ответа на запрос, он использует основные данные. Это означает, что прикладным программам не нужно ничего менять, чтобы создавать и использовать сводные данные. Более того, конечным пользователям не нужно и знать о суще-

ствовании сводок, и сводки можно создавать и изменять без привлечения пользователей.

База Oracle поддерживает несколько типов переписывания запросов, что дает возможность использовать одну сводку при ответе на несколько запросов. Мы рассмотрим некоторые из них в следующих разделах, в том числе:

- совпадение текста (textmatch);
- вложение агрегатов (Aggregaterollup);
- обратное соединение (Join-back);
- фильтрованные данные (Filtered data);
- вложение с использованием объекта-измерения.

Чтобы разрешить переписывание запросов, нужно создать материализованное представление с условием ENABLE QUERY REWRITE. Кроме того, инициализационный параметр QUERY_REWRITE_ENABLED необходимо установить в TRUE, и должен использоваться оптимизатор, работающий на основе стоимости (cost-based optimizer).

```
-- разрешаем переписывание запросов
ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE;
-- разрешаем использование оптимизатора на основе стоимости
ALTER SESSION SET OPTIMIZER_MODE="all_rows";
```

В нескольких первых наших примерах мы будем использовать следующее материализованное представление, которое вычисляет сумму объемов продаж продуктов за месяц.

```
CREATE MATERIALIZED VIEW monthly_sales_mv
 ENABLE QUERY REWRITE
 AS
 SELECT t.month, t.year, p.product_id,
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales
      FROM time t, product p, purchases ps
      WHERE t.time_key = ps.time_key AND
           ps.product_id = p.product_id
      GROUP BY t.month, t.year, p.product_id;
```

4.6.1. Как можно узнать, что запрос был переписан?

Чтобы определить, был ли запрос переписан, используйте процедуру EXPLAIN PLAN (описанную в Главе 3) и посмотрите на план обработки запроса. В выходных данных процедуры показано имя материализованного представления, которое было использовано для переписывания запроса. Мы будем использовать эту возможность в приводимых примерах. Кроме этого, вы можете использовать для слежения за планом обработки запроса оптимизатором опцию SET AUTOTRACE в SQLPLUS. Если вы обнаруживаете, что ожидаемого переписывания запросов не происходит, вы можете использовать для диагностики проблемы утилиту DBMS_MVIEW.EXPLAIN_REWRITE, которая будет обсуждаться ниже.

Совет: Давайте материализованным представлениям имена с суффиксом _mv. Это позволит вам быстро находить их в выходных данных плана обработки.

4.6.2. Совпадение текста SQL

Самый простой тип переписывания запросов — это когда текст SQL в запросе, формирующем материализованное представление, точно совпадает с поступившим запросом. Совпадение текста не учитывает регистр букв и игнорирует различия в числе пробелов.

План обработки запроса в следующем фрагменте кода показывает, что оптимизатор решил осуществить доступ к материализованному представлению MONTHLY_SALES_MV с помощью полного сканирования таблицы.

```
-- полное совпадение текста
SELECT t.year, t.month, p.product_id,
   sum (ps.purchase_price) as sum_of_sales,
   count (ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = f.time_key AND
f.product_id = p.product_id
GROUP BY t.year, t.month, p.product_id;
```

```
MONTH    YEAR         PRODUCT_    SUM_OF_SALES    TOTAL_SALES
-----    ----------   --------    ------------    -----------
   1     2002           SP1000    54966.53        332
   1     2002           SP1001    48861.17        312
   1     2002           SP1010    45432.43        268
...
-------------------------------------------------------------------
|Id| Operation              |Name              |Rows| Bytes |Cost |
-------------------------------------------------------------------
| 0| SELECT STATEMENT       |                  | 327| 18966 |  2  |
| 1|  TABLE ACCESS FULL     | MONTHLY_SALES_MV | 327| 18966 |  2  |
-------------------------------------------------------------------
```

База Oracle также попробует выявить совпадение текста, начиная с ключевого слова FROM в запросе. При этом принимаются во внимание различия в расположении столбцов в списке оператора SELECT и вычисленные значения выражений. В следующем примере функции SUM(purchase_price) и COUNT(purchase_price) используются для вычисления среднего, а расположение столбцов t.month и t.year в перечне оператора SELECT обратное. Можно видеть, что для переписывания этого запроса было использовано материализованное представление.

```
SELECT t.month, t.year, p.product_id,
     AVG(ps.purchase_price) avg_of_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
     ps.product_id = p.product_id
GROUP BY t.year, t.month, p.product_id;

MONTH       YEAR          PRODUCT_ID   AVG_OF_SALES
----------  ----------    ----------   ------------
1           2002              SP1000   165.561837
1           2002              SP1001   156.606314
1           2002              SP1010   169.523993
...
-------------------------------------------------------------------
|Id| Operation          | Name              | Rows | Bytes |Cost |
-------------------------------------------------------------------
| 0|  SELECT STATEMENT  |                   | 327  | 18966 | 2   |
| 1|  TABLE ACCESS FULL | MONTHLY_SALES_MV  | 327  | 18966 | 2   |
-------------------------------------------------------------------
```

Если тексты запроса и материализованного представления не совпадают, база Oracle сравнит характеристики соединения, условия GROUP BY, агрегаты запроса и материализованного представления и определит, можно ли переписать этот запрос под использование материализованного представления. В следующих разделах мы проиллюстрируем использование этих правил.

4.6.3. Вложение агрегатов

Вложение агрегатов (aggregate rollup) (часто называемое просто вложением) получается, когда агрегаты материализованных представлений можно агрегировать с целью получения требуемого в запросе агрегата. Простой пример такой ситуации — когда агрегат содержит только некоторые из сгруппированных столбцов материализованного представления. Например, если материализованное представление группируется по годам и месяцам, а в запросе указывается только год, тогда агрегаты материализованного представления по каждому месяцу данного года агрегируются, образуя одну строку по данному году.

```
-- вложение по столбцу месяца
SELECT t.year, p.product_id,
     SUM (ps.purchase_price) as sum_of_sales,
     COUNT (ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
     ps.product_id = p.product_id
GROUP BY t.year, p.product_id;
YEAR    PRODUCT_ID         SUM_OF_SALES       TOTAL_SALES
------  ----------         -------------      -----------
2002    SP1000               142173.51        659
2002    SP1001                97910.04        475
2002    SP1010               122422.72        514
...
-------------------------------------------------------------------------
|Id| Operation            | Name              | Rows | Bytes | Cost |
-------------------------------------------------------------------------
| 0| SELECT STATEMENT     |                   | 327  | 14715 | 5    |
| 1|  SORT GROUP BY       |                   | 327  | 14715 | 5    |
| 2|    TABLE ACCESS FULL |MONTHLY_SALES_MV   | 327  | 14715 | 2    |
-------------------------------------------------------------------------
```

Когда произойдет вложение, переписанный запрос будет содержать, как отмечено в предыдущем плане обработки запроса, условие GROUP BY.

Более интересный случай вложения — когда данные описываются в иерархическом виде в объекте-измерении. В этом случае система переписывания запросов может вкладывать данные более низких уровней иерархии в более высокие. Мы опишем это позже в этой же главе.

4.6.4. Обратное соединение

Чтобы выполнилось переписывание запроса, все столбцы запроса должны либо появляться в материализованном представлении, либо происходить от одного из столбцов материализованного представления. В последнем случае материализованное представление для получения требуемого столбца должно быть соединено с таблицей основных данных. Такое соединение называется обратным (join-back). В простейшем случае, чтобы произошло обратное соединение, необходимо, чтобы материализованное представление содержало либо первичный ключ, либо идентификатор строки таблицы основных данных. Например, столбец product_id является первичным ключом таблицы продуктов (product). На следующий запрос, в котором запрашивается название продукта (product_name), можно ответить, используя материализованное представление MONTHLY_SALES_MV и обратное соединение. В плане оптимизатора видно, что запрос был переписан с использованием материализованного представления MONTHLY_SALES_MV, соединенного с таблицей product.

```
-- обратное соединение с таблицей продуктов с помощью первичного ключа
SELECT t.year, t.month, p.product_name,
     SUM (ps.purchase_price) as sum_of_sales
FROM time t, product p, purchases ps
 WHERE t.time_key = ps.time_key AND
     ps.product_id = p.product_id
GROUP BY t.year, t.month, p.product_name;

YEAR    MONTH        PRODUCT_NAME    SUM_OF_SALES
----    -----        -------------   ------------
2002        1        CD LX1             40208.33
2002        1        CD LX2             54031.52
2002        1        CD LX3              9145.54
...
```

```
-------------------------------------------------------------------------
|Id| Operation              | Name              I Rows | Bytes | Cost  |
-------------------------------------------------------------------------
| 0| SELECT STATEMENT       |                     | 157 | 9734  | 9     |
| 1|  SORT GROUP BY         |                     | 157 | 9734  I 9     |
| 2|   HASH JOIN            |                   I 327 | 20274 I 5     |
| 3|    TABLE ACCESS FULL   | PRODUCT           I 162 | 2754  I 2     |
| 4|    TABLE ACCESS FULL   | MONTHLY_SALES_MV I 327 | 14715 I 2     |
-------------------------------------------------------------------------
```

4.6.5. Фильтрованные данные

Материализованное представление MONTHLY_SALES_MV, определенное ранее, содержит данные обо всех продуктах по каждому месяцу и году. Иногда может понадобиться составить сводку по определенному продукту, региону или году. В таком случае материализованное представление будет содержать только часть данных, определяемую условием отбора в условии WHERE запроса. В Oracle 9i возможно выполнение сложного переписывания запросов под такие материализованные представления. База Oracle определяет, можно ли дать ответ на запрос с помощью материализованного представления, путем анализа условий WHERE материализованного представления и запроса.

Следующее материализованное представление содержит суммарный объем продаж и число продаж для категории электроники (ELEC) за годы с 1997 по 2002.

```
CREATE MATERIALIZED VIEW elec_sales_mv
ENABLE QUERY REWRITE
AS
SELECT t.month, t.year, p.product_id,
     SUM(ps.purchase_price) as sum_of_sales,
     COUNT (ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.purchase_date AND
     ps.product_id = p.product_id AND
     p.category = 'ELEC' AND t.year BETWEEN 1997 and 2002
GROUP BY t.month, t.year, p.product_id;
```

Это материализованное представление можно использовать при ответе на следующий запрос, в котором запрашивается суммарный объем и количество продаж в категории электроники за 1999 год. Отметьте, что следующий план обработки запроса является предикатным информационным разделом. Эта часть выходных данных утилиты EXPLAIN PLAN выводит и показывает предикаты, используемые на каждом этапе выполнения плана обработки запроса. Здесь мы приводим их, чтобы показать, что после переписывания запроса данные материализованного представления фильтруются, чтобы выбрать только строку, соответствующую 1999 году.

```
SELECT t.month, p.product_id,
     SUM(ps.purchase_price) as sum_of_sales,
     COUNT (ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.purchase_date AND
     ps.product_id = p.product_id AND
     p.category = 'ELEC' AND t.year = 1999
GROUP BY t.month, t.year, p.product_id;

MONTH      PRODUCT_      SUM_OF_SALES      TOTAL_SALES
---------- --------      ------------      -----------
1          SP1000          54966.53              332
1          SP1001          48861.17              312
1          SP1010          45432.43              268
...

-------------------------------------------------------------------
|Id I Operation           | Name          I Rows  I Bytes I Cost  I
-------------------------------------------------------------------
I 0| SELECT STATEMENT     I               | 2     I 116   I 2      I
|*1|  TABLE ACCESS FULL  |ELEC_SALES_MV | 2     I 116   I 2      I
-------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter("ELEC_SALES_MV"."YEAR"=1999)
```

С другой стороны, ответ на запрос, в котором запрашиваются суммарный объем продаж за 1999 год по всем категориям, нужно будет получать по основным таблицам, поскольку материализованное представление не содержит доста-

точно данных для вычислений по категориям продуктов иным, нежели электроника.

Запрос может иметь дополнительные условия, не упомянутые в материализованном представлении. Например, в следующем запросе нам нужны помесячные объемы продаж цифровых камер (Digital Camera) в категории электроники за 2002 год. Материализованное представление содержит данные по всем продуктам в этой категории, и мы можем определить название продукта (product_name) по идентификатору продукта (product_id) с помощью обратного соединения. Следовательно, запрос будет переписан под использование материализованного представления с соединением с таблицей продуктов.

```
SELECT t.month, SUM(ps.purchase_price) as sum_of_sales,
  COUNT (ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.purchase_date AND
  ps.product_id = p.product_id AND p.category = 'ELEC' AND
  t.year = 2002 AND product_name = 'Digital Camera'
GROUP BY t.month;
MONTH    SUM_OF_SALES        TOTAL_SALES
---------- ------------ -----------
1        54966.53                 332
2        32861.22                 204
4        24575.66                  82
6        29770.1                   41

-------------------------------------------------------------------
|Id | Operation            | Name           I Rows      1 Bytes I
Cost|
-------------------------------------------------------------------
| 0 | SELECT STATEMENT     I                  | 1   | 75  | 4    I
| 1 |  TABLE ACCESS BY     | PRODUCT          | 1   I 17  | 1    I
|   |      INDEX ROWID     |                  |     |     |      |
| 2 |   NESTED LOOPS       |                  | 1   I 75  I 4    |
| 3 |     TABLE ACCESS FULL | ELEC_SALES_MV   | 2   | 116 | 2    |
| 4 |     INDEX RANGE SCAN  | PRODUCT_PK_INDEX|     I 162 |      I
-------------------------------------------------------------------
```

Этот пример показывает, как можно вместе применять два механизма — обратное соединение и фильтрованные данные.

4.6.6. Вычисление других агрегатов в запросе

Агрегаты в запросе можно вычислить с помощью других агрегатов в материализованном представлении. Мы уже рассмотрели пример этого в разделе, посвященном совпадению SQL-текста. Однако сила переписывания запросов состоит в том, что можно одновременно выполнять много разных преобразований. Например, в следующем запросе мы хотим узнать среднюю цену продаж каждого товара за год. В материализованном представлении имеется функция получения суммы и подсчета количества по столбцу purchase_price на уровнях месяца и года. Однако можно вычислить среднее, сначала выполнив вложение месяцев в годы, а затем разделив сумму на количество строк столбца purchase_price. Запрос, следовательно, переписывается под использование материализованного представления MONTHLY_SALES_MV.

— вычислимость агрегата

```
SELECT t.year, p.product_id, AVG(ps.purchase_price) as ave_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
ps.product_id = p.product_id
GROUP BY t.year, p.product_id;

YEAR       PRODUCT_      AVE_SALES
---------- --------      ----------
2002       SP1000        215.74129
2002       SP1001        206.1264
2002       SP1010        238.176498
...
-----------------------------------------------------------------------
|Id| Operation           | Name              | Rows  I Bytes I  Cost I
-----------------------------------------------------------------------
| 0| SELECT STATEMENT    |                   | 327   I 14715 | 5    |
| 1|  SORT GROUP BY      |                   | 327   I 14715 I 5    |
| 2|   TABLE ACCESS FULL | MONTHLY_SALES_MV  | 327   I 14715 I 2    |
-----------------------------------------------------------------------
```

Чтобы вычислить агрегаты в запросе с вложениями, может потребоваться, чтобы в материализованном представлении присутствовали вспомогательные агрегаты. Например, чтобы вложить агрегат AVG, в материализованном представлении должны быть агрегаты SUM и COUNT или AVG и COUNT.

Для всех случаев переписывания запросов, которые мы пока рассматривали, никакой дополнительной информации от пользователя не требуется. Однако, чтобы получить максимум пользы от переписывания запросов, вы должны проинформировать систему переписывания запросов о взаимосвязях столбцов данных с помощью ограничений и измерений.

4.6.7. Переписывание запросов с помощью измерений

Одним из наиболее мощных свойств переписывания запросов является возможность использовать одно материализованное представление для работы с очень разнообразными запросами. Объект-измерение (dimension), который мы обсуждали в разделе 4.5, в этом отношении чрезвычайно полезен. Позволяя вам объявлять о взаимосвязях между столбцами таблиц измерений, он обеспечивает систему переписывания запросов информацией для осуществления вложений нижних уровней иерархии в верхние. Для примера, предположим, что вашим пользователям нужно узнать суммарный объем продаж по дням, месяцам и годам. Для ответа на этот запрос вы можете создать три материализованных представления или же создать одно уровня дней, а затем определить объект-измерение и показать взаимосвязь между днями, месяцами и годами. Теперь, когда поступает запрос уровня месяцев, будет использоваться материализованное представление уровня дней с вложенностью до уровня месяцев.

Вспомните, что взаимоотношения, объявляемые в измерениях, автоматически базой Oracle не проверяются. Следовательно, вы должны установить параметр QUERY_REWRITE_INTEGRITY в TRUSTED или STALE_TOLERATED, показывая тем самым, что взаимоотношениям, объявленным в объекте-измерении, можно доверять.

Рассмотрим определение измерения-времени. Условие HIERARCHY говорит системе переписывания запросов, что ключ времени (time_key) вкладывается в номер недели (week_number), который, в свою очередь, вкладывается в квартал (quarter). Это означает, что если мы знаем значение ключа времени для некоторого столбца таблицы TIME, мы знаем, к какой неделе он относится. Точно так же, если мы знаем месяц (скажем, январь 2002), мы знаем и год, к которому он относится (2002).

```
CREATE DIMENSION time
  LEVEL time_key                  is time.time_key
  LEVEL month                     is time.month
  LEVEL quarter                   is time.quarter
```

```
  LEVEL year                    is time.year
  LEVEL week_number             is time.week_number
  HIERARCHY fiscal_rollup (
     time_key                   CHILD OF
     week_number                CHILD OF
     quarter )
  HIERARCHY calendar_rollup(
     time_key                   CHILD OF
     month                      CHILD OF
     year);
```

Предположим, что наше материализованное представление MONTHLY_SALES_MV создано для вывода общего объема продаж по продуктам и месяцам следующим образом:

```
CREATE MATERIALIZED VIEW monthly_sales_mv
  ENABLE QUERY REWRITE
  AS
  SELECT t.month, p.product_id,
     SUM(ps.purchase_price) as sum_of_sales,
     COUNT (ps.purchase_price) as total_sales
  FROM time t, product p, purchases ps
  WHERE t.time_key = ps.time_key AND
     ps.product_id = p.product_id
  GROUP BY t.month, p.product_id;
```

В следующем запросе мы хотим узнать общий объем продаж по продуктам и по годам. Поскольку у нас есть материализованное представление с общим объемом продаж по продуктам и месяцам, а месяца можно вкладывать в годы, как это указывается в иерархии calendar_rollup в измерении-времени, оптимизатор перепишет запрос под использование материализованного представления MONTHLY_SALES_MV. Заметим, что для того, чтобы определить значение года для некоторого месяца, база Oracle произведет обратное соединение материализованного представления с таблицей времени.

```
— вложение в более высокий уровень иерархии
SELECT t.year, p.product_id,
     COUNT (ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
```

```
WHERE t.time_key = ps.time_key AND
      ps.product_id = p.product_id
GROUP BY t.year, p.product_id;
-------------------------------------------------------------------
|Id| Operation                | Name              | Rows | Bytes | Cost |
-------------------------------------------------------------------
| 0| SELECT STATEMENT         |                   |    1 |    39 |    9 |
| 1|  SORT GROUP BY           |                   |    1 |    39 |    9 |
| 2|   HASH JOIN              |                   |  176 |  6864 |    7 |
| 3|    VIEW                  |                   |    5 |    35 |    4 |
| 4|     SORT UNIQUE          |                   |    5 |    35 |    4 |
| 5|      TABLE ACCESS FULL   | TIME              |   15 |   105 |    2 |
| 6|    TABLE ACCESS FULL     | MONTHLY_SALES_MV  |  246 |  7872 |    2 |
-------------------------------------------------------------------
```

Использование условия *ATTRIBUTE* в измерении

Когда создается хранилище данных, объект-измерение часто упускают из виду, поскольку его значение для переписывания запросов оценивают недостаточно. Однако объект-измерение предоставляет огромные возможности для переписывания запросов без дополнительных затрат. Мы уже видели, как система переписывания запросов может использовать преимущества условия HIERARCHY для переписывания нескольких запросов с использованием одного материализованного представления. Система переписывания запросов также может использовать условие ATTRIBUTE измерения. В следующем примере мы хотим узнать суммарный объем продаж по покупателям на основе их занятия (occupation) и пола (sex).

```
SELECT c.sex, c.occupation, SUM(ps.purchase_price) as sum_of_sales
FROM purchases ps, customer c
WHERE c.customer_id = ps.customer_id
GROUP BY c.sex, c.occupation;
```

Мы могли бы вставить столбцы пола и занятия в материализованное представление. Однако мы знаем, что, имея идентификатор покупателя (customer_id), мы можем найти такую информацию, как имя, пол и занятие. Такие взаимосвязи внутри таблицы, которые не иерархичны по своей природе, определяются условием ATTRIBUTE в измерении. Условие ATTRIBUTE говорит нам, что каждый идентификатор покупателя имеет связь типа «один к одному» со столбцами имени покупателя, его пола и занятия.

```
-----------------------------------------------------------------
| Id | Operation              | Name          | Rows | Bytes | Cost |
-----------------------------------------------------------------
|  0 | SELECT STATEMENT       |               |    8 |   320 |    7 |
|  1 |  SORT GROUP BY         |               |    8 |   320 |    7 |
|  2 |   HASH JOIN            |               |   82 |  3280 |    5 |
|  3 |    TABLE ACCESS FULL   | CUSTOMER      |   40 |   800 |    2 |
|  4 |    TABLE ACCESS FULL   | CUST_SALES    |   82 |  1640 |    2 |
-----------------------------------------------------------------
```

При проектировании хранилища вам следует попробовать выявить связи между таблицами измерений и создать везде, где возможно, объекты-измерения. Это приведет к значительной экономии места и повысит возможности переписывания запросов, улучшая, таким образом, производительность их обработки.

4.6.8. Переписывание запросов и ограничения

В OLTP-системах ограничения используются для обеспечения целостности данных. В хранилище ограничения можно использовать для формирования взаимосвязей между таблицей фактов и таблицами измерений. Как правило, такое ограничение, как первичный ключ, создается в столбце уникального ключа в каждой таблице измерения. Такое ограничение, как внешний ключ, и ограничение NOT NULL создаются для каждого соответствующего ключа в таблице фактов.

Как уже говорилось в главе 2, в схеме EASYDW имеются первичные ключи для всех таблиц измерений — покупатель (customer), продукт (product) и время (time). Кроме того, существуют внешние ключи и ограничение NOT NULL в столбцах таблицы покупок (purchases), которые соединяют ее с таблицами измерений.

Взаимоотношение, определяемое в этих ограничениях, указывает системе переписывания запросов, что соединение между таблицей purchases и, например, таблицей TIME создает одну строку для каждой строки таблицы purchases. Такое соединение называется соединением без потерь (losslessjoin), поскольку ни одна строка таблицы покупок не будет потеряна (или дублирована) при соединении. Вы можете использовать материализованное представление при переписывании запросов, содержащих меньше соединений, чем материализованное представление, если дополнительные соединения материализованного представления являются соединениями без потерь. Например, мы можем переписать приводимый ниже запрос под использование материализованного представления MONTHLY_SALES_MV, даже если таблица времени вообще пропущена!

Это возможно потому, что дополнительное соединение между таблицами purchases и time в материализованном представлении является соединением без потерь.

```
SELECT p.product_id, SUM(ps.purchase_price) as sum_of_sales
FROM product p, purchases ps
WHERE ps.product_id = p.product_id
GROUP BY p.product_id;

PRODUCT_      SUM_OF_SALES
--------      ------------
SP1000        142324.24
SP1001        104369.24
SP1010        122422.72
...
-------------------------------------------------------------------------
I IdI Operation            I Name              | Rows | Bytes I Cost |
-------------------------------------------------------------------------
| 0| SELECT STATEMENT      |                   |  327  | 10464 | 4     1
| 1|  SORT GROUP BY       I                   |  327  | 10464 I 4     |
|  2|   TABLE ACCESS FULL I MONTHLY_SALES_MV  |  327  | 10464 | 2     |
-------------------------------------------------------------------------
```

Как обсуждалось в главе 2, когда ограничение задействовано, вы можете заставить базу Oracle проверить целостность данных. Как правило, данные для хранилища проверяются в операционной системе или в ходе процесса ETL. В таком случае вы можете использовать условие NOVALIDATE, что укажет базе Oracle на то, что данные уже проверены.

```
ALTER TABLE purchases ENABLE NOVALIDATE CONSTRAINT fk_customer_id;
```

Объявление ограничения сообщает системе переписывания запросов о связи между таблицей фактов и таблицей измерения. Чтобы информировать базу Oracle о том, что корректности ограничений можно доверять и использовать их при переписывании запросов даже в отсутствии проверки, следует использовать дополнительное условие RELY. Это позволяет администратору базы данных сказать: «Верь мне. Я уже проверил правильность данных». Система переписывания запросов может положиться на правильность связи.

```
ALTER TABLE purchases MODIFY CONSTRAINT fk_customer_id RELY;
```

Совет. Используйте ограничения для формирования связи между таблицами фактов и измерений. Используйте объект-измерение при объявлении связей внутри таблиц измерений, таких, как иерархия по времени или регионам.

4.6.9. Режимы целостности данных при переписывании запросов

В наших примерах мы уже несколько раз упоминали параметр QUERY_REWRITE_INTEGRITY. Этот параметр указывает для Oracle ту степень, в которой переписанные запросы должны отражать данные основных таблиц. Этот параметр может иметь три значения:

- ENFORCED (Принудительный);
- STALE_TOLERATED (Допускающий устаревшие данные);
- TRUSTED (С доверием к данным).

Если для параметра QUERY_REWRITE_INTEGRITY установлен режим ENFORCED (который устанавливается по умолчанию), база Oracle обеспечит то, что переписанный запрос вернет те же самые результаты, что и запрос без переписывания. Если установлен режим STALE_TOLERATED, база Oracle будет использовать материализованные представления, которые не содержат всех основных данных. Если установлено значение TRUSTED, база Oracle будет использовать данные и связи, на которые получено «благословение» администратора базы, а именно, объекты-измерения, ограничения RELY и материализованные представления, созданные по таблицам PREBUILT. База Oracle не проверяет правильность связей, объявленных в измерении и соответствие заранее созданной таблицы запросу, создающему материализованное представление.

Вы должны решить, какой уровень целостности данных при переписывании запросов соответствует вашей цели. Большую часть времени нужно получать результаты самым быстрым из возможных способов, переписывая запросы под использование материализованных представлений. Однако, если ваши материализованные представления устаревают и более не отражают все основные данные, вы можете предпочесть получать результаты из основных таблиц, пока не сможете произвести очередное обновление.

Совет. Когда вы в первый раз тестируете материализованное представление и выясняете, будет ли происходить переписывание запросов, установите параметр QUERY_REWRITE_INTEGRITY в STALE_TOLERATED, поскольку если запрос не будет переписываться в этом режиме, он не будет переписываться ни в каком другом. Как только вы определите, что все работает, можете установить режим параметра в нужное вам значение.

Если результаты, полученные из материализованного представления, достаточно близко соответствуют вашим целям, вы можете использовать это представление, даже если оно устаревает. Чтобы определить помесячный прирост продаж в он-лайновой системе, вам не требуется иметь все транзакции в материализованном представлении. Пока данные являются достаточно свежими, вы можете получать ответы, достаточно близкие к истине. Или же, если приложение знает, что пропущенные данные не входят в область данного запроса, оно вполне может использовать данное материализованное представление. Например, если пропущенные данные касаются последнего месяца, а запросу не требуются эти данные, вы можете использовать материализованное представление. Или же можно использовать материализованное представление, когда устаревает таблица фактов, но не тогда, когда обновляется таблица измерения.

Чтобы указать оптимизатору, что вы хотите использовать материализованное представление, даже если оно устаревает, установите значение параметра QUERY_REWRITE_INTEGRITY в STALE_TOLERATED. Вы можете определить, является ли материализованное представление свежим (FRESH) или устаревшим (STALE) по столбцу STALENESS представления каталога USER_MVIEWS.

В следующем примере показано различие между двумя режимами целостности STALE_TOLERATED и ENFORCED в отношении устаревших данных. Мы вставили четыре новые строки в таблицу фактов purchases.

```
INSERT INTO purchases VALUES ( 'SP1061','1-FEB-2002',
   'AB123456','1-FEB-2002', 0024,28.01, 4.50, 'Y');

INSERT INTO purchases VALUES ( 'SP1062','1-FEB-2002',
   'AB123457','1-FEB-2002', 1024,28.01, 4.50, 'Y');

INSERT INTO purchases VALUES ( 'SP1063','2-FEB-2002' ,
   'AB123457','2-FEB-2002', 0024,28.01, 4.50, 'N');
```

```
INSERT INTO purchases VALUES ( 'SP1064','2-FEB-2002',
    'AB123457','2-FEB-2002', 1024,28.01, 4.50, 'N');
COMMIT;
```

Теперь материализованное представление MONTHLY_SALES_MV является устаревшим. В нем не отражаются все данные основной таблицы.

```
SELECT staleness FROM user_mviews
WHERE mview_name = 'MONTHLY_SALES_MV';

STALENESS
-------------------
STALE
```

Если вы установите параметр QUERY_REWRITE_INTEGRITY в ENFORCED (или TRUSTED), база Oracle будет использовать основные таблицы, а не материализованное представление, а количество продаж будет модифицировано для включения в него новых строк.

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY=ENFORCED;
SELECT t.month, p.product_id,
, SUM(ps.purchase_price) as sum_of_sales,
  COUNT(ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
 ps.product_id = p.product_id
GROUP BY t.month, p.product_id;
```

MONTH	PRODUCT_	SUM_OF_SALES	TOTAL_SALES
1	SP1000	19683.57	209
1	SP1001	17377.68	189
1	SP1010	12500.83	145
1	SP1011	25094.03	249
...			
2	SP1061	362092.71	248 <- новое значение
2	**SP1062**	54775.9	**292**
2	SP1063	45862.38	**286**
2	SP1064	53637.11	**248**
...			

```
-------------------------------------------------------------------------
|Id| Operation                 | Name                | Rows | Bytes | Cost|
-------------------------------------------------------------------------
|0 | SELECT STATEMENT          |                     | 567  | 23814 | 415 |
|1 |  SORT GROUP BY            |                     | 567  | 23814 | 415 |
|2 |   NESTED LOOPS            |                     | 70585| 2895K | 43  |
|3 |    HASH JOIN              |                     | 70585| 2412K | 43  |
|4 |     TABLE ACCESS FULL     | TIME                | 15   | 225   | 2   |
|5 |     PARTITION RANGE ALL   |                     |      |       |     |
|6 |      TABLE ACCESS FULL    | PURCHASES           | 94113| 1838K | 40  |
|7 |     INDEX RANGE SCAN      | PRODUCT_PK_INDEX    | 1    | 7     |     |
-------------------------------------------------------------------------
```

Отметьте, что режим TRUSTED в отношении устаревших данных ведет себя аналогично ENFORCED: он не использует устаревшее материализованное представление. Однако если вы можете мириться с устаревшими данными и установили значение параметра QUERY_REWRITE_INTEGRITY в STALE_TOLERATED, то материализованное представление начинает использоваться. Тем не менее, новые вставленные строки в общем значении не учитываются!

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY=STALE_TOLERATED;

SELECT t.month, p.product_id,
  SUM(ps.purchase_price) as sum_of_sales,
  COUNT(ps.purchase_price) as total_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
  ps.product_id = p.product_id
GROUP BY t.month, p.product_id;

MONTH   PRODUCT_ SUM_OF_SALES TOTAL_SALES
------  -------- ------------ -----------
   1    SP1000       19683.57         209
   1    SP1001       17377.68         189
   1    SP1010       12500.83         145
   1    SP1011       25094.03         249
   ...
   2    SP1061      362064.7          247      <-старое значение
   2    SP1062       54747.89         291
   2    SP1063       45834.37         285
   2    SP1064       53609.1          247
   ...
```

```
-------------------------------------------------------------------
|Id| Operation           | Name              | Rows | Bytes | Cost |
-------------------------------------------------------------------
| 0| SELECT STATEMENT    |                   |  327 | 18966 |    2 |
| 1|  TABLE ACCESS FULL  | MONTHLY_SALES_MV  |  327 | 18966 |    2 |
-------------------------------------------------------------------
```

Вы, вероятно, думаете, что установка режима ENFORCED будет наилучшим решением, поскольку оно гарантирует одни и те же результаты при использовании материализованного представления и основных таблиц. Проблема в том, что необходимо проверить все ограничения, и объекты-измерения не используются. Пока вы не проверите все созданные связи, такие, как ограничения, переписывание запросов не произойдет. Следовательно, в хранилище вы, скорее всего, будете использовать режим TRUSTED. В этом режиме материализованные представления используются, только если они являются свежими. Ограничения используются, если в них имеется условие RELY, и измерения также принимаются во внимание, даже если они не проверены.

Предостережение: Переписывание запросов зависит от целостности ваших определений измерений и ограничений. Вкладывается ли каждый продукт таблицы продуктов в одну и только одну категорию, как это указано в вашем определении измерения? Имеет ли каждый продукт таблицы покупок соответствующий идентификатор продукта в таблице продуктов, как это определяется в ссылочных ограничениях целостности? Если ваши данные не отражают взаимосвязей, определяемых в ограничении или измерении, вы можете получить непредвиденные результаты. То же касается и материализованных представлений для заранее созданных таблиц. Если заранее созданная таблица не соответствует в точности запросу, формирующему материализованное представление, тогда результаты могут быть неопределенными. Если подвести итог, то мы рекомендуем режим TRUSTED для использования в хранилище, но при этом нужно обеспечить корректность данных.

4.6.10. Переписывание запросов и оптимизатор

Переписывание запросов возможно только при использовании оптимизатора, работающего на основе стоимости (cost-based optimizer). Хотя, как правило,

получение результатов из материализованного представления происходит быстрее, возможны случаи, когда быстрее будет получать результаты из основной таблицы. Оптимизатор запросов сравнивает стоимость плана обработки запроса при наличии и в отсутствии переписывания запросов и использует тот, который имеет меньшую стоимость. Чтобы обеспечить правильность этого выбора, вы должны собрать статистическую информацию по основным таблицам, привлекаемым к выполнению запроса, и по материализованным представлениям с помощью пакета DBMS_STATS.

Однако, если вашему приложению нужно, чтобы оптимизатор использовал материализованное представление и не использовал основные таблицы, вы можете установить параметр QUERY_REWRITE_ENABLED в FORCE. В этом режиме если существует материализованное представление, удовлетворяющее запросу, то оптимизатор будет его использовать. Он не будет сравнивать стоимость плана с наличием и отсутствием переписывания запроса. Этот параметр является новинкой Oracle 9i Release 2.

Если для переписывания запроса подходят несколько материализованных представлений, система переписывания выберет то, для которого требуется читать меньший объем данных. Тем не менее, вы можете потребовать использовать конкретное материализованное представление, применив инструкцию /*+ REWRITE(mv) ,*/. Можно отменить переписывание запроса для выражения с помощью инструкции /*+ NOREWRITE */. Предположим, например, что у нас есть два подходящих материализованных представления: MONTHLY_SALES_MV, вычисляющее суммарный объем продаж по месяцам, и YEARLY_SALES_MV, вычисляющее суммарный объем продаж по годам. Для переписывания следующего ниже запроса можно ожидать выбора последнего из них, поскольку оно меньше. Тем не менее, вы можете принудительно заставить переписывать запрос с использованием MONTHLY_SALES_MV с помощью инструкции.

```
SELECT t.year, p.product_id, SUM(ps.purchase_price) sum_of_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
      ps.product_id = p.product_id
GROUP BY t.year, p.product_id;

-------------------------------------------------------------------
|Id| Operation          | Name            | Rows | Bytes | Cost |
-------------------------------------------------------------------
| 0| SELECT STATEMENT   |                 | 82   | 2624  | 2    |
| 1|  TABLE ACCESS FULL | YEARLY_SALES_MV | 82   | 2624  | 2    |
-------------------------------------------------------------------
```

```
SELECT /*+ REWRITE(monthly_sales_mv) */ t.year, p.product_id,
     SUM(ps.purchase_price) as sum_of_sales
FROM time t, product p, purchases ps
WHERE t.time_key = ps.time_key AND
     ps.product_id = p.product_id
GROUP BY t.year, p.product_id;

-------------------------------------------------------------------------
|Id| Operation                 | Name             | Rows | Bytes |Cost |
-------------------------------------------------------------------------
| 0| SELECT STATEMENT          |                  |  115 |  2415 |  10 |
| 1|  SORT GROUP BY            |                  |  115 |  2415 |  10 |
| 2|   HASH JOIN               |                  |  589 | 12369 |   7 |
| 3|    VIEW                   |                  |    5 |    35 |   4 |
| 4|     SORT UNIQUE           |                  |    5 |    35 |   4 |
| 5|      TABLE ACCESS FULL    |TIME              |   15 |   105 |   2 |
| 6|    TABLE ACCESS FULL      |MONTHLY_SALES_MV  |  825 | 11550 |   2 |
-------------------------------------------------------------------------
```

4.6.11. Переписывание запросов и слежение за изменениями разделов

Выше мы обсуждали возможность Oracle 9i по слежению за изменениями разделов, которая позволяет быстро обновлять материализованные представления после операций над разделами. Также, слежение за изменениями разделов повышает для материализованного представления возможности по переписыванию запросов. Как правило, когда обновляется основная таблица, материализованное представление становится устаревшим и не может использоваться системой переписывания запросов в режимах ENFORCED и TRUSTED. Тем не менее, если основная таблица секционирована, а материализованное представление поддерживает слежение за изменениями разделов таблицы (включая в себя ключ или маркер секционирования для данной таблицы), то база Oracle может определить, является ли данная часть материализованного представления свежей. Если ответ на запрос может быть дан с использованием только этой, свежей части материализованного представления, система переписывания запросов будет использовать это материализованное представление. Например, на рис. 4.4 (см. раздел 4.3.3) свежая часть материализованного представления соответствует разделам за январь 2002 и март 2002. Нельзя использовать ту часть

материализованного представления, которая относится к февралю 2002 (раздел с изменением) и апрелю 2002 (новый раздел).

Рассмотрим следующее материализованное представление, содержащее данные об объемах продаж продуктов. Таблица покупок purchases секционирована по ключу времени time_key, который включен в материализованное представление.

```
CREATE MATERIALIZED VIEW product_category_sales_mv
ENABLE QUERY REWRITE
AS
SELECT ps.time_key, p.category,
     SUM(ps.purchase_price) as sum_of_sales
FROM product p, purchases ps
WHERE ps.product_id = p.product_id
GROUP BY ps.time_key, p. category;
```

Если мы просмотрим представление user_mviews, мы увидим, что материализованное представление является свежим (FRESH).

```
SELECT staleness FROM user_mviews
WHERE mview_name = 'PRODUCT_CATEGORY_SALES_MV';

STALENESS
-------------------
FRESH
```

Материализованное представление содержит только данные по март 2002. Теперь предположим, что мы добавили к таблице покупок новый раздел и загрузили данные за апрель 2002.

```
ALTER TABLE purchases ADD PARTITION purchases_apr2002
     values less than (TO_DATE('01-05-2002', 'DD-MM-YYYY'));

INSERT INTO purchases VALUES ( 'SP1063','2-APR-2002',
     'AB123457','7-APR-2002', 0024,28.01, 4.50, 'N');
INSERT INTO purchases VALUES ( 'SP1064','2-APR-2002',
     'AB123457','8-APR-2002', 1024,28.01, 4.50, 'N');
...

COMMIT;
```

Если мы теперь посмотрим представление user_mviews, мы увидим, что материализованное представление устарело.

```
SELECT staleness FROM user_mviews
WHERE mview_name = 'PRODUCT_CATEGORY_SALES_MV';

STALENESS
-------------------
STALE
```

Теперь рассмотрим следующий запрос, в котором запрашивается суммарный объем продаж с января по март 2002. Оптимизатор базы Oracle определит, что для выполнения запроса необходим доступ только к разделам таблицы purchases за январь, февраль и март 2002 года. Поскольку материализованное представление для данной таблицы доступно для системы слежения за изменениями разделов, база Oracle определит, что в отношении этих разделов материализованное представление является свежим. Поэтому запрос может быть переписан под использование материализованного представления, как это показано в приведенном плане обработки запроса.

```
SELECT ps.time_key, p.category,
     SUM(ps.purchase_price) as sum_of_sales
FROM product p, purchases ps
WHERE ps.product_id = p.product_id and
     ps.time_key BETWEEN TO_DATE('01-01-2002', 'DD-MM-YYYY').
          AND TO_DATE('31-03-2002', 'DD-MM-YYYY')
GROUP BY ps.time_key, p.category;
-------------------------------------------------------------------------
|Id| Operation              |Name              I Rows  I Bytes I Cost  I
-------------------------------------------------------------------------
|0 | SELECT  STATEMENT      |                  | 1     | 26    | 2     |
|1 |  TABLE ACCESS  FULL    |PRODUCT_CATEGORY_| 1     |  26   | 2     |
|  |                        |         SALES_MV |       |       |       |
-------------------------------------------------------------------------
```

С другой стороны, ответ на запрос, который требует суммарный объем продаж за апрель 2002, как показано в следующем примере, будет дан по основным данным.

```
SELECT ps.time_key, p.category,
     SUM(ps.purchase_price) as sum_of_sales,
FROM product p, purchases ps
WHERE ps.product_id = p.product_id and
     ps.time_key BETWEEN TO_DATE('01-01-2002', 'DD-MM-YYYY')
          AND TO_DATE('30-04-2002', 'DD-MM-YYYY')
GROUP BY ps.time_key, p. category;

-------------------------------------------------------------------------
|Id  | Operation                | Name        I  Rows  I Bytes I Cost  I
-------------------------------------------------------------------------
| 0  | SELECT STATEMENT         |             |     43  | 1376 | 442   |
| 1  |  SORT GROUP BY           |             |     43  | 1376 | 442   |
| 2  |   HASH JOIN              |             |  94113  | 2941KI  43    |
|
| 3  I    TABLE ACCESS FULL    | PRODUCT     |    162  | 1944 |   2   |
| 4  |    PARTITION RANGE ALL   |             |         |      |       |
| 5  |     TABLE ACCESS FULL    I PURCHASES   |  94113  | 1838KI  40    |
-------------------------------------------------------------------------
```

4.6.12. Переписывание запросов и группировки GROUPING SETS

SQL-операторы агрегации, CUBE, ROLLUP и GROUPING SETS предоставляют механизм для вычисления в одном запросе нескольких группировок. Вы можете создать материализованное представление с помощью запроса, содержащего эти операторы, и хранить несколько уровней агрегации, вместо того чтобы создавать материализованные представления для каждого уровня. Переписывание запросов можно использовать для запросов по любому из этих уровней агрегации.

В следующем примере показано материализованное представление с операторами GROUPING SETS, которое вычисляет суммарный объем продаж по трем группировкам (category, time_key), (category, time_key, town) и (time_key, country). Заметим, что материализованное представление должно содержать функцию GROUPING_ID или GROUPING для группы по столбцам, чтобы можно было распознавать строки, относящиеся к разным группам.

```
CREATE MATERIALIZED VIEW sales_mv
ENABLE QUERY REWRITE
AS
SELECT p.category, t.time_key, c.county, c.town,
      SUM(f.purchase_price) sales,
      GROUPING_ID(p.category, t.time_key, c.town, c.county) gid
FROM product p, purchases f, time t, customer c
WHERE p.product_id = f.product_id AND
      t.time_key = f.time_key AND
     . c.customer_id = f.customer_id
GROUP BY GROUPING SETS ( (p.category, t.time_key),
(p.category, t.time_key, c.town),
(t.time_key, c.county));
```

Переписывание запроса с простым условием GROUP BY

Это материализованное представление можно использовать для переписывания запроса, которому требуется любая группировка из имеющихся в материализованном представлении, или которая может быть получена из него путем вложений. Например, следующий запрос, которому требуется общий объем продаж по категориям и ключу времени, может быть переписан с использованием материализованного представления SALES_MV.

```
SELECT p.category, t.time_key, SUM(f.purchase_price) sales
FROM product p, purchases f, time t, customer c
WHERE p.product_id = f.product_id AND
      t.time_key = f.time_key AND
     , c.customer_id = f.customer_id
GROUP BY p.category, t.time_key;
```

Из информации о предикате в выходных данных процедуры EXPLAIN PLAN мы видим, что было выполнено переписывание путем отбора строк с GID=3, что соответствует группировке (p.category, t.time_key).

```
-------------------------------------------------------------
I  Id I Operation          I Name     I Rows  I Bytes I Cost I
-------------------------------------------------------------
I   0 I SELECT STATEMENT   I          I    1 |    39 I    2 I
|* 1|  TABLE ACCESS FULL  I SALES_MV |    1 I    39 I    2 |
-------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter("SALES_MV"."GID"=3)
```

Переписывание запросов с множественной группировкой

Если в запросе используется множественная группировка, база Oracle попытается найти материализованное представление, удовлетворяющее всем группировкам. Например, следующий запрос может быть переписан под материализованное представление SALES_MV, созданное ранее. Выходные данные процедуры EXPLAIN PLAN показывают, что при переписывании запроса были использованы две группировки материализованного представления — gid = 3, соответствующее (p.category, t.time_key), и gid = 10, которая соответствует группировке (t.time_key, c.county).

```
SELECT p.category, t.time_key, c.county,
      SUM(f.purchase_price) sales
FROM product p, purchases f, time t, customer c
WHERE p.product_id = f.product_id AND
      t.time_key = f.time_key AND
      c.customer_id = f.customer_id
GROUP BY GROUPING SETS ((p.category, t.time_key),
      (t.time_key, c.county));

-------------------------------------------------------------------
|Id | Operation            | Name     | Rows | Bytes | Cost |
-------------------------------------------------------------------
| 0 | SELECT STATEMENT     |          |    1 |    46 |    2 |
|* 1|  TABLE ACCESS FULL   | SALES_MV |    1 |    46 |    2 |
-------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
1 - filter("SALES_MV"."GID"=3 OR "SALES_MV"."GID"=10)
```

Переписывание запросов с использованием нескольких материализованных представлений

Если база Oracle не может найти одного отвечающего на запрос материализованного представления, она попытается переписать каждую группу отдельно.

(Заметим, что запрос с операторами GROUPING SETS можно выразить в виде UNION ALL всех запросов с отдельными группировками). В результате для переписывания запросов может использоваться несколько материализованных представлений, а некоторые группы могут остаться непереписанными и использовать основные данные. Такое преобразование переписывания запросов впервые появляется в Oracle 9i release 2.

Приведенный ниже запрос нельзя переписать с использованием одного только материализованного представления SALES_MV. Группировка (p.category, t.time_key) в этом представлении присутствует, а группировку (t.time_key, c.town) можно получить путем вложения (p.category, t.time_key, c.town). Однако группировки (p.category, c.county, t.year) в этом материализованном представлении нет.

```
SELECT p.category, t.time_key, c.county, t.year, c.town,
  SUM(f.purchase_price) sales
FROM product p, purchases f, time t, customer c
WHERE p.product_id = f.product_id AND
      t.time_key = f.time_key AND
      c.customer_id = f.customer_id
GROUP BY GROUPING SETS ((p.category, t.time_key),
     (t.time_key, c.town),
          (p.category, c.county, t.year));
```

```
-------------------------------------------------------------------------
|  Id I Operation                  | Name      I Rows I Bytes  I Cost I
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |           |    5 |   290   |  120 |
|   1 |  VIEW                      |           |   •5 |   290   I  120 |
|   2 |   UNION-ALL                |           |      |         |      |
|*  3 |    TABLE ACCESS FULL       | SALES_MV  |    1 |    39   |    2 |
|   4 |    SORT GROUP BY           |           |    1 |    47   I    4 |
|*  5 |     TABLE ACCESS FULL      | SALES_MV  |    1 |    47   |    2 |
|   6 |    SORT GROUP BY           |           |    3 |   186   |  114 |
|*  1 |     HASH JOIN              |           | 13374 |   809K |   16 |
|   8 |      TABLE ACCESS FULL     | PRODUCT   |  162 |  1944   |    2 |
|*  9 |       HASH JOIN            |           | 13374 |   653K |   13 |
|  10 |        TABLE ACCESS FULL   | CUSTOMER  |   82 |   820   |    2 |
|* 11 |        HASH JOIN           |           | 13374 |   522K |   10 |
|  12 |         TABLE ACCESS FULL  | TIME      |    7 |    84   |    2 |
|  13 |         TABLE ACCESS FULL  | PURCHASES | 13379 |  365K I    7 |
-------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------
3 - filter("SALES_MV"."GID"=3)
5 - filter("SALES_MV"."GID"=1)
7 - access("P"."PRODUCT_ID"="F"."PRODUCT_ID")
9 - access("C"."CUSTOMER_ID"="F"."CUSTOMER_ID")
11 - access("T"."TIME_KEY"="F"."TIME_KEY")
```

В выходных данных процедуры EXPLAIN PLAN показано, что переписывание запросов производилось с помощью материализованного представления SALES_MV для двух группировок (gid = 3 и gid = 1) и с помощью основных таблиц для группировки (p.category, c.county, t.year).

Теперь предположим, что у нас есть простое материализованное представление SALES_MV2, в котором есть группировка (p.category, c.county, t.year).

```
CREATE MATERIALIZED VIEW sales_mv2
ENABLE QUERY REWRITE
AS
SELECT p.category, c.county, t.year, SUM(f.purchase_price) sales
FROM product p, purchases f, time t, customer c
WHERE p.product_id = f.product_id AND
      t.time_key = f.time_key AND
      c.customer_id = f.customer_id
GROUP BY p.category, c.county, t.year;
```

Теперь для переписывания запросов база Oracle может использовать комбинацию SALES_MV и SALES_MV2, как показано в приводимом плане обработки запроса.

```
-------------------------------------------------------------------
|Id  | Operation            | Name       | Rows | Bytes | Cost |
-------------------------------------------------------------------
|  0 | SELECT STATEMENT     |            |   84 |  4872 |    8 |
|  1 |  VIEW                |            |   84 |  4872 |    8 |
|  2 |   UNION-ALL          |            |      |       |      |
|* 3 |    TABLE ACCESS FULL | SALES_MV   |    1 |    39 |    2 |
|  4 |    SORT GROUP BY     |            |    1 |    47 |    4 |
|* 5 |     TABLE ACCESS FULL| SALES_MV   |    1 |    47 |    2 |
|  6 |    TABLE ACCESS FULL | SALES_MV2  |   82 |  3034 |    2 |
-------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------
3 - filter("SALES_MV"."GID"=3)
5 - filter("SALES_MV"."GID"=1)
```

4.6.13. Поиск проблем в переписывании запросов

В предыдущих примерах мы использовали процедуру EXPLAIN PLAN, показывающую, что запрос был переписан под использование материализованного представления. Однако иногда можно обнаружить, что запрос не был переписан, как мы того ожидали. Правила, управляющие переписыванием запросов, могут быть чрезвычайно сложными, а причины отсутствия использования материализованных представлений могут быть неочевидны. В некоторых случаях причина совершенно тривиальна, например неустановленный в TRUE параметр QUERY_REWRITE_ENABLED. В других случаях причина может быть более трудноуловимой, например, нет ограничения, или оно не проверено, или же требуемый в запросе столбец отсутствует в материализованном представлении. В Oracle 9i предлагается утилита, которая помогает диагностировать причины таких нарушений переписывания запросов. Эта утилита PL/SQL API DBMS_MVIEW.EXPLAIN_REWRITE.

Чтобы использовать утилиту EXPLAIN_REWRITE, вы указываете запрос и, по желанию, материализованное представление, которое нужно использовать для переписывания. Эта процедура сообщит вам, будет ли данный запрос использовать это материализованное представление и, если нет, то почему. Перед тем как использовать эту процедуру, вам нужно создать в своей схеме таблицу с именем REWRITE_TABLE с помощью скрипта utlxrw.sql, который находится в директории rdbms/admin. В эту таблицу будут помещены результаты выполнения утилиты EXPLAIN_REWRITE. Существует также интерфейс типа varray, который позволяет осуществлять доступ к результатам через программу PL/SQL.

Теперь мы проиллюстрируем использование этой утилиты. В первом примере пользователь забыл установить параметр QUERY_REWRITE_ENABLED в TRUE. Для диагностики этой проблемы вы выполняете процедуру EXPLAIN_REWRITE и забираете результаты из таблицы REWRITE_TABLE.

```
BEGIN
 dbms_mview.explain_rewrite('
 SELECT t.month, t.year, p.product_id,
```

```
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales
 FROM time t, product p, purchases ps
 WHERE t.time_key = ps.time_key AND
 ps .product_id = p.product_id
 GROUP BY t.month, t.year, p.product_id' , ' MONTHLY_SALES_MV' ) ;
END;
/

SELECT mv_name, message FROM rewrite_table;

MV_NAME                        ; MESSAGE
---------------    ------------------------------
MONTHLY_SALES_MV   QSM-01001: query rewrite not enabled
```

Утилиту EXPLAIN_REWRITE можно использовать для выяснения того, почему данное материализованное представление не используется при переписывании запроса. Запрос, приведенный в следующем примере, уже использовался ранее, чтобы проиллюстрировать использование ограничений в переписывании запросов. Материализованное представление имеет дополнительные соединения между таблицей покупок (purchases) и таблицей времени (time). Однако в этом примере ограничение flctime не помечено условием RELY, и поэтому соединение таблиц покупок и времени не может быть определено как соединение без потерь (lossless join). Следовательно, переписывание запроса не удается. Без этой утилиты такую проблему было бы очень трудно диагностировать.

```
BEGIN
dbms_mview.explain_rewrite('
SELECT p.product_id,
      SUM (ps.purchase_price) as sum_of_sales,
      COUNT (ps.purchase_price) as total_sales
   FROM product p, purchases ps
   WHERE p.product_id = ps .product_id
   GROUP BY p.product_id', ' MONTHLY_SALES_MV' ) ;
END;
/

SELECT mv_name, message FROM rewrite_table;
```

```
MV_NAME                    MESSAGE
-------------------------  ----------------------------------------
MONTHLY_SALES_MV           QSM-01110: a lossy join in MV,
                           MONTHLY_SALES_MV, between tables,
                           PURCHASES and TIME, not found in query

MONTHLY_SALES_MV           QSM-01053: NORELY referential integrity
                           constraint/ on table, PURCHASES, in
                           TRUSTED/STALE TOLERATED integrity mode
```

Теперь давайте исправим эту проблему и снова запустим утилиту EXPLAIN_REWRITE. Теперь мы увидим, что запрос переписывается.

```
ALTER TABLE purchases MODIFY constraint fk_time rely;

BEGIN
dbms_mview.explain_rewrite('
SELECT p.product_id,
     SUM (ps.purchase_price) as sum_of_sales,
     COUNT (ps.purchase_price) as total_sales
   FROM product p, purchases ps
   WHERE p.product_id = ps .product_id
   GROUP BY p.product_id', 'MONTHLY_SALES_MV');
 END;
/

SELECT mv_name, message FROM rewrite_table;

MV_NAME                    MESSAGE
-------------------------  ----------------------------------
MONTHLY_SALES_MV           QSM-01033: query rewritten with
                           materialized view,
                           MONTHLY_SALES_MV
```

Иногда переписывание запросов с помощью затребованного материализованного представления может быть возможным, но при этом может быть лучше использовать другое материализованное представление. Предположим, мы создали материализованное представление PRODUCT_SALES_EXACT_MATCH для запроса, приведенного в предыдущем примере, настроенное на точное соот-

ветствие тексту запроса. Теперь, вместо предыдущего, система переписывания запросов будет использовать это материализованное представление, поскольку оно более оптимально. Утилита EXPLAIN_REWRITE скажет вам, если у вас именно такой случай.

```
begin
dbms_mview.explain_rewrite('
SELECT p.product_id,
     SUM(ps.purchase_price) as sum_of_sales,
     COUNT(ps.purchase_price) as total_sales
  FROM product p, purchases ps
  WHERE p.product_id = ps.product_id
  GROUP BY p.product_id', 'MONTHLY_SALES_MV');
end;
/

SELECT mv_name, message FROM rewrite_table;

MV_NAME                        MESSAGE
------------------------------ --------------------------------
MONTHLY_SALES_MV               QSM-01009: materialized view,
                               PRODUCT_SALES_EXACT_MATCH,
                               matched query text
```

4.7. Summary Advisor

Теперь, когда мы обсудили все преимущества, которые дают материализованные представления, мы подходим к такому вопросу: как узнать, какое материализованное представление создавать. Приложения, связанные с принятием решений, обычно используют четко определенный набор запросов. Простейшим решением для повышения производительности этих запросов будет создать по одному материализованному представлению для каждого запроса. На практике же требования к свободному месту на диске и ресурсам для обновлений скоро становятся непомерными. При помощи системы переписывания запросов вы можете использовать одно и то же материализованное представление для переписывания обширного класса запросов. Однако определение оптимального набора создаваемых материализованных представлений для группы запросов может оказаться довольно хитрой задачей. Инструмент Summary Advisor (Консультант по сводкам) создан

для того, чтобы помочь в решении проблемы выбора материализованных представлений для прикладной задачи.

Summary Advisor можно использовать при выяснении того, какие материализованные представления нужно создать для данного набора запросов. В Oracle 9i Summary Advisor тесно интегрирован с системой переписывания запросов и у него есть информация о разнообразных правилах переписывания. Summary Advisor оптимизирует материализованные представления, стараясь уложиться в лимиты места на диске, установленные пользователем.

Чтобы работать с Oracle 9i Summary Advisor, в вашей базе данных должно быть разрешено использование Java, должна быть собрана статистическая информация по всем нужным таблицам и их столбцам. Если вы определите в своей схеме ограничения и измерения, выражающие ваши бизнес-правила, Summary Advisor даст лучшие рекомендации, поскольку сможет воспользоваться преимуществами более тонких правил переписывания, которые обсуждались ранее.

Существует два способа запуска Summary Advisor: с помощью мастера Summary Advisor Wizard в Oracle Enterprise Manager и с помощью пакета DBMS_OLAP из SQL*Plus.

4.7.1. Summary Advisor Wizard

В Oracle Enterprise Manager предлагается мастер Summary Advisor Wizard, который проведет вас через выполнение этапов, необходимых для анализа. Добраться до Summary Advisor Wizard можно, щелкнув по пункту Warehouse (Хранилище), затем — Summary Management (Управление сводными данными), а потом выбрав пункт Summary Advisor (Консультант по сводкам) из меню Object (Объект). Когда анализ закончится, вы можете принять некоторые или все рекомендации и реализовать их. Давайте рассмотрим типичный случай работы с Summary Advisor Wizard.

Первый этап — это сообщить мастеру о том, какие операторы SQL используются в вашей задаче. Это называется рабочей нагрузкой (workload). Рабочую нагрузку можно получать из разных источников, например кэша SQL в базе Oracle, системы трассировки или же из пользовательской таблицы. Summary Advisor в процессе составления набора материализованных представлений изучит правила переписывания запросов и статистические данные об операторах (такие как приоритет, время выполнения и частоту встречаемости).

На Рисунке 4.9 показано окно Summary Advisor Wizard, предназначенное для ввода данных о рабочей нагрузке.

Поскольку в рабочую нагрузку могут входить несколько операторов, в Summary Advisor предусмотрен механизм отбора наиболее существенных с помощью

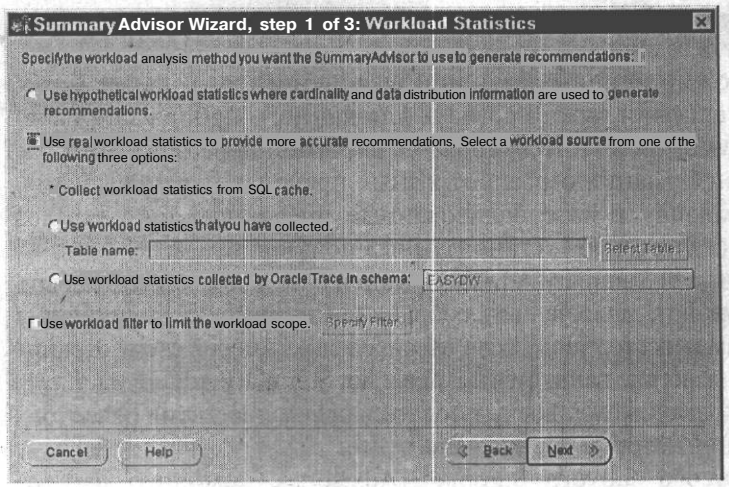


Рис. 4.9. *Summary Advisor Wizard — ввод рабочей нагрузки*

фильтра. Фильтр можно настроить на разные критерии, например на пользователя, на таблицы, используемые в запросе, на время ответа или частоту. Фильтр можно применять либо при вводе рабочей нагрузки, либо в процессе получения рекомендаций. На рис. 4.10 мы настраиваем фильтры так, чтобы рассматривались только SQL-запросы, подаваемые пользователем EASYDW и к таблицам CUSTOMERS и PURCHASES. Затем у вас будет возможность выбрать таблицы фактов и пометить для сохранения уже имеющиеся материализованные представления.

Когда анализ закончен, рекомендации отображаются графически, как это показано на рис. 4.11. На оси X графика показано используемое дисковое пространство, а на оси Y — совокупная производительность для данной рекомендации. Пользователь, применяя движущуюся стрелку, может указать, сколько места на диске он может выделить.

Рекомендации, удовлетворяющие выбранным лимитам дискового пространства, можно просмотреть подробно, щелкнув мышкой по кнопке View/Modify Recommendations (Просмотреть/Изменить рекомендации). На рис. 4.12показано окно мастера, посвященное рекомендациям. Пользователь может изменить имя материализованного представления и опции всех рекомендованных материализованных представлений.

Когда необходимые изменения сделаны, вы можете нажать кнопку Next (Далее) в окне, показанном на рис. 4.11, и увидеть код SQL, необходимый для реализации рекомендаций (показано на рис. 4.13). Кроме того, вы можете сохранить

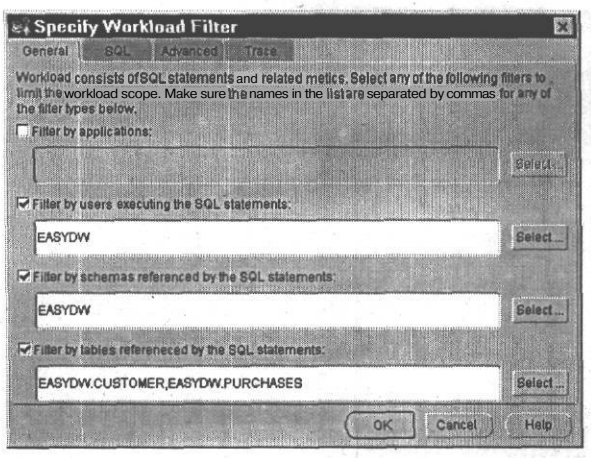


Рис. 4.10. *Summary Advisor Wizard — Фильтры*

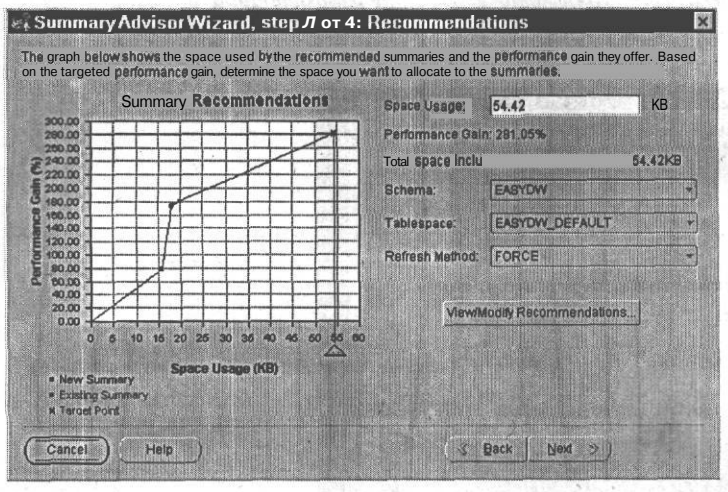
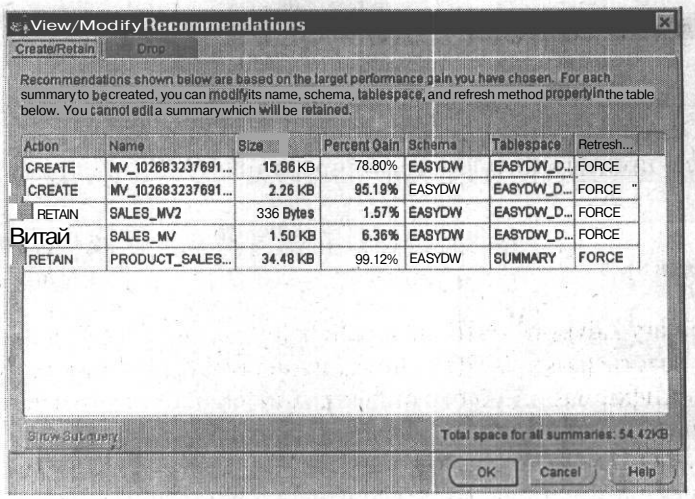


Рис. 4.11. *Summary Advisor Wizard — график рекомендации*

сгенерированный код в файле. После нажатия кнопки Finish (Готово), рекомендации будут реализованы. Чтобы реализовывать рекомендации, вы должны иметь необходимые привилегии.

Теперь мы рассмотрим, как можно запускать Summary Advisor из командной строки.



Витай

Рис. 4.12. *Summary Advisor Wizard — Просмотр/Изменение рекомендаций*

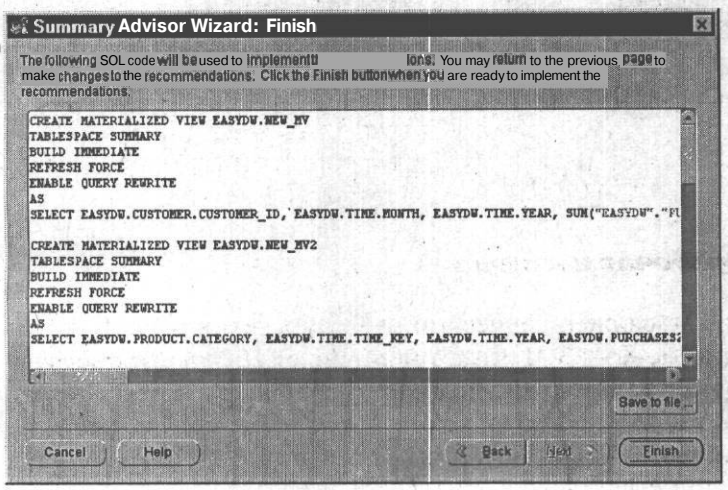


Рис. 4.13. *Summary Advisor Wizard — реализация рекомендаций*

4.7.2. Использование пакета DBMS_OLAP

Summary Advisor можно запускать без графического интерфейса, с помощью процедур пакета DBMS_OLAP. Мы покажем вам типичный сеанс работы с использованием интерфейса прикладного программирования (API) Summary Advisor.

Ввод рабочей нагрузки

Первый этап запуска Summary Advisor — это ввод рабочей нагрузки (workload). Рабочую нагрузку можно ввести из кэша SQL, пользовательской таблицы или сгенерировать с помощью информации об ограничениях и измерениях, имеющихся в базе данных. В следующем примере мы вводим рабочую нагрузку из кэша SQL с помощью процедуры LOAD_WORKLOAD_CACHE. Перед тем как испрльзовать API для ввода рабочей нагрузки, вы должны создать идентификатор этой рабочей нагрузки с помощью процедуры CREATE_ID. Этот идентификатор затем будет передан в процедуру RECOMMEND_MVIEW_STRATEGY.

```
Variable  workload_id  number;
EXECUTE DBMS_OLAP.CREATE_ID(:workload_id);

EXECUTE DBMS_OLAP.LOAD_WORKLOAD_CACHE  (:workload_id,
                                        DBMS_OLAP.WORKLOAD_NEW,
                                        DBMS_OLAP.FILTER_NONE,
                                        'Easydw',1);
```

Пользовательская рабочая нагрузка

Иногда у вас может оказаться специфический набор инструкций SQL, которые вы хотели бы настроить. Кэш SQL может не точно отражать эти инструкции. В этом случае вы можете поместить инструкции в таблицу и использовать процедуру LOAD_WORKLOAD_USER для того, чтобы передать эту рабочую нагрузку в Summary Advisor. В пользовательской таблице рабочей нагрузки должен, как минимум, определяться текст запроса и пользователь, который его выполняет. Чтобы получать лучшие рекомендации, следует предоставить некоторую статистическую информацию, такую, как частота или время ответа (если возможно).

В следующем примере показан формат пользовательской таблицы рабочей нагрузки и то, как ее загружать.

```
CREATE TABLE my_workload
 (query                          varchar2(2000)      not null,
  owner                          varchar2(30) not null,
  application                    varchar2(30),
  frequency                      number,
  lastuse                        number,
  priority                       number,
  responsetime                   number,
  resultsize                     number,
  sql_addr                       number,
  sql_hash                       number
);
<insert workload statements into my_workload>

variable workloa'd_id number;

execute dbms_olap.create_id (:workload_id);

execute DBMS_OLAP.LOAD_WORKLOAD_USER  (:workload_id,
                                       DBMS_OLAP.WORKLOAD_NEW,
                                       DBMS_OLAP.FILTER_NONE,
                                       'EASYDW', 'MY_WORKLOAD');
```

Как только рабочая нагрузка введена, вы проходите через следующие этапы, которые обсуждаются ниже, и получаете рекомендации.

Фильтры

Как уже говорилось ранее, фильтры можно использовать для того, чтобы уменьшить число анализируемых запросов. Фильтры могут основываться на таблицах или схемах, на которые ссылается запрос, частоте, времени ответа, кардинальном числе таблиц или определенном пользователем поле приоритета (для пользовательской рабочей нагрузки). В приводимом примере мы создадим фильтр, который выбирает запросы, содержащие таблицы схемы EASYDW. Это сужает область исследования для Summary Advisor, уменьшая, таким образом, время анализа. Далее мы выбираем только те запросы, которые возникали, по меньшей мере, десять раз. Как и в случае рабочей нагрузки, вы должны создать идентификатор фильтра с помощью процедуры CREATE_ID. Этот идентификатор будет использоваться для передачи фильтра в процедуру создания рекомендации.

```
Variable filter_id number;
EXECUTE DBMS_OLAP.CREATE_ID(:filter_id);

EXECUTE DBMS_OLAP.ADD_FILTER_ITEM(:filter_id, 'SCHEMA', 'EASYDW',
            NULL, NULL, NULL, NULL);

EXECUTE DBMS_OLAP.ADD_FILTER_ITEM (:filter_id, 'FREQUENCY',
            NULL, 10, NULL, NULL, NULL);
```

Получение рекомендации по материализованным преставлениям

Теперь все готово к запуску процесса выдачи рекомендаций. Чтобы сгенерировать рекомендации, мы используем процедуру DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY. Перед запуском этой процедуры вы должны сгенерировать идентификатор запуска (run_id) с помощью процедуры CREATE_ID. Идентификатор используется для поиска результатов анализа в представлении каталога SYSTEM.MVIEW_RECOMMENDATIONS.

Процедуре RECOMMEND_MVIEW_STRATEGY необходимо предоставить рабочую нагрузку и фильтры, используя для этого соответствующие идентификаторы — workload_id и filter_id, сгенерированные ранее. При желании, вы можете также указать лимиты дискового пространства для материализованных представлений (в приводимом примере — 500MB) и список материализованных представлений, которые не следует отбрасывать.

```
Variable run_id number;
EXECUTE DBMS_OLAP.CREATE_ID(:run_id) ;

EXECUTE DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY(:run_id, :workload_id,
              :filter_id, 500000, 100,
             NULL, NULL);
```

Отчеты и скрипты Summary Advisor

Как только рекомендации сгенерированы, вы можете создать отчет или скрипт, используя процедуры DBMS_OLAP.GENERATE_MVIEW_REPORT или GENERATE_MVIEW_SCRIPT, соответственно. В скрипте содержатся инструкции SQL, используемые для генерации рекомендуемых материализованных представлений.

Чтобы сгенерировать скрипт и файлы отчета, пользователь, запускающий эти процедуры, должен иметь права на чтение и запись в специфические дирек-

тории. Чтобы дать пользователю EASYDW права на чтение и запись директории h:\oracle\reports нужно выполнить следующую команду:

```
EXECUTE DBMS_JAVA.GRANT_PERMISSION('EASYDW',
        'java.io.FilePermission',
        'h:\oracle\reports\*',
        'read, write');
```

Следующий фрагмент кода представляет собой скрипт, сгенерированный Summary Advisor для предыдущего примера. Мы указали, что скрипт должен использовать для сгенерированных операторов создания материализованных представлений табличное пространство SUMMARY. Скрипт размещается в директории, разрешения на чтение и запись в которой были даны выше.

```
EXECUTE DBMS_OLAP.GENERATE_MVIEW_SCRIPT ('h:\oracle\reports\mview_rec.sql',
                        :run_id, 'SUMMARY');
/*****************************************************************
** Oracle Summary Advisor 9i - Production
**
** Summary Advisor Recommendation Script
*****************************************************************/

/*****************************************************************
** Recommendations for run ID #59
*****************************************************************/

/*****************************************************************
** Rank 1
** Storage 16,240 bytes
** Gain 78.72%
** Benefit Ratio 201,344.00
**
** SELECT easydw.customer.customer_id, easydw.time.month, easydw.time.year,
**       sum(easydw.purchases.purchase_price), count(
**       easydw.purchases.purchase_price), count(*)
**    FROM easydw.purchases, easydw.time, easydw.customer
**    WHERE easydw.time.time_key = easydw.purchases.time_key
**      AND easydw.customer.customer_id = easydw.purchases.customer_id
**    GROUP BY easydw.customer.customer_id, easydw.time.month,
**       easydw.time.year
*****************************************************************/
```

```
CREATE MATERIALIZED VIEW mv_id_59_rank_1
 TABLESPACE summary
 BUILD IMMEDIATE
 REFRESH COMPLETE
 ENABLE QUERY REWRITE AS
   SELECT EASYDW.CUSTOMER.CUSTOMER_ID, EASYDW.TIME.MONTH,
EASYDW.TIME.YEAR,
SUM("EASYDW"."PURCHASES"."PURCHASE_PRICE"),
COUNT("EASYDW"."PURCHASES"."PURCHASE_PRICE"), COUNT(*) FROM
EASYDW.PURCHASES,
EASYDW.TIME,
EASYDW.CUSTOMER WHERE EASYDW.TIME.TIME_KEY = EASYDW.PURCHASES.TIME_KEY
AND
EASYDW.CUSTOMER.CUSTOMER_ID
= EASYDW.PURCHASES.CUSTOMER_ID GROUP BY EASYDW.CUSTOMER.CUSTOMER_ID,
EASYDW.TIME.MONTH,
EASYDW.TIME.YEAR;

<оставшаяся часть скрипта опущена>
```

Вы также можете получить подробный отчет по анализу, проведенному Summary Advisor, в виде HTML, выполнив процедуру DBMS_OLAP.GENERATE_MVIEW_REPORT, как это показано в приведенном примере. Вы можете настраивать отчет, включая в него различные разделы с подробной информацией, например данные о рабочей нагрузке, фильтрах, рекомендациях и использовании имеющихся материализованных представлений. Если идентификатор запуска run_id не передан, генерируется отчет по всем данным, относящимся к пользователю, запустившего процедуру. На рис. 4.14 показан раздел отчета Summary Advisor в web-браузере.

```
EXECUTE DBMS_OLAP.GENERATE_MVIEW_REPORT
      ('h:\oracle\reports\report_rec.html',
         :run_id, DBMS_OLAP.RPT_ALL);
```

Гипотетическая рабочая нагрузка

Если вы находитесь еще в процессе разработки приложения, у вас может не оказаться набора выполняемых операторов SQL. Вы, тем не менее, можете использовать Summary Advisor, если у вас есть схема для запросов и некоторая допол-

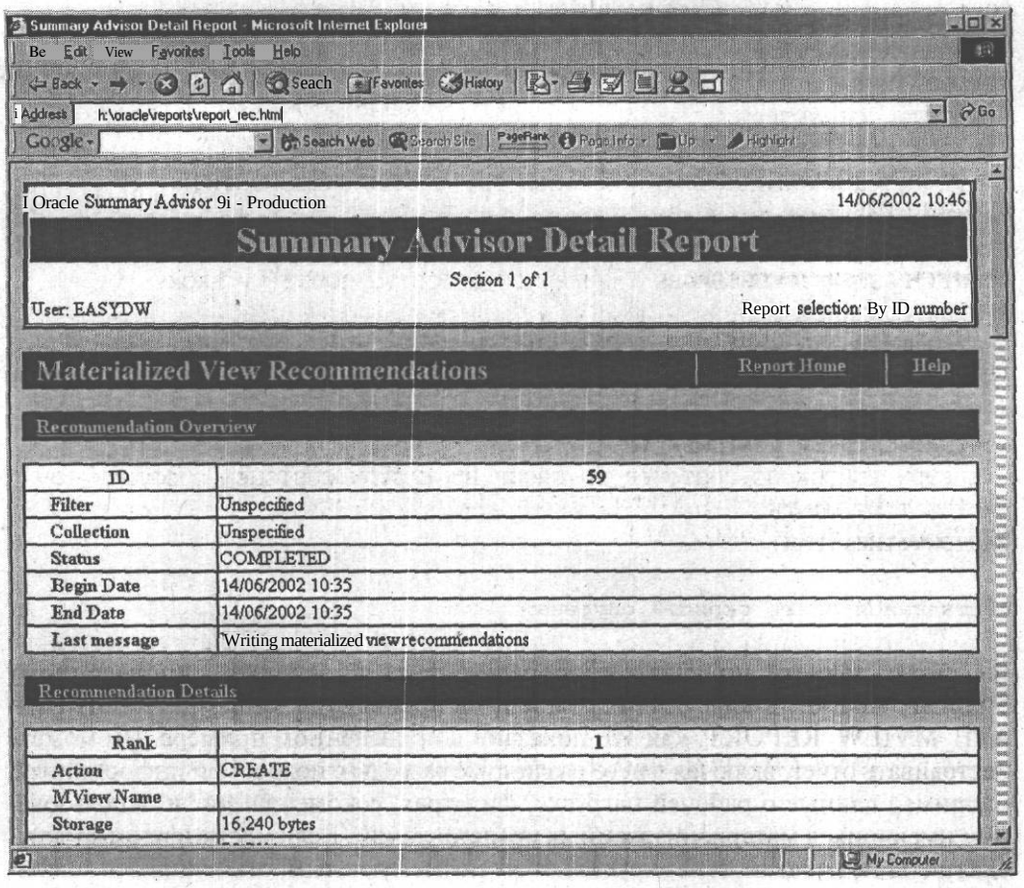


Рис. 4.14. *Отчет Summary Advisor*

нительная информация, например определения ограничений и измерений. В данном случае Summary Advisor определит гипотетическую рабочую нагрузку, используя взаимоотношения между различными столбцами, а потом рекомендует для этой нагрузки материализованные представления. Такой подход хорошо работает в звездообразных схемах, являющихся типичными для хранилищ. Чтобы сгенерировать рекомендации с использованием гипотетической рабочей нагрузки, передайте идентификатор рабочей нагрузки (workload_id) для DBMS_OLAP.WORKLOAD_NONE в процедуру DBMS_OLAP.RECOMMEND_MV

```
EXECUTE DBMS_OLAP.RECOMMEND_MVIEW_STRATEGY(:run_id,
                                           DBMS_OLAP.WORKLOAD_NONE,
                                           DBMS_OLAP.FILTER_NONE,
                                           500000, 100, NULL, NULL);
```

Вы можете просматривать рекомендации и генерировать отчеты и скрипты, как описано выше.

Очистка результатов

Вы можете очистить результаты анализа, вызвав процедуру DBMS_OLAP.PURGE_RESULTS.

```
EXECUTE DBMS_OLAP.PURGE_RESULTS(:run_id);
```

Чтобы очистить все результаты работы Summary Advisor, используйте процедуру DBMS_OLAP.RUNID_ALL.

```
EXECUTE DBMS_OLAP.PURGE_RESULTS(DBMS_OLAP.RUNID_ALL);
```

Вы можете удалить более ненужную рабочую нагрузку с помощью процедуры PURGE_WORKLOAD, передав ей конкретный идентификатор рабочей нагрузки или значение DBMS_OLAP.WORKLOAD_ALL (для удаления всех нагрузок). Точно так же вы можете удалить фильтр с помощью процедуры PURGE_FILTER.

Мониторинг использования существующих материализованных представлений

Иногда может понадобиться узнать, какие сводки находятся в использовании. Для администратора базы данных крупная проблема — определить, когда можно безопасно удалить сводку из хранилища. В пакете DBMS_OLAP есть процедура EVALUATE_MVIEW_STRATEGY, которая позволяет оценить использование имеющихся материализованных представлений. Это поможет определить, есть ли материализованные представления, не используемые при текущей рабочей нагрузке. Результаты этой оценки можно увидеть в представлении SYSTEM.MVIEW_EVALUATIONS.

```
EXECUTE DBMS_OLAP.EVALUATE_MVIEW_STRATEGY(:run_id, :workload_id,
                                          :filter_id);
```

```
SELECT runid, mview_owner, mview_name, storage, freq,
    cumulative_benefit as benefit
FROM SYSTEM.MVIEW_EVALUATIONS;

RUNID   MVIEW_OWNER    MVIEW_NAME                STORAGE   FREQ     BENEFIT
-----   -----------    ------------------        -------   ----  ----------
141     EASYDW         SALES_MV                    .1539     12    41351428
141     EASYDW         PRODUCT_SALES_BY_MON        35303      5     4295396
```

Оценка размера *материализованного представления*

Если вы создаете материализованное представление, будет неплохо знать, сколько времени оно, скорее всего, займет. Вы можете получить эту информацию, вызвав процедуру DBMS_OLAP.ESTIMATE_SUMMARY_SIZE, как это показано в следующем фрагменте кода.

Использовать эту процедуру очень легко: просто передайте ей инструкцию SELECT, используемую для создания материализованного представления.

```
SET SERVEROUTPUT on

DECLARE
no_of_rows NUMBER;
mv_size NUMBER;
BEGIN
no_of_rows :=0;
mv_size :=0;

DBMS_OLAP.estimate_summary_size ('purchases_by_county',
   'SELECT SUM(ps.purchase_price), ct.county
   FROM PURCHASES ps, CUSTOMER ct                    ' ,
   WHERE ps.customer_id = ct.customer_id
   GROUP BY ct.county' , no_of_rows, mv_size ) ,-

DBMS_OUTPUT.put_line ( 'SELECT on Purchases GROUP by County ');
DBMS_OUTPUT.put_line ( 'No of Rows: ' || no_of_rows ) ;
DBMS_OUTPUT.put_line ( 'Size of Summary (bytes): ' || mv_size) ;
DBMS_OUTPUT.put_line ( '');
END;
/
```

```
SELECT on Purchases GROUP by County
No of Rows: 3
Size of Materialized view (bytes): 96
```

В этом примере мы выводим количество строк и ожидаемый размер материализованного представления. Так же, отметьте, что в этом примере мы используем для отображения результатов работы пакета инструкцию

```
DBMS_OUTPUT.PUT_LINE
```

4.8. Заключение

Управление сводными данными в Oracle 9i предоставляет вам набор очень мощных инструментов для повышения производительности обработки Запроса. Материализованные представления можно использовать для предварительного вычисления результатов часто подаваемых запросов. При помощи системы переписывания запросов запросы могут явным образом переписываться под использование материализованных представлений. Быстрое обновление и система слежения за изменениями разделов позволяет поддерживать сводные таблицы в соответствии с данными, загружаемыми в хранилище. Измерения позволяют представлять иерархическую информацию о данных и пользоваться преимуществами вложений при переписывании запросов. Summary Advisor можно применять для получения рекомендаций о том, какие материализованные представления будут верными для вашей прикладной задачи.

В следующей главе мы подробно обсудим различные задачи, связанные с загрузкой данных в хранилище.

5

Загрузка данных в хранилище

5.1. ETL-процесс

Заполнение хранилища данными включает в себя все задачи, связанные с получением данных от рабочей системы-источника, их очисткой и преобразованием в нужный формат и нужную степень детализации, загрузку данных в хранилище и подготовку их к использованию в аналитических целях.

На рис. 5.1. показаны этапы выполнения извлечения, преобразования и загрузки данных (extraction, transformation, load, ETL). Данные извлекаются из рабочей системы-источника и транспортируются в промежуточную область (staging area). Эта область представляет собой место временного хранения данных для их подготовки. Промежуточная область может представлять собой набор простых последовательных файлов, временные таблицы в хранилище данных Oracle или и то, и другое сразу. Эти данные интегрируются с другими, очищаются и преобразуются в общий формат. Затем данные загружаются в целевые таблицы хранилища. Иногда этот процесс называют аббревиатурой ETT (от extraction, transportation, transformation, извлечение, транспортировка, преобразование).

В ходе первоначального заполнения хранилища загружаются данные предыстории. Данные в рабочих системах могут иметь разный формат. Если, например, рабочую систему кассовых терминалов меняли два года назад, данные последних двух лет будут иметь один формат, а данные старше двух лет — другой.

После загрузки первоначальных данных периодически нужно будет вносить данные о новых транзакциях и событиях. Как правило, это делается регулярно, по расписанию, например в конце дня, недели или месяца. В ходе загрузки и в то время, когда обновляются индексы и материализованные представления, данные, как правило, недоступны для выполнения запросов пользователей. Период времени, отведенный на внесение новых данных, называется «окном пакетной загрузки» (batch window). Это окно непрерывно уменьшается по мере того, как все

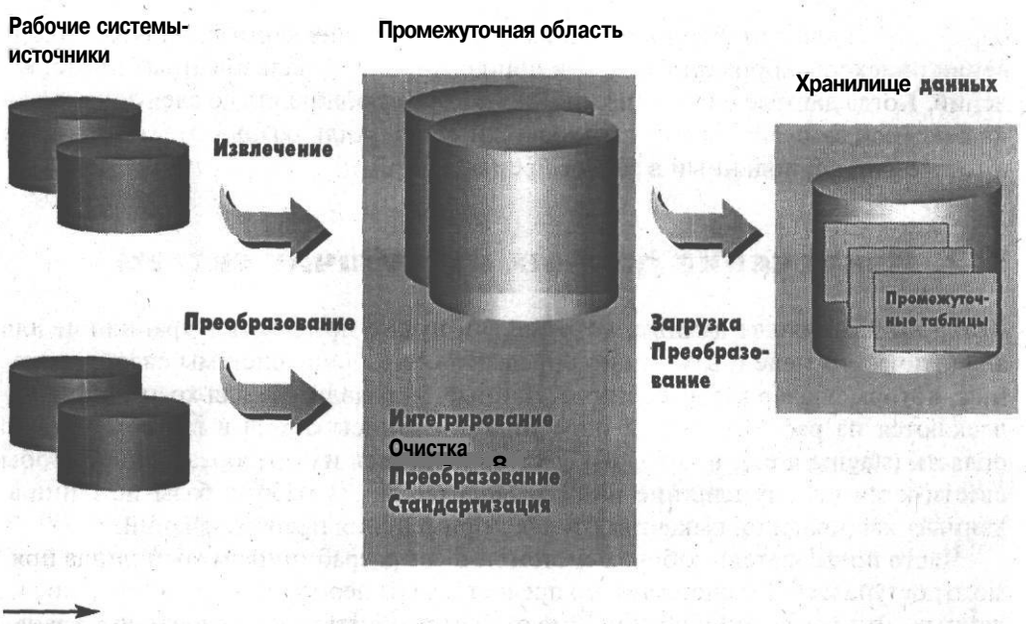


Рис. 5.1. *ETL-процесс*

больше он-лайновых услуг начинают занимать все больше времени. Более высокую доступность можно получить, проведя секционирование таблицы фактов. Пока данные загружаются в новый раздел, оставшиеся разделы остаются доступны для использования.

Значительная часть работы по созданию хранилища посвящена ETL-процессу. Поиск данных в рабочих системах, формирование процесса извлечения данных, транспорт данных, фильтрование, очистка, интегрирование данных из нескольких источников и загрузка данных в хранилище — все это занимает значительное время.

Если вы еще не знакомы с данными компании, то добавляется доля трудностей, связанных с пониманием данных. Один и тот же объект может иметь в разных системах разные названия. Что еще хуже, две разные вещи могут называться одинаково. Выявление всего этого может представлять собой трудную задачу. Для каждого столбца в хранилище нужно найти соответствующий столбец в системе-источнике. Соответствие некоторых данных будет прямым, другие данные нужно будет получить и перевести в другой формат.

Как только данные загружены в хранилище, до того как они будут «опубликованы» в готовом для использования виде, нужно произвести их дальнейшую

обработку с целью интегрирования их в существующие данные, а также обновления индексов, сбора статистики и обновления материализованных представлений. Когда данные опубликованы, их не следует обновлять до следующего окна пакетной загрузки, чтобы пользователи не получали разных ответов на один и тот же запрос, поданный в разное время.

5.2. Извлечение данных из рабочих систем

Когда вы выявили те данные, которые вам нужно поместить в хранилище для аналитических целей, вам нужно определить те рабочие системы своей компании, которые содержат эти данные. Данные, необходимые для хранилища, извлекаются из рабочих систем-источников и записываются в промежуточную область (staging area), в которой будет проводиться их преобразование. Чтобы свести к минимуму влияние на производительность работы базы-источника, данные, как правило, выкачиваются без применения преобразований.

Часто пользователи рабочих систем не дают разработчикам хранилища прямого доступа к своим системам, но предоставляют периодические извлечения из данных. Эти извлечения обычно имеют форму простых, последовательно записанных файлов рабочей системы, которые и представляют собой промежуточную область.

Нужно разработать прикладные программы, чтобы отбирать поля и записи, требующиеся для хранилища. Если данные сохраняются в традиционной системе, эти программы могут быть написаны на языке COBOL и требовать специальных логических операций для того, чтобы обрабатывать такие вещи, как повторяющиеся поля и условия «COBOL occurs clause». Разработчики хранилища данных должны работать в тесном контакте с разработчиками приложений для OLTP-систем, которые создают скрипты для извлечения данных и выдают необходимые столбцы и форматы данных.

Для разработки ETL-процесса вам нужно определить, как часто нужно извлекать данные из рабочих систем. Это можно делать по окончанию определенного периода или бизнес-события, например в конце дня или недели или по завершении финансового квартала. Нужно четко определить, что означают слова «конец дня» или «последний день недели», особенно если ваша система охватывает несколько часовых поясов. Извлечение данных можно проводить в разных системах в разное время и помещать в промежуточную область для последующей загрузки в хранилище в ближайшее окно пакетной загрузки. Еще один аспект процесса разработки хранилища — это принятие решения о том, какой уровень агрегации будет использоваться при ответе на

бизнес-запросы. Это влияет на то, какие данные и в каком объеме будут передаваться по сети.

Некоторые рабочие системы могут работать на основе реляционных баз данных, таких, как Oracle 8i или 9i, Oracle Rdb, DB2/MVS, Microsoft SQL Server, Sybase или Informix. Другие могут быть в формате традиционных систем, таких, как IMS или Oracle DBMS. Другими форматами могут быть VSAM, индексированные файлы RMS или какие-нибудь еще структурированные файловые системы.

Если вы имеете прямой доступ к системам-источникам, вы можете получить данные разнообразными способами, зависящими от типа системы, в которой находятся данные. При небольших объемах данных вы можете использовать шлюз (gateway) или ODBC. Для более значительных объемов можно написать свою программу, которая напрямую соединяется с базой-источником, используя ее собственный интерфейс прикладного программирования (Application Programming Interface, API). Многие средства для ETL упрощают процесс извлечения данных, предоставляя связь с источником.

5.2.1. Выявление изменившихся данных

После первоначальной загрузки данных в хранилище, по мере того как меняются исходные данные, данные в хранилище необходимо будет обновлять на регулярной основе, чтобы они отражали происходящие изменения. Нужно установить механизм, позволяющий отслеживать и фиксировать интересующие изменения в рабочих системах. Нужно вносить изменения, а не перестраивать периодически все хранилище. Если выявлять изменения на этапе извлечения данных, то меньше информации нужно будет передавать по сети и загружать в хранилище.

К изменениям данных относится как добавление новых, так и изменение и удаление существующих. Например, в случае хранилища данных EASYDW нас интересуют новые заказы, а также обновление имеющейся информации о продуктах и покупателях. Если мы больше не продаем какой-то продукт, он удаляется из системы ввода заказов, но исторические сведения нужно оставить в хранилище. Именно поэтому для хранилищ рекомендуют использовать суррогатные ключи. Если ключ продукта будет повторно использоваться в рабочей системе, это не повлияет на записи хранилища.

В хранилищах не является нетипичным и изменение таблиц измерений, поскольку столбцы, например описание продукта, могут изменяться. Разработка хранилища включает в себя принятие решения о том, как будут отражаться изменения, вносимые в таблицы измерений. Если вам нужно сохранить старую версию описания продукта, вы можете сделать в таблице дополнительный столбец,

хранящий как новую, так и старую версии. Если вам нужно хранить все старые описания продуктов, вам придется создать новую строку для каждого изменения, назначив им разные значения ключей. Как правило, следует избегать внесения изменений в таблицу фактов.

Существуют разные методы выявления новых данных. Один из методов — это ставить отметку времени, когда строка данных в рабочей системе меняется. Программа извлечения данных отбирает данные на основе отметки времени транзакции и извлекает все строки, которые обновились с момента последнего извлечения. Можно использовать этот метод при перемещении заказов из системы обработки заказов в хранилище EASYDW, отбирая строки на основе значения даты покупки (purchase_date). Мы проиллюстрируем это ниже в этой главе.

Если источником данных является реляционная база, для выявления измененных строк можно использовать триггеры. Триггеры представляют собой хранимые процедуры, которые могут вызываться до или после события, такого, как вставка, изменение или удаление. Триггер можно использовать для сохранения измененных записей в таблице, из которой измененные строки можно получить в ходе извлечения информации. Будьте осторожны при использовании триггеров в приложениях, связанных с большими объемами данных, поскольку они могут создать значительную нагрузку на рабочую систему.

Иногда у вас может не оказаться возможности изменить схему и добавить отметку времени или триггер. Система может быть уже сильно нагружена, и вы не хотите никоим образом ухудшить производительность. Или же источником данных может быть традиционная система, в которой триггеры не предусмотрены. Следовательно, вам может потребоваться для выявления изменений использовать сравнение файлов. Для этого нужно сохранять на разных этапах образы извлеченных файлов. Например, чтобы выявить изменения, вам может понадобиться сравнить последнее извлечение с текущим списком продуктов или покупателей.

Также возможно выявлять изменения метаданных и определений данных. Изменение структуры рабочей системы, такое, например, как добавление или удаление столбца, влияет на программы извлечения и загрузки, которые придется переписать, чтобы принять в расчет изменения.

5.2.2. Фиксирование изменений данных в Oracle

В Oracle 9i предлагается новый метод, облегчающий выявление изменений в тех случаях, когда системой-источником является база данных Oracle 9i. Результаты всех операций INSERT, UPDATE и DELETE можно сохранять в таблицах,

называемых таблицами изменений (change tables). Программы извлечения данных могут отбирать данные из этих таблиц.

Фиксирование изменений данных, которое часто называют аббревиатурой CDC (change data capture), использует интерфейс публикация-подписка (publish-subscribe), показанный на рис. 5.2. Тот, кто осуществляет публикацию (publisher), обычно это администратор базы, определяет, какие пользовательские таблицы в рабочей системе используются для загрузки хранилища, и настраивает систему на фиксацию и публикацию измененных данных. Для каждой таблицы источника с данными, которые требуется ввести в хранилище, создается таблица изменений.

Затем программы извлечения данных подписываются (subscribe) на таблицы-источники. Следовательно, возможно любое количество подписчиков (subscribers). Каждому подписчику выдается собственное представление таблицы изменений. Это изолирует подписчиков друг от друга, если они одновременно осуществляют доступ к одним и тем же данным. Подписчики для отбора измененных данных из представлений подписчика (subscriber view) используют SQL. Они видят только те столбцы, которые их интересуют, и только те строки, которые он еще не обрабатывали. Если обновления нескольких таблиц зависят друг от друга, таблицы изменений могут быть сгруппированы в набор изменений (change set). Если, например, у вас есть таблица-заголовок заказа и таблица сведений о заказе, эти таблицы будут сгруппированы в один набор изменений, что обеспечивает связность транзакций.

В Oracle 9i изменения фиксируются синхронно, в реальном времени, как часть транзакции в рабочей системе. Данные об изменениях генерируются в виде операций DML над исходными таблицами. Когда в пользовательскую таблицу

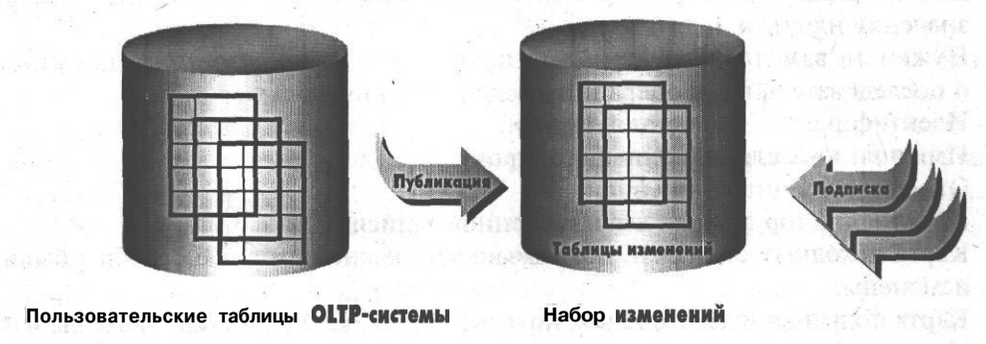


Рис. 5.2. *Архитектура публикация-подписка системы фиксирования изменений данных*

вставляется новая строка, она также сохраняется в таблице изменений. Когда строка в пользовательской таблице обновляется, измененные столбцы сохраняются в таблице изменений. В таблицу изменений можно записывать старые или новые значения или и то, и другое сразу. Когда строка из пользовательской таблицы удаляется, удаленная строка также сохраняется в таблице изменений. Это дает определенную нагрузку на каждую транзакцию, но упрощает процесс извлечения данных.

Публикация изменений данных

В хранилище EASYDW нас интересуют все новые заказы из системы ввода заказов. Администратор базы создает таблицы изменений с помощью предоставляемой процедуры DBMS_LOGMINER_CDC_PUBLISH.CREATE_CHANGE_TABLE и указывает список столбцов, который следует в них включить. Таблица изменений содержит изменения одной исходной таблицы.

Во многих примерах в этой главе мы будем использовать схему с именем OLTP, являющуюся частью рабочей базы данных. В приводимом примере администратор базы с помощью процедуры CREATE_CHANGE_TABLE фиксирует столбцы PRODUCT_ID, CUSTOMER_ID, PURCHASE_DATE, PURCHASE_TIME, PURCHASE_PRICE, SHIPPING_CHARGE, и TODAY_SPECIAL_OFFER из таблицы ORDERS схемы OLTP. Название таблицы изменений — ORDERS_CHANGE_TABLE. Поскольку в нашем примере только одна таблица изменений, используется единственный поддерживаемый в Oracle 9i и установленный по умолчанию набор SYNC_SET.

После списка столбцов идут несколько параметров, которые позволяют указывать следующее.

- Хотите ли вы, чтобы таблица изменений содержала старые значения, новые значения или и то, и другое.
- Нужен ли вам последовательный номер строки, дающий представление о последовательности операций в ходе транзакции.
- Идентификатор измененной строки.
- Имя пользователя, изменившего строку.
- Отметка времени изменения.
- Идентификатор объекта для измененной записи.
- Карта исходных столбцов, показывающая, какие исходные столбцы были изменены.
- Карта получившихся столбцов, показывающая, какие столбцы таблицы изменений были изменены.
- Опциональный столбец для присоединения к оператору CREATE TABLE DDL.

Приложение может проверять карту исходного или получившегося столбца и определять, какие столбцы были изменены.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_PUBLISH.CREATE_CHANGE_TABLE -
('oltp', 'orders_change_table', 'SYNC_SET', 'oltp', 'orders', -
'product_id VARCHAR2(8), -
 customer_id VARCHAR2(10),-
 purchase_date date, -
 purchase_time NUMBER(4,0), -
 purchase_price NUMBER(6,2), -
 shipping_charge NUMBER(5,2), -
 today_special_offer VARCHAR2(1)', -
 'BOTH', 'Y','N','Y','Y','N','Y','Y',NULL);
```

Пример выводимых данных мы увидим позже в этой же главе. Чтобы увидеть список опубликованных таблиц изменений, пошлите запрос к таблице-словарю CHANGE_TABLES.

```
SQL> SELECT CHANGE_TABLE_NAME FROM CHANGE_TABLES;

CHANGE_TABLE_NAME
------------------
ORDERS_CHANGE_TABLE
```

Затем администратор базы выдает подписчикам привилегии SELECT для таблицы изменений.

Подписка на измененные данные

Программы извлечения данных для доступа к таблицам изменений создают подписки. Подписка может содержать данные от одной или более таблиц изменений из одного набора изменений.

В словарном представлении all_source_tables перечисляются исходные таблицы, уже опубликованные администратором базы. В данном примере опубликованы изменения таблицы ORDERS схемы OLTP.

```
SQL> SELECT * FROM ALL_SOURCE_TABLES;

SOURCE_SCHEMA_NAME  SOURCE_TABLE_NAME

OLTP                ORDERS
```

Создание подписки

Есть несколько этапов создания подписки.

1. Получение дескриптора подписки
2. Перечисление всех таблиц и столбцов, на которые подписывается программа извлечения данных.
3. Активизация подписки.

Чтобы создать подписку на изменения, нужно получить дескриптор подписки на набор изменений с помощью процедуры GET_SUBSCRIPTION_HANDLE. Этот дескриптор будет передаваться в другие используемые процедуры. В следующем примере вначале создается переменная с именем subhandle.

```
SQL> VARIABLE subhandle NUMBER;
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.GET_SUBSCRIPTION_HANDLE -
(CHANGE_SET => 'SYNC_SET', -
 DESCRIPTION => 'Changes to orders table' , -
 SUBSCRIPTION_HANDLE => :subhandle);
```

Затем создается подписка с помощью процедуры SUBSCRIBE. Подписка может содержать одну таблицу или несколько из одного набора изменений. В процедуре SUBSCRIBE перечисляются схема, таблица и столбцы измененных данных, которые программа извлечения будет использовать для загрузки в хранилище. В данном примере процедура SUBSCRIBE используется для получения изменений всех столбцов таблицы ORDERS схемы OLTP. Процедура подписки выполняется по одному разу для каждой таблицы. В этом примере нас интересуют изменения только одной таблицы.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.SUBSCRIBE -
(SUBSCRIPTION_HANDLE => :subhandle, -
  SOURCE_SCHEMA => 'oltp', -
  SOURCE_TABLE => 'orders', -
  COLUMN_LIST => 'product_id,customer_id,purchase_date, -
purchase_time,purchase_price,shipping_charge,today_special_offer');
```

После создания подписки на все таблицы изменений подписка активизируется с помощью процедуры ACTIVATE_SUBSCRIPTION. Активизация делается для того, чтобы показать, что все таблицы добавлены и процесс подписки завершен.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.ACTIVATE_SUBSCRIPTION -
  (SUBSCRIPTION_HANDLE => :subhandle);
```

Когда подписка создана и новые данные добавляются в исходные таблицы, они становятся доступны для обработки.

Обработка измененных данных

Для иллюстрации процесса обработки данных мы добавляем две строки в таблицу заказов (ORDERS). Как только данные вставляются в таблицу заказов, эти изменения также сохраняются в таблице ORDERS_CHANGE_TABLE.

```
SQL> INSERT INTO oltp.orders(order_id,product_id, -
customer_id, purchase_date, purchase_time, purchase_price, -
shipping_charge,today_special_offer -
sales_person_id,payment_method) -
VALUES ('123', 'SP1031','AB123495','01-JAN-02',
1031,156.45,6.95,'N','SMITH','VISA');

1 row created.

SQL> INSERT INTO oltp.orders(order_id,product_id, -
customer_id, purchase_date, purchase_time, purchase_price, -
shipping_charge,today_special_offer -
sales_person_id,payment_method) -
VALUES ('123','SP1031','AB123495','01-FEB-02',
1031,156.45,6.95,'N','SMITH','VISA');

1 row created.
```

Чтобы обработать измененные данные, программа проходит все шаги, перечисленные ниже в тексте и проиллюстрированные на рис. 5.3. Таблица изменений является динамической, новые измененные данные добавляются к ней в то же самое время, когда программа извлечения читает ее. Чтобы обеспечить последовательность в отображении содержимого таблицы изменений, данные просматриваются во временном окне. Перед доступом к данным окно разворачивается. На рис. 5.3 в первом окне доступны строки 1–8. Пока программа обрабатывала эти строки, в таблицу изменений были добавлены строки 9–15. Получить доступ к строкам 9–15 можно, свернув (purge) окно и развернув (extend) его снова.

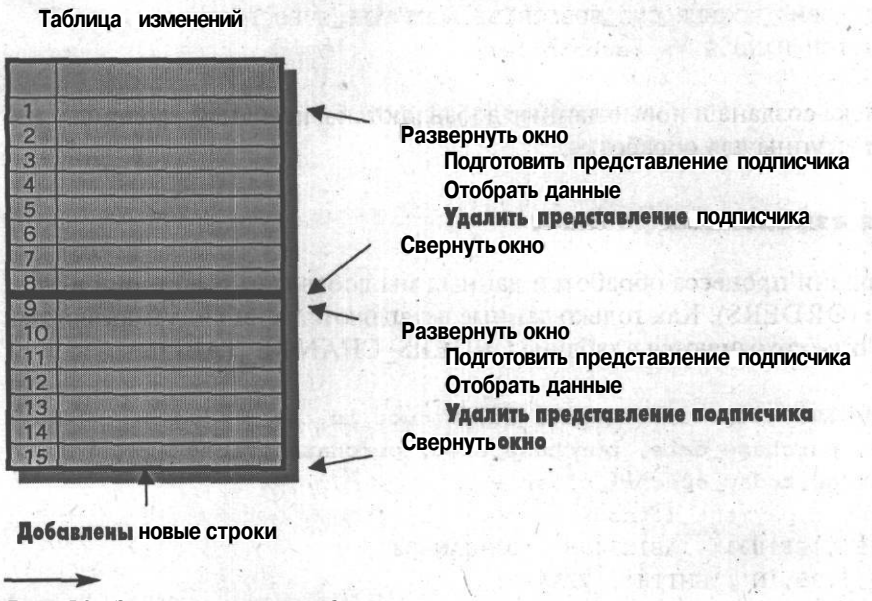


Рис. 5.3. *Запрос измененных данных*

Подписчики вместо прямого доступа к таблице изменений должны создавать представление. Программа подготавливает представление подписчика, отбирает данные из таблицы, используя это представление, а затем удаляет его.

Шаг 1. Развертывание окна

Измененные данные доступны только во временном окне — от момента вызова процедуры EXTEND_WINDOW до момента вызова процедуры PURGE_WINDOW. Чтобы увидеть новые данные, добавленные в таблицу изменений, окно нужно развернуть процедурой EXTEND_WINDOW.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.EXTEND_WINDOW -
(SUBSCRIPTION_HANDLE => :subhandle);
```

Шаг 2. Подготовка представления подписчика

Подписчики вместо прямого доступа к таблице изменений должны создавать представление. В следующем примере сначала создается переменная viewname.

```
SQL> VARIABLE viewname VARCHAR2(4000);
```

Представление создается с помощью процедуры PREPARE_SUBSCRIBER_VIEW. Содержимое представления не будет меняться даже при добавлении в таблицу изменений новых данных.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.PREPARE_SUBSCRIBER_VIEW -
 (SUBSCRIPTION_HANDLE => :subhandle, -
SOURCE_SCHEMA => 'oltp', -
SOURCE_TABLE => 'orders', -
VIEW_NAME => :viewname);
```

Шаг 3. Отбор данных из представления

В данном примере будет проверено содержимое представления. Сначала определяется имя представления, которое затем используется в инструкции SELECT.

```
SQL> PRINT viewname;
VIEWNAME
--------------
CDC#CV$2131489
```

Первый столбец выводимых данных содержит значение «I» от insert (вставка). Затем идут столбцы cscn и commit_time (время выполнения), после чего идет имя пользователя, внесшего изменение. Для каждой строки выводятся новые данные. Идентификатор строки-источника (RSID$) показывает порядок выражений в транзакции.

```
SQL> SELECT OPERATION$, CSCN$, COMMIT_TIMESTAMP$, .
   USERNAME$, CUSTOMER_ID, PRODUCT_ID, RSID$
   FROM cdc#cv$2131489;

OP  CSCN$   COMMIT_TI  USERNAME$  CUSTOMER_ID  PRODUCT_ID  RSID$
--  ------  ---------  ---------  -----------  ----------  ----
I   345643  28-FEB-02  OLTP       AB123495     SP1031      1
I   345643  28-FEB-02  OLTP       AB123495     SP1031      2
```

Шаг 4. Удаление представления подписчика

Когда обработка данных представления подписчика заканчивается, оно удаляется с помощью процедуры DROP_SUBSCRIBER_VIEW.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIBER_VIEW -
   (SUBSCRIPTION_HANDLE => :subhandle, -
    SOURCE_SCHEMA => 'oltp', -
    SOURCE_TABLE => 'orders');
```

Шаг *S*. Свертывание окна

Когда данные больше не нужны, окно сворачивается с помощью процедуры PURGE_WINDOW. Когда все подписчики сворачивают свои окна, данные в этих окнах автоматически удаляются.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.PURGE_WINDOW -
    (SUBSCRIPTION_HANDLE => :subhandle);
```

Завершение подписки

Когда программа извлечения данных больше не требуется, вы можете закончить подписку с помощью процедуры DROP_SUBSCIPTION.

```
SQL> EXECUTE DBMS_LOGMNR_CDC_SUBSCRIBE.DROP_SUBSCRIPTION -
    (SUBSCRIPTION_HANDLE => :subhandle);
```

Транспортировка изменений в промежуточную область

Теперь, когда изменения были забраны из рабочей системы, их нужно переместить в промежуточную область. Программа извлечения может записать их в файл данных, не входящий в базу, скопировать их с помощью FTP и с помощью SQL*Loader или внешних таблиц загрузить измененные данные в промежуточную область. В качестве альтернативы можно предложить записать изменения в таблицу и переместить ее в промежуточную область с помощью транспортируемых табличных пространств (transportable tablespaces). Оба эти метода обсуждаются ниже в этой главе.

5.3. Преобразование данных в общий формат

Когда данные извлечены из рабочих систем, они готовы к очистке и преобразованию в общий формат. Нужно устранить противоречия между соглашениями об именах, форматами хранения, типами данных и схемами кодирования. Удаляются дубликаты, проверяются взаимосвязи и добавляются идентификаторы уникальных ключей. В этом разделе мы познакомимся с различными типами преобразований, позже мы покажем конкретные примеры преобразований.

5.3.1. Интеграция данных из нескольких источников

Часто информация, необходимая для создания таблицы хранилища, поступает из нескольких систем-источников. Если у систем есть общее поле, данные можно соединить с помощью этого столбца.

Интегрирование данных из нескольких источников может быть весьма трудной задачей. Разработку рабочих систем могли осуществлять разные люди в разное время, используя разные стили, стандарты и методологии. Они могли использовать и разные технологии (например, аппаратные платформы, системы управления базами данных и программное обеспечение операционных систем). Если данные поступают с мейнфрейма IBM, может понадобиться конвертировать их из EBCDIC в ASCII или из big endian в little endian или наоборот.

Что еще усложняет проблему, в системах-источниках может не быть общего идентификатора. Например, при создании измерения Покупатели идентификатор покупателя может оказаться не во всех системах. Вам может понадобиться просматривать имена и адреса покупателей, определяя, относится ли запись к одному и тому же человеку. Могут быть различия в расстановке пробелов, в регистре и пунктуации. Oracle Warehouse Builder позволяет решить проблему дупликаций покупателей.

5.3.2. Очистка данных

В большинстве рабочих систем содержаться «загрязненные» данные. Это значит, что в них могут присутствовать:

- дублирующиеся записи;
- пропуски данных;
- данные, содержащие недопустимые значения;
- данные, ссылающиеся на несуществующие первичные ключи.

Иногда выполнение бизнес-правил обеспечивают приложения, иногда — ограничения целостности данных базы, а иногда никакого контроля нет совсем.

Данные должны быть стандартизированы. Например, любой данный почтовый адрес с названием улицы, например 1741 ColemanAve, может быть представлен множеством способов. Слово «Avenue» (Авеню) можно хранить в виде «Ave,» «Ave.,» «Avenue» или «AVE.» Преобразование «найти и заменить» позволяют найти все эти значения и заменить их на стандартное, выбранное для хранилища.

Вы можете захотеть провести проверку некоторых типов данных. Если продается продукт, имеющий только три возможных цвета, вы можете проверить соответствие данных списку значений, например «красный, желтый и зеленый». Со временем этот список может изменяться, включая новые значения. Например, в феврале 2002 года появился также продукт синего цвета. Вы можете проверять данные по более крупным спискам значений, например по списку штатов США.

Некоторые типы очистки включают в себя комбинирование и разделение символьных данных. Может потребоваться сцепить два строковых столбца, например объединить LAST_NAME, запятую и FIRST_NAME в столбец CUSTOMER_NAME. Или же может понадобиться с помощью оператора разрезания разделить строку на части, например отделить код города от номера телефона.

Важный момент, относящийся к целостности данных, — обеспечение взаимосвязей типа «один к одному» и «один к многим». Часто эти связи проверяются в ходе процесса преобразования, а не с помощью ограничений базы данных, связанных с целостностью.

5.3.3. Генерация новых данных

При загрузке данных может оказаться нужно выполнить вычисления или создать новые данные на основе имеющихся. Например, может понадобиться сохранять промежуточную сумму или количество записей, по мере того как они перемещаются из базы-источника в целевую базу.

При разработке хранилища определяется должный уровень подробности его данных. Часто лучше всего хранить данные на разных уровнях подробности, с разными периодами хранения и архивирования. Период хранения наиболее подробных данных о транзакциях обычно короче, чем данных, агрегированных на более высокий уровень. Подробные транзакционные данные о продажах необходимы для анализа того, какие продукты покупаются вместе. Ежедневные объемы продаж продуктов по магазинам используются для анализа закономерностей по регионам и определения продаваемости продукта.

Данные можно агрегировать в ходе процесса преобразования. Если вы не хотите сохранять подробные транзакционные данные в хранилище, их можно агрегировать перед отправкой в хранилище.

5.3.4. Генерация ключей хранилища

Вместо того чтобы использовать ключи, применяемые в рабочих системах, обычной практикой является создание для использования в хранилище нового ключа, называемого суррогатным или синтетическим. Обычно суррогатный ключ представляет собой сгенерированную последовательность целых чисел.

Суррогатные ключи применяются по многим причинам. Ключи, которые используются в рабочих системах, могут представлять собой длинные символьные строки, в компоненты которых вкладывается определенный смысл. Поскольку суррогатные ключи — это целые числа, таблицы фактов и иерархические индексы будут меньше по размерам, будут иметь меньше уровней и занимать меньше места на диске, что улучшит время ответа. Суррогатные ключи обеспечивают некоторую независимость от изменений в рабочей системе.

Если в рабочей системе меняется соглашение о названиях для кода продукта или формат, то не нужно будет менять все данные в хранилище. Когда одна компания поглощает другую, может потребоваться ввести в хранилище продукты поглощенной компании. Очень маловероятно, что обе компании использовали одну и ту же схему кодирования продуктов. Если существует возможность, что две компании использовали один и тот же код для разных продуктов, тогда к коду продукта в хранилище нужно добавлять идентификатор компании. Использование суррогатных ключей может значительно облегчить интеграцию данных в ситуациях такого типа.

И суррогатные ключи, и ключи рабочей системы хранятся в таблице измерения, как это показано на рис. 5.4, где продукт с кодом SR125 известен в хранилище как PRODUCT_ID 1. Следовательно, можно видеть, что в таблице фактов ключ продукта хранится в виде 1. Тем не менее, пользователи спокойно могут использовать в запросах и код SR125, даже и не подозревая о преобразовании, проведенном в хранилище.

Суррогатный ключ используется в таблице фактов в виде столбца, соединяющего таблицу фактов с таблицей измерения. В приведенном примере имеется два формата кодов продукта. Часть их — числовые коды, разделенные с помощью тире: 123–118. Другие представляют собой смесь буквенных и числовых символов: SR125. В ходе ETL-процесса, когда загружается очередная запись о факте, суррогатный ключ ищется в таблицах измерений и записывается в таблицу фактов.

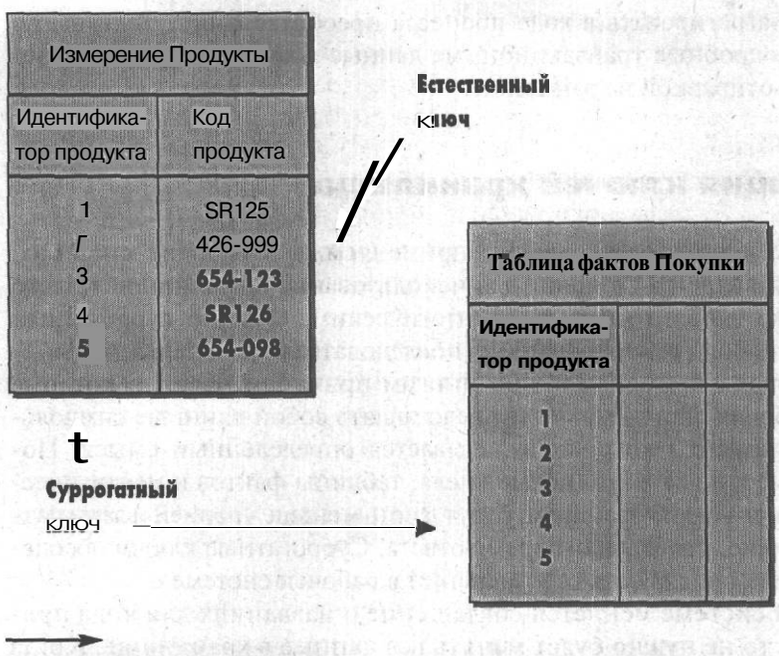


Рис. 5.4. *Использование суррогатных ключей в хранилище*

5.3.5. Выбор оптимального места проведения преобразований

Преобразования данных можно осуществлять на любом этапе ETL-процесса. Нужно выбрать наиболее эффективное место для проведения преобразования: в базе-источнике, в промежуточной области, в ходе операции загрузки или во временных таблицах сразу после загрузки в хранилище. В Oracle 9i для облегчения проведения преобразований было добавлено несколько новых свойств.

- Преобразования можно выполнять в ходе процесса извлечения данных. Вообще, если это возможно, лучше всего выполнять преобразования, связанные с фильтрацией в базе-источнике. Это позволяет отбирать для загрузки в хранилище только интересующие данные. В идеале, нужно извлекать только те данные, которые изменились с момента последнего извлечения. Хотя преобразования и можно выполнять в рабочих системах-источниках, важно свести к минимуму дополнительную нагрузку, которая накладывается процессом извлечения на рабочую систему.

- Преобразования можно выполнять в промежуточной области, перед загрузкой данных в хранилище. Если нужно интегрировать данные из многих систем, то в ходе извлечения сделать это не получится. В качестве промежуточной области можно использовать последовательные файлы, базу Oracle или и то, и другое вместе. Если поступающие данные имеют форму последовательного файла, скорее всего, наиболее эффективным будет завершить промежуточный процесс до загрузки их в хранилище. Преобразования, требующие сортировки, последовательной обработки и построчных операций, можно эффективно проводить в промежуточном последовательном файле.

Если вашим источником служит база данных Oracle 8i или 9i, перемещение данных в хранилище облегчат транспортируемые табличные пространства, и извлекать данные во внешнюю таблицу не придется. В данном случае имеет смысл провести преобразования во временных промежуточных таблицах сразу, как только данные окажутся в хранилище. При проведении преобразований в базе Oracle, если возможна массированная обработка данных с помощью набора операций SQL, их можно выполнять параллельно. Если какие-то данные, необходимые для преобразования, уже находятся в хранилище, возможно, более эффективным будет выполнение преобразований в хранилище. Например, поиск суррогатных ключей в таблице измерения можно эффективно проводить в промежуточных таблицах базы данных. В разделе 5.5 описаны некоторые примеры выполнения преобразований в промежуточной области базы данных Oracle.

- Преобразования можно выполнять в ходе процесса загрузки. Некоторые простые виды преобразований можно выполнять по мере того, как данные загружаются через SQL*Loader. Это можно использовать, например, для перевода букв столбца в верхний регистр. Лучше всего это работает, если нужно добавить небольшое число строк, например при первоначальной загрузке таблицы измерения. В Oracle 9i предлагается новое свойство — внешние таблицы, которые помогают проводить более сложные преобразования данных в ходе процесса загрузки. Ниже приводится описание некоторых примеров преобразований при загрузке.

5.4. Загрузка данных в хранилище

При загрузке хранилища таблицы измерений, как правило, загружаются первыми. Таблицы измерений содержат суррогатные ключи и прочую описательную информацию, необходимую для таблиц фактов. При загрузке таблицы

фактов информация ищется в таблицах измерений и вставляется в столбцы таблицы фактов.

При загрузке таблицы измерения нужно и добавлять новые строки, и изменять существующие. Например, измерение Покупатели может содержать десятки тысяч записей о покупателях. Обычно изменяется 10 и менее процентов этой информации. Вы будете добавлять новых покупателей и иногда менять информацию о существующих.

При добавлении в таблицу измерений новых данных, нужно выяснить, не существует ли уже такая запись. Если нет, можно добавлять ее в таблицу. Если запись существует, есть разнообразные способы обработки изменений, в зависимости от того, нужно ли вам хранить старую информацию для аналитических целей.

Если меняется домашний адрес покупателя, обычно нет нужды хранить старый адрес, так что запись можно просто обновить. В быстро растущей компании часто меняются регионы, в которых проводится продажа. Например, «Канада» до 1990 года входила в «весь остальной мир», а после реорганизации стала входить в «Американский континент». Если вы хотите иметь информацию и о старой, и о новой географической иерархии, вы можете создать новую запись в измерении, содержащую все старые данные плюс новую иерархию, и дать этой записи новый суррогатный ключ. В качестве альтернативы можно предложить создать в первоначальной записи столбцы для хранения как предыдущих, так и текущих значений.

Измерение-время содержит по одной строке для каждой единицы времени, представляющей интерес для хранилища. В примере с торговой компанией EASYDW покупки могут совершаться в режиме он-лайн 365 дней в году, так что интерес будет представлять каждый день. Для каждой конкретной даты хранится информация о дне, в том числе день недели, номер недели, месяц, квартал, год, и является ли этот день нерабочим. Измерение-время можно загружать ежегодно.

При загрузке таблицы фактов новая информация, как правило, добавляется в конец имеющейся таблицы фактов. Менять имеющиеся строки не стоит, поскольку эти данные и нужно хранить. Например, таблица фактов Покупки содержит данные за три месяца. Новые данные из источника — системы ввода заказов — добавляются к таблице фактов ежемесячно. Секционирование данных по месяцам облегчает такой режим работы.

В следующих разделах мы рассмотрим различные способы загрузки данных, включая:

- SQL*Loader, который вставляет данные в новую таблицу или добавляет к имеющейся, если данные находятся в последовательном файле, внешнем по отношению к базе данных;

- внешние таблицы, которые вставляют данные в новую таблицу или добавляют к уже имеющейся, если данные находятся в последовательном файле, внешнем по отношению к базе данных, и если вы хотите провести преобразование данных при загрузке;
- транспортируемые табличные пространства, которые используются для перемещения данных между двумя базами Oracle, такими, как рабочая система и хранилище.

5.4.1. Использование SQL*Loader для загрузки хранилища

Одним из наиболее популярных средств загрузки данных является SQL*Loader, поскольку он специально разработан для как можно более быстрой загрузки записей. Его можно использовать либо с помощью командной строки рабочей систем, либо с помощью мастера в Oracle Enterprise Manager (OEM), который мы сейчас и обсудим.

Использование мастера Oracle Enterprise Manager Load Wizard

На рис. 5.5 показан мастер загрузки Load Wizard в Oracle Enterprise Manager, который может использоваться для создания расписания автоматического выполнения задач SQL*Loader. Этот мастер проведет вас по процессу загрузки данных из внешнего файла в базу данных в соответствии с набором инструкций, содержащимся в управляющем файле. Для выполнения загрузки запускается пакетное задание Enterprise Manager.

Вы можете загружать инструменты управления данными, выбирая их в меню инструментов после того, как выберете целевую базу данных в древовидном каталоге Навигатора, как это показано на рис. 5.6.

Управляющий файл

Управляющий файл представляет собой текстовый файл, описывающий процесс загрузки. Функция управляющего файла (показанного на рис. 5.7) состоит в том, чтобы сообщить программе SQL*Loader, какой файл загружать, как интерпретировать записи и столбцы и в какие таблицы вставлять данные.

Управляющий файл написан на языке определения данных SQL*Loader. Ниже приводится пример управляющего файла, который можно использовать для добав-

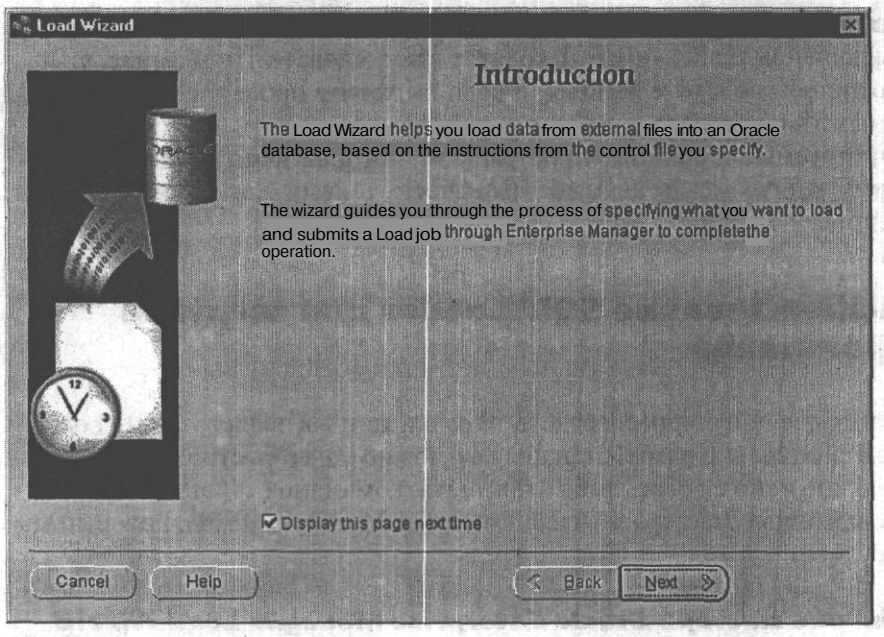


Рис. 5.5. *Мастер загрузки Load Wizard*

ления новых данных о продукте в хранилище EASYDW. Данные сохраняются в файле **product.dat**. К имеющейся таблице будут добавлены новые строки.

```
- Загрузка измерения product
LOAD DATA
INFILE 'product.dat' append
INTO TABLE product
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY "'"
(product_id,
 product_name,
 category,
 cost_price,
 sell_price,
 weight,
 shipping_charge,
 manufacturer,
 supplier)
```

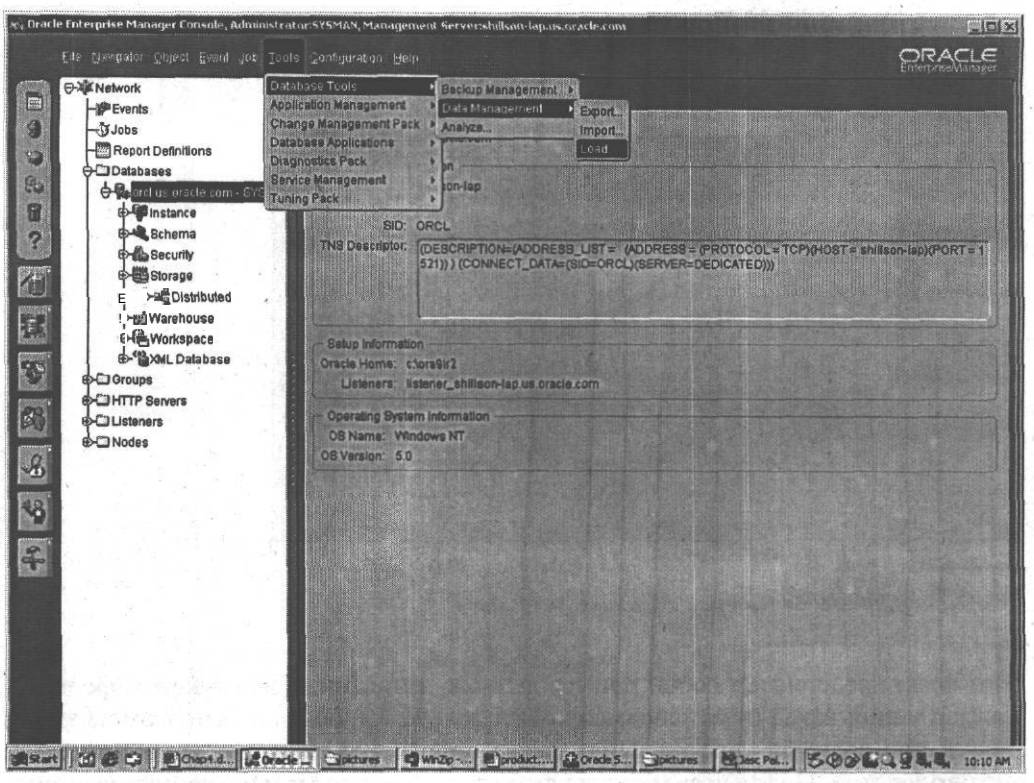


Рис. 5.6. *Как найти мастер загрузки*

Файл данных

В следующем примере приводится небольшой образец файла данных product.dat. Поля отделяются запятыми и, при желании, могут заключаться в одиночные кавычки. Каждое поле входящего файла соответствует определенному столбцу таблицы. При чтении данные конвертируются из формата этого файла в формат столбца базы данных.

```
'SP1242', 'CD LX1','MUSC',8.90, 15.67, 2.5,  2.95,   'RTG', 'CD Inc'
'SP1243', 'CD LX2','MUSC',8.90, 15.67, 2.5,2.95, 'RTG', 'CD Inc'
'SP1244', 'CD LX3','MUSC',8.90, 15.67, 2:5,  2.95,   'RTG', 'CD Inc'
'SP1245', 'CD LX4','MUSC',8.90, 15.67, 2.5,  2.95,   'RTG', 'CD Inc'
```

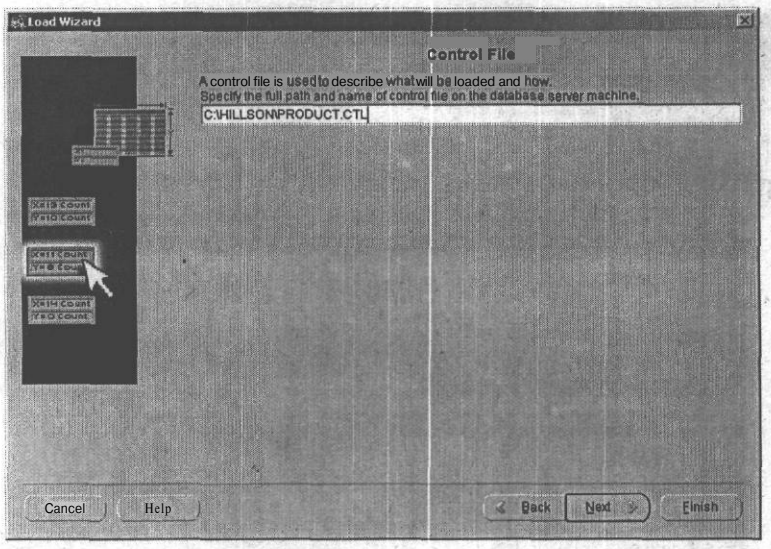


Рис. 5.7. *Управляющий файл*

Этот файл представляет собой пример данных, записанных в потоковом формате. Каждая запись заканчивается символом-разделителем, обычно символом перевода строки или возврата каретки. Поля также отделяются символом-ограничителем, обычно запятой. Поля также могут заключаться в одиночные или двойные кавычки.

Помимо потокового формата, SQL*Loader поддерживает форматы файлов с фиксированной или переменной длиной. В файле фиксированной длины все записи имеют одинаковую длину. Как правило, одинакова и длина полей. В управляющем файле вводимые записи описываются путем указания начального положения, длины и типа данных. В файле переменной длины записи могут иметь разную длину. Первое поле каждой записи используется для указания длины этой записи.

Режимы работы SQL*Loader

На рис. 5.8 показаны два режима работы SQL*Loader: режим условного пути (conventional path) и режим прямого пути (direct path).

Загрузку в режиме условного пути следует использовать только для ввода небольших объемов данных, например при первоначальной загрузке небольшой таблицы измерения или же при загрузке данных, типы которых не поддерживаются в режиме прямого пути, например типа varray. Загрузка в режиме услов-

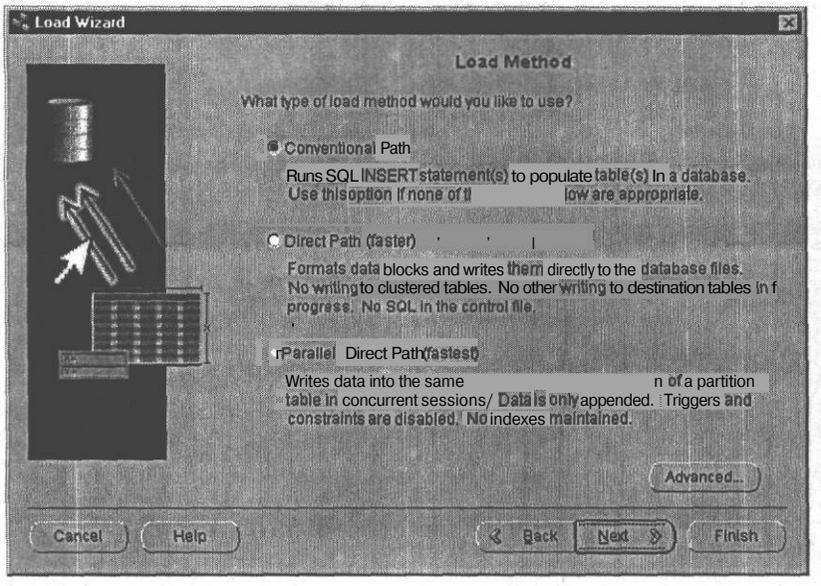


Рис. 5.8. *Режимы работы SQL*Loader*

ного пути использует операторы SQL INSERT. При добавлении каждой строки обновляются индексы, запускаются триггеры и проверяются ограничения. При вводе больших объемов данных в периоды небольших окон, предназначенных для пакетной загрузки, для оптимизации производительности можно использовать загрузку в режиме прямого пути. В этом режиме уровень SQL в расчет не принимается. Блоки данных форматируются напрямую и записываются в файлы базы данных. Если работать на системе с несколькими процессорами, загрузку можно выполнять параллельно, что дает заметный выигрыш в производительности.

Обработка ошибок в SQL*Loader

В ходе операции загрузки можно создавать дополнительные файлы, помогающие в диагностировании и исправлении ошибок, которые могут возникнуть в ходе загрузки. Для записи состояния операции ввода данных создается файл протокола (LOG-файл). Следует всегда просматривать его, чтобы убедится, что загрузка прошла успешно. Копии записей, которые нельзя ввести в базу из-за нарушений целостности данных, можно записать в файл ошибок (BAD-файл).

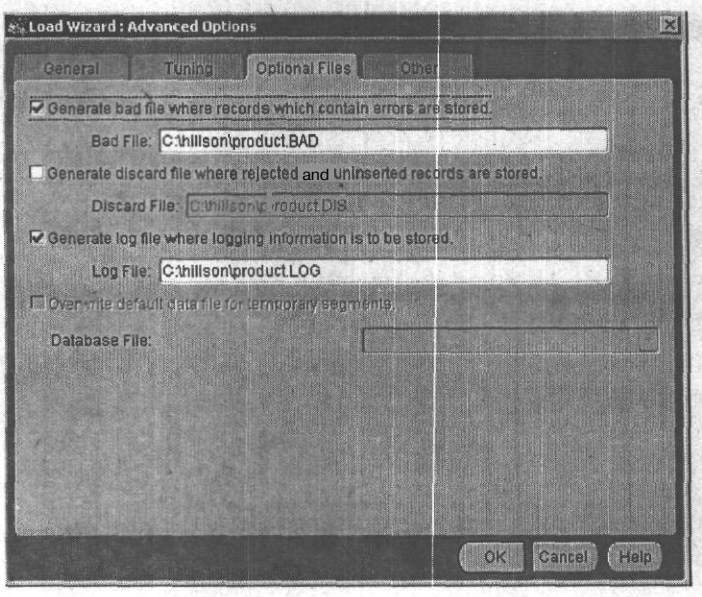


Рис. 5.9. *Дополнительные опции SQL*Loader*

Этот файл можно будет ввести позже, когда будут исправлены проблемы с целостностью. Если вы получили файл извлеченных данных, содержащий больше записей, чем вам нужно, можно ввести только его часть. Для выбора загружаемых записей используется условие WHEN управляющего файла. Пропущенные записи будут записаны в файл отклоненных данных (discard file).

Выберите дополнительные файлы, которые вы бы хотели создать, используя окно Advanced options (Дополнительные опции). В данном случае мы выбрали BAD-файл и LOG-файл.

Составление расписания операций загрузки

Система составления расписаний выполнения заданий в Enterprise Manager позволяет создавать задания и управлять ими, устанавливать время выполнения заданий и следить за их выполнением. Можно запустить задание однократно или указать, как часто его следует выполнять. Если вы запускаете задание несколько раз, можно сохранить его в библиотеке заданий Enterprise Manager, чтобы к нему можно было вернуться впоследствии. На рис. 5.10 задается однократный запуск задания в 11:31 p.m.

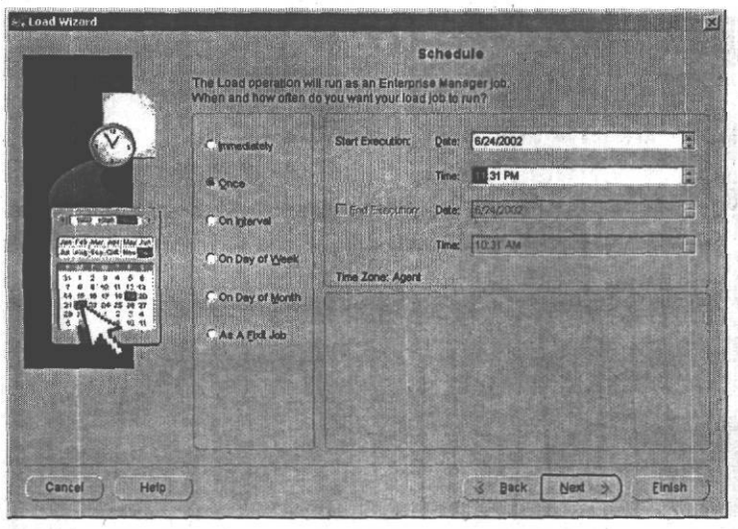


Рис. 5.10. *Задание по загрузке*

Слежение за выполнением операции загрузки

Следить за заданием в ходе его выполнения можно, щелкнув по пункту Jobs (Задания) в древовидном каталоге Навигатора в Enterprise Manager Console (как показано на рис. 5.11). На странице Active (Активные задания) перечислены все работающие или назначенные задания. Если щелкнуть по заданию на странице активных заданий, можно получить информацию о его состоянии и ходе выполнения. Задание Load 0016 было назначено и будет запущено в 11:31 p.m. Страница History (История) содержит список ранее выполненных заданий. По завершении задания вы можете увидеть, выполнено ли оно успешно, и если выполнение не удалось, то просмотреть список, ошибок.

Проверка протокола SQL*Loader

В следующем примере показана часть файла протокола (LOG-файла) для примерного сеанса загрузки. К таблице product было всего добавлено 159 строк. 10 записей были отброшены из-за неверных данных. Копии этих записей можно найти в файле product.bad.

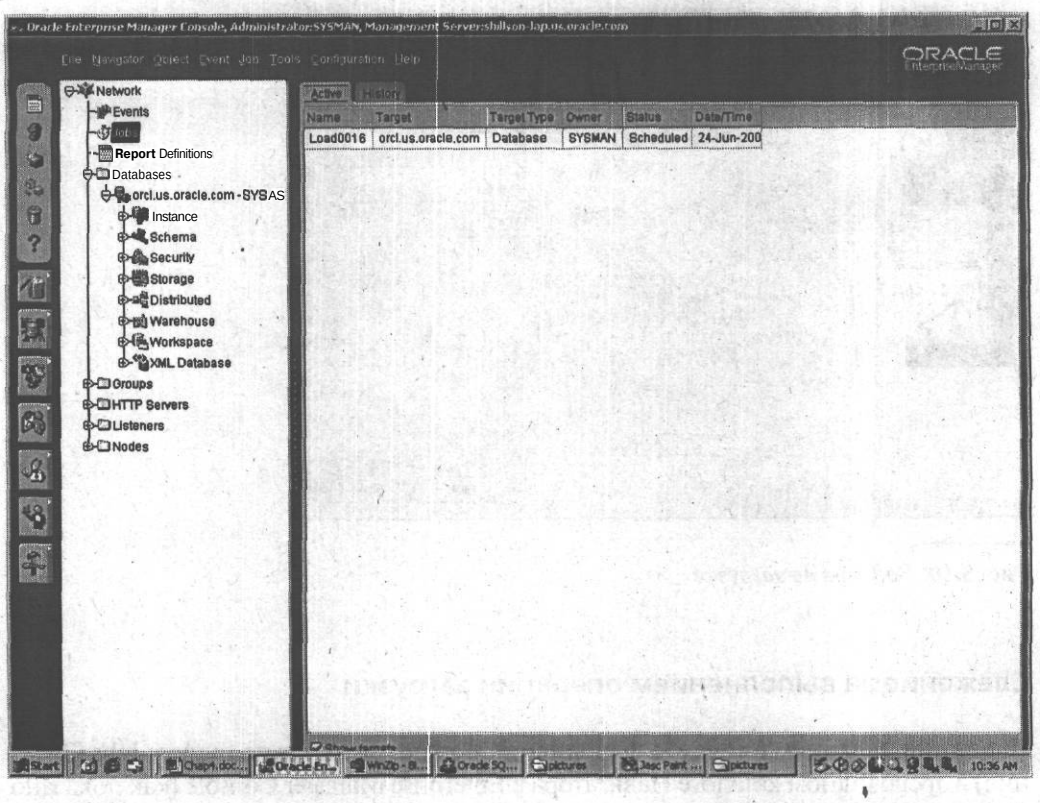


Рис. 5.11. *Слежение за заданиями в OEM console*

```
SQL*Loader: Release 9.2.0.1.0 -
Production on Mon Jun 24 06:45:47 2002

Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.

Control File: C:\hillson\product.ctl
Data File: c:\hillson\product.dat
Bad File: C:\hillson\product.BAD
Discard File: none specified

(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array: 64 rows, maximum of 256000 bytes
Continuation: none specified
Path used: Conventional

Table EASYDW.PRODUCT, loaded from every logical record.
Insert option in effect for this table: APPEND

Column Name          Position      Len    Term    Encl     Datatype
---------------      --------      ----   -----   ----     ---------
PRODUCT_ID             FIRST       *       ,     O(')     CHARACTER
PRODUCT_NAME           NEXT        *       ,     O(')     CHARACTER
CATEGORY               NEXT        *       ,     O(')     CHARACTER
COST_PRICE             NEXT        *       ,     O(')     CHARACTER
SELL_PRICE             NEXT        *       ,     O(')     CHARACTER
WEIGHT                 NEXT        *       ,     O(')     CHARACTER
SHIPPING_CHARGE        NEXT        *       ,     O(')     CHARACTER
MANUFACTURER           NEXT        *       ,     OO       CHARACTER
SUPPLIER               NEXT        *       ,     OO       CHARACTER

Record 1: Rejected - Error on table EASYDW.PRODUCT,
column PRODUCT_ID. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 3: Rejected - Error on table EASYDW.PRODUCT,
column PRODUCT_ID. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 4: Rejected - Error on table EASYDW.PRODUCT,
column WEIGHT. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 5: Rejected - Error on table EASYDW.PRODUCT,
column SELL_PRICE. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 6: Rejected - Error on table EASYDW.PRODUCT, column
SHIPPING_CHARGE. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 7: Rejected - Error on table EASYDW.PRODUCT,
column COST_PRICE. Column not found before end of logical record
 (use TRAILING NULLCOLS)
```

```
Record 8: Rejected — Error on table EASYDW.PRODUCT, column
SHIPPING_CHARGE. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 9: Rejected - Error on table EASYDW.PRODUCT,
column COST_PRICE. Column not found before end of logical record
 (use TRAILING NULLCOLS)
Record 106: Rejected - Error on table EASYDW.PRODUCT.
ORA-00001: unique constraint (EASYDW.PK_PRODUCT) violated

Record 134: Rejected - Error on table EASYDW.PRODUCT.
ORA-00001: unique constraint (EASYDW.PK_PRODUCT) violated

Table EASYDW.PRODUCT:
  159 Rows successfully loaded.
  10 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Space allocated for bind array: 148608 bytes(64 rows)
Read buffer bytes: 1048576

Total logical records skipped: 0
Total logical records read: 169
Total logical records rejected: 10
Total logical records discarded: 0

Run began on Mon Jun 24 06:45:47 2002
Run ended on Mon Jun 24 06:46:14 2002

Elapsed time was: 00:00:27.56
CPU time was: 00:00:00.09
```

Оптимизация работы SQL*Loader

При выполнении ввода больших объемов данных в небольшой период времени, отведенный под пакетную загрузку, можно использовать различные методы оптимизации производительности.

- Применение загрузки в режиме прямого пути. Прямое форматирование блоков данных и запись их в файлы базы данных делает ненужной большую часть

работы, связанную с выполнением операторов вставки языка SQL. Режим прямого пути требует полного доступа к таблице или загружаемому разделу. Кроме того, автоматически отключаются триггеры, а проверка ограничений откладывается до окончания загрузки.

- Отключение проверки ограничений, связанных с целостностью данных, перед загрузкой. При вводе данных в режиме прямого пути SQL*Loader автоматически отключает все ограничения CHECK и REFERENCES. При использовании параллельной загрузки в режиме прямого пути или при загрузке в единственный раздел нужно отключить прочие типы ограничений. Можно также вручную отключить проверку на NOT NULL, уникальность и первичные ключи. Когда загрузка завершается, можно заставить SQL*Loader вновь включить ограничения или включить их вручную самостоятельно.

■ Загрузка отсортированных данных. Предварительная сортировка снижает объем временных областей хранения, необходимых для загрузки и позволяет проводить оптимизацию для уменьшения вычислений при объединении данных. Чтобы дать программе SQL*Loader информацию о том, по каким индексам отсортированы данные, используйте инструкцию SORTED INDEXES управляющего файла.

- Отсрочка обработки индексов. При вставке или удалении данных или обновлении столбца первичных ключей индексы обрабатываются автоматически. При вводе больших объемов данных в режиме прямого пути процесс загрузки можно ускорить, если отложить обработку индексов до тех пор, пока данные не будут введены. Можно либо удалить индексы перед началом загрузки, либо опустить операцию обработки индексов, установив условие SKIP_INDEX_MAINTENANCE = TRUE в командной строке SQL*Loader. Разделы индексов, для которых иначе проводилось бы обновление, помечаются как «непригодный индекс», поскольку сегмент индекса не соответствует данным, которые он индексирует. После загрузки данных индекс необходимо будет перестроить.
- Отключение протоколов повторного выполнения (redo logging) с помощью опции управляющего файла UNRECOVERABLE. По умолчанию все изменения, вносимые в базу данных, записываются в протокол повторного выполнения. Этот протокол можно использовать для восстановления базы данных после аварий. Восстановление среды (media recovery) — это восстановление после утраты файла базы данных, часто из-за аппаратной проблемы, например после сбоя головки диска. При отключении протокола повторного выполнения загрузка данных осуществляется быстрее.

Однако, если системная ошибка возникает в середине загрузки данных, нужно будет перезапускать загрузку, поскольку нельзя будет использовать протокол повторного выполнения для восстановления. Если вы используете программу Oracle Data Guard, которая защищает данные, используя логическую или физическую резервную базу данных, вы можете не захотеть отключать этот протокол. Не занесенные в протокол данные нельзя автоматически внести в резервную базу.

После того как данные введены с использованием опции UNRECOVERABLE, важно сделать резервную копию, чтобы гарантировать себе возможность восстановления данных в будущем в случае необходимости.

- Загрузка данных в один раздел. Если вы загружаете данные в раздел или подраздел секционированной таблицы, другие пользователи могут продолжать использовать остальные разделы таблицы. Загрузка данных о транзакциях за апрель не блокирует пользователям доступ к имеющимся данным с января по март. Таким образом, повышается общая доступность хранилища.
- Параллельная загрузка данных. Если таблица секционирована, загружать данные можно сразу в несколько разделов параллельно. Также можно устанавливать несколько параллельных сеансов загрузки в одну таблицу или в один раздел секционированной таблицы.
- Увеличение значения параметра STREAMSIZE может улучшить время загрузки в режиме прямого пути, потому что от клиента SQL*Loader будет проходить больше данных в потоке к серверу базы Oracle.
- Если загружаемые данные содержат много повторяющихся дат, то применение параметра DATE_CACHE может улучшить производительность загрузки в режиме прямого пути. Используйте статистическую информацию о кэше данных (введенные данные, удачные и неудачные), содержащуюся в файле протокола SQL*Loader для настройки кэша под будущие похожие загрузки.

Загрузка данных в один раздел в режиме прямого пути SQL*Loader

Теперь мы рассмотрим пример ввода данных в один раздел. В хранилище EASYDW таблица фактов секционирована по дате. В конце апреля в хранилище вводятся транзакционные данные о произведенных покупках за апрель.

В следующем примере мы создаем табличное пространство, добавляем раздел к таблице покупок (purchases) и используем режим прямого пути SQL*Loader для ввода данных в раздел за апрель 2002 года.

Шаг 1. Создаем табличное пространство

```
SQL> CREATE TABLESPACE purchases_apr2002
   DATAFILE 'c:\ora9ir2\oradata\orcl\PURCHASESAPR2002.f' SIZE 5M
   REUSE AUTOEXTEND ON
   DEFAULT STORAGE
    (INITIAL 64K NEXT 64K PCTINCREASE Q MAXEXTENTS UNLIMITED);
```

Шаг 2. Добавляем раздел таблицы

```
SQL> ALTER TABLE easydw.purchases
   ADD PARTITION purchases_apr2002
   VALUES LESS THAN (TO_DATE('30-04-2002', 'DD-MM-YYYY'))
   PCTFREE 0 PCTUSED 99
   STORAGE (INITIAL 64K NEXT 64K PCTINCREASE 0)
   TABLESPACE purchases_apr2002;
```

Шаг 3. Отключаем все ссылочные ограничения, связанные с целостностью, а также триггеры

При использовании для загрузки данных в раздел режима прямого пути нужно отключить ссылочные и проверочные ограничения в этом разделе, а также все триггеры.

```
SQL> ALTER TABLE purchases DISABLE CONSTRAINT fk_time;
SQL> ALTER TABLE purchases DISABLE CONSTRAINT fk_product_id;
SQL> ALTER TABLE purchases DISABLE CONSTRAINT fk_customer_id;
```

Для того чтобы выяснить, выключено ограничение или нет, можно использовать столбец статуса в представлении user_constraints. В приведенном примере можно видеть, что все еще включено ограничение SPECIAL_OFFER.

```
SQL> SELECT TABLE_NAME, CONSTRAINT_NAME, STATUS
   FROM USER_CONSTRAINTS
   WHERE TABLE_NAME = ' PURCHASES ' ;
TABLE_NAME  CONSTRAINT_NAME          STATUS
----------  -----------------------  --------
PURCHASES   NOT_NULL_PRODUCT_ID      DISABLED
PURCHASES   NOT_NULL_TIME            DISABLED
```

```
PURCHASES    NOT_NULL_CUSTOMER_ID     DISABLED
PURCHASES    SPECIAL_OFFER            ENABLED
PURCHASES    FK_PRODUCT_ID            DISABLED
PURCHASES    FK_TIME DISABLED
PURCHASES    FK_CUSTOMER_ID           DISABLED
7 rows selected.opt
```

Чтобы определить, нужно ли отключать какие-либо триггеры, можно использовать столбец статуса в представлении user_triggers. В таблице PURCHASES триггеров нет.

```
SQL> SELECT TRIGGER_NAME, STATUS
   FROM ALL_TRIGGERS WHERE TABLE_NAME = 'PURCHASES';
no rows selected
```

Шаг 4. Загрузка данных

В следующем примере показан управляющий файл SQL*Loader для загрузки новых данных в один раздел. Отметьте использование условия PARTITION.

```
UNRECOVERABLE LOAD DATA
INFILE 'purchases.dat' BADFILE 'purchases.bad'
APPEND
INTO TABLE purchases
PARTITION (purchases_apr2002)
   (product_id position (1-6) char,
  time_key position (7-17) date "DD-MON-YYYY",
  customer_id position (18-25) char,
 ,purchase_date position (26-36) date "DD-MON-YYYY",
  purchase_time position (37-40) integer external,
  purchase_price position (41-45) decimal external,
  shipping_charge position (46-49) integer external,
  today_special_offer position (50) char)
```

Ключевое слово unrecoverable отключает восстановление среды (media recovery) для загружаемой таблицы. Изменения, вносимые в базу данных другими пользователями, будут продолжать протоколироваться. После отключения восстановления среды важно делать резервную копию, чтобы позже можно было в случае необходимости восстановить данные. Если вы попытаетесь провести восстановление

среды перед сохранением резервной копии, вы обнаружите, что загруженные блоки данных помечены как логически поврежденные. Чтобы восстановить данные, придется удалить раздел и перезагрузить данные.

Все данные, ввод которых оказался невозможен, будут записаны в файл purchases.bad. Данные будут загружаться в раздел purchases_apr2002. В приведенном примере показана загрузка файла фиксированной длины с именем purchases.dat. Каждое поле во вводимой записи описывается значением начального положения, конечного положения и типа данных. Заметьте, что это будут типы данных SQL*Loader, а не таблицы Oracle. Когда данные загружаются в таблицу, все эти поля при необходимости конвертируются в типы данных для столбцов.

В следующем примере показан пример файла данных purchases. Идентификатор продукта (PRODUCT_ID) начинается с первого столбца и занимает 6 байт. Ключ времени Time_key начинается со столбца 7 и занимает 17 байт. Для описания формата полей даты используется маска DD_MON_YYYY.

```
1234567890123456789012345678901234567890123456789012345678901
      I          I          I          I    I    I    I
SP100101-APR-2002AB12345601-APR-2002002428.014.50Y
SP100101-APR-2002AB12345701-APR-2002102428.014.50Y
```

Чтобы вызвать режим прямого пути SQL*Loader в командной строке, укажите direct=true. В этом примере значение skip_index_maintenance установлено в true, чтобы индексы перестраивались после загрузки.

```
sqlldr USERID=easydw/easydw CONTROL=purchases.ctl
LOG=purchases.log DIRECT=TRUE SKIP_INDEX_MAINTENANCE = TRUE
```

Шаг 5. Проверка протокола

В следующем примере показан фрагмент файла протокола (log-файла) SQL*Loader для операции загрузки. Вместо генерации протокола повторного выполнения для возможности последующего восстановления генерируется аннулирование протокола (invalidation redo), чтобы дать базе Oracle знать, что данную таблицу восстановить нельзя. Индексы сделаны непригодными. Указано начальное положение столбца и его длина.

```
SQL*Loader: Release 9.2.0.1.0 -
Production on Sun Jun 9 12:43:45 2002
```

```
Copyright (c) 1982, 2002, Oracle Corporation.
All rights reserved.

Control File: purchases.ctl
Data File: purchases.dat
  Bad File: purchases.bad
  Discard File: none specified

(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Continuation: none specified
Path used: Direct

Load is UNRECOVERABLE; invalidation redo is produced.

Table PURCHASES, partition PURCHASES_APR2002,
loaded from every logical record.
Insert option in effect for this partition: APPEND

Column Name          Position      Len    Term   Encl   Datatype
---------------      ----------    ----   ----   ----   --------
PRODUCT_ID               1:6       6                   CHARACTER
TIME_KEY                 7:17      11                  DATE DD-MON-YYYY
CUSTOMER_ID             18:25      8                   CHARACTER
PURCHASE_DATE           26:36      11                  DATE DD-MON-YYYY
PURCHASE_TIME           37:40      4                   CHARACTER
PURCHASE_PRICE          41:45      5                   CHARACTER
SHIPPING_CHARGE         46:49      4                   CHARACTER
TODAY_SPECIAL_OFFER      50        1                   CHARACTER

The following index(es) on table PURCHASES were processed:
index EASYDW.PURCHASE_PRODUCT_INDEX partition PURCHASES_APR2002
was made unusable due to:
SKIP_INDEX_MAINTENANCE option requested
index EASYDW.PURCHASE_CUSTOMER_INDEX partition PURCHASES_APR2002
```

```
was made unusable due to:
SKIP_INDEX_MAINTENANCE option requested
index EASYDW.PURCHASE_SPECIAL_INDEX partition PURCHASES_APR2002
was made unusable due to:
SKIP_INDEX_MAINTENANCE option requested
```

Шаг 6. Включение всех ограничений и триггеров и перестройка индексов

После загрузки данных в один раздел нужно вновь задействовать все ссылки на ограничения и триггеры. Обработку всех локальных индексов раздела можно провести с помощью SQL*Loader. Глобальные индексы при загрузке раздела или подраздела в прямом режиме не обрабатываются и должны быть перестроены. В предыдущем примере индексы нужно перестроить, потому что была опущена обработка индексов. Эти шаги описываются далее в этой главе более подробно.

*Параллельная загрузка в режиме прямого пути SQL*Loader*

Если таблица является секционированной, загрузку в режиме прямого пути можно использовать для параллельного ввода данных в несколько разделов. Для минимизации нагрузки на каналы ввода-вывода все процессы параллельной загрузки в режиме прямого пути следует запускать для разделов, хранящихся на отдельных дисках.

Поскольку данные извлекаются из нескольких рабочих систем, вы часто будете иметь несколько файлов, которые нужно ввести в хранилище. Эти файлы можно загружать параллельно и распределять рабочую нагрузку по нескольким параллельным сеансам SQL*Loader.

На рис. 5.12 показан пример того, как можно использовать параллельную загрузку в режиме прямого пути для первоначальной загрузки исторической информации о транзакциях в таблицу покупок. Нужно запустить несколько сеансов SQL*Loader. Каждый сеанс принимает в качестве входных данных свой файл данных. В этом примере имеется три файла данных, каждый из которых содержит информацию о покупках за один месяц — январь, февраль или март. Все эти файлы будут загружены в таблицу покупок (purchases), которая также секционирована по месяцам. Все файлы данных загружаются параллельно, каждый — в собственный раздел.

Предлагается выполнить следующие шаги для проведения параллельной загрузки в режиме прямого пути:

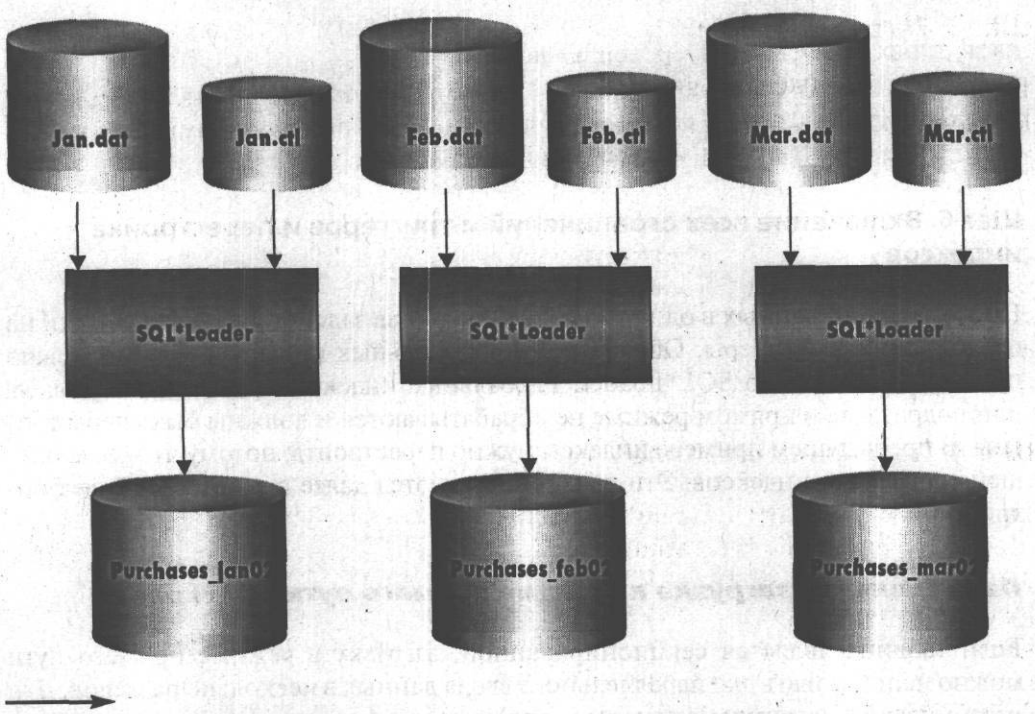


Рис. 5.12. *Параллельная загрузка в режиме прямого пути SQL*Loader*

Шаг 1. Отключение всех ограничений и триггеров

В ходе параллельной загрузки в режиме прямого пути нельзя проверять ограничения и запускать триггеры. Если вы забудете об этом, SQL*Loader выдаст сообщение об ошибке.

Шаг 2. Отказ от всех индексов

В ходе параллельной загрузки в режиме прямого пути нельзя проводить обработку индексов. Однако, если загружаются лишь несколько из многих разделов, возможно, лучше пропустить обработку индексов, и они будут помечены как непригодные.

Шаг 3. Загрузка данных

Если запустить несколько сеансов SQL*Loader и установить опции direct и parallel в true, процессы будут работать параллельно. В некоторых типах рабочих систем может потребоваться поставить в конце каждой строки символ «&».

```
sqlldr userid=easydw/easydw
CONTROL=jan.ctl DIRECT=TRUE PARALLEL=TRUE
sqlldr userid=easydw/easydw
CONTROL=feb.ctl DIRECT=TRUE PARALLEL=TRUE
sglldr userid=easydw/easydw
CONTROL=mar.ctl DIRECT=TRUE PARALLEL=TRUE
```

Шаг 4. Проверка протокола

Ниже приведен фрагмент одного из файлов протокола. Заметьте, что режим установлен в direct с опцией parallel.

```
SQL*Loader: Release 9.2.0.1.0 — Production on
Sat Jun 15 11:01:21 2002
Copyright (c) 1982, 2002, Oracle Corporation.
All rights reserved.

Control File:       c:\feb.ctl
Data File:          c:\feb.dat
   Bad File:         c:\feb.bad
   Discard File:  none specified

 (Allow all discards)

Number to load:     ALL
Number to skip:     0
Errors allowed:     50
Continuation: none specified
Path used: Direct — with parallel option.

Load is UNRECOVERABLE; invalidation redo is produced.
Table PURCHASES, partition PURCHASES_FEB2002,
loaded from every logical record.
Insert option in effect for this partition: APPEND
```

Шаг 5. Включение всех ограничений и триггеров

Создайте заново все индексы. После использования параллельной загрузки в режиме прямого пути задействуйте снова все ограничения и триггеры, отключенные перед загрузкой. Создайте заново все удаленные индексы.

Проведение преобразований с помощью SQL*Loader

Если вы получили извлеченные файлы с данными, которые вы не хотите видеть в хранилище, можно использовать SQL*Loader для того, чтобы отфильтровать интересующие строки. Записи, соответствующие критерию загрузки, отбираются путем указания условия WHEN, тестирующего соответствие или несоответствие записи условию. Если запись не удовлетворяет условию WHEN, она заносится в файл отклоненных данных (discard file). Файл отклоненных данных содержит записи, отброшенные из-за несоответствия ни одному из критериев отбора записей, указанных в управляющем файле. Заметьте, что эти записи отличаются от тех, которые заносятся в файл ошибок (bad-файл). Отклоненные записи не обязательно содержат неверные данные. Условие WHEN можно использовать и в режиме прямого и условного пути. ,

SQL*Loader можно использовать для проведения простых преобразований символьных данных. Например, можно вставить фрагмент строки с помощью подстрочной функции. Два поля можно сцепить друг с другом оператором CONCAT. Вы можете отсечь первые или последние символы, используя оператор обрезания. В управляющем файле, приведенном ниже, иллюстрируется использование условия WHEN для отклонения тех строк, в которых значение PRODUCT_ID является пустым. Показано, как перевести содержимое столбца PRODUCT_ID в верхний регистр.

```
LOAD DATA
INFILE 'product.dat' append
INTO TABLE product WHEN product_id != BLANKS
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY "'"
(product_id "upper(:product_id)",
  product_name,
  category,
  cost_price,
  sell_price,
  weight,
  shipping_charge,
```

```
  manufacturer,
  supplier)
```

При вызове SQL*Loader из командной строки (как показано в примере) или из Мастера загрузки Oracle Enterprise Manager (как мы видели ранее) указывается файл для отклоненных данных (discard-файл).

```
sqlldr userid=easydw/easydw CONTROL=product.ctl
LOG=product.log BAD=product.bad DISCARD=product.dis DIRECT=true
```

В Oracle 9i преобразования такого типа можно выполнять и в режиме прямого, и в режиме условного пути. Тем не менее, поскольку они применяются к каждой отдельной записи, они серьезно влияют на производительность процесса загрузки.

*Операции, выполняемые SQL*Loader после загрузки*

После того как данные введены, может потребоваться обработать исключения, восстановить ограничения и перестроить индексы.

Проверка протоколов

Всегда просматривайте протоколы, чтобы убедиться, что загрузка прошла успешно. Проверьте правильность числа добавленных строк.

Обработка исключений процесса загрузки

Просмотрите файл с расширением .bad и выясните, какие строки не были добавлены. Записи, не соответствующие ограничению NOT NULL, отбрасываются и заносятся в bad-файл SQL*Loader.

Включение ограничений, связанных с целостностью данных

Проверьте ссылочную целостность, если вы еще этого не сделали. Убедитесь, что каждому внешнему ключу таблицы фактов соответствует первичный ключ таблицы измерения. В хранилище EASYDW каждая строка таблицы покупок PURCHASES должна иметь корректный идентификатор покупателя (CUSTOMER_ID) из таблицы покупателей CUSTOMERS и верный идентификатор продукта (PRODUCT_ID) из таблицы продуктов (PRODUCTS).

При использовании загрузки в режиме прямого пути были отключены ограничения CHECK и REFERENCES. При использовании параллельной загрузки в режиме прямого пути были отключены все ограничения.

Обработка нарушений ограничений

Чтобы выявить строки, содержащие неверные данные, вы можете создать таблицу исключений (exceptions table). Создайте таблицу с именем «exceptions», запустив скрипт UTLEXCPT.SQL. После включения ограничения, выведите таблицу, в которую должны были быть записаны исключения.

```
SQL> ALTER TABLE purchases ENABLE CONSTRAINT fk_product_id
       EXCEPTIONS INTO exceptions;
```

В нашем примере в двух строках имеется неверный идентификатор продукта. Была сделана продажа продукта, не существующего в измерении Продукты.

```
ROW_ID                 .   OWNER  TABLE_NAME   CONSTRAI
------------------         ----------  ----------  ----------
AAAC/ZAAMAAAAADAAF   EASYDW        PURCHASES     FK_PRODUCT_ID
AAAC/ZAAMAAAAADAAI   EASYDW        PURCHASES     FK_PRODUCT_ID
```

Чтобы определить, какие строки нарушают ссылочную целостность, выберите строки таблицы покупок (purchases) с идентификаторами строк (rowed), соответствующими таблице исключений. В данном примере есть две строки, не имеющие соответствующего продукта в измерении products.

```
SQL> SELECT * from purchases WHERE row_id in (select row_id from
exceptions);
PRODUCT_ TIME_KEY  CUSTOMER_I  PURCHASE_  PURCHASE_TIM PURCHASE_PRICE
-------  --------- ----------  ---------  ------------ ----------
XY1001   01-MAR-02 AB123457    Ol-MAR-02      1024        28 .01  4 .5 Y
AB1234   01-MAR-02 AB123456    Ol-MAR-02        24        28 .01  4 .5 Y
```

Совет. Важно исправлять все проблемы ссылочной целостности, особенно если вы используете систему управления сводными данными, которая полагается при переписывании запросов на правильность этих взаимосвязей.

Включение ограничений без проверки

Если целостность данных обеспечивает приложение или очистка данных уже проводилась и вы знаете, что нарушения ограничений, связанных с целостностью отсутствуют, включайте ограничения с опцией NOVALIDATE. Используйте условие RELYдля переписывания запросов. Поскольку управление сводными данными и прочие средства зависят от взаимоотношений, определяемых ссылочными ограничениями, нужно всегда определять ограничения, даже если они не проверяются.

```
SQL>ALTER TABLE purchases ENABLE NOVALIDATE CONSTRAINT fk_product_id;
SQL>ALTER TABLE purchases MODIFY CONSTRAINT fk_product_id RELY;
```

Выявите непригодные индексы

Перед публикацией данных в хранилище нужно провести проверку и выяснить, не являются ли какие-либо индексы непригодными. Индекс становится непригодным (unusable), если он больше не соответствует всем данным. Индекс может быть помечен как непригодный в силу различных причин, в том числе следующих.

- Вы указали отсрочить обработку индекса с помощью параметра skip_index_maintenance при запуске SQL*Loader.
- Не были отключены ограничения UNIQUE при использовании режима прямого пути SQL*Loader. По окончании загрузки ограничения проверяются после перестройки индексов. Если обнаруживаются дубликаты, индекс оказывается неверным и остается помеченным как непригодный.

Индекс нужно удалить и создать заново или перестроить, чтобы он вновь стал пригодным. Если лишь один из разделов помечен как непригодный, другие разделы этого индекса будут верными.

Чтобы выявить непригодные индексы, сделайте запрос к таблице USER_INDEXES так, как это показано в приведенном фрагменте кода. Можно видеть, что индекс PRODUCT_PK_INDEX является непригодным. Если индекс секционирован, его статус будет обозначен как N/A.

```
SQL> SELECT INDEX_NAME, STATUS FROM USER_INDEXES;

INDEX_NAME                     STATUS
-----------------              --------------
CUSTOMER_PK_INDEX              VALID
I_SNAP$_CUSTOMER_SUM           VALID
PRODUCT_PK_INDEX               UNUSABLE
PURCHASE_CUSTOMER_INDEX        N/A
PURCHASE_PRODUCT_INDEX         N/A
PURCHASE_SPECIAL_INDEX         N/A
PURCHASE_TIME_INDEX            N/A
TIME_PK_INDEX                  VALID
TSO_PK_INDEX                   VALID
9 rows selected.
```

Затем проверьте, не являются ли непригодными какие-нибудь из разделов индексов, прочитав таблицу USER_IND_PARTITIONS. В следующем примере непригодным является индекс PURCHASE_PRODUCT_INDEX для раздела purchases_apr2002.

```
SQL> SELECT INDEX_NAME, PARTITION_NAME, STATUS FROM
    USER_IND_PARTITIONS WHERE STATUS != 'VALID';
INDEX_NAME                PARTITION_NAME          STATUS
----------------------    --------------------    --------
PURCHASE_PRODUCT_INDEX    PURCHASES_JAN2002       USABLE
PURCHASE_PRODUCT_INDEX    PURCHASES_FEB2002       USABLE
PURCHASE_PRODUCT_INDEX    PURCHASES_MAR2002       USABLE
PURCHASE_PRODUCT_INDEX    PURCHASES_APR2002       UNUSABLE
```

Перестройка непригодных индексов

Если некоторые ваши индексы оказались непригодны, их нужно перестроить перед тем, как осуществлять доступ к таблице или разделу. В следующем примере перестраивается индекс purchases_product_index для только что добавленного раздела purchase_apr2002.

```
SQL> ALTER INDEX purchase_product_index
    REBUILD PARTITION purchases_apr2002;
```

5.4.2. Загрузка данных в хранилище с использованием внешних таблиц

Внешняя таблица (external table) — это таблица, хранимая вне базы данных в последовательном файле. К данным внешней таблицы можно обращаться точно так же, как и к данным таблицы, хранящейся в базе. Можно выбирать столбцы, строки и проводить соединение данных с другими таблицами с помощью SQL. К данным внешней таблицы, как и к таблицам, хранящимся в базе, можно обращаться с использованием параллельных процессов.

Внешние таблицы доступны только для чтения. Не позволяются никакие операции DML, и вы не можете создавать индексы по внешней таблице. Если вам нужно обновить эти данные или обратиться к ним более чем один раз, вы можете загрузить данные в базу, где можно обновить их, и добавить индексы для улучшения производительности обработки запросов. Загружая данные в базу, вы получаете возможность управлять ими как частью базы данных. Утилита RMAN не делает резервных копий данных внешних таблиц.

Поскольку вы можете посылать запросы к данным внешних таблиц, которые вы только что определили с помощью команд SQL, вы также можете загружать в них данные с помощью инструкции INSERT SELECT. Внешние таблицы являются альтернативой SQL*Loader при загрузке данных из внешних файлов. Их можно использовать для выполнения более сложных преобразований при загрузке данных и для упрощения многих рабочих аспектов параллельной загрузки, управления триггерами, ограничениями и индексами. Однако во многих случаях режим прямой загрузки Oracle 9i SQL*Loader может оказаться быстрее. Между этими двумя методами есть много сходств, и если вы использовали SQL*Loader, вы легко научитесь использовать внешние таблицы.

Указание положения файла данных и файлов протоколов для внешней таблицы

Перед тем как создавать внешнюю таблицу, нужно создать объект-директорию, в котором указывается положение файлов данных и файлов протоколов. Файлы данных и протоколов должны располагаться на той же машине, что и сервер базы данных, или должны быть доступны для сервера. В этом отличие от SQL*Loader, в котором клиент SQL*Loader посылает данные серверу. Директории создаются в одном пространстве имен и не принадлежат к схемам отдельных пользователей. В следующем примере администратор базы создает директории для файлов данных и файлов протоколов.

```
SQL> CREATE OR REPLACE DIRECTORY data_file_dir as 'C:\datafiles\';
SQL> CREATE OR REPLACE DIRECTORY log_file_dir as 'C:logfiles\';
```

Пользователям, осуществляющим доступ к файлам данных в директории, необходимо дать права для чтения. В следующем примере администратор базы дает права для чтения пользователю EASYDW.

```
SQL> GRANT READ ON DIRECTORY data_file_dir to easydw;
SQL> GRANT READ ON DIRECTORY log_file_dir to easydw;
```

Посмотрев в представление DBA_DIRECTORIES, мы можем увидеть созданные директории.

```
SQL> SELECT * FROM DBA_DIRECTORIES;
OWNER          DIRECTORY_NAME       DIRECTORY_PATH
----------     ----------------     --------------
SYS            LOG_FILE_DIR         c:\logfiles\
SYS            DATA_FILE_DIR        c:\datafiles\
```

Создание внешней таблицы

Чтобы создать внешнюю таблицу, нужно указать следующие данные.

- Метаданные, описывающие, как данные будут выглядеть для базы Oracle, включая имя внешней таблицы, названия столбцов и типы данных базы Oracle. Именно эти имена вы будете использовать в инструкциях SQL для доступа к внешней таблице. Метаданные хранятся в словаре данных базы Oracle (Oracle data dictionary). В примере, показанном на рис. 5.13, имя внешней таблицы — new_products. Данные хранятся вне базы. В этом состоит отличие от других обычных таблиц базы, для которых и данные, и метаданные хранятся в базе.
- Параметры доступа (access parameters), описывающие, каким образом данные хранятся во внешнем файле, где он расположен, его формат и как идентифицировать поля и записи. Информация похожа на ту, которая содержится в управляющем файле SQL*Loader. Параметры доступа использует драйвер доступа (access driver). Пример параметров доступа приводится в выражении CREATE EXTERNAL TABLE.

В Oracle 9i есть драйвер доступа только одного типа, называемый ORACLE_LOADER, обеспечивающий доступ только для чтения к последова-

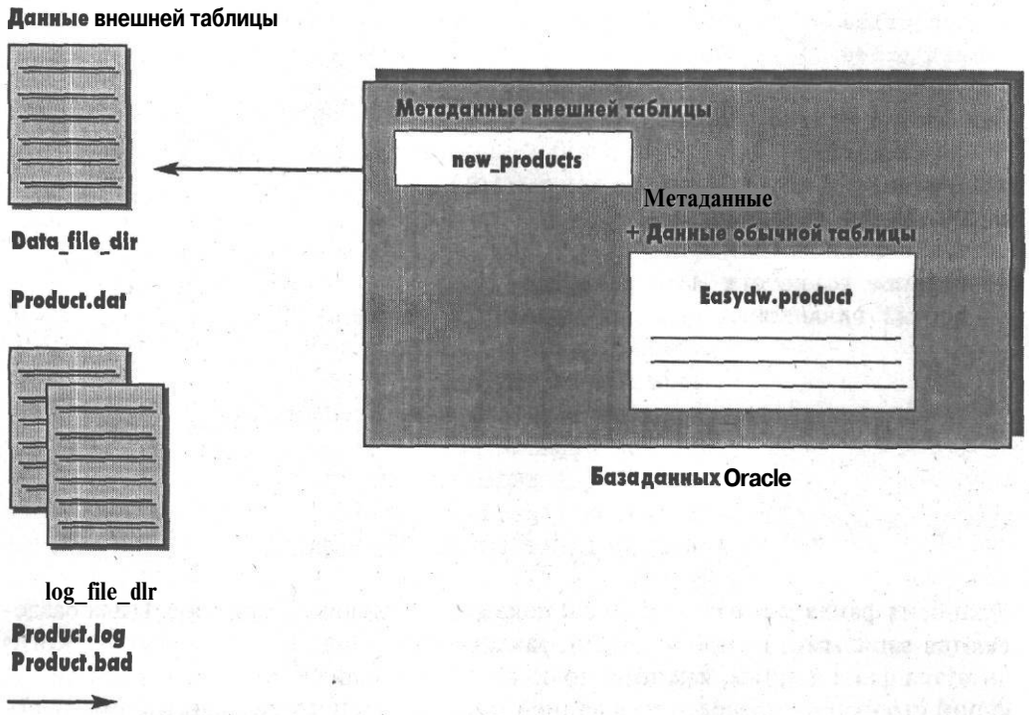


Рис. 5.13. *Внешняя таблица*

тельным файлам. Он указан в условии TYPE выражения CREATE EXTERNAL TABLE.

В следующем примере показан код SQL, создающий внешнюю таблицу. Используется условие CREATE TABLE ORGANIZATION EXTERNAL. Перечисляются метаданные, описывающие названия столбцов и типы данных. В условии TYPE указывается тип драйвера доступа и начинается описание того, как файл хранится на диске. Каждое поле отделяется запятой и, по желанию, заключается в апострофы. Указывается положение файла данных, файла протокола и файла ошибок. При указании условия parallel данные будут вводиться с использованием параллельных процессов.

```
CREATE TABLE new_products
 (product_id          VARCHAR2(8),
  product_name              VARCHAR2(30) ,
  category                   VARCHAR2(4),
```

```
  cost_price        NUMBER ' (6,2) ,
  sell_price         NUMBER  (6,2),
  weight                    NUMBER  (4,2),
  shipping_charge   NUMBER  (5,2),
  manufacturer              VARCHAR2(20),
  supplier                  VARCHAR2(10))
ORGANIZATION EXTERNAL
 (TYPE ORACLE_LOADER
   DEFAULT DIRECTORY data_file_dir
   ACCESS PARAMETERS (RECORDS DELIMITED BY NEWLINE
                      CHARACTERSET US7ASCII
                      BADFILE log_file_dir:'product.bad'
                      LOGFILE log_file_dir:'product.log'
                      FIELDS TERMINATED BY ','
                      OPTIONALLY ENCLOSED BY.....)
                      LOCATION ('product.dat'))
                      REJECT LIMIT UNLIMITED PARALLEL;
```

Фрагмент файла данных product.dat показан в следующем примере. Поля разделяются запятыми, и, при желании, заключаются в одиночные кавычки. Когда читается файл данных, каждому полю вводимого файла находится соответствующий столбец в определении внешней таблицы. При чтении данные при необходимости конвертируются из типа данных вводимого файла в тип столбца базы.

```
'SP1000','Camera','ELEC',45.67,67.23,15.00,4.50,'Ricoh','Ricoh'
'SP1001','APS Camera','ELEC',24.67,36.23,5.00,4.50,'Ricoh','Ricoh'
'SP1010','Camera','ELEC',35.67,47.89,5.00,4.50,'Agfa','Agfa'
```

После выполнения команды CREATE TABLE, показанной ранее, метаданные для таблицы new_products сохраняются в базе данных. Эта таблица выглядит следующим образом.

```
SQL> DESCRIBE new_products;

Name                 Null?        Type
-------------        --------     ------------
PRODUCT_ID                        VARCHAR2(8)
PRODUCT_NAME                      VARCHAR2(30)
CATEGORY                          VARCHAR2(4)
```

```
COST_PRICE                    NUMBER(6,2)
SELL_PRICE                    NUMBER(6,2)
WEIGHT                        NUMBER(4,2)
SHIPPING_CHARGE               NUMBER(5,2)
MANUFACTURER                  VARCHAR2(20)
SUPPLIER                      VARCHAR2(10)
```

Представление-словарь USER_EXTERNAL_TABLES показывает созданные внешние таблицы, а также их описание. Представление-словарь USER_EXTERNAL_LOCATIONS показывает местоположение файла данных.

```
SQL> SELECT TABLE_NAME, TYPE_NAME, DEFAULT_DIRECTORY_NAME
    FROM USER_EXTERNAL_TABLES;

TABLE_NAME           TYPE_NAME           DEFAULT_DIRECTORY_NAME
-------------        ---------------     ----------------------
NEW_PRODUCTS         ORACLE_LOADER       DATA_FILE_DIR

SQL> SELECT * FROM. USER_EXTERNAL_LOCATIONS;

TABLE_NAME           LOCATION       DIR   DIRECTORY_NAME
-------------        ------------   ---   --------------
NEW_PRODUCTS         product.dat    SYS   DATA_FILE_DIR
```

Доступ к данным, хранимым во внешней таблице

После создания внешней таблице к ней можно осуществлять доступ с помощью SQL так же, как и к любой другой таблице базы данных, хотя данные и будут читаться из файла вне базы. Фрагмент выводимой информации приведен ниже.

```
SQL> SELECT * FROM NEW_PRODUCTS;

PRODUCT   PRODUCT      CAT     COST    SELL  WEIGHT  SHIPPING  MANUF  SUPPL
ID        NAME         PRICE   PRICE                 CHARGE
-------   ----------   ------  -----   ----  ------  --------  -----  -------
SP1000    Camera       ELEC    45.67   67.23   15      4.5      Ricoh  Ricoh
SP1001    APS Camera   ELEC    24.67   36.23    5      4.5      Ricoh  Ricoh
SP1010    Camera       ELEC    35.67   47.89    5      4.5      Agfa   Agfa
```

Ввод данных из внешней таблицы

В следующем примере для загрузки данных в таблицу измерения PRODUCT используется инструкция INSERT/SELECT. Выполнять преобразования при загрузке с помощью функций и арифметических операторов SQL очень легко. В данном примере после того как файл данных был создан, возросла стоимость топлива и компания-поставщик повысила ставки на 10%. В этом примере данные вводятся в таблицу EASYDW.PRODUCT путем отбора столбцов из внешней таблицы NEW_PRODUCTS. При загрузке данных стоимость доставки каждого товара увеличивается на 10%.

```
SQL> INSERT INTO easydw.product
       (product_id, product_name, category, cost_price,
        sell_price, weight, shipping_charge, manufacturer,
        supplier)
     SELECT product_id, product_name, category, cost_price,
            sell_price, weight,
            (shipping_charge + (shipping_charge * .10)),
            manufacturer, supplier FROM new_products;
```

Данные загружены в таблицу EASYDW.PRODUCT, и мы можем видеть, что стоимость доставки показанных товаров повысилась с $4.50 до $4.95.

```
SQL> select * from easydw.product;

PRODUCT  PRODUCT      CAT    COST   SELL WEIGHT  SHIPPING MANUF SUPPL
IP       NAME         PRICE  PRICE               CHARGE
-------  ------------ ------ ------ ---- ------  -------- ----- -------
SP1000   Camera       ELEC   45.67  67.23   15   4.5      Ricoh Ricoh
SP1001   APS Camera   ELEC   24.67  36.23    5   4.5      Ricoh Ricoh
SP1010   Camera       ELEC   35.67  47.89    5   4.5      Agfa  Agfa
```

Параллельная загрузка данных с использованием внешних таблиц

Еще одно преимущество использования внешних таблиц — возможность параллельной загрузки данных без необходимости разрезать файл на меньшие части и запускать несколько сеансов, как это приходится делать при использовании SQL*Loader. Степень параллелизма устанавливается с помощью стандартных инструкций или с помощью условия PARALLEL при создании

внешней таблицы, как это показано выше. Выводные данные процедуры EXPLAIN PLAN показывают наличие параллельного доступа.

```
SQL> EXPLAIN PLAN FOR INSERT INTO easydw.product
     (product_id, product_name, category, cost_price,
      sell_price, weight, shipping_charge, manufacturer,
      supplier)
     SELECT product_id, product_name, category, cost_price,
      sell_price, weight,
       (shipping_charge + (shipping_charge * .10)),
      manufacturer, supplier FROM new_products;
```

Процедура utlxplp.sql используется для вывода данных процедуры EXPLAIN PLAN, содержащих отредактированные столбцы, подлежащие параллельной обработке и выделенные для удобства жирным шрифтом.

```
SQL> @c:\ora9iR2\rdbms\admin\utlxplp.sql

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
|Id|Operation|Name     |Rows|Bytes|Cost|TQ    |IN-OUT|PQ Dis.trib|
------------------------------------------------------------------------
| 0|INSERT   |         |8168| 781K| 4  |      |      |           |
| 1|EXTERNAL |NEW_     |8168| 781K| 4  |20,00|P->S   |QC         |
|  |TABLE    |PRODUCTS |    |     |    |      |      |RANDOM     |
|  |ACCESS   |         |    |     |    |      |      |           |
|  |FULL     |         |    |     |    |      |      |           |
------------------------------------------------------------------------
```

Загрузка измерений с помощью команды SQL MERGE

При загрузке новых данных в существующие таблицы измерений может потребоваться добавлять новые строки и вносить изменения в имеющиеся. В прошлом для того чтобы отличить новую строку от измененной, была нужна специальная программная логика. В Oracle 9i была добавлена новая возможность, значительно облегчающая этот процесс, — SQL MERGE. Часто это называется «подстановка» (upsert). Если строка существует, она обновляется. Если строка отсутствует, она вставляется.

В следующем примере в измерение customer (покупатели) добавляются новые данные. Вводимый файл содержит как данные о новых покупателях, так и изменения имеющихся данных. В этом случае не нужно хранить старую информацию, так что данные будут обновлены. Для файла, содержащего обновления к измерению customer, создается внешняя таблица с именем CUSTOMER_CHANGE.

```
CREATE TABLE easydw.customer_changes
 (customer_id        VARCHAR2(10),
  town               VARCHAR2(10),
  county             VARCHAR2(10),
  postal_code        VARCHAR2(10),
  dob                DATE,
  country            VARCHAR2(20),
  occupation         VARCHAR2(10))
ORGANIZATION EXTERNAL
   (TYPE oracle_loader
      DEFAULT DIRECTORY data_file_dir
      ACCESS PARAMETERS  (RECORDS DELIMITED BY NEWLINE
                          CHARACTERSET US7ASCII
                          BADFILE log_file_dir:'cus_changes.bad'
                          LOGFILE log_file_dir:'cust_changes.log'
                          FIELDS TERMINATED BY  ','
                          OPTIONALLY ENCLOSED BY. . . . .)
                          LOCATION  ('customer_changes.dat'))
                          REJECT LIMIT UNLIMITED NOPARALLEL;
```

Здесь вместо команды INSERT используется команда MERGE (слияние). Эта команда состоит из двух частей. Когда customer_id из таблицы customer_table соответствует customer_id из таблицы customer_changes, строка обновляется. Если эти значения не соответствуют, вставляется новая строка.

```
MERGE INTO easydw.customer c
USING easydw.customer_changes cc
ON (c.customer_id = cc.customer_id)
WHEN MATCHED THEN UPDATE SET
c.town=cc.town,
c.county=cc.county,
c.postal_code=cc.postal_code,
    c.dob=cc.dob,
```

```
c.country=cc.country,
c.occupation=cc.occupation
WHEN NOT MATCHED THEN INSERT
     (customer_id, town, county, postal_code,
      dob, country, occupation)
    VALUES
     (cc.customer_id, cc.town, cc.county, cc.postal_code,
      cc.dob, cc.country, cc.occupation);
```

Перед слиянием исходной и измененной информации о покупателях у нас было 45 покупателей.

```
SQL> SELECT COUNT(*) FROM customer;

  COUNT(*)
----------
        45
```

Когда создана внешняя таблица, мы можем использовать SQL и посмотреть файл данных таблицы customer_changes. Первыми идут две строки, которые будут обновлены. Первая клиентка, которая ранее была домашней хозяйкой, вернулась к работе инженера. Второй покупатель переехал из Soton2 в Soton. Последние две строки соответствуют двум новым покупателям.

```
SQL> SELECT * FROM customer_changes;

CUSTOMER_ID TOWN    COUNTY   POSTAL_CODE DOB       COUNTRY    OCCUPATION
----------- ------  -------  ----------- --------- -------    ----------
AB123459    Soton   Hants    SO11TF      23-SEP-27     UK     ENGINEER
AB123460    Soton   Hants    S011TF      23-SEP-27     UK     HOUSEWIFE
AA114778    London  London   W11QC       14-APR-56     UK     ENGINEER
AA123478    London  London   W11QC       14-APR-56     UK     ENGINEER
```

Посмотрев на фрагмент измерения CUSTOMER, мы можем увидеть строки, в которых столбец CUSTOMER_ID соответствует столбцу CUSTOMER_ID внешней таблицы CUSTOMER_CHANGE. Эти строки и будут обновлены.

```
SQL> SELECT * FROM customer;
```

```
CUSTOMER_ID TOWN    COUNTY   POSTAL_CODE DOB       COUNTRY    OCCUPATION
----------- ------  -------  ----------- --------- -------    ----------
AB123459    Soton   Hants    SO11TF      23-SEP-27     UK     HOUSEWIFE
AB123460    Soton2  Hants    S011TF      23-SEP-27     UK     HOUSEWIFE
```

После выполнения оператора MERGE у нас будет 47 покупателей. Ниже выведены только измененные строки таблицы CUSTOMER.

```
SQL> SELECT COUNT(*) FROM customer;
COUNT(*)
--------
47
SQL> SELECT * FROM customer;

"CUSTOMER_ID TOWN    COUNTY   POSTAL_CODE DOB       COUNTRY    OCCUPATION
------------ ------  -------  ----------- --------- -------    ----------
AB123459     Soton   Hants    SO11TF      23-SEP-27     UK     ENGINEER
AB123460     Soton   Hants    SO11TF      23-SEP-27     UK     HOUSEWIFE
AA114778     London  London   W11QC       14-APR-56     UK     ENGINEER
AA123478     London  London   W11QC       14-APR-56     UK     ENGINEER
```

В этом примере оператор MERGE был использован с внешними таблицами. Его также можно использовать с любыми пользовательскими таблицами.

5.4.3. Загрузка данных в хранилище с использованием транспортируемых табличных пространств

Самым быстрым способом переноса данных из одной базы данных Oracle в другую является использование транспортируемых табличных пространств (transportable tablespaces). Транспортируемые табличные пространства обеспечивают механизм переноса одного или более табличного пространства из одной базы Oracle в другую. Вместо того чтобы обрабатывать за раз по одной строке, весь файл или набор файлов копируется в другую базу, а потом интегрируется в нее путем импортирования метаданных, которые описывают таблицы в файлах. Помимо данных, можно также перемещать индексы.

Поскольку данные не выгружаются и не загружаются, операция переноса может быть быстрой, однако данные нельзя конвертировать в другой формат. Таким образом, в Oracle 9i вы можете транспортировать табличные пространства между базами Oracle, использующими один и тот же набор символов и одну операционную систему. Если требуются какие-либо операции преобразования, нужно использовать другой механизм.

Транспортируемые табличные пространства можно использовать для перемещения данных из рабочей базы данных в промежуточную область, если вы используете в качестве таковой другую базу Oracle. Транспортируемые табличные пространства также полезны при переносе данных из большого хранилища в зависимое тематическое хранилище (data mart).

На рис. 5.14 показаны шаги, выполняемые в ходе использования транспортируемых табличных пространств. Эти шаги таковы.

1. Создать новое транспортируемое табличное пространство.
2. Перенести данные, которые вы хотите переместить, в собственное табличное пространство.
3. Сделать это табличное пространство доступным только для чтения.
4. Использовать утилиту экспорта для получения метаданных, описывающих объекты табличного пространства.
5. Скопировать файлы данных и экспортируемый дамп-файл, содержащий метаданные, в целевую систему.
6. Использовать утилиту импорта для ввода описаний метаданных в целевую базу.
7. Сделать табличное пространство доступным для чтения и записи.
8. Выполнить преобразование.
9. Перенести данные из промежуточной области в таблицу фактов хранилища.

В этом разделе мы поговорим о шагах 1–7. В последующих разделах мы рассмотрим шаги 8 и 9.

Одним из источников данных для хранилища EASYDW является система обработки заказов на базе Oracle 9i. В этом примере система ввода заказов содержит запись для каждого заказа, хранимого в таблице PURCHASES (покупки). В конце апреля месяца все заказы (orders) за апрель 2002 года будут скопированы в новую таблицу, называемую APR_ORDERS в табличном пространстве ORDERS, которое хранится в файле данных с именем orders.f.

Вот шаги, необходимые для переноса табличного пространства из одной базы данных в другую.

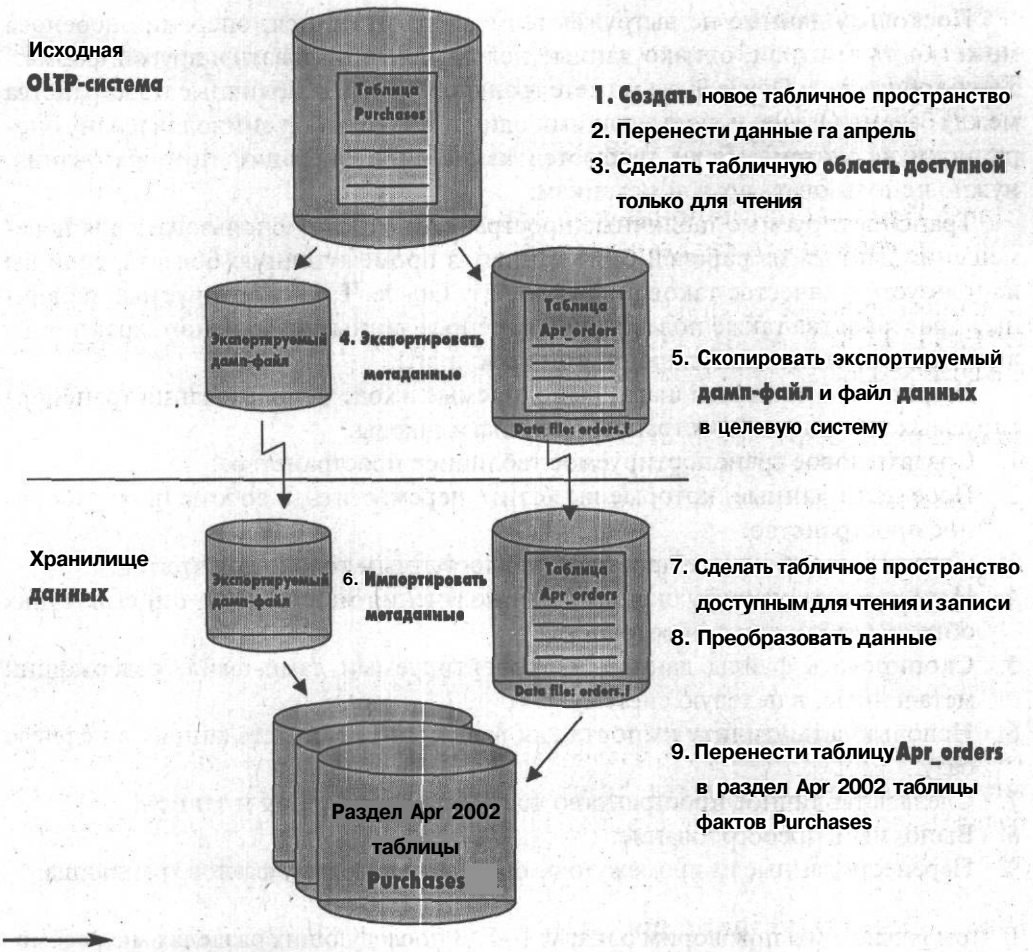


Рис. **5.14.** *Транспортируемые табличные пространства*

Шаг 1. Создание табличного пространства в OLTP-системе

Выберите имя, являющееся уникальным как для исходной, так и для целевой системы. В данном примере создается табличное пространство ORDERS и соответствующий ему файл данных orders.f.

```
CREATE TABLESPACE orders
datafile 'c:\ora9ir2\oradata\orcl\orders.f' SIZE 5M REUSE
AUTOEXTEND ON
DEFAULT STORAGE
(INITIAL 64K PCTINCREASE 0 MAXEXTENTS UNLIMITED);
```

Шаг 2. Перемещение данных за апрель 2002 года в таблицу в только что созданном табличном пространстве

В данном примере таблица создается и заполняется с помощью инструкции CREATE TABLE AS SELECT. Таблица создается в табличном пространстве ORDERS. В данном примере за апрель было сделано 150004 заказа.

```
SQL> CREATE TABLE apr_orders TABLESPACE orders AS
     SELECT * FROM purchases WHERE
     purchase_date BETWEEN '01-APR-2002' AND '30-APR-2002';

SQL> SELECT COUNT(*) FROM apr_orders;

COUNT(*)
--------
   15004
```

Каждая таблица должна быть изолированной и не должна ссылаться на что-либо вне табличного пространства. Если имеется глобальный индекс по таблице, он будет неизолированным, и его перед перемещением табличного пространства нужно отбросить.

Шаг 3. Сделать таблицу доступной только для чтения

Если вы этого не сделаете, операция экспортирования, выполняемая на следующем этапе, не пройдет.

```
ALTER TABLESPACE orders READ ONLY;
```

Шаг 4. Экспортирование метаданных

Определения метаданных для табличного пространства ORDERS извлекаются и сохраняются в экспортируемом дамп-файле с именем expdat.dmp с помощью команды EXPORT.

```
exp 'system/manager AS SYSDBA' TRANSPORT_TABLESPACE=y
    TABLESPACES=orders TRIGGERS=n CONSTRAINTS=n GRANTS=n
    FILE=expdat.dmp LOG=export.log
```

При переносе набора табличных пространств можно включить в них ссылочные ограничения, связанные с целостностью данных. Однако, если вы решите сделать это, вы должны будете перемещать таблицы как с первичными, так и с внешними ключами.

Совет. При экспортировании и импортировании табличных пространств вы должны обязательно входить в систему как системный администратор.

Приведенный пример представляет собой копию файла протокола export.log, сгенерированного в ходе экспорта. Экспортируются только метаданные табличного пространства, а не данные.

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 -
Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
Export done in WE8MSWIN1252 character set and AL16UTF16 NCHAR
character set
Note: table data (rows) will not be exported
Note: grants on tables/views/sequences/roles will not be exported
Note: constraints on tables will not be exported
About to export transportable tablespace metadata...
For tablespace ORDERS ...
. exporting cluster definitions
. exporting table definitions
. . exporting table APR_ORDERS
. end transportable tablespace metadata export
Export terminated successfully without warnings.
```

Шаг 5. Перенос табличного пространства

Теперь копируем файл данных orders.f и экспортируемый файл expdat.dmp в физическую область системы, содержащей промежуточную базу данных. Можно ис-

пользовать любые функции для копирования последовательных файлов, такие, как функции операционной системы или FTP. Копировать нужно в двоичном режиме, поскольку эти файлы не являются файлами ASCII.

В нашем примере файл orders.f был скопирован в файл c:\ora9ir2\oradata\orcl\orders.fв промежуточную систему с помощью FTP.

Шаг 6. Импортирование метаданных

При импортировании метаданных вы вставляете табличное пространство в целевую базу данных.

```
imp 'system/manager AS SYSDBA' TRANSPORT_TABLESPACE=Y
Datafiles='c:\ora9ir2\oradata\orcl\orders.f'
FILE=expdat.dmp LOG=import.log
```

Проверьте протокол импортирования, чтобы убедиться, что не возникло никаких ошибок. Отметьте, что были импортированы метаданные промежуточного табличного пространства.

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 -
 Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 — Production

Export file created by EXPORT:V09.02.00 via conventional path
About to import transportable tablespace(s) metadata...
import done in WE8MSWIN1252 character set and AL16UTF16 NCHAR
character set
. importing SYS's objects into SYS
. importing SYSTEM's objects into SYSTEM
. . importing table "APR_ORDERS"
Import terminated successfully without warnings.
```

Проверьте соответствие суммарных данных данным OLTP-системы.

```
SQL> SELECT COUNT(*) FROM apr_orders;

COUNT(*)
--------
   15004
```

Шаг 7. Сделайте табличное пространство ORDERS доступным для чтения и записи

Теперь все готово для проведения преобразований!

```
SQL> ALTER TABLESPACE orders READ WRITE;
```

Как можно видеть в ходе прохождения этих этапов, строки таблицы в базе данных не выгружаются и не перезагружаются. Таким образом, применение транспортируемых табличных пространств является самым быстрым способом переноса данных между двумя базами Oracle.

5.5. Преобразование данных в Oracle Server

Раздел 5.3 знакомит читателя с преобразованиями и помогает выбрать оптимальное место для проведения этих преобразований. Мы видели примеры преобразования данных в ходе их загрузки с помощью SQL*Loader и внешних таблиц. В этом разделе мы обсудим проведение преобразований в Oracle Server. Если вы выполняете преобразования в хранилище данных Oracle, вы, скорее всего, загружаете данные во временные промежуточные таблицы, преобразуете их, а затем переносите их в таблицу фактов хранилища.

При использовании транспортируемых табличных просранств данные (как говорилось в предыдущем разделе) переносятся из OLTP-системы в промежуточную область хранилища Oracle. Данные, загруженные с помощью внешних таблиц и SQL*Loader, можно далее преобразовывать в промежуточной области в Oracle Server. Oracle 9i предоставляет процедуры для проведения преобразований на SQL, PL/SQL и Java. В этом разделе мы рассмотрим процедуры SQL для проведения следующих простых преобразований:

- удаление символа «-» из идентификатора продукта и повышение платы за поставку на 10%;
- проверка на предмет ошибочных идентификаторов продуктов;
- поиск ключа хранилища и замена его на идентификатор продукта.

Теперь мы запишем первое преобразование в виде табличной функции.

5.5.1. Преобразования, связанные с очисткой имеющихся данных и образованием новых

Инструкцию SQL UPDATE и встроенные функции можно использовать для проведения некоторых простых преобразований. В качестве продолжения нашего примера нам нужно очистить таблицу APR_ORDERS. Некоторые идентификаторы продуктов (PRODUCT_ID) содержат символ «-», который нужно удалить. Повысилась цена на топливо, и к стоимости поставки необходимо добавить 10%. Обе эти операции можно выполнить за один раз с помощью оператора UPDATE. Приведенные несколько строк были обновлены.

```
SQL> SELECT * FROM apr_orders;

PRODUCT   TIME_KEY   CUSTOMER   PURCHASE   PURCH   PURCH   SHIP   TODAYS_
ID                   ID         Date       Time    Price   Charge Special
-------   ---------  ---------  ---------  -----   -----   ------ --------
SP1001    02-APR-02  AB123457   02-APR-02  1024    28.01   5.45   N
SP-1000   01-APR-02  AB123456   01-APR-02  1001    67.23   5.45   N
SP-1000   01-APR-02  AB123457   01-APR-02  1002    67.23   5.45   N
```

Для обновления значения shipping charge (стоимость доставки) используется оператор умножения (*) SQL, а функция REPLACE используется для замены символа «-» на пустые кавычки «» (т. е. для его удаления). Измененные поля показаны жирным шрифтом. Стоимость доставки повысилась с $5.45 до $6. Из идентификатора продукта убран дефис.

```
SQL> UPDATE apr_orders SET
shipping_charge = (shipping_charge + shipping_charge *.10),
product_id=REPLACE(product_id, '-','');

PRODUCT   TIME_KEY   CUSTOMER   PURCHASE   PURCH   PURCH   SHIP   TODAYS_
ID                   ID         Date       Time    Price   Charge Special
-------   ---------  ---------  ---------  -----   -----   ------ --------
SP1001    02-APR-02  AB123457   02-APR-02  1024    28.01   6      N
SP1000    01-APR-02  AB123456   01-APR-02  1001    67.23   6      N
SP1000    01-APR-02  AB123457   01-APR-02  1002    67.23   6      N
```

5.5.2. Проверка данных с помощью измерения

Часто входящие данные требуется проверять с помощью информации, содержащейся в таблицах измерений. Хотя в действительности эта операция и не является преобразованием, она описывается здесь, поскольку она часто производится перед загрузкой данных в таблицы хранилища. В нашем примере мы хотим убедиться, что все идентификаторы продуктов для апрельских заказов верны. Идентификаторы продуктов (PRODUCT_ID) заказов должны совпадать с идентификаторами продуктов измерения product.

Приводимый запрос показывает, что в данных за апрель месяц есть неверный идентификатор продукта, не совпадающий с PRODUCT_CODE в измерении product. Все неверные данные должны быть исправлены до ввода их из промежуточной области в хранилище.

```
SQL> SELECT DISTINCT product_id FROM apr_orders WHERE product_id
NOT IN (SELECT product_id FROM product);

PRODUCT_ID
----------
     SP1036
```

5.5.3. Поиск ключа хранилища

Теперь, когда мы очистили столбец PRODUCT_ID, мы изменим его под использование ключа хранилища. В следующем примере к таблице product добавляется product_code, являющийся суррогатным ключом хранилища, который будет использоваться для поиска идентификатора продукта (product_id). На рис. 5.4 показано использование суррогатных ключей хранилища. Выведен фрагмент измерения product.

```
SQL> SELECT PRODUCT_ID, PRODUCT_CODE FROM PRODUCT;

PRODUCT_ID   PRODUCT_CODE
----------   ------------
         1    SP1000
         2    SP1001
         3    SP1010
         4    SP1011
         5    SP1012
```

В следующем преобразовании мы будем использовать PRODUCT_ID таблицы APR_ORDERS для поиска ключа хранилища в таблице product. Столбец PRODUCT_ID таблицы APR_ORDERS будет заменен на ключ хранилища.

```
SQL> SELECT * FROM APR_ORDERS;

PRODUCT   TIME_KEY   CUSTOMER   PURCHASE   PURCH   PURCH   SHIP   TODAYS_
ID                   ID         Date       Time    Price   Charge Special
-------   ---------  ---------  ---------  -----   -----   ------ --------
SP1001    01-APR-02  AB123456   Ol-APR-02  24      28.01   6      Y
SP1001    01-APR-02  AB123457   Ol-APR-02  1024    28.01   6      Y
SP1061    Ol-APR-02  AB123456   Ol-APR-02  24      28.01   8.42   Y
SP1062    Ol-APR-02  AB123457   Ol-APR-02 .1024    28.01   3.58   Y

SQL> UPDATE APR_ORDERS A SET A.PRODUCT_ID =
 (SELECT P.PRODUCT_ID FROM PRODUCT P
WHERE A.PRODUCT_ID = P.PRODUCT_CODE);

4 rows updated.
```

Заметьте, что исходное значение PRODUCT_ID было заменено на ключ хранилища.

```
SQL> SELECT * FROM APR_ORDERS;

PRODUCT   TIME_KEY   CUSTOMER   PURCHASE   PURCH   PURCH   SHIP   TODAYS_
ID                   ID         Date       Time    Price   Charge Special
-------   ---------  ---------  ---------  -----   -----   ------ --------
2         Ol-APR-02  AB123456   Ol-APR-02  24      28.01   6      Y
2         Ol-APR-02  AB123457   Ol-APR-02  1024    28.01   6      Y
54        Ol-APR-02  AB123456   Ol-APR-02  24      28.01   8.42   Y
55        Ol-APR-02  AB123457   Ol-APR-02  1024    28.01   3.58   Y
```

5.5.4. Табличные функции

Результаты одного преобразования часто сохраняют в таблице базы данных. Затем эта таблица используется как входная информация для следующего преобразования. Такой процесс преобразований и сохранений промежуточных результатов повторяется для каждого преобразования в последовательности.

Слабое место такого подхода — производительность. Цель в том, чтобы выполнить все преобразования таким образом, чтобы каждая запись была прочитана, преобразована и обновлена только один раз. Конечно, бывают ситуации, когда это невозможно и по данным нужно пройти несколько раз.

Табличная функция (table function) — это функция, входные данные которой представляют собой набор строк, и выходные данные — также набор строк, которые могут быть в виде таблицы (отсюда и название — табличная функция). Наборы строк можно обрабатывать параллельно, а результаты одной можно передать по каналу к другой, пока преобразование не будет закончено для всех строк в наборе, что устраняет необходимость проходить по данным несколько раз.

Табличные функции используют объектную технологию Oracle и пользовательские типы данных. Во-первых, нужно определить новые типы данных для входящей записи и выходящей таблицы. В приводимом примере определяется тип данных PURCHASES_RECORD для описания записей таблицы PURCHASES.

```
SQL> CREATE TYPE purchases_record as OBJECT
(product_id                 VARCHAR2(8),
time_key                    DATE,
customer_id                 VARCHAR2(10),
purchase_date               DATE,
purchase_time               NUMBER(4,0),
purchase_price              NUMBER(6,2),
shipping_charge             NUMBER(5,2),
today_special_offer         VARCHAR2(1));
```

После этого определяется тип данных PURCHASES_TABLE. В него входит коллекция данных типа PURCHASES_RECORD, которые будут возвращаться функцией как выходные данные.

```
SQL> CREATE TYPE purchases_table AS TABLE of purchases_record;
```

Теперь определим тип данных для переменной-курсора, которая будет использоваться для передачи набора строк в табличную функцию. Переменные-курсоры представляют собой указатели, хранящие адрес некоторого элемента, а не сам этот элемент. В PL/SQL указатель создается с помощью типа данных REF. Поэтому переменная-курсор имеет тип REF CURSOR. Чтобы создавать переменные-курсоры, нужно сначала определить тип REF CURSOR.

```
CREATE PACKAGE cur_pack AS TYPE ref_cur_type IS REF CURSOR;
END cur_pack;
```

Ниже приводится наша табличная функция TRANSFORM, выполняющая операции поиска и замены, удаляющая тире из идентификатора продукта и увеличивающая стоимость доставки. Есть некоторые моменты, отличающие эту функцию от других. В этой функции в определении используется условие PIPELINED, а в теле — условие PIPE ROW. Это заставляет функцию возвращать строки по мере завершения работы с ними и не ждать обработки всех строк. Входом функции служит переменная-курсор INPUTRECS типа REF_CUR_TYPE, который был определен ранее. Выходными данными функции являются записи таблицы покупок (purchases) типа PURCHASES_TYPE, который был определен ранее. Для получения входящих строк используется переменная типа REF_CURSOR, INPUTREC, потом выполняется преобразование, а результаты по каждой строке выводятся в канал. Функция завершается оператором RETURN без указания возвращаемого значения.

```
CREATE OR REPLACE FUNCTION transform (inputrecs
IN cur_pack.ref_cur_type)
RETURN purchases_table
PIPELINED
IS
   product_id              VARCHAR2(8);
   time_key         DATE;
   customer_id             VARCHAR2(10);
   purchase_date           DATE;
   purchase_time           NUMBER(4,0);
   purchase_price   NUMBER(6,2);
   shipping_charge NUMBER(5,2);
   today_special_offer     VARCHAR2(1);
BEGIN
LOOP
FETCH inputrece INTO product_id,
      time_key,customer_id,purchase_date,purchase_time,
      purchase_price,shipping_charge,today_special_offer;
      EXIT WHEN INPUTRECS%NOTFOUND;
      product_id := REPLACE(product_id, '-','');
      shipping_charge :=(shipping_charge+shipping_charge*.10);
      PIPE ROW(purchases_record( product_id,
        time_key,
```

```
        customer_id,
        purchase_date,
        purchase_time,
        purchase_price,
        shipping_charge,
        today_special_offer));
END LOOP;
CLOSE inputrecs;
RETURN;
END;
```

Мы отменили изменения, сделанные в ходе предыдущего преобразования и выполним преобразование заново, с использованием табличной функции. Вот данные перед преобразованием.

```
SQL> SELECT * FROM APR_ORDERS;

PRODUCT   TIME_KEY   CUSTOMER   PURCHASE   PURCH   PURCH   SHIP   TODAYS_
ID                   ID         Date       Time    Price   Charge Special
--------- ---------- ---------- ---------- ------- ------- ------ -------
SP1001    01-APR-02  AB123456   01-APR-02  24      28.01   5.45   Y
SP1001    01-APR-02  AB123457   01-APR-02  1024    28.01   5.45   Y
SP1061    01-APR-02  AB123456   01-APR-02  24      28.01   8.42   Y
SP1062    01-APR-02  AB123457   01-APR-02  1024    28.01   3.58   Y
```

Для вызова функции включите ее в оператор SELECT. Перед именем функции в условии FROM используется ключевое слово TABLE. Изменения в коде выделены жирным шрифтом.

```
SQL> SELECT * FROM
     TABLE(transform(CURSOR(SELECT * FROM apr_orders)));

PRODUCT   TIME_KEY   CUSTOMER   PURCHASE   PURCH   PURCH   SHIP   TODAYS_
ID                   ID         Date       Time    Price   Charge Special
--------- ---------- ---------- ---------- ------- ------- ------ -------
SP1001    01-APR-02  AB123456   01-APR-02  24      28.01   6      Y
SP1001    01-APR-02  AB123457   01-APR-02  1024    28.01   6      Y
SP1061    01-APR-02  AB123456   01-APR-02  24      28.01   9.26   Y
SP1062    01-APR-02  AB123457   01-APR-02  1024    28.01   3.94   Y
```

Если нам будет нужно сохранить эти данные, в следующем фрагменте кода показано, как создать таблицу для сохранения результатов табличной функции.

```
SQL> CREATE TABLE TEST AS
SELECT * FROM TABLE(transform(CURSOR(SELECT * FROM apr_orders)));

Table created.
```

5.5.5. Преобразование, разделяющее источник данных на несколько частей

Иногда при преобразованиях один источник данных разделяется на несколько частей, как показано на рис. 5.15. Новая возможность, появившаяся в Oracle 9i, многотабличный оператор INSERT, помогает проводить преобразования такого типа.

В следующем примере таблица APR_ORDERS будет разделена на две таблицы. Все заказы, пользующиеся сегодняшним специальным предложением, будут записаны в таблицу SPECIAL_PURCHASES. Все обычные продажи будут записываться в таблицу PURCHASES. Такую информацию можно в дальнейшем использовать для нацеливания рекламы. Создается новая таблица, SPECIAL_PURCHASES, с теми же столбцами, что и таблицы APR_ORDERS и PURCHASES.

В операторе вставки указывается условие, результат проверки которого определяет, в какую таблицу будут вставляться данные. В этом примере есть только одно условие WHEN, но вы можете использовать несколько условий WHEN, если нужно поверить несколько вариантов. Если ни одно из условий WHEN не возвращает значение true, выполняется условие ELSE.

При указании ключевого слова FIRST база Oracle прекращает проверку условий WHEN при истинности первого из условий. И наоборот, если указано ключевое слово ALL, для каждой строки проверяются все условия. Указывать ALL полезно, если в нескольких таблицах хранится одинаковая строка.

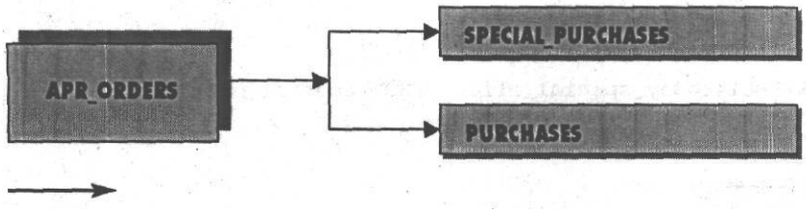


Рис. 5.15. *Многотабличная вставка*

```
SQL> INSERT FIRST WHEN today_special_offer = 'Y'
   THEN INTO special_purchases
   ELSE INTO purchases
   SELECT * FROM apr_orders;

15004 rows created.
```

В таблицу SPECIAL_PURCHASES были вставлены четыре строки, в которых today_special_orders = 'Y'. Оставшиеся строки были помещены в таблицу PURCHASES.

```
SQL> SELECT COUNT(*) FROM purchases;

COUNT(*)
---------
    15000

SQL> SELECT COUNT(*) FROM special_purchases;

COUNT(*)
---------
        4
```

Из приведенных запросов видно, что данные разделены на две таблицы. Покупки, для которых значение TODAY_SPECIAL_OFFER равно «N» сохранены в таблице PURCHASES, а те, у которых это значение равно «Y» — в таблице SPECIAL_PURCHASES.

```
SQL> SELECT DISTINCT(today_special_offer) FROM purchases;

Today_special_offer
--------------------
N

SQL> SELECT DISTINCT(today_special_offer) FROM special_purchases;

Today_special_offer
--------------------
Y
```

5.5.6. Перенос данных из промежуточной таблицы в таблицу фактов

Когда данные преобразованы, они готовы к переносу в таблицы хранилища. На рис. 5.14 данные о покупках, сделанных в апреле, были помещены во временные промежуточные таблицы хранилища. Когда данные были очищены и преобразованы, их можно переносить (как показано на Рисунке 5.16) в таблицу фактов.

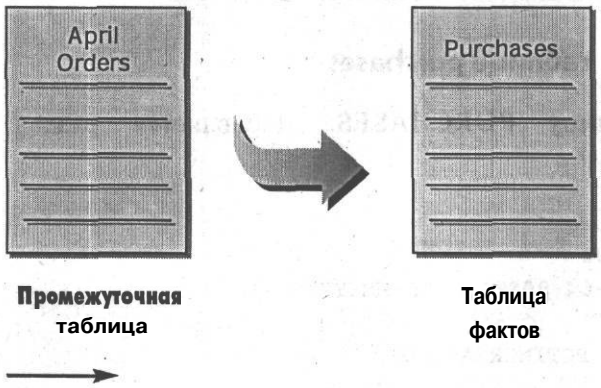


Рис. 5.16. *Данные переносятся из промежуточной таблицы в таблицу фактов*

Если таблица фактов секционирована, создаются новые табличные пространства для файлов данных и индексов, новый раздел добавляется к таблице фактов и данные переносятся в новый раздел. Эти этапы проиллюстрированы с помощью примера EASYDW.

Шаг 1. Создание нового табличного пространства для информации о покупках, сделанных в апреле, и соответствующего индекса

Поскольку таблица покупок (PURCHASES) секционирована по месяцам, для хранения апрельских покупок будет создано новое табличное пространство. Другое табличное пространство создается для индексов.

```
SQL> CREATE TABLESPACE purchases_apr2002
  DATAFILE 'c:\ora9ir2\oradata\orcl\purchasesapr2002.f'
  SIZE 5M
```

I1-I559

```
    REUSE AUTOEXTEND ON
    DEFAULT STORAGE
     (INITIAL 64K NEXT 64K PCTINCREASE 0 MAXEXTENTS UNLIMITED)

SQL> CREATE TABLESPACE purchases_apr2002_idx
    DATAFILE 'C:\ora9ir2\oradata\orcl\PURCHASESAPR2002_IDX.f'
    SIZE 3M REUSE
    AUTOEXTEND ON
    DEFAULT STORAGE
     (INITIAL 16K NEXT 16K PCTINCREASE 0 MAXEXTENTS UNLIMITED)
```

Шаг 2. Добавление раздела к таблице purchases

Вносятся изменения в таблицу PURCHASES. Добавляется раздел purchases_apr2002.

```
SQL>ALTER TABLE easydw.purchases
ADD PARTITION purchases_apr2002
VALUES LESS THAN (TO_DATE('30-04-2002', 'DD-MM-YYYY'))
PCTFREE 0 PCTUSED 99
STORAGE (INITIAL 64K NEXT 164 PCTINCREASE 0)
TABLESPACE purchases_apr2002;
```

На рис. 5.17 показан новый раздел, добавленный к таблице PURCHASES, и соответствующий индекс PURCHASE_PRODUCT_INDEX.

Шаг 3. Перенос таблицы в новый раздел

Существуют разнообразные способы переноса данных из одной таблицы в другую в пределах одной базы данных:

- обмен разделами (если таблица фактов секционирована);
- вставка в режиме прямого пути;
- CREATE TABLE AS SELECT.

Перенос *данных* с *помощью обмена разделами*

Условие ALTER TABLE EXCHANGE PARTITION, как правило, является самым быстрым способом переноса данных из несекционированной таблицы в раздел секционированной таблицы. Это условие можно использовать для переноса как

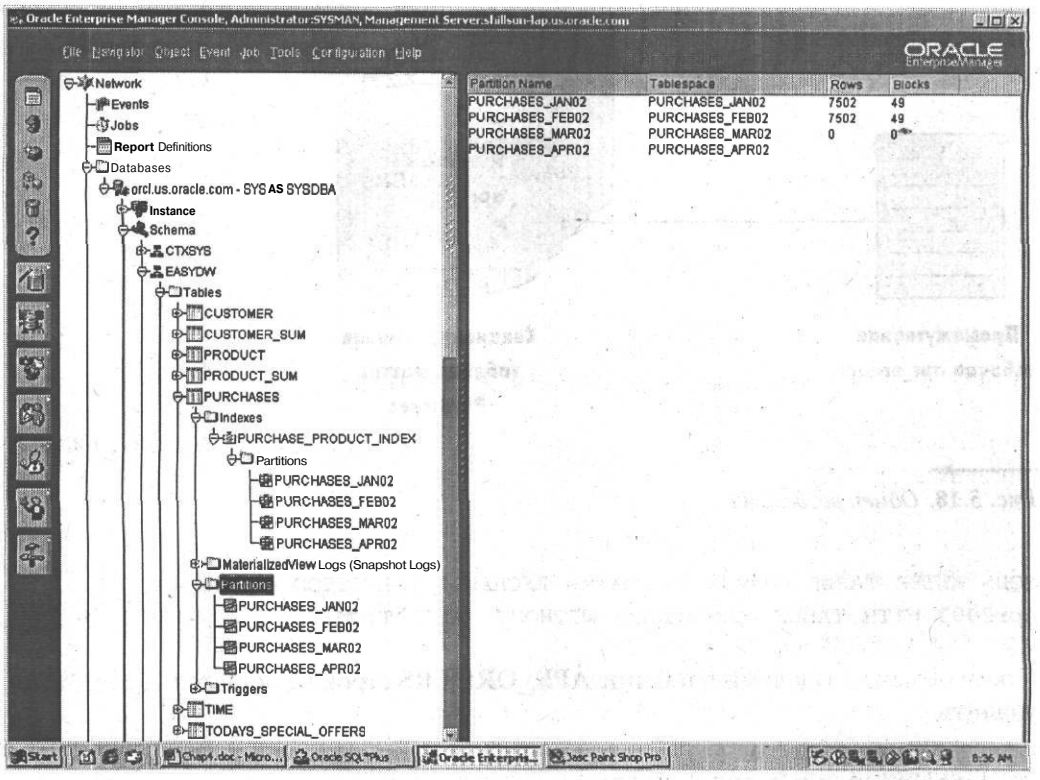


Рис. 5.17. *Просмотр разделов таблицы с помощью Oracle Enterprise Manager*

данных, так и локальных индексов из промежуточной таблицы в секционированную таблицу фактов. Причиной того, что это осуществляется так быстро, является то, что в действительности сами данные не переносятся, а лишь соответствующим образом обновляются метаданные.

Если очистка данных ранее не производилась, их можно проверить на соответствие критерию секционирования. Или же можно этого не делать, указав условие WITHOUT VALIDATION. На рис. 5.18 показан перенос данных из промежуточной таблицы APR_ORDERS в таблицу фактов PURCHASES с использованием обмена разделами.

В следующем примере показано перемещение данных из таблицы APR_ORDERS в раздел PURCHASES_APR2002 таблицы EASYDW.PURCHASES.

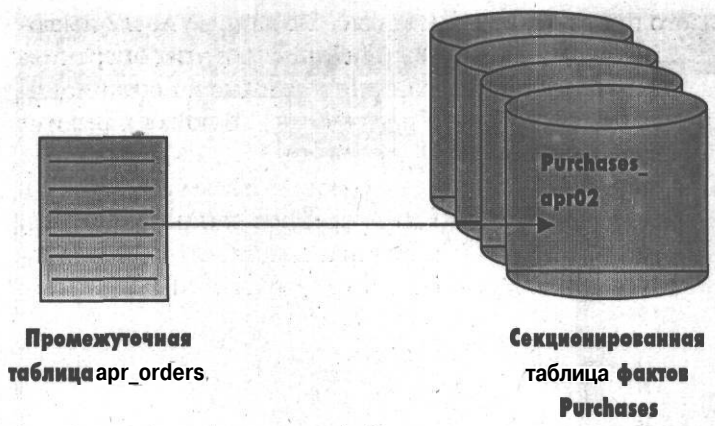


Рис. 5.18. *Обмен разделами*

```
SQL> ALTER TABLE easydw.purchases EXCHANGE PARTITION purchases_
apr2002 WITH TABLE apr_orders WITHOUT VALIDATION;
```

После обмена разделами в таблице APR_ORDERS строк не остается, и ее можно удалить.

```
SQL> SELECT * FROM apr_orders;

no rows selected

SQL> DROP TABLE apr_orders;

Table dropped.
```

Перенос данных между таблицами с помощью вставки в режиме прямого пути

Если таблица фактов не секционирована, вы можете добавлять в нее новые данные с помощью вставки в режиме прямого пути (direct path insert). Такой режим повышает производительность операций вставки путем форматирования и записи данных напрямую в файлы данных без использования буферного кэша. Эта функция похожа на режим прямого пути SQL*Loader.

Вставка в режиме прямого пути добавляет вставляемые данные после имеющихся в таблице. Свободные места в существующей таблице при этих операциях не используются. Данные можно вставлять в секционированные и несекционированные таблицы, как последовательно, так и параллельно. Вставка в режиме прямого пути обновляет индексы таблицы.

В базе данных EASYDW, поскольку таблица покупок purchases уже существует и секционирована по месяцам, мы будем для переноса данных в таблицу использовать вставку в режиме прямого пути. Вставка выполнится, когда вы включите инструкцию APPEND и используете для оператора INSERT синтаксис SELECT.

```
SQL>INSERT /*+ APPEND */ INTO easydw.purchases
      SELECT * FROM apr_orders
```

В нашем примере таблица фактов purchases уже существует, и новые данные добавляются в отдельный раздел.

Совет. Обязательно отключите все ссылочные ограничения перед выполнением загрузки в режиме прямого пути. Если этого не сделать, инструкция APPEND будет проигнорирована и без предупреждений будет использоваться условная вставка (conventional insert). Кроме всего прочего, эта вставка займет длительное время, если данных много.

Совет. Условная вставка используется, если применить INSERT... с условием VALUES, даже при использовании инструкции APPEND.

Создание новой таблицы с помощью CREATE TABLE AS SELECT

Если основная таблица фактов еще не существует, вы можете создать новую, отобрав одну или более таблицу с помощью оператора CREATE TABLE AS SELECT. Создание таблицы можно выполнять с использованием параллельных операций. Кроме того, вы можете отключить протокол повторного выполнения. В следующем примере, имя промежуточной таблицы — TEMP_PRODUCTS. После выполнения преобразований и очистки данных, путем копирования из TEMP_ PRODUCTS создается таблица PRODUCTS.

```
SQL> CREATE TABLE products PARALLEL NOLOGGING AS
     SELECT * FROM temp_products;
```

5.6. Операции, проводимые после загрузки

После загрузки данных, перед тем как они станут доступны для пользователей, может потребоваться проверка их качества, включение отключенных ограничений и триггеров, перестройка индексов, обновление статистики стоимостного (cost-based) оптимизатора и обновление материализованных представлений.

Шаг 1. Сбор статистики оптимизатора для таблиц

Запустите пакет dbms_stats и обновите статистическую информацию оптимизатора по всем таблицам, в которые вы загрузили значительный объем данных. Статистические сведения можно собрать для индекса, раздела индекса, таблицы, раздела или подраздела, а также для материализованного представления.

Поскольку мы добавили к таблице фактов PURCHASES новый раздел, в следующем примере мы соберем статистическую информацию по этому разделу, purchases_apr2002.

```
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS('easydw','purchases',
'purchases_apr2002',DBMS_STATS.AUTO_SAMPLE_SIZE);
```

Шаг 2. Проверка измерений

При использовании Управления сводными данными запустите для каждого из измерений процедуру DBMS_OLAP.VALIDATE_DIMENSION, чтобы проверить правильность иерархических атрибутов и взаимоотношений в соединениях. Также этот пакет можно использовать для определения того, имеют ли столбцы какого-либо уровня значение NULL. Вы можете проверять либо только что добавленные строки, либо все вообще. Все возникшие исключения записываются в таблицу MVIEW$_EXCEPTIONS, которая создается в пользовательской схеме.

В следующем примере проверяется правильность измерения PRODUCT (параметр 1) схемы EASYDW (параметр 2). Проверяться будут все строки (параметр 3 установлен в false), а также наличие нулей (NULL) (параметр 4 установлен в true).

```
SQL> EXECUTE dbms_olap.validate_
dimension('product','easydw',false,true);
```

```
SQL> select * from mview$_exceptions;
no rows selected
```

Шаг 3. Обновление материализованных представлений

После загрузки данных за апрель в таблицу покупок, материализованное представление PRODUCT_SUM становится устаревшим, как показано на рис. 5.19. Оно содержит информацию по продажам только с января по март, и его нужно обновить, включив данные за апрель.

На рис. 5.20 показаны опции, которые можно выбирать при обновлении материализованных представлений. В данном случае мы выбрали тип обновления force (форсированный), который, по возможности, будет выполнять инкрементное

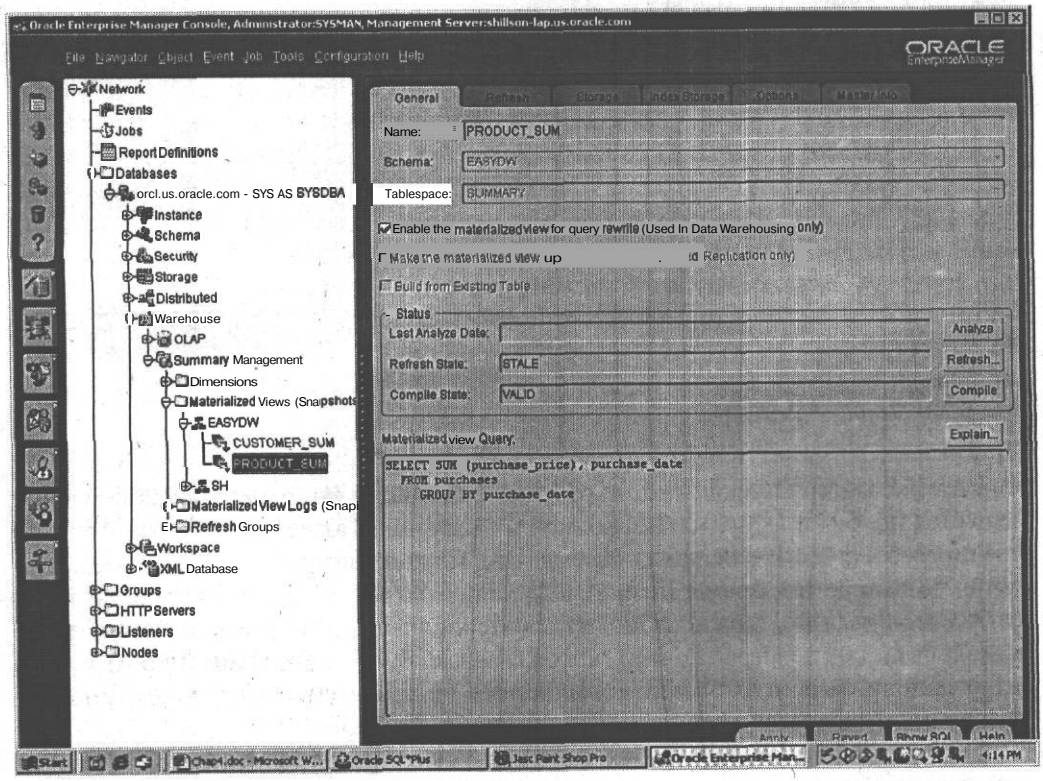


Рис. 5.19. *Использование Oracle Enterprise Manager для поиска устаревших материализованных представлений*

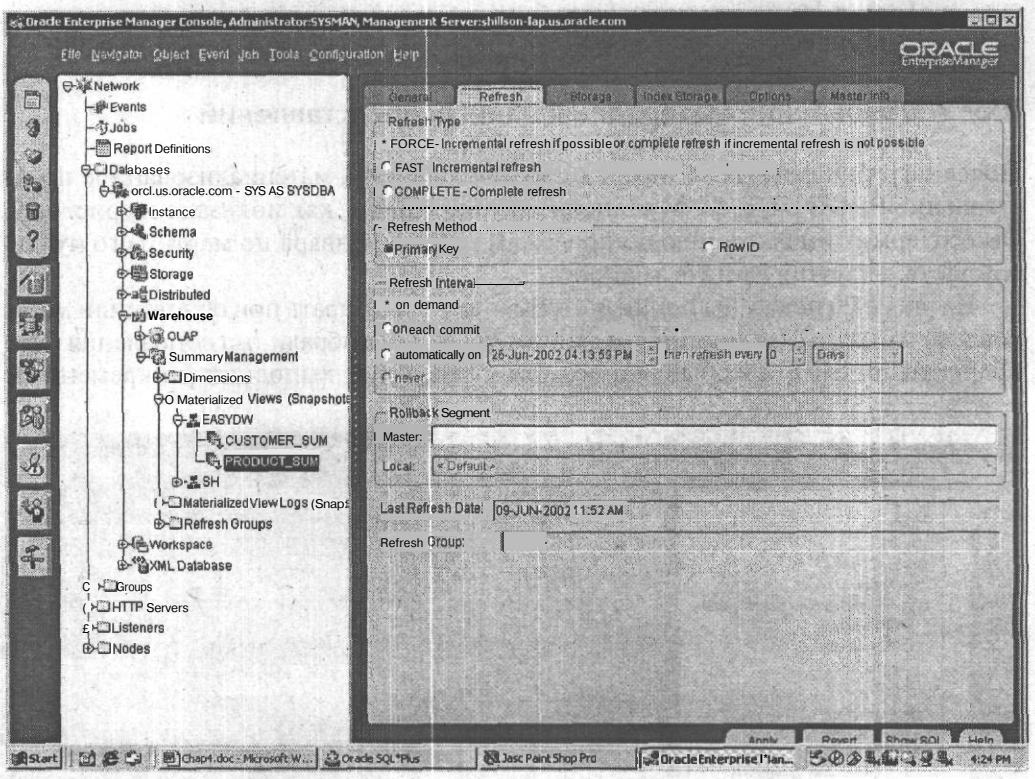


Рис. 5.20. *Опции обновления*

обновление материализованного представления. Если инкрементное обновление невозможно, будет выполняться полное обновление. Также мы выбрали обновление по требованию (on demand). Кроме того, можно настроить проведение обновления на конкретное время. Если данные обновляются в хранилище, а не загружаются массированно, можно обновлять материализованные представления после вышолнения каждой транзакции, которая делает их устаревшими. Чтобы можно было выполнять обновление по совершению, нужно создавать протокол материализованных представлений.

Нажав кнопку Explain (Объяснить), вы сможете определить возможности вашего материализованного представления, включая тип обновления, которое ваша система сможет выполнять. На рис. 5.21 показано, что данное материализованное представление будет обновляться полностью. В главе 4 описана система

слежения за изменениями разделов (Partition Change Tracking, PCT). Система PCT позволяет проводить быстрое обновление материализованных представлений для секционированных таблиц после операций над разделами. Чтобы систему PCT можно было использовать, нужно включить в материализованное представление либо ключ секционирования таблиц данных, либо специальный столбец, называемый маркером секционирования (partition marker). В данном примере использовать PCT нельзя, поскольку в списке выбранного нет ключа или маркера секционирования. Еще одна причина невозможности быстрого обновления состоит в том, что в определении материализованного представления пропущены некоторые столбцы. Можно видеть, что есть SUM(expr) без COUNT(expr). Быстрое обновление после вставки и других операторов DML невозможно, поскольку отсутствует протокол материализованного представления. Чтобы эти оптимизации можно было использовать, нужно изменить определение материализованного представления. Функция объяснения возможностей материализованного представления может оказать большую помощь для наилучшего формирования материализованных представлений. ,

Еще одним способом обновления материализованных представлений является использование процедуры DBMS_MVIEW.REFRESH, описанной ранее в главе 4.

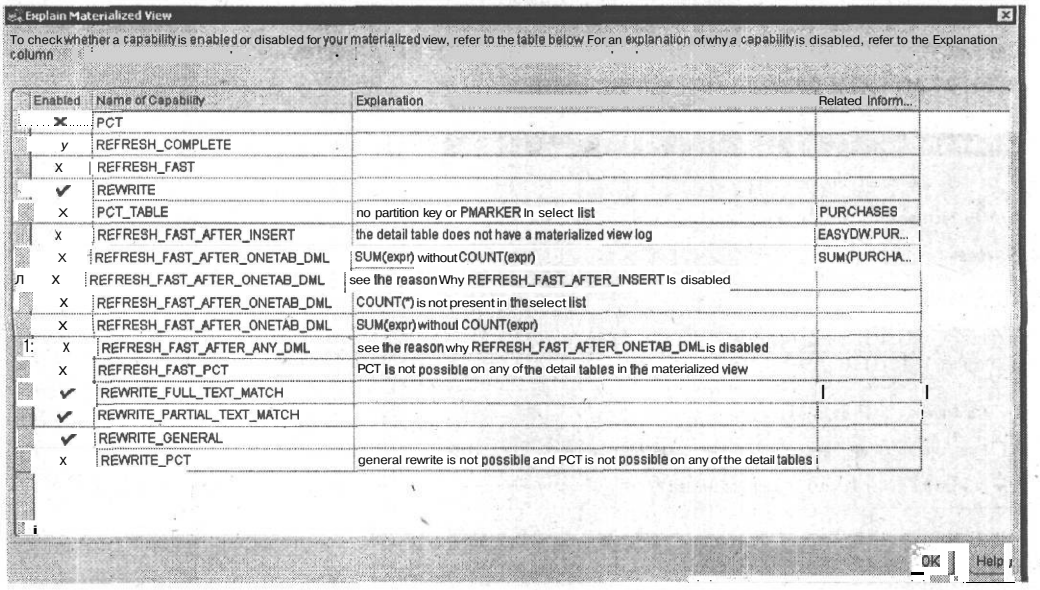


Рис. 5.21. *Объяснение возможностей материализованного представления*

Шаг 4. Сбор статистики работы оптимизатора для материализованных представлений

Теперь обновим статистику оптимизатора для всех материализованных представлений, которые могли существенно измениться в размерах. Если для обновления представлений вы использовали Oracle Enterprise Manager, у вас есть возможность проанализировать представление после его обновления, как это показано на рис. 5.22.

Чтобы собрать статистику без использования Oracle Enterprise Manager, применяйте пакет DBMS_STATS. В данном примере будут собраны статистические данные для материализованного представления PRODUCT_SUM в схеме EASYDW.

```
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS('easydw','product_sum');
```

Шаг 5. Резервное копирование базы данных, таблицы или раздела после завершения загрузки (если вы использовали опцию UNRECOVERABLE)

Поскольку восстановление среды для загружаемой таблицы отключено, вы не сможете выполнить восстановление при сбое. Обратитесь к главе 8, где обсуждаются методы резервного копирования.

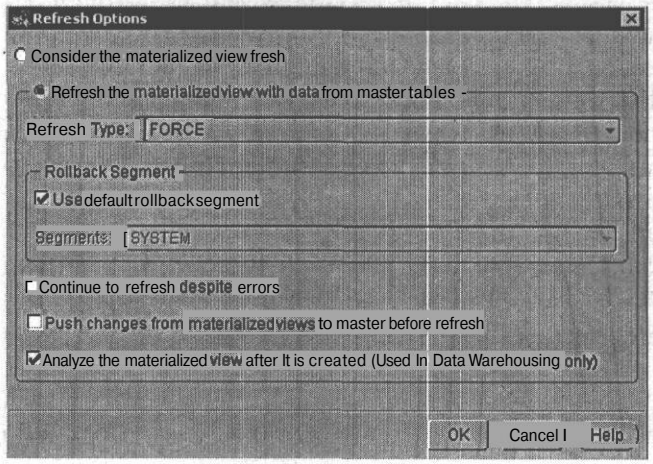


Рис. 5.22. *Анализ материализованного представления после его обновления*

Шаг 6. Публикация данных

Пошлите электронное письмо, уведомляющее пользователей, что данные накануне были загружены и теперь готовы к использованию.

5.7. Использование доступных инструментов для выполнения процесса ETL

На рынке имеются несколько инструментов, помогающих частично автоматизировать ETL-процесс, такие, как Oracle Warehouse Builder, Informatica, AbInitio, Ascential и Sagent. Эти средства предоставляют механизмы доступа к гетерогенным источникам данных, обычно к реляционным базам данных или последовательным файлам, и функции извлечения данных. Они управляют передачей данных через сеть, предоставляют возможности для проведения преобразований и очистки данных и загружают данные в ваше хранилище.

5.8. Заключение

В этой главе мы рассмотрели процесс извлечения, преобразования и загрузки данных (extraction, transformation, load, ETL). Мы обсудили различные методы ввыявления измененных строк в системе-источнике в процессе извлечения и познакомились с системой Oracle change data capture (Фиксирование изменений данных). Были рассмотрены несколько типов преобразований, наиболее типичных для хранилищ данных. Были показаны примеры проведения преобразований в ходе загрузки и использования промежуточных таблиц в базе Oracle. Были описаны методы загрузки данных в хранилище, включая SQL*Loader, внещние таблицы и транспортируемые табличные пространства. Перенос данных из промежуточной системы в таблицы хранилища можно осуществить при помощи обмена разделами, вставки в режиме прямого пути или оператора CREATE TABLE AS SELECT. В главе 6 мы рассмотрим Oracle Warehouse Builder, который применяет многие из описанных в этой главе функций и помогает проводить загрузку данных в хранилище.

6

Инструменты для работы с хранилищами Oracle

6.1. Какие существуют инструменты?

До сих пор мы строили наши базы данных и управляли ими с помощью интерфейса SQL*Plus. Однако существует множество доступных средств, помогающих пользователю, администратору базы или разработчику облегчить создание хранилища и работу с ним. Сейчас мы рассмотрим некоторые из таких средств, предоставляемых компанией Oracle. К ним относятся:

- Warehouse Builder;
- Discoverer;
- Reports (описывается в Главе 8);
- OLAP и Data Mining (описывается в Главе 9).

Oracle Warehouse Builder представляет собой средство, помогающее администраторам баз данных разрабатывать хранилища данных и управлять ими. Oracle Discoverer — это инструмент конечного пользователя для посылки запросов к хранилищу.

6.2. Oracle Warehouse Builder

Oracle Warehouse Builder (OWB) — это инструмент, созданный компанией Oracle для разработки и внедрения хранилищ масштаба предприятия (warehouse), тематических хранилищ (data mart) и приложений, ориентированных на бизнес-информацию (BI, Business Intelligence). Он является частью пакета Oracle 9i Developer Suite, куда входят продукты для разработки приложений, такие, как 9i Jdeveloper, 9i Designer и 9i Forms Developer. К области бизнес-информации относятся такие продукты, как Oracle Discoverer (описываемый ниже в этой главе), Oracle Reports (описываемый в главе 8) и Oracle Warehouse Builder.

Создание приложения любого типа — это не та задача, которая дается легко, и нужно пройти множество этапов, чтобы можно было облегчить построение хранилища или BI-приложения с помощью Oracle Warehouse Builder. Этот инструмент поможет:

- проектировать и создавать потоки данных между источниками и конечными базами;
- проектировать, создавать, управлять, обновлять и совершенствовать схему хранилища данных;
- управлять определениями баз-источников и обновлять их;
- импортировать определения баз-источников;
- разрабатывать и создавать OLAP и специализированную среду создания запросов;
- использовать функциональные возможности баз данных Oracle.

В OWB вы найдете репозиторий, который хранится в базе Oracle и где OWB держит все свои метаданные. Существует OWB-клиент, который используется при разработке и создании хранилища данных и приложения. Имеется генератор кода, который создает скрипты в соответствии с вашим проектом и применяет их к хранилищу, и есть среда создания отчетов, чтобы метаданные можно было просматривать в браузере.

Вы можете спросить, зачем использовать такой инструмент, как Oracle Warehouse Builder для разработки создания хранилища и связанных с ним приложений? Почему нельзя разработать их вручную? Конечно, сделать это можно, но Warehouse Builder имеет следующие преимущества.

- Снижается время разработки системы благодаря графическому интерфейсу (GUI).
- Проект хранится в одном месте, так что есть гарантия, что пользователи не будут работать с устаревшей моделью.
- Код, сгенерированный OWB, свободен от ошибок и работает с первого раза.

Теперь мы рассмотрим использование OWB при разработке большого или тематического хранилища.

6.2.1. Настройка Warehouse Builder

Перед тем как можно будет разрабатывать хранилище с помощью Warehouse Builder, требуется произвести небольшие настройки, чтобы создать репозиторий, который Warehouse Builder будет использовать при работе по созданию

хранилища. Запуск Repository Assistant, окно которого приведено на рис. 6.1, покажет нам шаги, которые нужно выполнить для того, чтобы создать репозиторий Warehouse Builder. Хотя этих шагов довольно много, их выполнение не займет много времени, и скоро вы начнете работать с Warehouse Builder. Вы можете выбрать, в какой базе данных будет располагаться репозиторий OWB, а затем вы должны войти в систему от имени пользователя, имеющего привилегии SYSDBA. Для репозитория нужно указать имя пользователя и пароль, а также табличное пространство для хранения данных и то, какой язык вы хотите использовать. Нажмите кнопку Finish (Готово), и репозиторий будет создан. Теперь Oracle Warehouse Builder готов к использованию.

6.2.2. Клиент Oracle Warehouse Builder

Когда репозиторий Oracle Warehouse Builder создан, пора переходить к использованию этого инструмента в разработке нашего хранилища. Мы можем:

- определить логическое представление для схемы нашего хранилища;

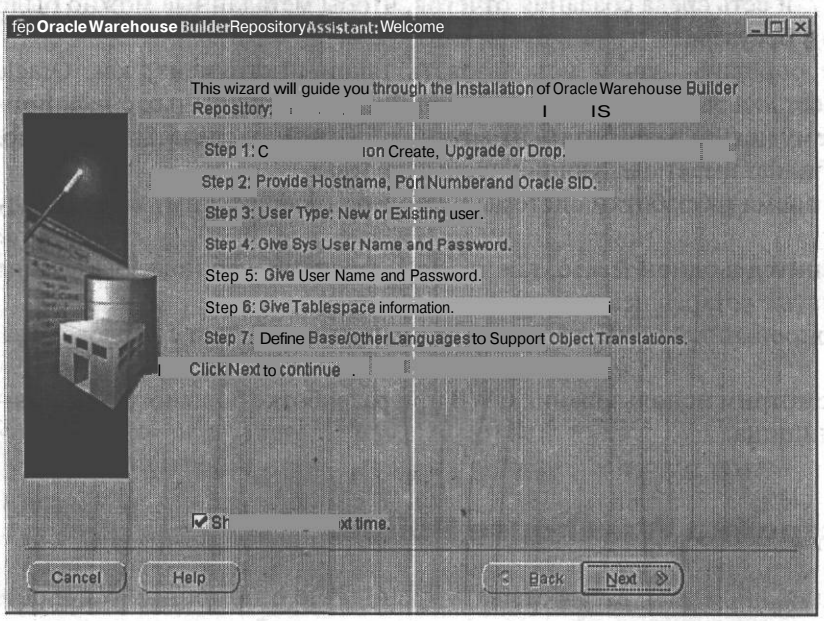


Рис. 6.1. *WarehouseBuilder — этапы настройки*

- определить источники данных и целевые базы;
- описать процессы ETL (extract, transform, load — извлечения, преобразования, загрузки);
- сгенерировать код SQL, создающий наше хранилище;
- составить распорядок задач по загрузке и обновлению с помощью Oracle Enterprise Manager.

В OWB хранилище определяется внутри проекта (project), а в проекте есть набор формальных определений, в которых указывается, как хранилище будет выглядеть с логической точки зрения. Эти определения называются модулями (module).

В те этапы, которые мы должны пройти при создании хранилища, входит определение базы-источника и базы-цели. Затем нам нужно будет описать, как данные будут переноситься из этих источников в хранилище. Для переноса данных нужно будет описать преобразования, которые будут переводить данные в тот формат, который мы используем в хранилище. Затем нам нужно будет сгенерировать проектную модель и код, реализующий наше хранилище и загружающий его данными.

6.2.3. Создание модулей

Одним из первых этапов является указание места, откуда будут браться данные для хранилища. Делается это путем определения модуля с помощью мастера New Module Wizard. Этот процесс включает четыре этапа. Один из первых показан на рис. 6.2, где мы даем модулю имя Order_Entry_system. Затем из раскрывающегося меню мы можем выбрать статус модуля. Поскольку мы определяем источник данных, мы выбираем статус Production.

Совет. Если вы определяете физический объект, пробелы в имени модуля не допускаются.

OWB предоставляет возможность работать с несколькими источниками данных, включая последовательные файлы и SAP. На рис. 6.3 мы выбрали базу данных Oracle.

Поскольку мы указали, что одним из наших источников данных является база Oracle, мы должны теперь уточнить, является ли этот источник собственно базой Oracle или мы хотим взять проектную модель из репозитория Oracle Designer. После этого все, что нам нужно, — это создать связь (dblink) с базой с помощью предоставленного окна, как показано на рис. 6.4, и наш источник данных готов.

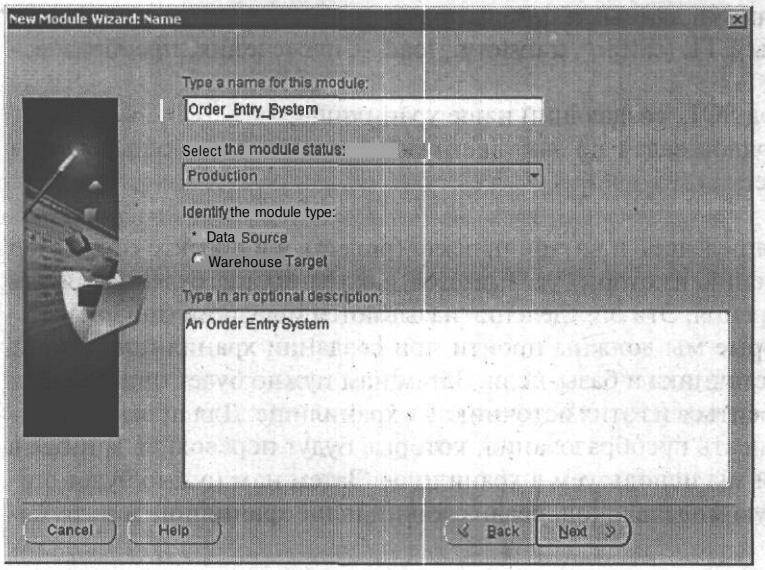


Рисунок 6.2 *Warehouse Builder — создание модуля*

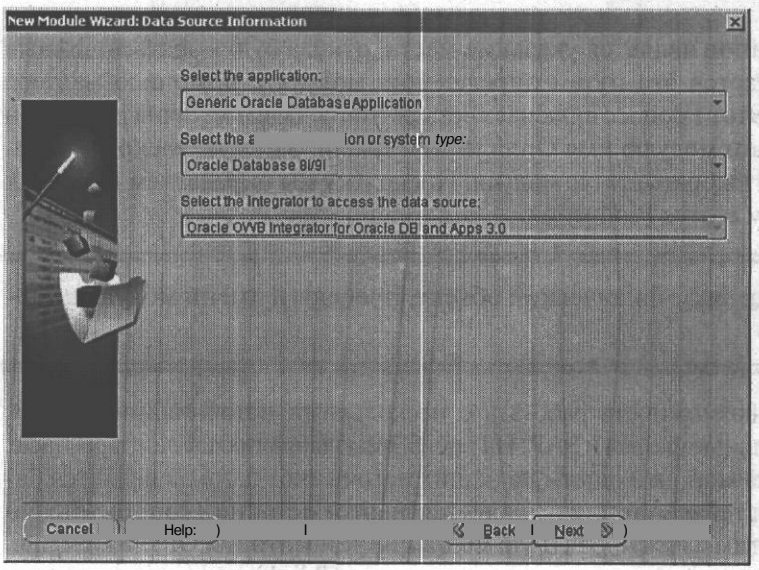


Рисунок 6.3 *Warehouse Builder — тип источника данных*

Затем этот процесс можно повторить, указав место, в котором будет располагаться наше хранилище. На рис. 6.5 можно видеть главную навигационную панель OWB Client, показывающую, что мы создали два модуля: EASYDW_DW, который представляет собой собственно хранилище, и ORDER_ENTRY_SYSTEM, являющийся одним из источников данных для хранилища. Еще одно полезное свойство — поскольку мы действительно создаем хранилище данных, OWB дает нам возможность указать, какой тип хранилища нам нужен, и в этот момент оно физически может не существовать.

После того как мы проделали всю эту работу, чрезвычайно важно сохранить все это, поскольку автоматического сохранения OWB не выполняет. Следовательно, если вас удовлетворяет состояние объектов, вы должны подтвердить изменения, либо щелкнув по иконке Commit (Подтверждение), либо выбрав Commit в пункте File (Файл) панели меню в верхней части окна.

6.2.4. Формирование таблиц в хранилище

В предыдущем разделе мы определяли источники данных и наше хранилище данных. Теперь пора определить таблицы, которые потребуются нам в хранилище.

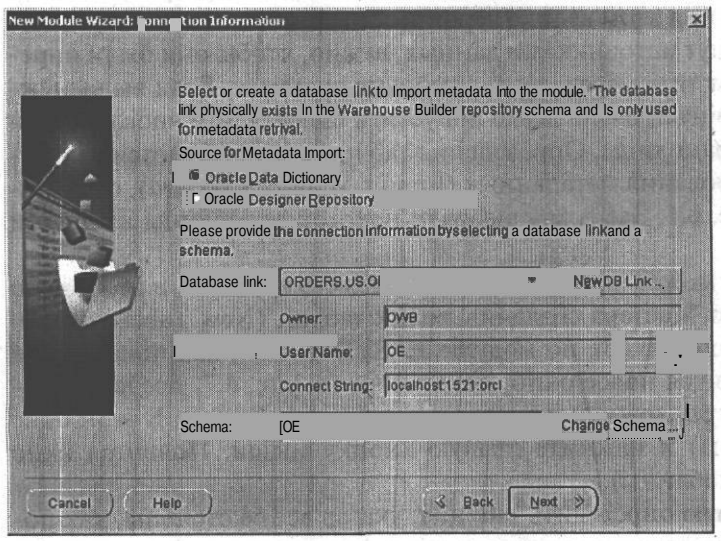


Рис. 6.4. *Warehouse Builder — определение источника данных*

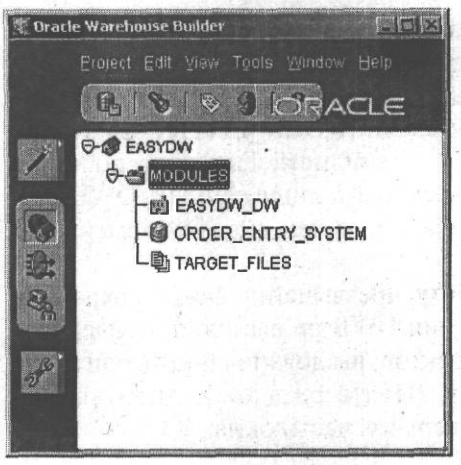


Рис. 6.5. *Warehouse Builder — созданы два модуля*

Это можно сделать либо вручную, либо путем импорта таблиц из другой базы данных.

Импорт определений таблиц показывает, как OWB может сэкономить время разработки, поскольку определение таблиц вручную — задача, требующая массу времени, и, кроме того, при этом велика вероятность ошибок. В ходе определения таблиц, которые будут источниками данных, важно, чтобы они были определены точно так же, как они определены в рабочих системах. Если вы вынуждены определять их вручную, очень легко допустить ошибку, что впоследствии задержит реализацию хранилища. Определения таблиц импортируются с помощью Мастера импортирования метаданных (Import Metadata Wizard), окно которого показано на рис. 6.6. Здесь мы выбрали некоторые таблицы из другой базы данных.

Некоторые таблицы может понадобиться определить вручную, и это можно осуществить с помощью Мастера создания новых таблиц (New Table Wizard). Просто сделайте двойной щелчок по модулю EASYDW_DW, как показано на рис. 6.5. При этом появится новое окно, показанное на рис. 6.7, представляющее собой редактор модуля хранилища (Warehouse Module Editor). В этом окне выберите Table (Таблица) и нажмите правую кнопку мыши. Появится окно Create Table (Создание таблицы).

Теперь можно вручную определить таблицу, указав все ее столбцы, как показано на рис. 6.8. работая с этим мастером, не забывайте, что вы также можете указывать ограничения, такие, как первичный ключ, внешний ключ и проверочные

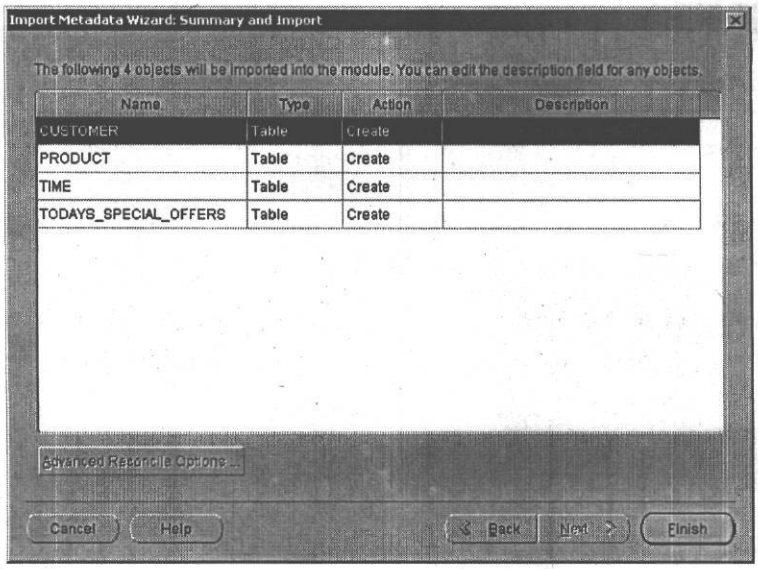


Рис. 6.6. *Warehouse Builder — импортирование таблицы*

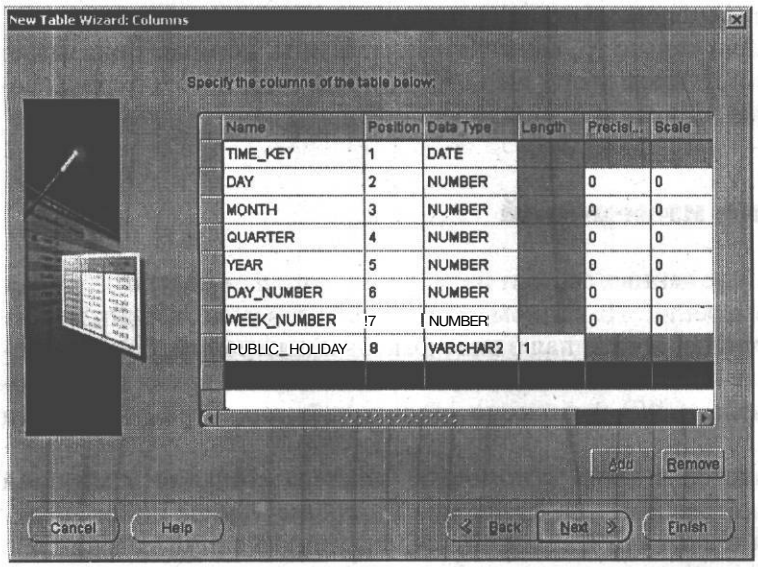


Рис. 6.7. *Warehouse Builder — определение таблицы вручную*

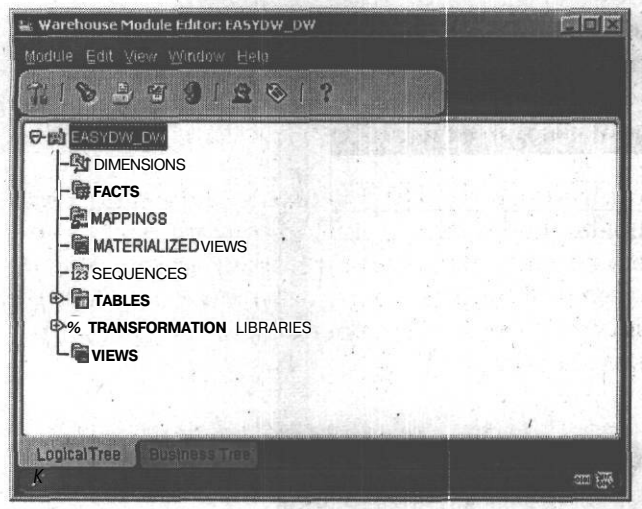


Рис. 6.8. *Warehouse Builder — редактор модулей Warehouse Module Editor*

ограничения (check constraints). Когда определение таблицы завершено, нажмите кнопку Finish (Готово) и завершите процесс определения таблицы.

Сейчас мы показали только то, как создавать таблицы, которые показывает Warehouse Module Editor. Кроме того, вы можете определить здесь материализованные представления, описанные в главе 4.

6.2.5. Создание измерений

Как правило, хранилище данных состоит из таблиц, к которым могут относиться таблицы фактов и измерений. На данном этапе разработки у нас, вероятно, есть только модель хранилища, описанная в виде фактов и измерений. OWB позволяет определять такие логические объекты, и когда наступит время генерировать проектную модель, OWB физически создаст таблицы, отражающие эти факты и измерения.

Измерения создаются в OWB с помощью Мастера создания измерений (Dimension Wizard), который можно вызвать из Warehouse Module Editor, щелкнув правой кнопкой по пункту Dimensions (измерения), а затем выбрав либо пункт Create Dimension (создать измерение), либо пункт Create Time Dimension (Создать измерение-время).

Этот процесс очень похож на тот, который мы уже видели в главе 4 при создании измерений. То есть вам нужно дать измерению имя, определить все уровни и их атрибуты, а затем описать иерархию. На рис. 6.9 мы видим одно из окон мастера создания измерений, в котором мы определяем иерархию измерения customer (покупатели).

Если вы хотите создать измерение-время, OWB автоматически предложит соответствующие уровни, атрибуты и иерархии, и вы сможете выбрать то, что нужно для вашей системы. Это означает, что измерение-время можно создать очень легко и быстро. Кроме того, OWB предоставляет функцию на PL/SQL, которая вставляет данные в измерение-время. Ее также можно сконфигурировать для генерации произвольного набора дат.

6.2.6. Создание таблиц фактов

Когда созданы измерения, следующим этапом нашего процесса разработки будет определение одной или более таблиц фактов. В OWB таблица фактов —

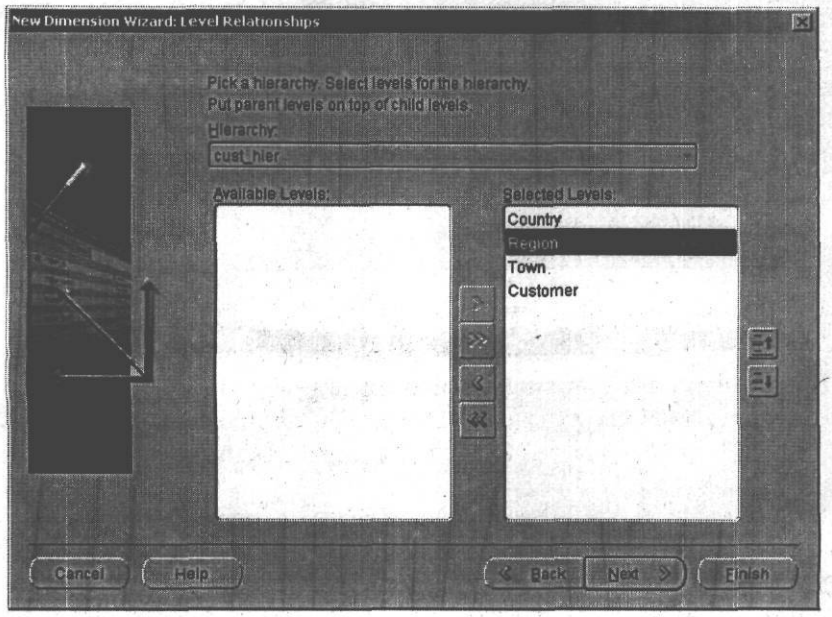


Рис. 6.9. *Warehouse Builder — иерархия измерения*

это логический объект в том смысле, что только когда генерируется физическая проектная модель, таблица фактов становится реальной физической таблицей. Создается она щелчком правой кнопки мыши по пункту Fact (Таблица фактов) в Warehouse Module Editor и затем выбором пункта Create Fact (Создать таблицу фактов). Появится мастер создания новой таблицы фактов (New Fact Wizard). Одно из окон мастера показано на рис. 6.10. В нем мы определяем внешние ключи нашей таблицы фактов. В данном примере OWB автоматически предлагает внешний ключ для каждого ранее определенного измерения.

Мастер позволяет добавлять единицы измерения. После нажатия кнопки Finish (Готово) наша таблица фактов создается. Теперь измерения и факты можно увидеть в редакторе таблиц фактов Warehouse Fact Table Editor (рис. 6.11).

Совет. Почаще проверяйте проект и отдельные компоненты.

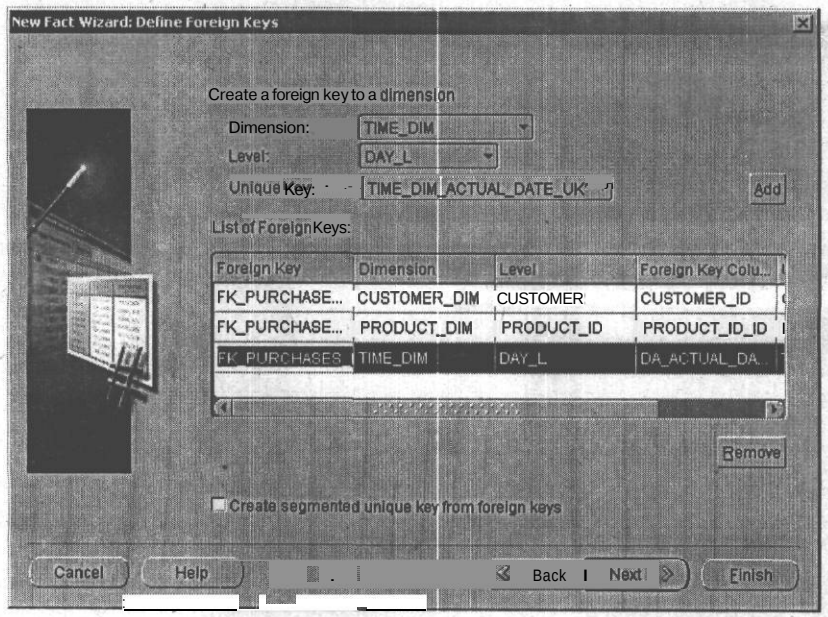


Рис. 6.10. *Warehouse Builder — создание таблицы фактов*

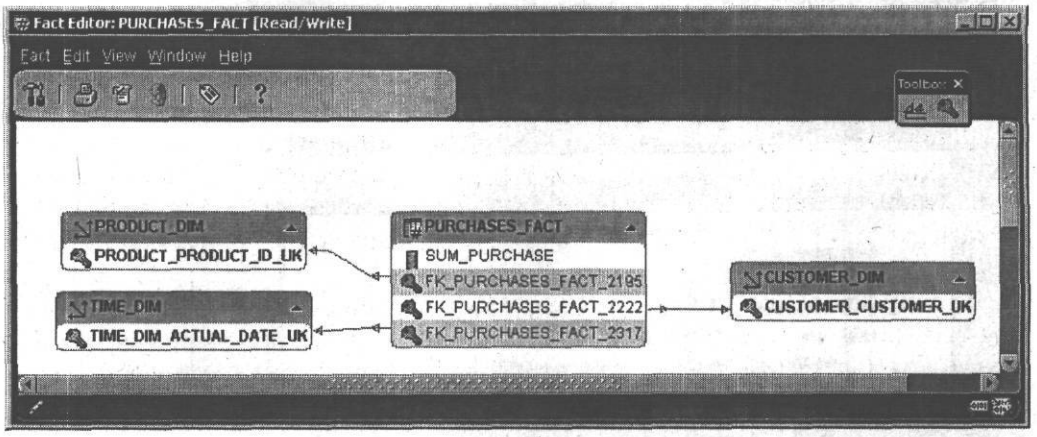


Рис. 6.11. *Warehouse Builder — Редактор таблиц фактов*

На любом этапе формирования хранилища можно проверить части модели, запустив функцию Validate (Проверить), описанную в разделе 6.2.9. Любые выявленные проблемы можно исправлять немедленно, до перехода к следующей части.

6.2.7. Преобразования

Проектирование хранилища — это больше, чем просто определение нескольких таблиц. Мы используем OWB, поскольку хотим иметь возможность легко указать, каким образом данные будут переноситься из базы-источника в наше хранилище. В главе 5 мы видели, что данные при перемещении из одной базы в другую, часто требуется преобразовывать. В OWB мы можем определять эти преобразования с помощью:

- процедур;
- пакетных процедур;
- функций PL/SQL;
- пакетныхфункций.

И опять, у нас не должно возникнуть необходимости писать ничего нового, поскольку с OWB поставляется много средств для преобразований, которые должны удовлетворить ваши требования. На рис. 6.12 мы можем видеть некоторые из этих поставляемых функций. В OWB эти функции распределены по

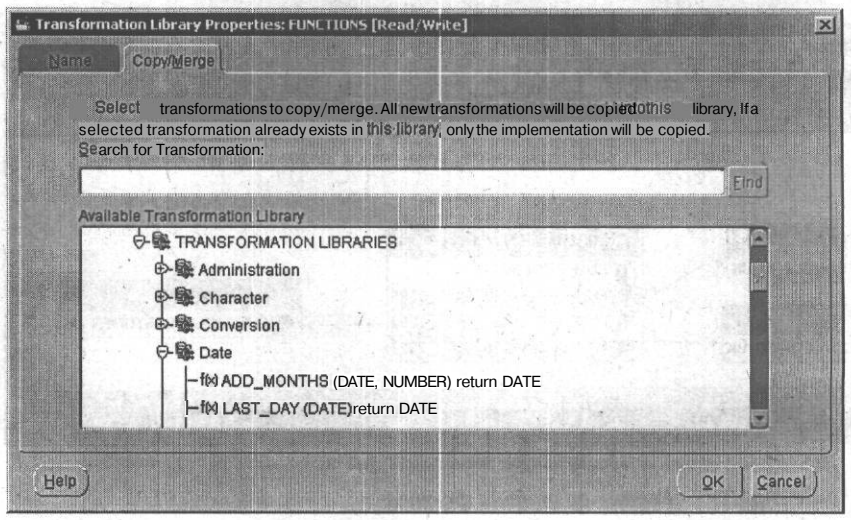


Рисунок 6.12. *Поставляемые функции для проведения преобразований*

группам, и здесь мы можем видеть некоторые функции для работы с датами, такие, как ADD_MONTH или LAST_DAY.

В том маловероятном случае, если вы не найдете ничего подходящего в предоставленных библиотеках, вы можете легко создать в Oracle Warehouse Builder свои собственные функции и процедуры. Однако мы надеемся, что такой необходимости не возникнет. Теперь, когда мы знаем о наличии этих функций в OWB, мы можем приступить к определению того, как данные будут переноситься из баз-источников в наше хранилище.

6.2.8. Определение соответствий между базами-источниками и базами-целями

Реальные возможности Oracle Warehouse Builder становятся очевидны, когда мы можем видеть то, как можно использовать его при определении способа переноса данных из баз-источников, например из OLTP-системы, в хранилище. В главе 5 мы узнали о различных методах, которые можно применять при проведении загрузки и преобразований. Теперь, в Oracle Warehouse Builder, используя графический интерфейс, мы можем графически изобразить эти процессы, определяя соответствия (mapping). Когда хранилище будет окончательно генерироваться,

OWB даже создаст требуемые процедуры для преобразования и загрузки данных. Сейчас мы рассмотрим лишь несколько из многих имеющихся типов соответствий.

Соответствие между источником и целью

Соответствие определяется в редакторе модулей, Warehouse Module Editor. Нужно щелкнуть правой кнопкой мыши по пункту Mappings (Соответствия), выбрать пункт Creating Mappings (Создание соответствий), и появится Редактор соответствий (Mapping Editor), показанный на рис. 6.13. Первое соответствие, которое мы определим, — это извлечение информации из нашей OLTP-системы ORDERS и помещение ее в хранилище данных. Также OWB может брать информацию из последовательных файлов или SAP.

Когда появится пустое окно Mapping Editor, щелкните по значку Mapping Table (Соответствие для таблицы) и перетащите его в редактор. Редактор спросит, где находится ваша таблица, и мы выберем таблицу ORDER_LINES из базы-источника ORDER_ENTRY_SYSTEM, которую мы определили ранее. Затем щелкаем по значку Mapping Fact (Соответствие для таблицы фактов) и выбираем PURCHASES_FACT. Теперь в нашем редакторе две таблицы — ORDER_ITEMS и PURCHASES_FACT.

Теперь мы должны указать, какие элементы каждой таблицы нужно перенести. Во-первых, мы перенесем элемент PRODUCT_ID из таблицы ORDERJTEMS в таблицу PURCHASES_FACT. Это можно сделать, протянув линию от пункта PRODUCT_ID в таблице ORDER_ITEM к пункту PRODUCT_ID в таблице PURCHASES_FACT. На рисунке 6.13 можно видеть линию, которую OWB нарисовал между этими двумя элементами.

Вычисление значения

Следующий столбец, который нужно определить в таблице PURCHASES_FACT — это стоимость заказа, а его можно получить, только рассчитав его по двум имеющимся столбцам таблицы ORDER_ITEMS. Для вычисления этого значения нужно создать промежуточную ячейку выражения, как показано на рис. 6.13.

Щелкните по значку Expression (Выражение) и перетащите его в Mapping Editor. Появится пустое окно с пустыми группами входящих и исходящих значений. Выберите два столбца из таблицы ORDER_ITEMS — unit_price и quantity и перетащите их в группу входящих значений. Теперь с помощью правой кнопки мыши добавим новый атрибут в группу исходящих значений, который назовем Total_Price. Теперь выберем свойства этого элемента и щелкнем по ячейке

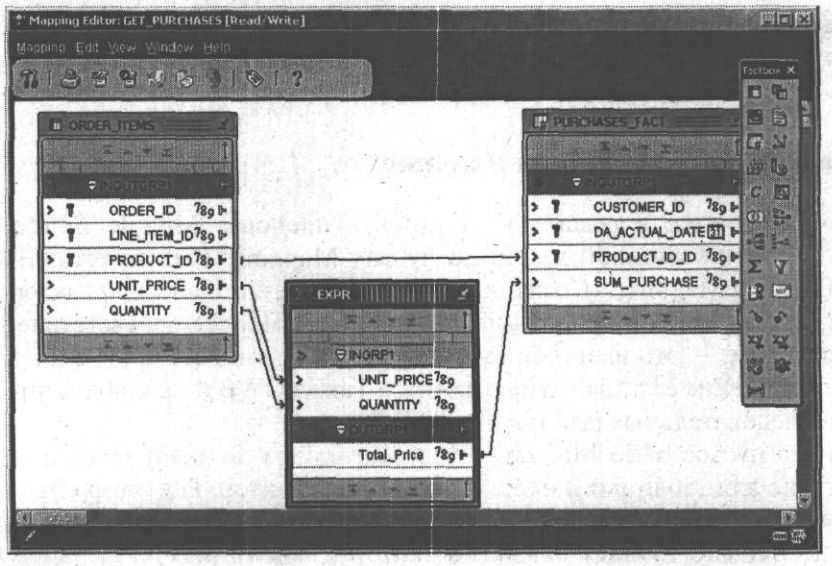


Рис. 6.13. *Редактор соответствий*

выражения (Expression) в Expression Builder. Здесь можно указать, как вычисляется данный атрибут. В данном случае два числа умножаются друг на друга. Теперь протяните линию от Total_Price к столбцу SUM_Purchase в таблице PURCHASES_FACT, и установление соответствия будет завершено.

Совет. Вероятно, стоит время от времени проверять, не были ли созданы неверные соответствия и преобразования.

Соединение таблиц и получение данных

Еще одной обычной задачей, которую приходится решать разработчику хранилища, является выполнение соединения данных двух и более таблиц и извлечение информации, которая используется в качестве входных данных для другой таблицы. На рис. 6.14 мы видим соединение, созданное между таблицами ORDERS и ORDER_ITEM, которое позволяет сохранять в хранилище запись, содержащую сведения об общей стоимости заказа. Заметьте, что созданное выше выражение (рис. 6.13) теперь используется как входящая информация для соединения.

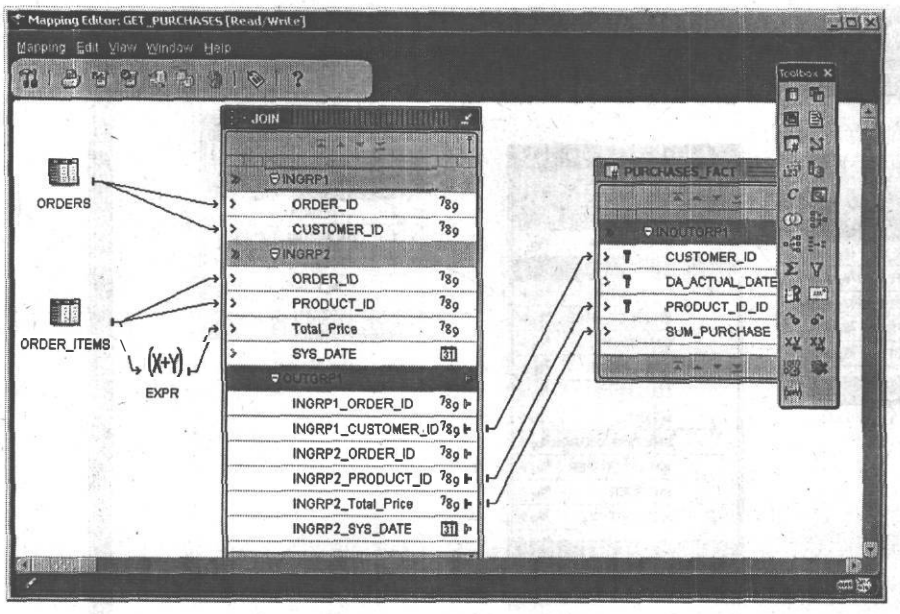


Рис. 6.14. *Соединения в Mapping Editor*

Совет. Когда вы закончите разрабатывать часть проекта, например выражение для вычисления значения объема продаж, можно свернуть окно, чтобы освободить рабочее место.

Поиск ключей

Ранее в этой книге мы описывали суррогатные ключи, служащие для преобразования ключей баз-источников в ключи, используемые в хранилище. В OWB мы можем точно указать то, как будет протекать такое преобразование, с помощью функции поиска ключей (key lookup).

В главе 5 мы описывали процесс конвертирования кода продукта, который использовался в OLTP-системе, в суррогатный ключ хранилища. На рис. 6.15 этот процесс реализуется в OWB, путем указания столбца PRODUCT_ID в таблице ORDER_ITEM в качестве входной информации для функции поиска ключей.

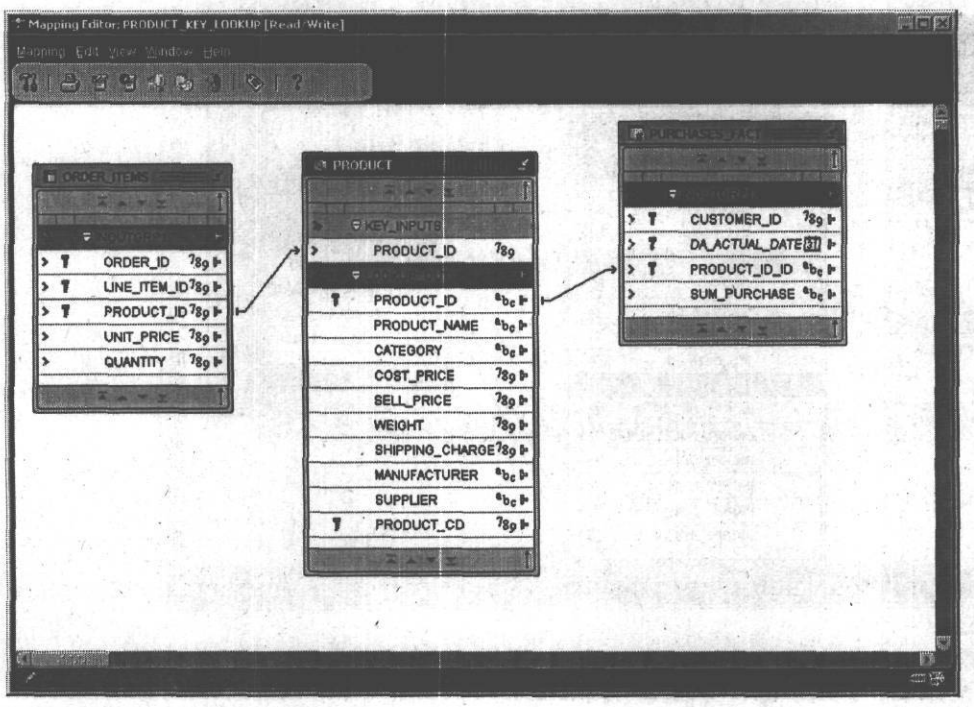


Рис. 6.15. *Поиск ключа продукта*

Хотя на рисунке этого не видно, было определено соответствие между product_id и product_cd. Результатом будет столбец product_id, посылаемый в таблицу PURCHASES_FACT.

Фильтрование данных

Когда данные извлекаются из таких источников, как OLTP-система, могут встретиться ситуации, когда не нужно пересылать в хранилище все данные, В OWB это не проблема, поскольку он позволяет фильтровать поступающие данные различными способами.

На рис. 6.16 мы осуществляем фильтрацию элементов из таблицы ORDER_ITEMS в соответствии с указанным нами критерием. Например, мы можем сказать, что нас интересуют только те данные, где общая стоимость заказа превышает некоторое значение. Кроме того, после фильтрации мы применяем

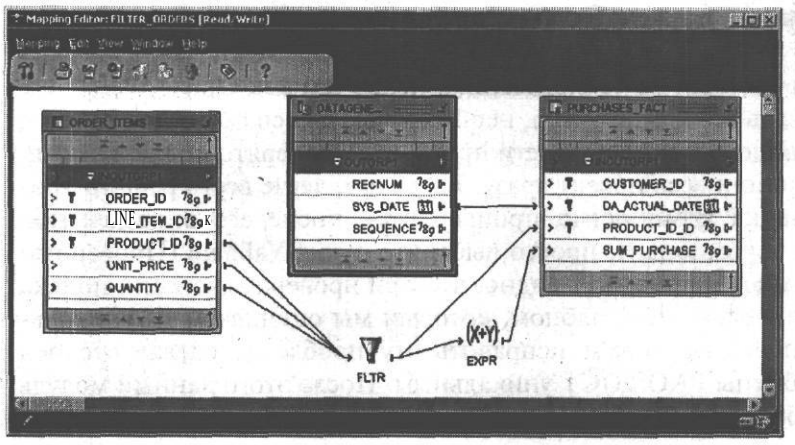


Рис. 6.16. *Фильтрация исходных данных*

к данным, прошедшим через фильтр, выражение, созданное в примере, показанном на рис. 6.13.

Применение генераторов данных

Бывают ситуации, когда нужно не извлекать данные из источника, а генерировать их автоматически. В OWB такая возможность существует, и можно автоматически сгенерировать:

- номер записи;
- номер последовательности;
- системную дату.

На рис. 6.16 можно видеть, что для указания даты заказа используется системная дата. Сделать это можно, перетащив значок генератора данных (Data Generator) в Редактор соответствий, выбрав нужный элемент, и присоединив его к таблице фактов.

Пока что, в этой главе мы лишь слегка затронули типы преобразований и соответствий, имеющиеся в Oracle Warehouse Builder. Мы придерживались простых примеров, но в реальном хранилище такие примеры можно объединять. Мы начали демонстрировать это на рис. 6.14 и 6.16, в которых выражение использовалось в качестве входной информации для другой стадии процесса загрузки.

6.2.9. Проверка разработанной модели

В процессе разработки, или по ее окончанию, перед тем как Oracle Warehouse Builder сможет создать все компоненты, необходимые для создания, загрузки и управления хранилищем, нужно провести проверку. Проверять можно каждый компонент отдельно или весь проект сразу. Вероятно, легче всего решать проблемы, если проверка компонента производится после его определения.

Чтобы проверить компонент, просто выберите пункт Validate (Проверить) из списка опций меню. На рис. 6.17 видно, что при проверке функции поиска ключей возникла проблема со столбцом, который мы оцениваем на уникальность данных. Теперь мы можем исправить эту проблему, сделав столбец PRODUCT_CD таблицы PRODUCT уникальным. После этого данный модуль пройдет проверку без ошибок.

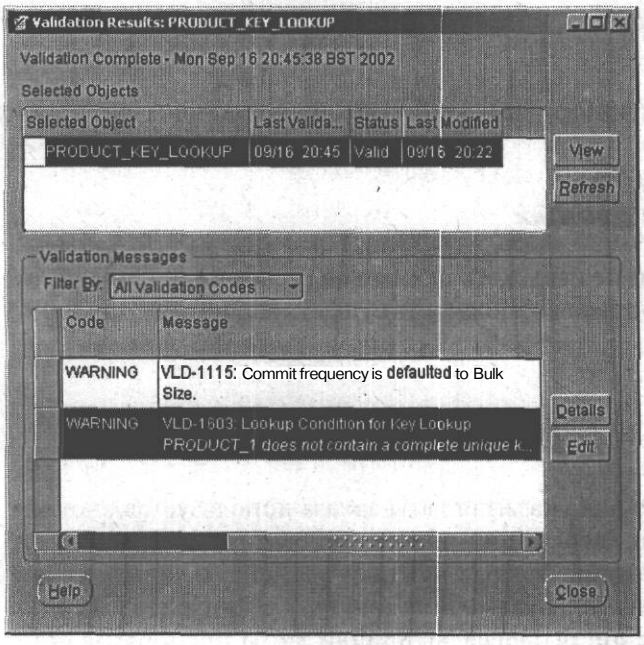


Рис. 6.17. *Проверка модели*

По ходу проверки Oracle Warehouse Builder отображает ее протекание (рис. 6.18). Это очень полезно, когда вы проверяете весь проект. Программа должна проделать

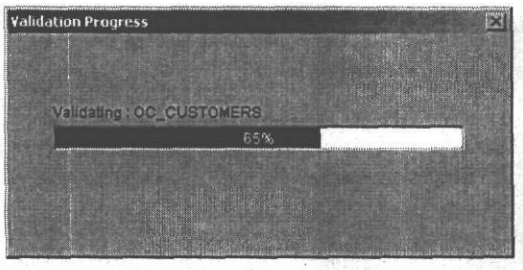


Рис. 6.18. *Выполнение процесса проверки*

массу работы, и весьма хорошо, что видно ее выполнение. Иначе вы можете подумать, что процесс завис.

6.2.10. Генерация и конфигурирование модели

Когда мы проверили нашу модель, мы готовы к тому, чтобы дать Oracle Warehouse Builder задание сгенерировать ее. Из Редактора модулей Warehouse Module Editor, окно которого показано на рис. 6.7, мы можем генерировать компоненты отдельно, выбрав пункт Module (Модуль), а затем — Generate (Сгенерировать). OWB спросит, желаем ли мы сгенерировать скрипты, которые могут создать:

- код SQL, создающий базу данных;
- код PL/SQL, исполняющийся внутри базы;
- процедуры для загрузки данных;
- файлы SQL*Loader для работы с последовательными файлами;
- TCL для Oracle Enterprise Manager.

На рис. 6.19 можно увидеть компоненты, генерируемые для нас OWB. Здесь есть объекты схемы, такие, как таблица PRODUCT. Нажав на кнопку View Code (Посмотреть код), мы можем увидеть код, определяющий таблицу. Также мы можем щелкнуть по кнопке Save as File (Сохранить в файл), чтобы можно было провести реализацию позже. Если вы решите выполнить ее сразу, нажмите на кнопку Deploy (Использовать). OWB спросит, в какой базе данных нужно создать компонент, и создаст его в этом месте.

Если мы перейдем к панели Mappings (Соответствия), мы увидим, что OWB создал пакет для загрузки данных, которые мы определили в примере на рис. 6.14. Щелкнув по кнопке View Code (Посмотреть код), мы можем увидеть код этой

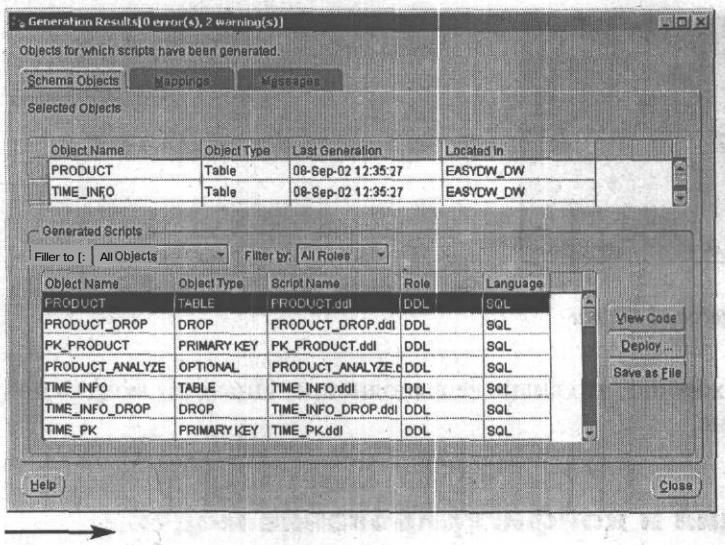


Рис. 6.19. *Реализация модели*

длинной процедуры, часть которого показана на рис. 6.20. Взгляните на ползунок и оцените, какой объем кода создал OWB. Показанный здесь фрагмент представляет собой часть оператора INSERT, но перед ним есть еще определения переменных и многое другое, что нам пришлось бы писать самим.

В данном примере OWB решил использовать оператор SQL INSERT, но поскольку OWB был разработан так, чтобы применять все возможности базы данных, он может использовать в генерируемом коде такие операторы, как MERGE, которые мы видели в главе 3.

Хотя мы и сгенерировали модель, скорее всего, в нее входят не все физические аспекты, такие, как сегментирование таблиц и необходимые индексы. Эти физические компоненты можно настроить в OWB, выбрав конфигурируемый модуль. Из пункта меню Edit (Редактирование) выберите пункт Configure (Конфигурировать). На рис. 6.21 показаны все свойства, которые вы можете конфигурировать. В данном примере для таблицы PURCHASES_FACT мы видим, что внешние ключи уже существуют, таблица будет сохраняться в табличном пространстве USERS и таблица еще не сегментирована.

В разделе Indexes (Индексы) видно наличие трех индексов на основе битовых карт. Их можно автоматически настроить с помощью OWB, как показано на рис. 6.22, нажав кнопку Add (Добавить), Generate (Сгенерировать) и выбрав рекомендованные индексы на основе битовых карт для столбцов внешних ключей.

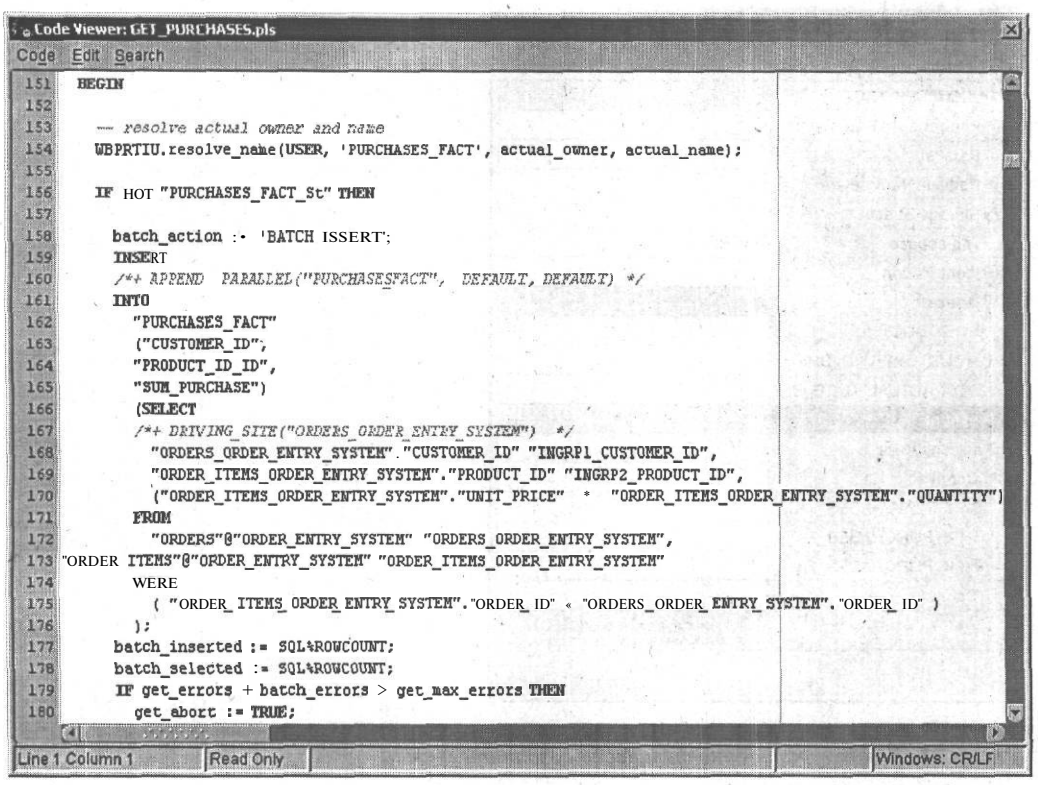


Рис. 6.20. *Код, сгенерированный Oracle Warehouse Builder*

Мы надеемся, вы оценили возможности OWB. Это очень мощный продукт, и мы смогли обрисовать лишь некоторые из его свойств. У использования таких средств, как Oracle Warehouse Builder, есть множество преимуществ. Вы можете сэкономить огромное количество времени, поскольку вам не придется писать код, реализующий хранилище и загружающий его данными. Код, который генерирует OWB, оптимизирован под базу данных Oracle и работает с первого раза. Имеется также возможность визуального представления хранилища, а использование имеющихся мастеров облегчает выполнение задач, связанных с его созданием. Кроме того, когда в среду вносятся изменения, вы можете визуально оценить их вклад. OWB позволяет легко включить изменения в среду. И наконец, если вы предпочитаете разрабатывать хранилище вручную, есть возможность быстро и точно выполнить некоторые однообразные задачи с помощью OWB.

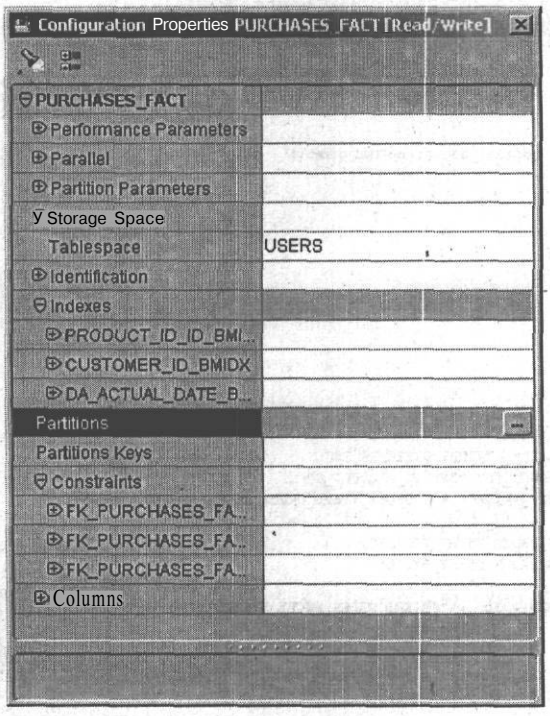


Рис. 6.21. *Конфигурирование физически созданной модели*

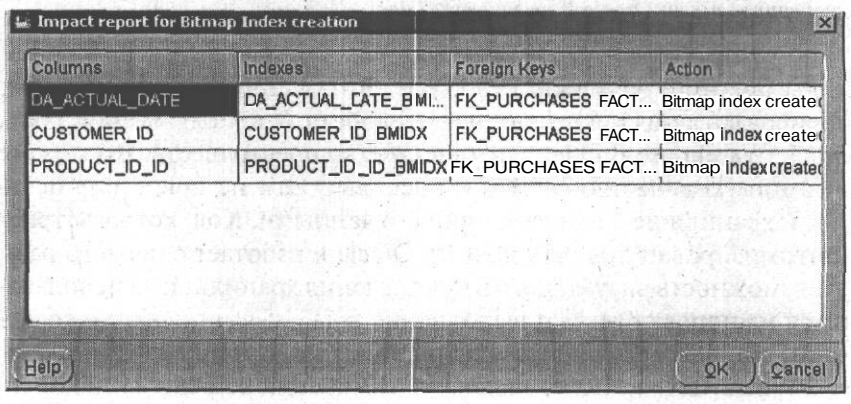


Рис. 6.22. *Рекомендованные индексы на основе битовых карт*

6.3. Oracle Discoverer

Oracle Discoverer — чрезвычайно популярный инструмент для создания запросов к хранилищу и генерации отчетов, поскольку он очень легок в использовании, интуитивно понятен и разработан для пользователей, плохо знакомых с компьютерами. Следовательно, для того чтобы плохо знакомые с компьютерами пользователи могли спокойно использовать Discoverer, группа администрирования должна провести некоторую настройку. Но как только вы это сделаете, вашим конечным пользователям работа с этим инструментом должна понравиться. У Oracle Discoverer есть четыре части:

- Administrator;
- Desktop;
- Plus;
- Viewer.

Administrator — это версия, которая используется администраторами хранилища для проведения настройки среды для обычных пользователей Discoverer. Обычные пользователи будут использовать либо версию Desktop, либо Pius, поскольку они создавались специально для тех людей, которые не знакомы с написанием компьютерных программ, а также для тех, кто не знаком с языком SQL и предпочитает работать с данными, используя обычные понятия бизнеса. Тем пользователям, которые опасаются, что они могут изменить данные в хранилище, — не бойтесь. Discoverer Viewer позволяет пользователю просматривать только заранее определенные отчеты. Привлекательность Discoverer Viewer в том, что пользователю для доступа к данным нужно иметь только персональный компьютер и браузер.

Oracle Discoverer интегрирован с Oracle Portal, о котором мы узнаем больше в главе 8. В этом методе доступ к данным осуществляется с помощью списка рабочих книг (workbooks) или портала рабочих листов и графиков (worksheet/graph portal).

Возможности применения Oracle Discoverer не ограничиваются базами Oracle, но мы покажем его использование именно на базах Oracle. К базам данных, сертифицированным для использования с Discoverer, относятся DB2 для NT и MVS, Microsoft SQL Server, Sybase и Terradata.

6.3.1. Почему Discoverer?

До того как мы узнаем, как настраивать Discoverer, давайте рассмотрим типы создаваемых им отчетов.

Создание запросов с помощью Discoverer Viewer

Discoverer — это мощное средство создания отчетов. В нем есть версия, называющаяся Discoverer Viewer, которая позволяет пользователю осуществлять доступ к заранее определенным отчетам и имеет возможность внесения в эти отчеты ограниченных модификаций. Преимущество такого подхода в том, что пользователю для доступа к данным в хранилище нужен лишь персональный компьютер и браузер. Администратор базы может быть уверен, что пользователь не сможет изменить данные.

Вы можете запустить Discoverer Viewer из своего браузера с помощью следующего URL:

```
http://easydw.com:7777/discoverer/viewer
```

Производится соединение с данной базой и выводится список рабочих книг, с которыми мы можем соединиться. Мы выбираем нашу книгу Easydw_workbook, и содержащийся в ней запрос немедленно выполняется. В примере на рис. 6.23 мы можем следить за процессом выполнения запроса. Когда обработка завершается, отображается отчет (рис. 6.24).

Поскольку мы определили отчет и как таблицу, и как диаграмму, отображаются обе версии. Заметьте, что мы можем в любой момент скрыть таблицу или график, щелкнув по пункту Hide Data (Скрыть данные) или Hide Chart (Скрыть график). Показанный здесь отчет Discoverer генерирует по умолчанию. Я уверен, что вы согласитесь с тем, что это очень приятный формат, хорошо представленный и легко понимаемый.

В рабочей книге может содержаться много отчетов, и добраться до любого из них можно, просто щелкнув мышкой по месту на панели слева, где стоит название отчета.

Одно из преимуществ использования Discoverer Viewer состоит в том, что пользователь не может изменить данные в хранилище или сгенерировать новый отчет. Однако он может внести в этот отчет небольшие изменения, например изменить способ сортировки или вывода таблицы, щелкнув мышью по пункту Presentation Option (Параметры вывода), как показано на рис. 6.25, и изменить внешний вид отчета.

Динамические отчеты с графиками

Одним из свойств, которые делают Discoverer приятным в использовании, является возможность сгруппировать несколько отчетов, которые можно легко

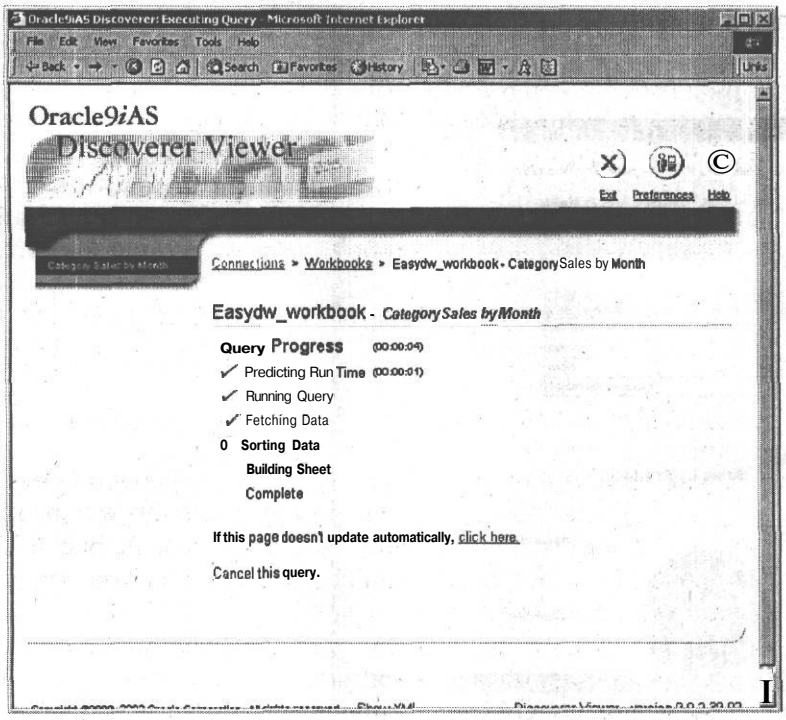
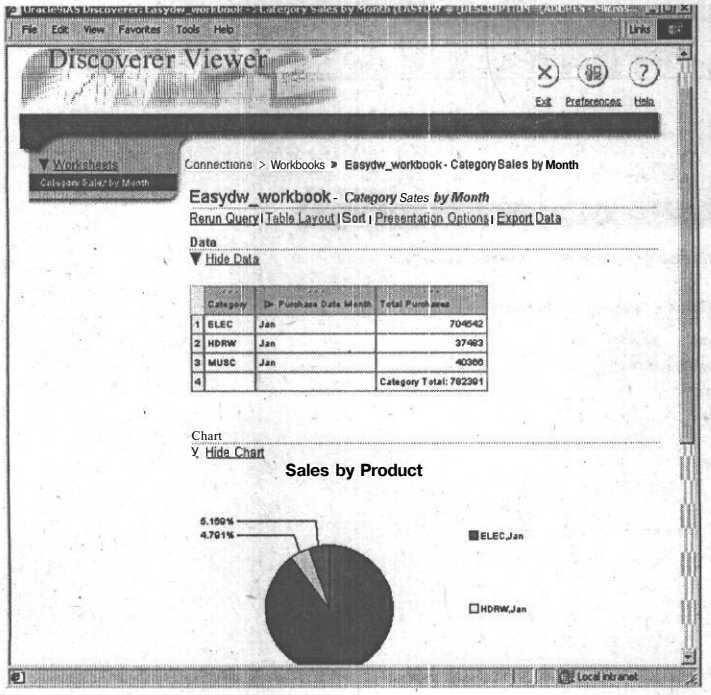


Рис. 6.23. *Discoverer Viewer — выполнение запроса*

получить и настроить под наши собственные требования. В приводимом примере у нас есть рабочая книга, названная Easydw_reports, и, если мы обратимся к рис. 6.26, в нижней части окна мы увидим три закладки для имеющихся отчетов:

- Total Sales by Customer (Общие объемы продаж по покупателям);
- Best Customers (Лучшие покупатели);
- Most Popular Products (Самые популярныепродукты).

На рис. 6.26 мы видим и таблицу (слева), и диаграмму (справа), сгенерированные по данным, содержащимся в хранилище. Когда этот отчет запускается, пользователю предлагается указать критерий того, что делает покупателя лучшим (например, покупки на сумму более 15000$), после чего данные отображаются. Мы видим, что на данный момент нашим лучшим покупателем является компания Harry Travel.



Рис, 6.24. *Discoverer Viewer — отчет по объемам продаж в категориях по месяцам*

Получение подробных сведений

Когда отчет выведен, люди, которые его читают,- могут увидеть, что он очень интересен, но им может потребоваться больше узнать о том, откуда взялись эти данные. Например, предположим, что пользователь просматривает отчет, показанный на рис. 6.27, и видит, что наиболее часто продаваемым товаром являются фотоаппараты. У него возникает вопрос: мы продаем одно и тоже число фотоаппаратов в течение всего года или это сезонное явление.

Тот, кто просматривает данный отчет, может сделать щелчок правой кнопкой мыши по товару, например фотоаппарату, и выбрать функцию Drill (Подробнее). Discoverer задаст вопрос о том, какие подробности вы хотите получить. На рис. 6.27 мы запросили подробные данные о продажах по месяцам и видим, что мы продаем почти одно и то же количество фотоаппаратов каждый месяц. При детализации сведений нужно получать только ту информацию, которая вам интересна.

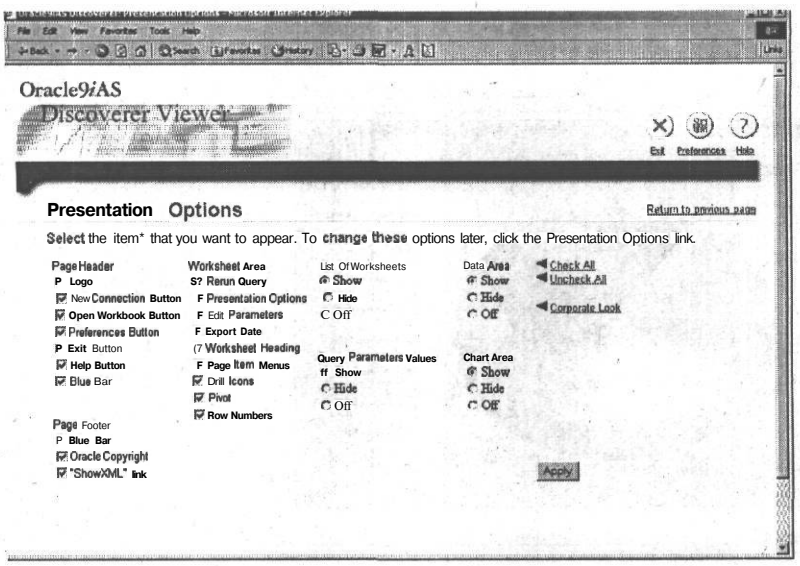


Рис. 6.25. *Discoverer Viewer — Параметры вывода (окно 1)*

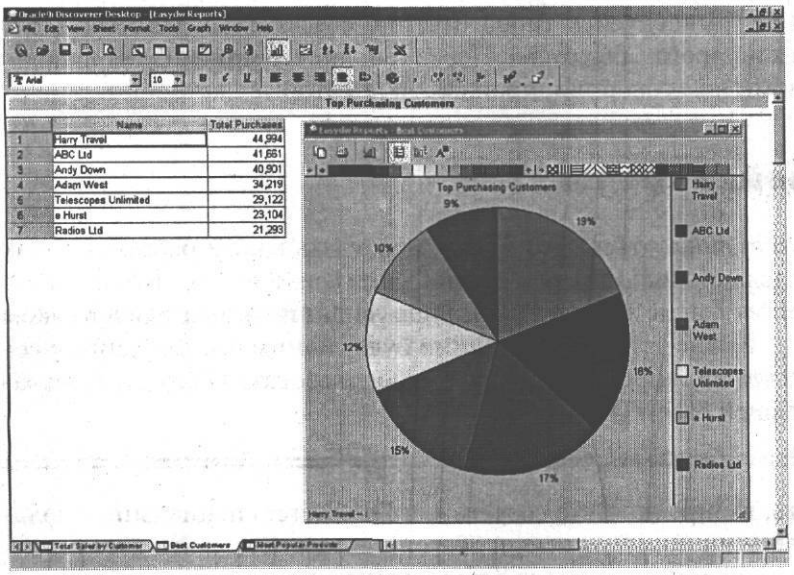


Рис. 6.26. *Discoverer Viewer — Параметры вывода (окно 2)*

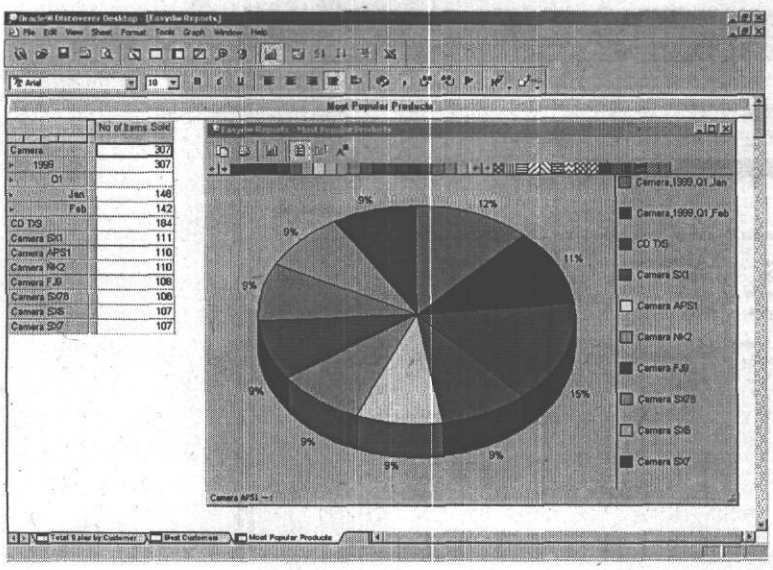


Рис. 6.27. *Discoverer Viewer — Параметры вывода (окно 3)*

Теперь, когда мы бегло коснулись типов отчетов, создаваемых Discoverer, мы должны вернуться к настройке Discoverer. Перед тем, как его можно будет использовать, нужно, как мы уже говорили, произвести некоторые настройки.

6.3.2. Настройка среды

Хотя поначалу может показаться, что нужно проделать много работы, но как только она будет сделана, пользователи оценят ваши усилия. Создаваемая среда в точности определяет, какие данные хранилища увидят пользователи и в каком виде они их увидят. Важно, что им и не нужно будет понимать, как данные соединяются при ответе на запрос. Чтобы начать использовать Discoverer, сначала соединитесь с вашей базой Oracle.

Совет. Может быть, вы предпочтете создать для Discoverer специального пользователя и соединяться с базой от имени этого пользователя, и это даст гарантию, что метаданные будут загружаться с использованием этого имени.

Уровень конечного пользователя

При первом запуске Discoverer Administrator вас попросят создать уровень конечного пользователя (end-user layer, EUL), заключающий в себе все метаданные, которые требуются для работы Discoverer. Это задание, следовательно, нужно выполнить только один раз. На рис. 6.28 показано окно, из которого вы можете управлять уровнем конечного пользователя, включая создание нового или удаление существующего.

После этого вам будет предложено ответить на вопрос о том, кому принадлежит данный EUL, а после того как вы нажмете на кнопку Finish (Готово), уровень будет создан. Затем придется подождать несколько секунд, пока метаданные будут загружаться в базу данных.

Настройка среды для Discoverer включает много этапов. Чтобы вам легче было их вспомнить, будет выведено окно, показанное на рис. 6.29, в котором все эти задачи перечислены. Щелкнув мышью по любому из пунктов этого списка, вы можете автоматически запустить данный компонент.

Область бизнес-задач

Когда определение уровня конечного пользователя завершено, нужно дать название области бизнес-задач (business area), в которой требуется определить всю информацию, необходимую для посылки запросов к хранилищу. Назначение области бизнес-задач — группирование информации по категориям, ориентированным на области бизнеса, знакомые конечным пользователям, такие, как

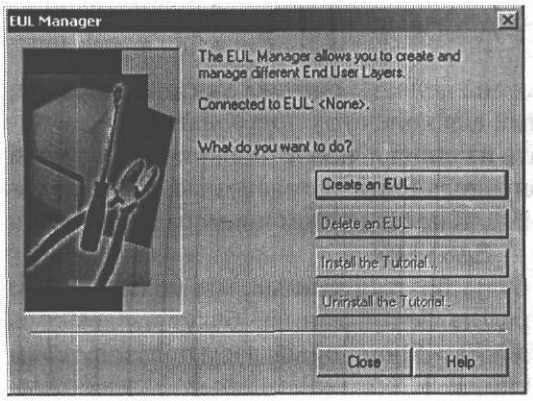


Рис. 6.28. *Discoverer Administrator — создание уровня конечного пользователя (EUL)*

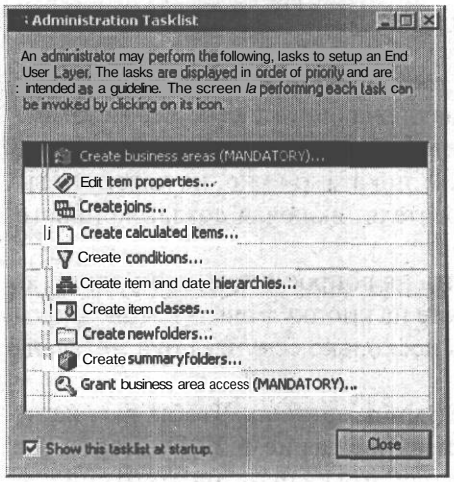


Рис. 6.29. *Discoverer Administrator — список задач*

продажи или финансы. Области бизнес-задач представляют собой элементы контроля доступа, и пользователю можно отвести одну или более область бизнес-задач. После этого пользователи будут иметь доступ ко всем объектам в выделенной им области.

В области бизнес-задач вы можете точно указать, к каким данным пользователь может иметь доступ, как эти данные могут соединяться, а также описать агрегирование данных и новые элементы данных, вычисляемые на основе уже имеющихся. Создание области бизнес-задач займет некоторое время, но позже принесет много пользы.

При создании области бизнес-задач мы должны сначала указать те схемы, данные из которых делаются доступными для конечных пользователей (как показано на рис. 6.30). Мы можем выбрать из списка любое число пользователей (схем), а также потребовать, чтобы выбирались только те элементы, которые соответствуют образцу, приведенному в окне. В приведенном примере мы выбрали только одного пользователя или схему — EASYDW.

Когда мы знаем схему, из которой будут браться данные, мы можем явным образом указать, из каких таблиц и представлений данные будут извлекаться (рис. 6.31). В нашем примере с компанией Easy Shopping мы даем пользователям доступ к четырем из пяти таблиц хранилища.

Чрезвычайно полезным свойством Discoverer является возможность с помощью мастера автоматически создавать соединения на основе первичных и внешних

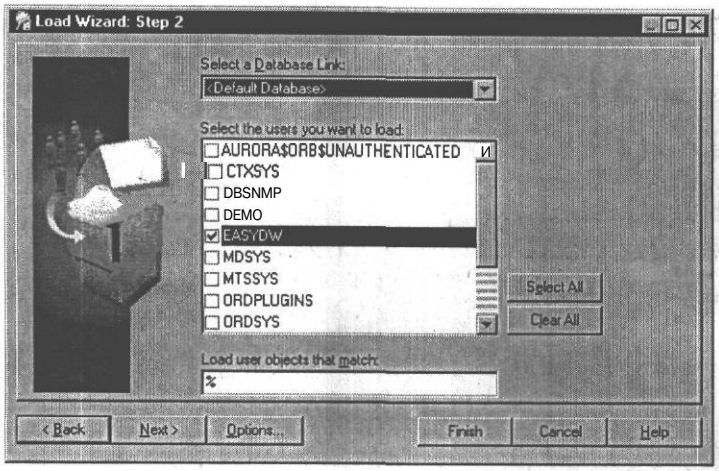


Рис. 6.30. *Discoverer Administrator — выбор схемы*

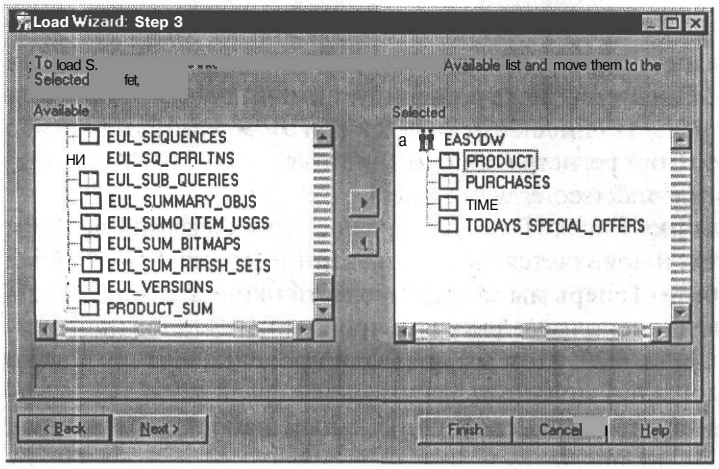


Рис. 6.31. *Discoverer Administrator — выбор таблиц и представлений*

ключей и создавать иерархии данных, как показано на рис. 6.32. Если вы позволите мастеру выполнить эту работу, вам останется меньше дел по настройке, и по умолчанию эти опции уже выставлены.

Есть еще одно, последнее окно, перед тем как будет завершена первая стадия создания области бизнес-задач. В окне, показанном на рис. 6.33, мы должны

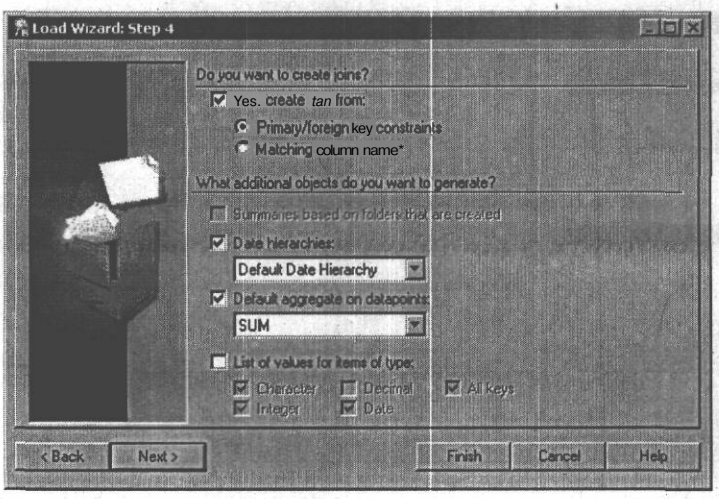


Рис. 6.32. *Discoverer Administrator — автоматическое создание соединений*

дать название области бизнес-задач и, при желании, предоставить описание того, что в ней содержится. Discoverer также сгенерирует имена для всех объектов, и на рис. 6.33 вы можете увидеть список имеющихся при этом опций, например перевести все буквы в верхний регистр (capitalize) или заменить знаки подчеркивания пробелами (replace underscores with spaces).

После нажатия на кнопку Finish (Готово) первая стадия создания области бизнес-задач завершается, и появляется окно, показанное на рис. 6.34. Сейчас начинается реальная работа. Теперь мы можем провести окончательную настройку области бизнес-задач, используя для учета пройденных этапов наш список задач, показанный на рис. 6.29. Хотя мы показали процесс создания только одной области бизнес-задач, вы можете создать в своей среде любое их количество, и каждая из них будет иметь свой уникальный набор требований к данным.

Ограничение числа отображаемых столбцов

По умолчанию, когда к таблице предоставляется доступ, он предоставляется ко всем столбцам этой таблицы. Очень приятное свойство Discoverer состоит в том, что пользователи могут осуществлять доступ к данным только через область бизнес-задач. Если таблица или столбец в эту область не входят, пользователь никогда и не узнает, что данные существуют.

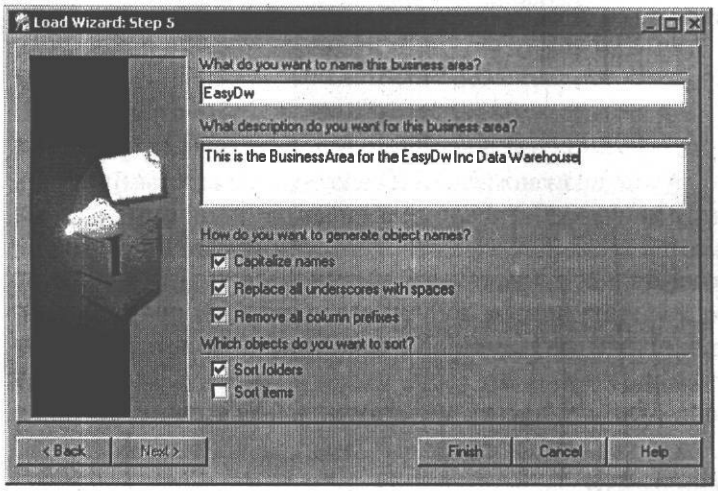


Рис. 6.33. *Discoverer Administrator — название для области бизнес-задач*

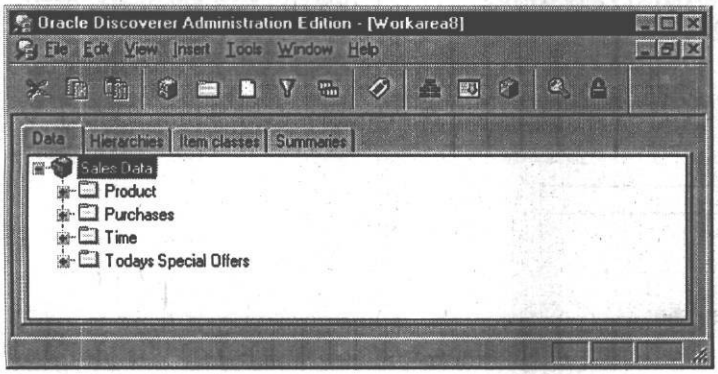


Рис. 6.34. *Discoverer Administrator — настройка области бизнес-задач*

На рис. 6.34 видны все таблицы базы данных, к которым у нас есть доступ. Отметьте, что таблица TODAYS_SPECIAL_OFFERS теперь называется Todays Special Offer в результате выполнения заданного изменения имени. По умолчанию конечный пользователь имеет доступ ко всем столбцам этих таблиц. Если вы щелкнете мышью по названию таблицы, отобразятся все столбцы. Чтобы убрать любой из столбцов, просто щелкните по элементу правой кнопкой мыши. Появится раскрывающийся список. Одним из пунктов этого списка будет пункт Delete

(Удалить). Выберите этот пункт, и элемент будет убран из области бизнес-задач, но не из базы данных.

Папки

Перед тем как идти дальше, нужно познакомить вас с некоторыми важными терминами. В Discoverer таблицы или представления называются папками (folders), а столбцы таблицы называются элементами (items). Папки могут быть двух типов: простые, которые основаны на одной таблице или представлении, и сложные, которые могут содержать элементы из других папок и которые могут быть вложенными.

Элемент соответствует столбцу реляционной базы данных. Простой элемент основывается на одном столбце, но элемент также может вычисляться или выводиться по формуле с использованием других элементов, функций и операторов.

Изменение свойств элемента

Атрибуты любого выбранного элемента можно изменить, щелкнув правой кнопкой мыши и выбрав пункт Properties (Свойства). Появится окно, показанное на рис. 6.35, и вы сможете изменять любые свойства. В приведенном примере

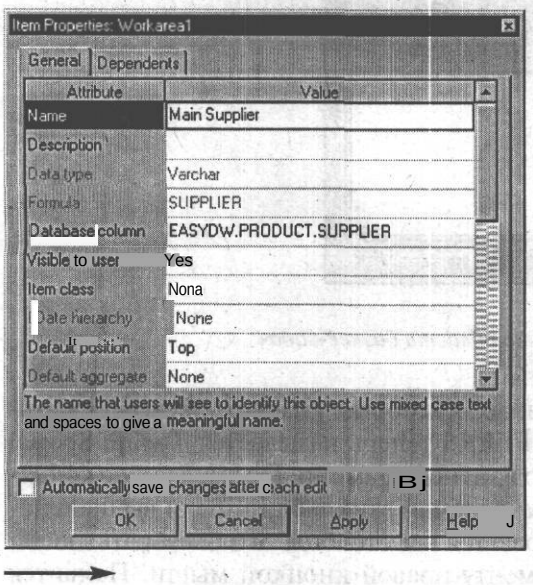


Рис. 6.35. *Discoverer Administrator — изменение свойств элемента*

мы изменили название элемента с Supplier на Main Supplier, и это означает, что нашим пользователям можно предоставить гораздо более «дружественные» и понятные названия, чем они есть в «компьютерном» формате.

Когда область бизнес-задач создается первоначально и если в базе данных определены первичные и внешние ключи, Discoverer автоматически создаст соединения между этими ключами. Вы, тем не менее, можете определять свои собственные соединения, выбрав из панели меню пункт Insert (Вставить), затем Joins (Соединения). Появится окно, показанное на рис. 6.36.

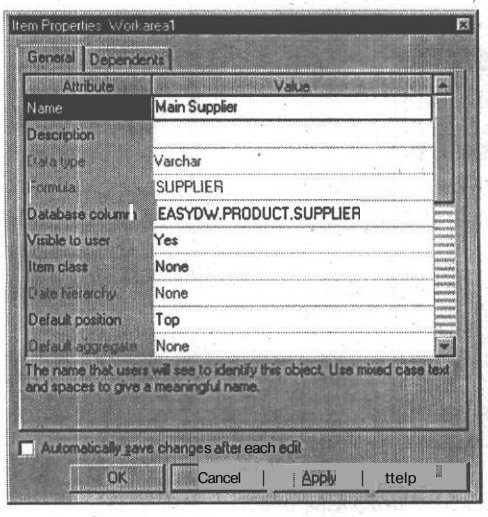


Рис. 6.36. *Discoverer Administrator — определение соединения*

Соединение определяется путем выбора в раскрывающемся меню элемента папки, который нужно соединить, и типа соединения. В нашем примере на рис. 6.36 мы определили соединение между таблицей Todays Special Offer и таблицей Product с помощью столбца Product_id. Преимущество, которое мы получаем, создав это соединение, состоит в том, что когда пользователь будет писать запрос, Discoverer будет знать, как соединять данные, и пользователю придется указывать меньше информации.

Создание новых элементов

Еще одно свойство, которое может потребоваться многим администраторам баз данных, — это возможность создавать в базе новые столбцы или элементы, вычисляя эти данные по другим столбцам.

Вычисление (calculation) создает новый элемент на уровне конечного пользователя. При этом столбцы в таблицы базы не добавляются. Эта возможность используется для создания нового элемента при отсутствии столбца базы, содержащего необходимые данные. Вычисления могут быть простыми, например weight*4.54, или же сложными математическими и статистическими выражениями. В примере на рис. 6.37 в таблице Purchases создается новый столбец Total Cost, для которого складываются данные столбцов purchase price и shipping charges. Вы можете создать столько вычислений, сколько вам нужно, и они будут отображаться как элементы таблицы.

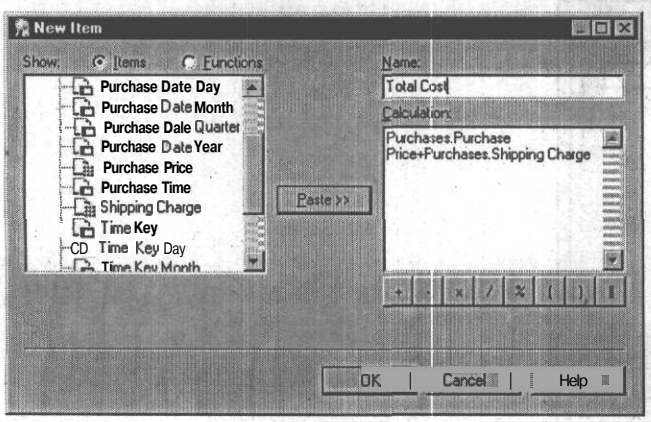


Рис. 6.37. *Discoverer Administrator — создание вычисляемого элемента*

Иерархии

Мы уже видели, что иерархии играют очень важную роль в хранилище. Хотя в базе Oracle мы можем создавать объекты-измерения, на момент написания этой книги в Discoverer они использоваться не могут, и мы должны создавать собственные измерения, которые в Discoverer носят название иерархий (hierarchies). Иерархии в Discoverer очень важны, поскольку, если элемент располагается в иерархии, пользователи могут передвигаться по иерархии вверх, изменяя запрос таким образом, чтобы увидеть больше деталей, или вниз — чтобы увидеть меньше деталей.

То, как можно перемещаться по иерархии, показано в отчете на рис. 6.27.

Иерархию можно создать, щелкнув по пункту Insert (Вставить) в верхней части окна, а потом выбрав пункт Hierarchy (Иерархия). Вам будет нужно ответить

на вопрос о типе иерархии, которую вы хотите создать: элементную или по датам. Когда область бизнес-задач создается первоначально, весьма вероятно, что Discoverer автоматически создаст иерархию по времени. Следовательно, вам, возможно, придется создавать только не-временные иерархии.

На рис. 6.38 показано, как легко можно создавать иерархии, выбирая элементы и определяя иерархические взаимоотношения. В примере на рис. 6.38 мы создали очень простую иерархию по идентификаторам продуктов (product_id) и категориям (category).

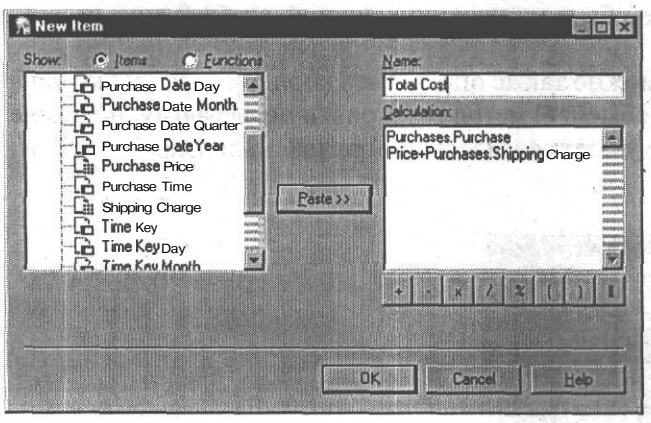


Рис. 6.38. *Discoverer Administrator — создание иерархии*

Классы элементов — список значений

Когда конечные пользователи реально посылают запросы, бывают ситуации, когда им очень полезно было бы видеть список возможных значений. Предположим, например, что им нужно выбрать все элементы — электрические приборы. Если бы они знали, к какой категории они относятся, это позволило бы ускорить создание запроса.

Класс элементов описывает иерархические взаимоотношения элементов, список значений, альтернативные ключи сортировки для элементов и методы отображения. Следовательно, класс элементов определяет все атрибуты элемента. После определения класса элементов к нему можно отнести элементы, имеющие общие свойства.

Класс элементов можно создать, щелкнув по пункту Insert (Вставить) в верхней части окна, а потом выбрав из списка пункт Insert Class или щелкнув по тексту

в списке задач, показанном на рис. 6.29. Можно создавать классы элементов нескольких разных типов, но если вы выберете пункт List of Values (Список значений), то вам придется выбрать только столбец, содержащий значения, и таблицы, в которых они будут использоваться. По завершении, щелкните по закладке Item Classes (Классы элементов), и появится окно, показанное на рис. 6.39. В данном примере был создан список значений с именем Categories по таблице измерения Products. Если мы раскроем пункт categories, будет послан запрос к хранилищу и будут выведены все возможные значения. На рис. 6.39 мы видим некоторые из продаваемых продуктов. Здесь у нас представлено всего несколько значений, но в реальном мире, где у вас может быть множество значений, отображение результатов можно не выполнять.

При определении класса можно также определить порядок сортировки данных. На рис. 6.39 был выбран обычный метод сортировки по алфавиту, но это не всегда оправдано, и вы можете указать логический порядок, например N, S, E, W, а не E, N, S, W.

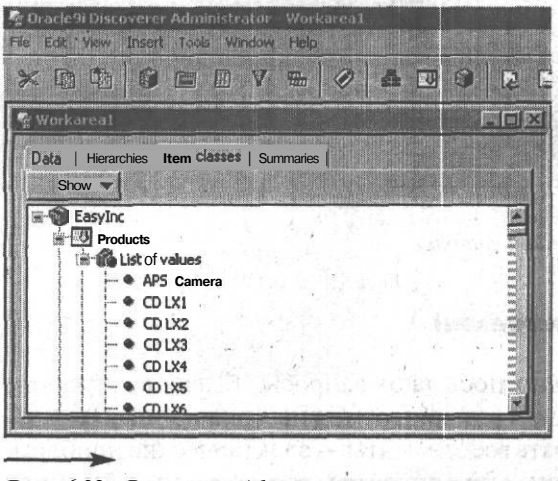


Рис. 6.39. *Discoverer Administrator — класс элементов*

Сводки

Мы уже рассматривали в главе 4, как важно создавать и использовать в хранилище материализованные представления. Discoverer позволяет вам создавать собственные материализованные представления, которые в Discoverer называются сводками (summary). Их можно создавать либо из Мастера создания сводок (Summary Wizard), либо следующими способами:

- использование статистики обработки запросов;
- создание сводок вручную;
- регистрация ранее созданной сводки.

Чтобы создать сводку, щелкните по пункту Insert (Вставить) в верхней части окна, а потом выберите из списка пункт Summary (Сводка). Появится окно мастера Summary Wizard, и вам будет задан вопрос о типе сводки, которую вы желаете создать. Если вы выберете пункт From Items (Из элементов) в уровне конечного пользователя, вы сможете создать собственную сводку.

Далее появится окно, показанное на рис. 6.40. Здесь вы выбираете папки и элементы внутри них, которые должны появится в сводке. В нашем примере мы выбрали только таблицу Purchases, но вы можете выбрать и несколько таблиц. Мы выбрали из этой таблицы четыре элемента данных и задали агрегировать значение элемента purchase price. Вы увидите, что Discoverer автоматически предоставит на выбор целый диапазон функций, из которых можно выбрать ту, которая применима к данному элементу. В примере на рис. 6.40 мы указали, что нужно применить к элементу purchase price функцию SUM. После этого от вас потребуется указать, какие вам необходимы группы элементов.

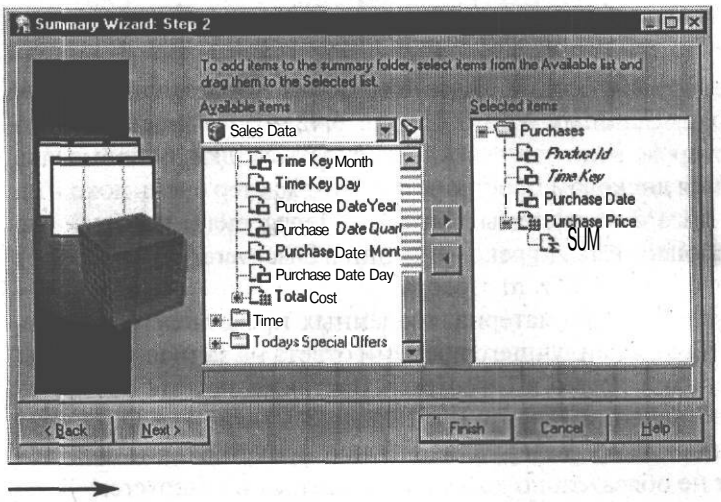


Рис. 6.40. *Discoverer Administrator — создание сводки*

Как мы уже видели, очень важно обеспечить соответствие материализованных представлений самым последним данным. При определении сводок в Discoverer можно указать частоту их обновления. Помните, что при поступлении в хранилище

новых данных, сводки нужно изменять, чтобы они отражали эти данные. В примере на рис. 6.41 мы объявляем, что данную сводку нужно обновлять каждый час.

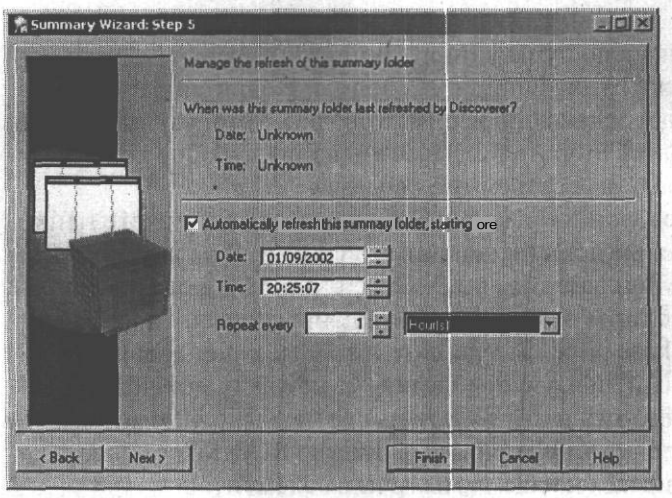


Рис. 6.41. *DiscovererAdministrator — обновление сводки*

Если вы не знаете, какую создать сводку, Discoverer может порекомендовать их вам через свой мастер сводок Summary Wizard. На рис. 6.42 можно видеть один из этапов работы с этим мастером, в котором можно выбирать сводки, основываясь на сведениях об имеющемся дисковом пространстве. Этот мастер очень похож на мастер Summary Advisor Wizard из системы Summary Management, который мы описывали в главе 4, но процесс выдачи рекомендаций в Discoverer отличается от того, который использует Summary Wizard сервера.

Создание сводок Discoverer или материализованных представлений очень важно, если вы хотите добиться наилучшего времени ответа на запрос. Здесь мы увидели, как создавать сводки прямо в Discoverer. Вы также можете создавать материализованные представления, как это описывалось в главе 4, и Discoverer будет использовать их, поскольку материализованное представление для использования в Discoverer не обязательно должно создаваться в Discoverer.

Вопросы безопасности

Последняя задача по настройке — это указание тех, кто может иметь доступ к только что созданной области бизнес-задач. Начать выполнение этой задачи

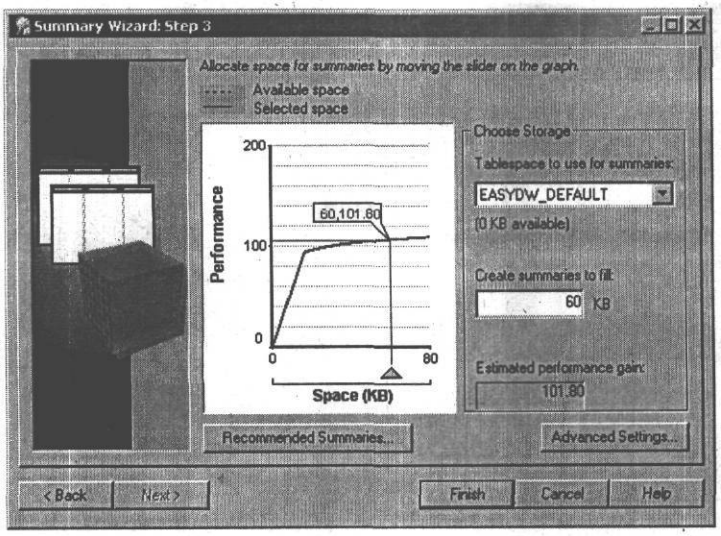


Рис. 6.42. *Discoverer Administrator — мастер Summary Wizard*

можно, сделав двойной щелчок по пункту Grant Business Area Access (Предоставить доступ к области бизнес-задач), показанному на рис. 6.29. Появится окно, которое позволит вам объявить, какие пользователи имеют доступ к данной области бизнес-задач или к каким областям бизнес-задач имеет доступ данный пользователь. В примере на рис. 6.43 мы видим, что к нашей области бизнес-задач имеет доступ только один пользователь — EASYDW. Не забывайте, что, хотя мы разрешили доступ к области бизнес-задач по имени пользователя, мы можем также выделять доступ по ролям, которые могут присваиваться пользователям. Когда вас будут устраивать все выданные права доступа, нажмите кнопку Apply (Применить) для завершения.

Рисунок 6.44 показывает нашу готовую область бизнес-задач. Просто щелкнув по таблице Purchases, мы увидим все элементы, доступные пользователям, новый, созданный нами вычисляемый элемент Total Cost, а также функции, которые мы можем применять к этому столбцу. В нижней части окна мы можем видеть, что были созданы соединения таблицы Purchases с таблицами Product и Time. Конечно в одном этом окне нельзя отобразить всю информацию, поэтому, чтобы увидеть иерархии, щелкните по закладке Hierarchies. По аналогии, переходите к классам элементов и к информации о сводках.

Теперь мы завершили выполнение всех основных настроечных задач для работы с Discoverer. Не забудьте сохранить свою работу и, пожалуйста, запомните, что данный инструмент имеет гораздо больше возможностей, чем мы здесь по-

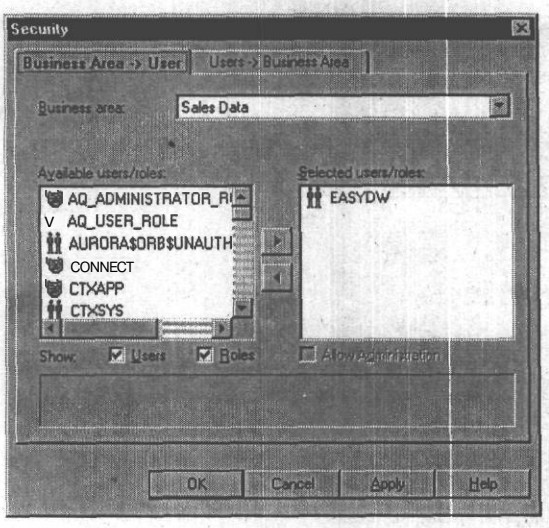


Рис. 6.43. *DiscovererAdministrator — определение прав доступа к области бизнес-задач*

казали. Например, мы не показали, что Discoverer полностью поддерживает аналитические функции Oracle, предназначенные для сложного анализа данных. Теперь мы приступим к работе с Discoverer Desktop, Discoverer Viewer и Discoverer Plus, и будем извлекать данные из нашего хранилища.

6.3.3. Посылка запросов с помощью Discoverer Plus и Discoverer Desktop

Когда среда настроена для посылки запросов с помощью Discoverer, можно запустить версию Desktop или версию Plus, которая разработана для использования в Web. В данном разделе в примерах будет использоваться web-версия — Discoverer Plus.

Отчеты в Discoverer хранятся в рабочей книге (workbook), так что первым шагом будет соединение с базой данных, в которой находится рабочая книга. Если вы загружаете Discoverer Plus из браузера по URL, например,

```
http://easydw.com:7777/discoverer/plus,
```

вы увидите список баз данных, к которым вы можете подключиться (рис. 6.45). Здесь мы видим, что имеется только одна база данных — EasyDw.

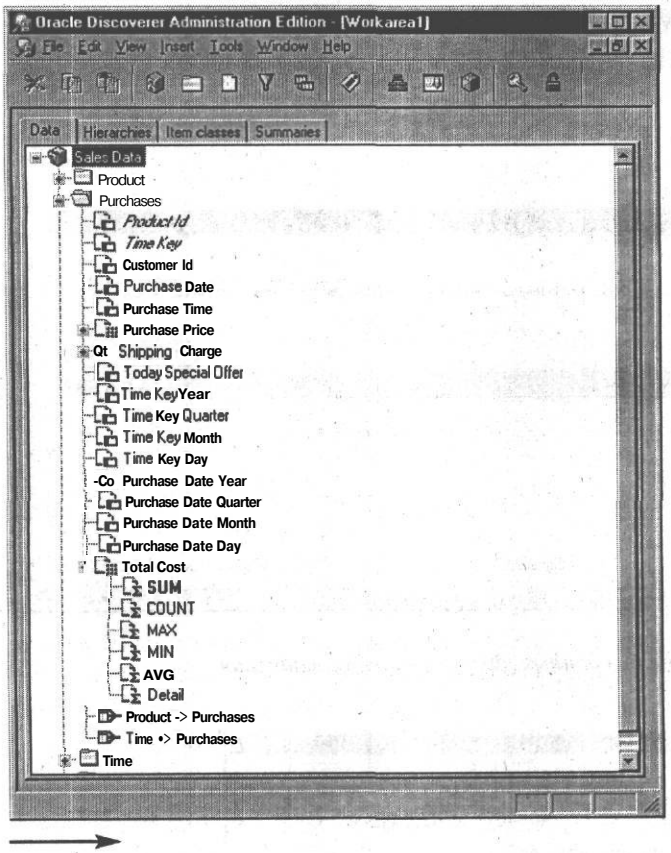


Рис. 6.44. *Discoverer Administrator — готовая область бизнес-задач*

Эти соединения определяются ранее и состоят из вашего пользовательского имени и имени базы данных. Следовательно, вам нужно указать только пароль, и вам будет задан вопрос о том, желаете ли вы создать новую или открыть имеющуюся рабочую книгу. На рис. 6.46 представлено стартовое окно, в котором указывается рабочая книга и определяется форма отображения результатов. Discoverer предлагает широкий выбор опций отображения, например представление данных в виде обычных или перекрестных таблиц. Использование перекрестных таблиц идеально при отображении многомерных данных.

Теперь пора конкретно определить, что будет сообщаться в запросе. Во-первых, нужно выбрать область бизнес-задач, созданную в версии Discoverer Administrator, которая определяет, какие данные вы можете просматривать.

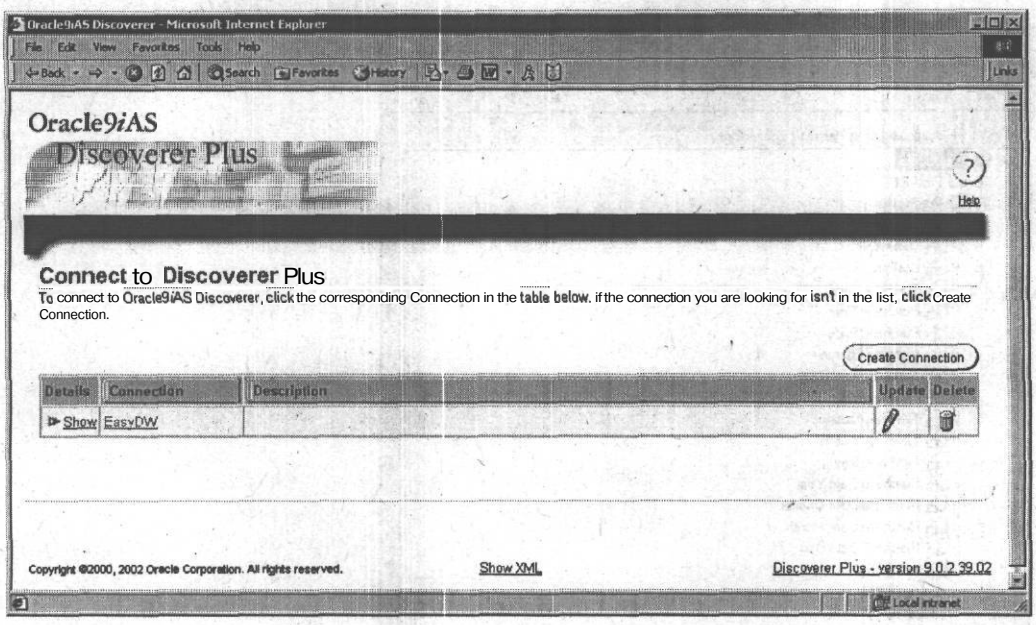


Рис. 6.45. *Discoverer Plus — список баз данных, доступных для соединения*

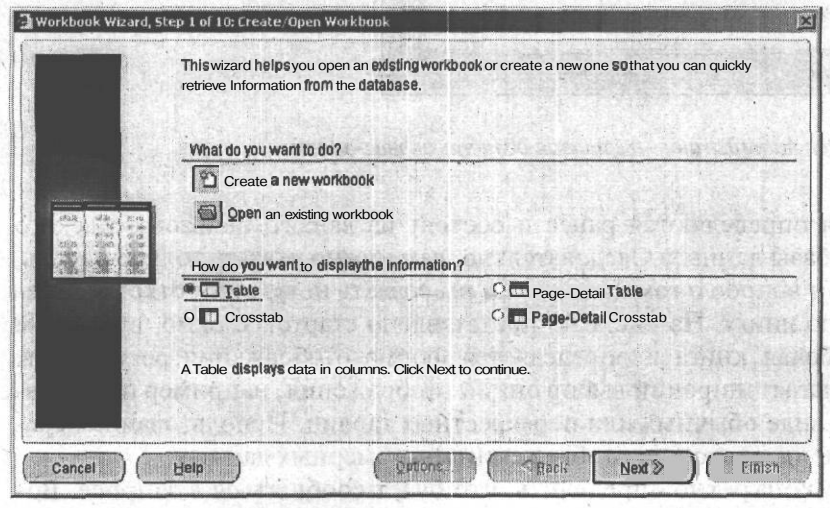


Рис. *6.46. Discoverer Plus — использование рабочей книги*

В нашем примере имеется только одна область бизнес-задач — Sales Data, но возможно наличие нескольких на выбор.

Теперь нам нужно выбрать элементы из папок. Процесс этот простой и выполняется путем переноса элементов из левого окна (Available, имеющиеся) в правое окно (Selected, выбранные). В примере на рис. 6.47 мы выбрали элемент category из папки product и элемент date из папки purchases и задали, чтобы элемент purchase price был агрегирован.

Когда Discoverer выявляет элемент, к которому можно применить функции, пользователю предоставляются все имеющиеся возможности. Например, на рис. 6.47 мы видим, что для элемента purchase_price применимы функции MAX и COUNT, и Discoverer так же может вычислить минимальное, максимальное и среднее значение. Мы уже выбрали суммирование по столбцу purchase price, просто развернув опции агрегирования в левой части окна, указав необходимый агрегат и нажав на кнопку «стрелка вправо». Этот процесс следует повторить для всех необходимых для отчета элементов.

Теперь мы должны указать расположение элементов отчета (layout). Это делается очень легко, путем перетаскивания необходимых столбцов туда, где вы хотите видеть их в отчете. В примере на рис. 6.48 мы указали следующий порядок: категория, дата и общая стоимость. Но мы можем с легкостью разместить столбец даты после столбца общей стоимости. На этом этапе мы лишь определяем, как данные будут представляться. В следующем окне мы определим форматы и заголовки.

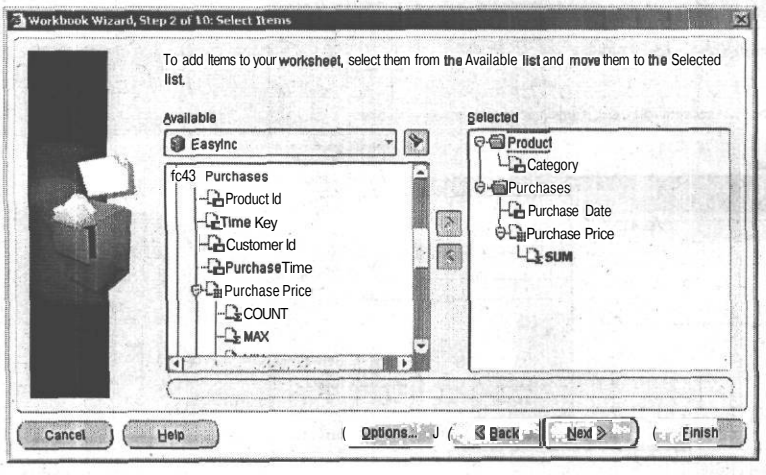


Рис. 6.47. *Discoverer Plus — Выбор отображаемых данных*

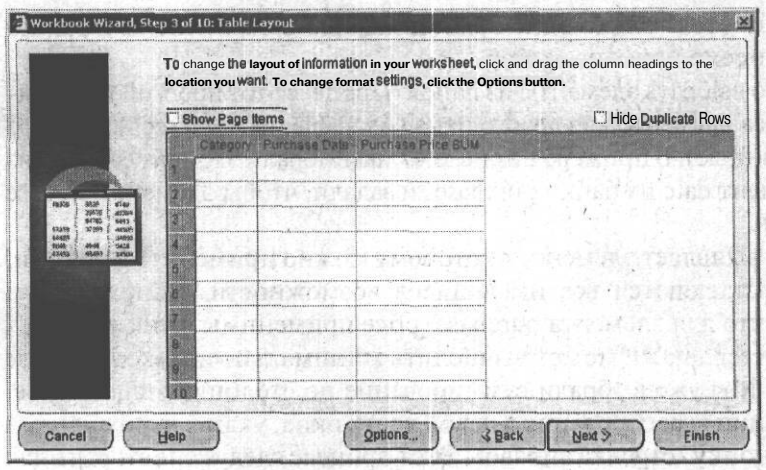


Рис. 6.48. *Discoverer Plus — расположение элементов отчета*

Для каждого отображаемого элемента мы можем определить способ форматирования данных и выбрать используемые в отчете заголовки. На рис. 6.49 мы заменили заголовок столбца с суммой (Purchase Price SUM) на Total Purchases. Параметры шрифта и выравнивания можно указать, щелкнув по кнопке Format Heading (Формат заголовка), а нажатие на кнопку Format Data (Формат данных)

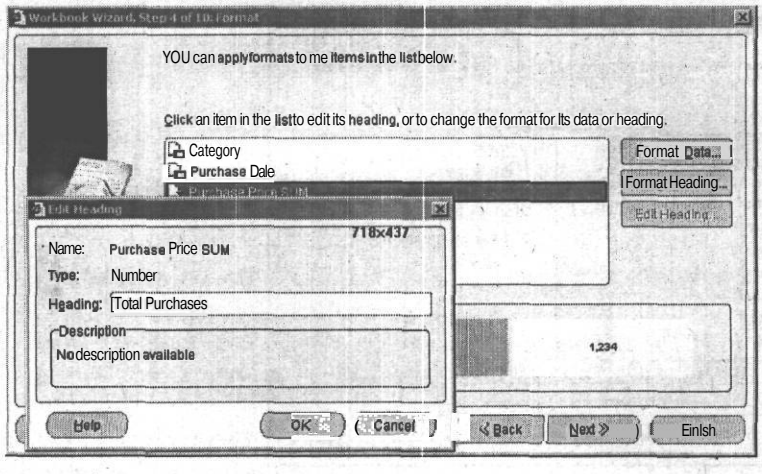


Рис. 6.49. *Discoverer Plus — формат заголовков*

позволит указать способ представления данных. В приведенном примере мы решили, что суммарное значение не будет иметь десятичных разрядов.

Теперь мы может указать условие, определяющее соответствие данных определенному критерию. Здесь имеется довольно много опций, но в данном примере мы хотим видеть все данные. После этого мы может указать способ сортировки данных, что можно сделать, нажав на кнопку Add (Добавить), а потом выбрав столбец. Столбец появится в списке, и вы сможете указать порядок сортировки. В примере на рис. 6.50 мы сортируем сначала по дате, а потом по категории.

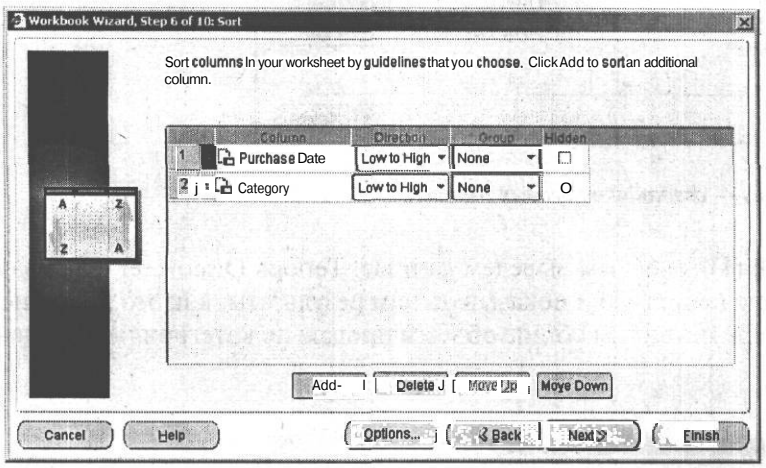


Рис. 6.50. *Discoverer Plus — сортировка данных*

На этом этапе мы также можем добавлять вычисляемые элементы (calculations), которые будут появляться в отчете, например, вычтя из себестоимости покупную цену мы получим прибыль. В наш сегодняшний отчет мы не будем включать вычисляемые элементы. Еще одна возможность — это создание процентной точки для всех элементов отчета, где это может оказаться полезным для понимания данных.

Мы уже говорили, что нам нужно просуммировать элемент purchase price, и в примере на рис. 6.51 мы также можем добавить к этому элементу итоговое значение (total). Как можно видеть, имеется множество опций, таких как, определение того, нужны ли нам промежуточные суммы (subtotals), определение типа суммирования и способа форматирования данных. И опять же, мы можем создавать столько итоговых значений, сколько нужно для отчета.

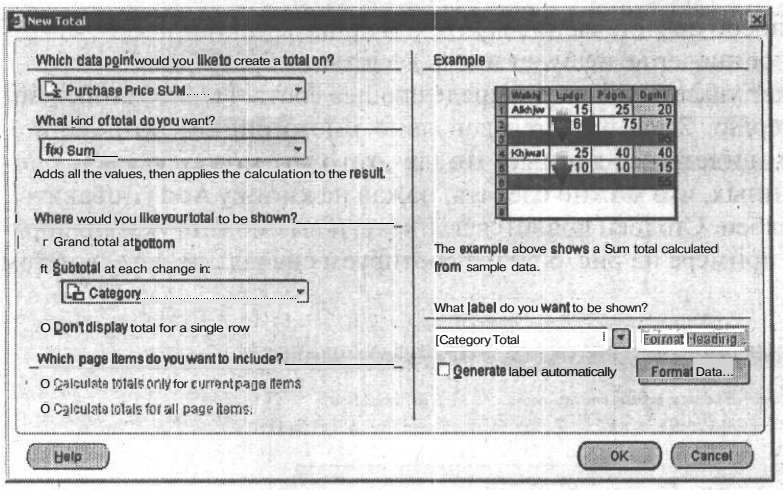


Рис. 6.51. *Discoverer Plus — создание итоговых значений*

Нажав кнопку Finish (Готово), мы выведем данные. Теперь Discoverer будет запрашивать хранилище напрямую и показывать нам результаты, как это показано на рис. 6.52. В примере мы видим общие объемы продаж по категориям для указанной даты.

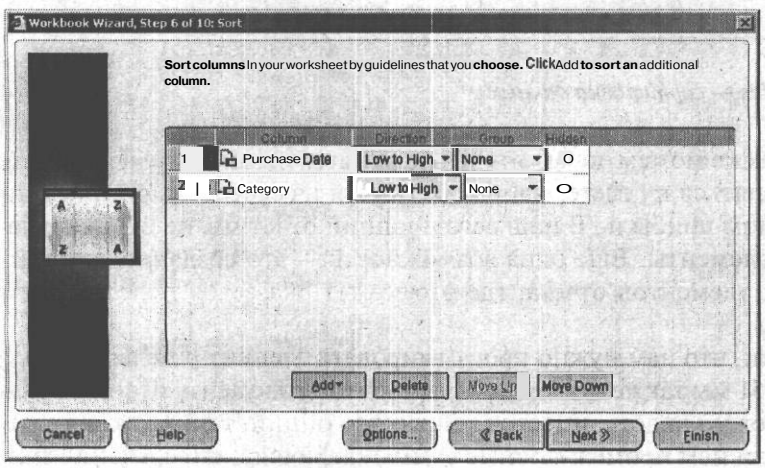


Рис. 6.52. *Discoverer Plus — сообщение о результатах*

Совет. Для того чтобы увидеть данные, может потребоваться отформатировать ячейки отчета, если числа велики, поскольку по умолчанию используются ячейки для небольших чисел.

Заметьте, что для создания отчета нам не потребовалось указывать, как нужно соединять выбранные таблицы, поскольку информация о соединениях уже указана в области бизнес-задач. Это одно из самых приятных преимуществ работы в Discoverer, поскольку конечный пользователь не должен знать ничего о реляционных соединениях. Вся эта работа делается за кулисами тем, кто создает область бизнес-задач с помощью версии Discoverer Administrator. Теперь наши пользователи могут только выбирать интересующие их данные, отвечать на вопросы в нескольких окнах и, щелкнув кнопку Finish, посылать запрос.

Теперь, когда у нас есть отчет, мы можем настраивать его под наши конкретные требования, либо щелкая мышью по элементам, либо выбирая меню в верхней части экрана. Чтобы изменить формат представления чисел, мы щелкаем по пункту Sheet (Лист), а потом — Format (Формат). Появится окно, показанное на рис. 6.53, в котором мы можем внести изменения в элемент. Обратите внимание, все окна, через которые мы проходили, создавая отчет, теперь представлены в виде закладок. Мы можем легко переходить к ним, а также вносить изменения.

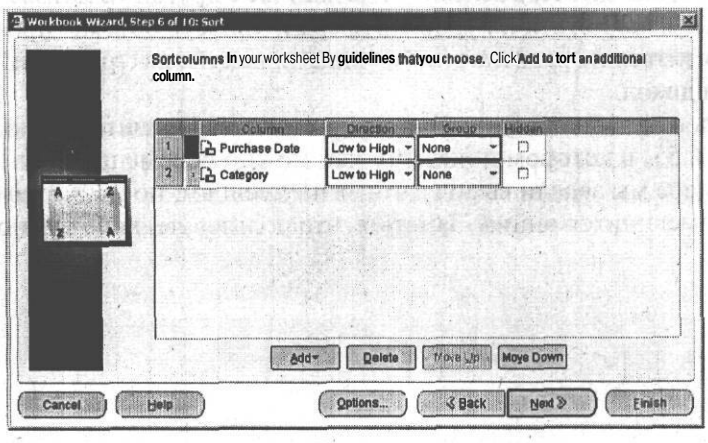


Рис. 6.53. *Discoverer Plus — customize the report*

Если вас интересует, как данный отчет будет выполняться в базе данных, выберите пункт Sheet (Лист), а потом — Show SQL (Показать код SQL). Будет вызван SQL Inspector, окно которого показано на рис. 6.54. В нем вы можете просмотреть либо код SQL, использующийся для выполнения запроса, либо, как показано на рис. 6.54, план обработки запроса.

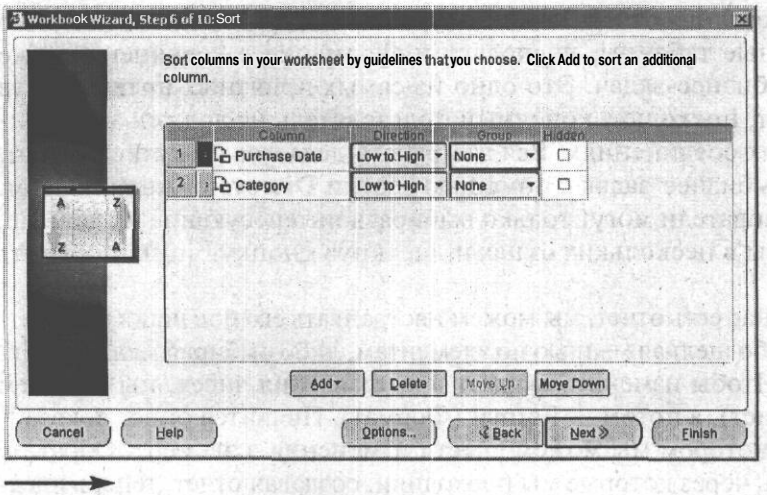


Рис. 6.54. *Discoverer Plus — SQL Inspector*

Теперь, имея отчет, мы можем захотеть взглянуть на данные с другой точки зрения. Например, мы можем подняться на уровень вверх и увидеть объемы продаж по месяцам или опуститься на уровень вниз и увидеть, какой электрический прибор дал наибольший доход.

Вы можете изменить отчет, щелкнув мышкой по стрелке. Появится окно, представленное на рис. 6.55, в котором показано, как можно перемещаться по уровням данных. В примере мы задали вывод данных по месяцам, но мы можем выбрать любую из пяти имеющихся опций. Заметьте, что пользователю не нужно



Рис. 6.55. *Discoverer Plus — перемещение по уровням данных*

указывать Discoverer, как создавать отчет. Вся эта информация была определена в ходе настройки, так что опять же нашим пользователям нужно знать очень немногое о том, как хранятся данные, чтобы получать необходимую информацию.

На рис. 6.56 мы теперь видим наш отчет с информацией на уровне месяцев. Поскольку у нас есть иерархия времени, мы можем изменить отчет и проводить суммирование по годам, а не по дням, просто выбрав пункт Year в списке. Мы надеемся, что вы начинаете ценить ту работу по настройке, которую мы провели, используя версию Discoverer Administrator.

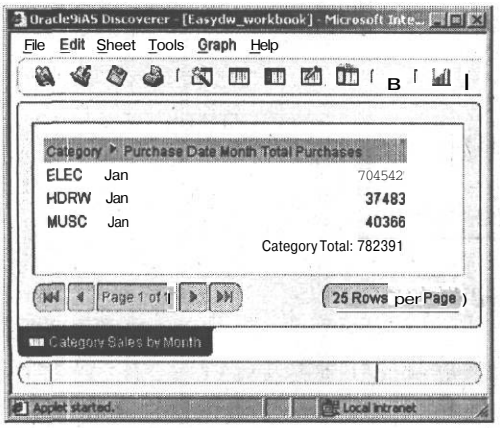


Рис. 6.56. *Discoverer Plus — отчет с данными уровня месяцев*

До сих пор мы просматривали данные в традиционном формате отчетов, однако Discoverer Plus может представить их графически, если вы ответите на несколько вопросов Мастера графиков Graph Wizard, окно которого показано на рис. 6.57.

В Discoverer Plus есть более дюжины разных типов графиков, и вы можете полностью настроить выводимую информацию, добавить свои заголовки и легенды. В примере на рис. 6.58 мы решили использовать для представления данных о месячных продажах по месяцам секторную диаграмму, которая помогает лучше видеть, что с момента начала продаж 1 января наиболее популярными товарами были электроприборы.

В приведенном примере отчета мы выводим все данные, однако вы можете отобрать часть их, указав определенное условие. Особенно приятным свойством Discoverer Plus является возможность установить несколько условий, а затем выбирать те, которые нужны для данного отчета. На рис. 6.59 мы видим одно из окон, в которых можно указывать условия.

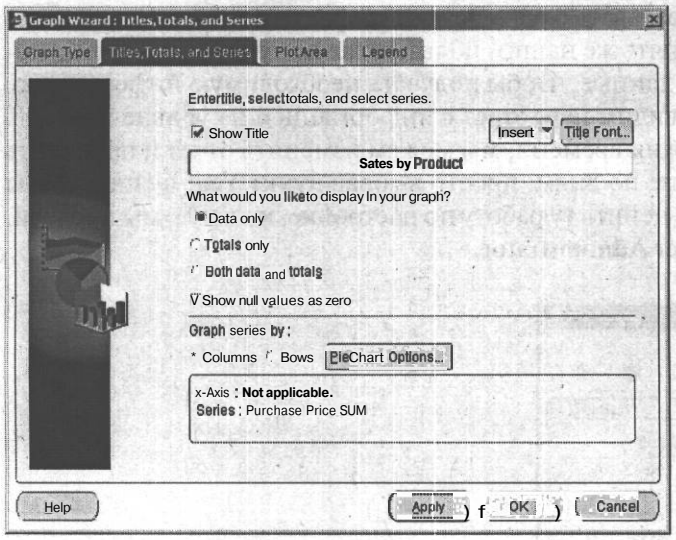


Рис. 6.57. *Discoverer Plus — мастер Graph Wizard*

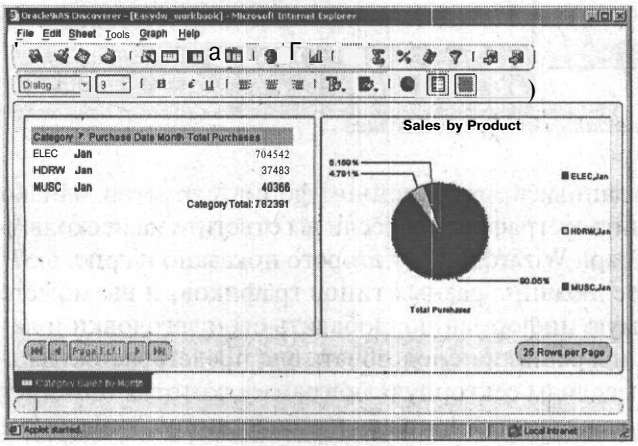


Рис. 6.58. *Discoverer Plus — диаграмма месячных продаж*

В действительности, есть еще одно окно, в котором вы можете определить условие с помощью довольно сложных выражений. В приводимом примере видно, что мы определили два условия, используем одно из них (стоимость товара

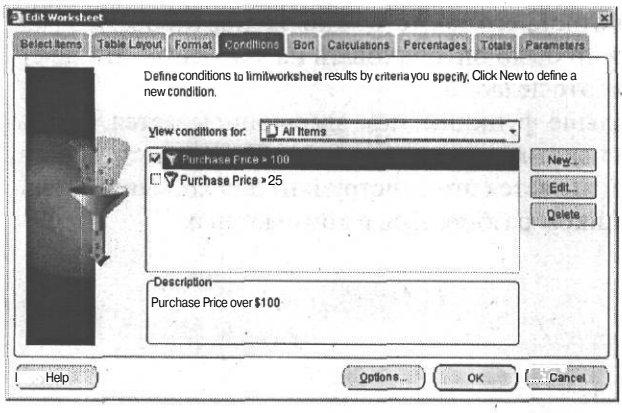


Рис. 6.59. *Discoverer Plus — специфический отбор данных*

больше $100), и проверяем какое влияние это условие окажет на наш отчет. Когда мы просматриваем новый отчет (рис. 6.60), мы видим, что больше $100 стоили только электроприборы, но даже при этом такие товары не составляют большей части наших продаж за этот месяц.

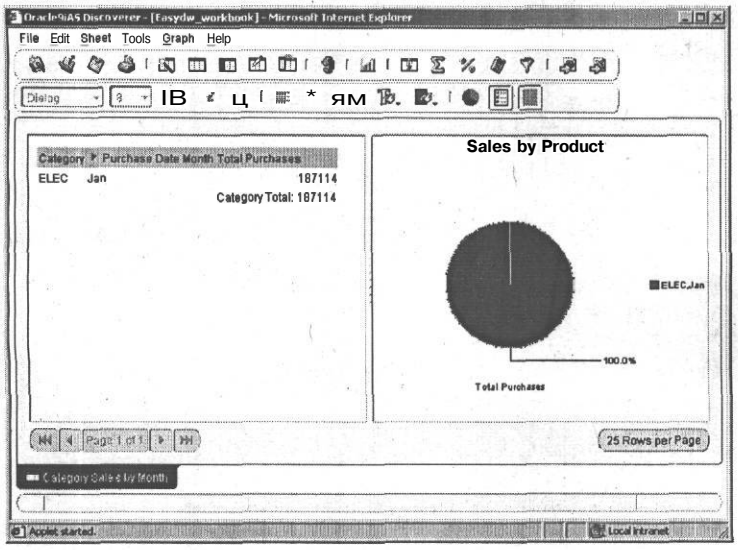


Рис. 6.60. *Discoverer Plus — отчет с использованием условий*

13-1559

После завершения создания отчета его также можно экспортировать в различные форматы, чтобы его можно было опубликовать на web-сайте. В главе 8 приводятся объяснения того, как это делается.

У Discoverer есть гораздо больше функций, чем здесь описывается, но мы надеемся, что теперь вы оценили некоторые из возможностей этого инструмента, а также то, насколько идеальным является этот инструмент для конечных пользователей, которые могут не слишком разбираться в компьютерах.

7

Управление хранилищем

7.1. Чем нужно управлять?

Когда хранилище данных создано и заполнено данными, очень важно обеспечить правильное управление им. Его нужно сконфигурировать для оптимальной производительности и доступности. Управлять нужно ресурсами дисков, памяти и процессоров. В этой главе мы опишем некоторые задачи, связанные с таким обслуживанием, и дадим советы по их выполнению.

Снова мы будем использовать средства с графическим пользовательским интерфейсом, предоставляемые в Oracle Enterprise Manager (OEM). Существуют также и альтернативные методы, но мы уверены, что вы согласитесь, что использование OEM делает управление хранилищем данных значительно более легким.

В этой главе мы сначала предложим вам введение в OEM, затем опишем различные методы резервного копирования хранилища с помощью Recovery Manager (RMAN), покажем важность разработки стратегии восстановления после аварий, обсудим реорганизацию хранилища, сбор статистики оптимизатора, тонкую настройку хранилища, поддержание безопасности и слежение за использованием дискового пространства.

7.2. Управление с помощью - Oracle Enterprise Manager

Oracle Enterprise Manager можно использовать для управления любой базой данных, расположенной в любом месте широкомасштабной сети. Следовательно, с помощью одного этого инструмента вы можете управлять всей системой, независимо от того, где находятся ее компоненты. Это чрезвычайно мощное свойство,

и вы должны помнить, что все то, что мы описываем, применимо не только к хранилищу, но и к другим базам данных вашей сети. В этом разделе мы рассмотрим некоторые концепции и шаги, необходимые для того, чтобы начать работать с OEM.

Enterprise Manager был создан для работы в трехуровневой архитектуре:

- консоль, предоставляющая клиентской системе графический пользовательский интерфейс;
- сервера управления (Oracle Management Servers, OMS), обеспечивающие административные функции, такие, как запуск заданий и событий, и работу на среднем уровне;
- интеллектуальные агенты (intelligent agents), которые следят за базой данных, запускают и останавливают ее работу и собирают данные о производительности. Интеллектуальные агенты работают на третьем уровне и располагаются на тех же узлах, что и базы данных Oracle.

Вся информация, необходимая для управления средой и ее администрирования, хранится в репозитории OEM, который может являться частью базы данных Oracle или храниться в базе Oracle, которую использует только Enterprise Manager. Именно здесь сервера управления сохраняют информацию, необходимую для управления конфигурацией сети, отслеживаемые события, выполняемые задания и учетные записи администраторов.

7.2.1. Консоль

Вся работа осуществляется через Oracle Enterprise Manager Console. Консоль может быль загружена соединенной с сервером управления или отдельно от него (рис. 7.1). Если вам нужно использовать задания и события, управление данными и мастера резервного копирования, вы должны соединиться с сервером управления (Oracle Management Server). Если вам нужно лишь поработать с основными функциями базы — управлять схемой, экземпляром, хранением или безопасностью, можно загрузить одну консоль.

Типичная консоль показана на рис. 7.2. Консоль содержит меню, панели инструментов и группы инструментов. Панель инструментов расположена в верхней левой части окна, а линейки инструментов — в левой части окна. Когда выбирается объект в проводнике, интерфейс этого объекта появляется в правой части окна, в информационной панели.

Когда вы запускаете консоль в первый раз или если в среду вынесены какие-то изменения, информацию, показанную на рис. 7.2 нужно обновить. Это

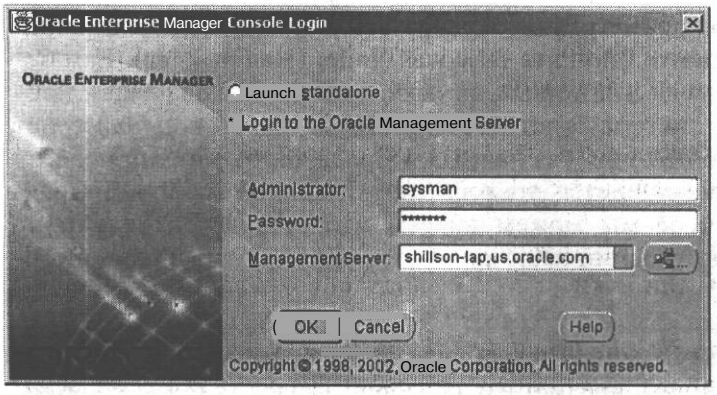


Рис. 7.1. *Загрузка Oracle Enterprise Manager Console*

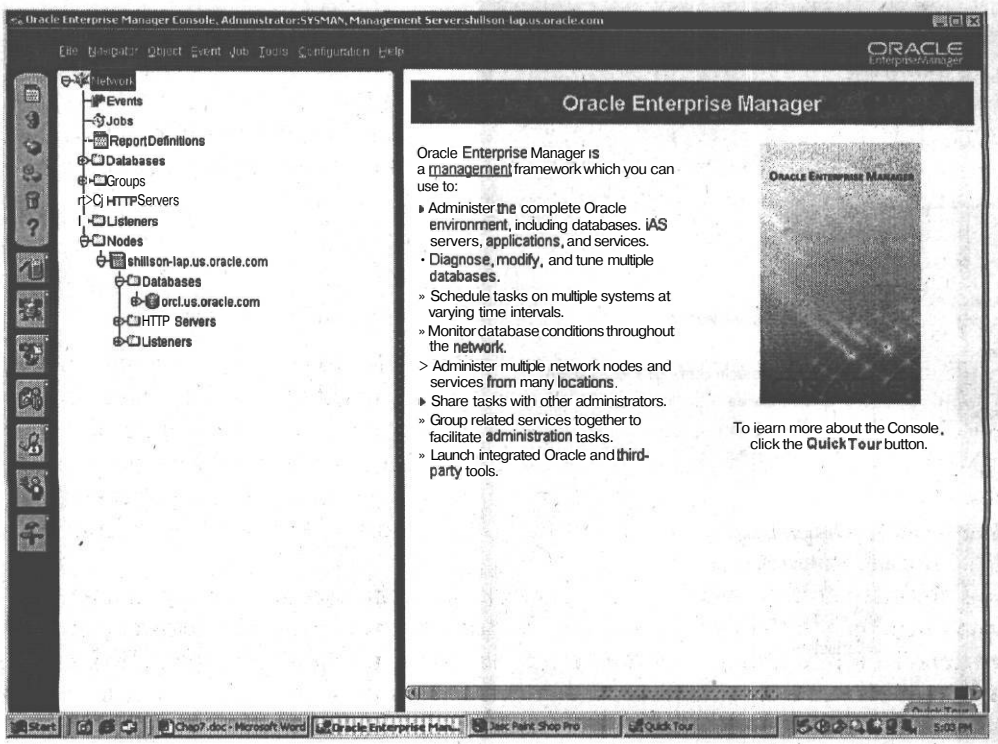


Рис. 7.2. *Oracle Enterprise Manager Console*

делается путем выбора в параметрах Проводника (Navigator) (самый верхний значок в панели инструментов) пункта Discover Nodes (Найти узлы). Вам будет предложено указать узлы для поиска, и, если на этих узлах запущен процесс Oracle Agent, ваша система будет исследована, и вашей консоли будет предоставлен список работающих баз данных и прослушивающих процессов (listeners). Так что вам не придется предоставлять никакой информации. В приводимом здесь примере мы можем видеть, что обнаружен экземпляр ORCL по умолчанию, содержащий наше хранилище EASYDW на узле shillson-lap.us.oracle.com.

Каталог Проводника, показанный на рис. 7.3 — это, возможно, наиболее важная часть экрана, поскольку именно отсюда запускается большинство операций. По умолчанию здесь есть пять предметных областей: Databases (базы данных), Groups (Группы), HTTP servers (HTTP-серверы), Listeners () и Nodes (Узлы).

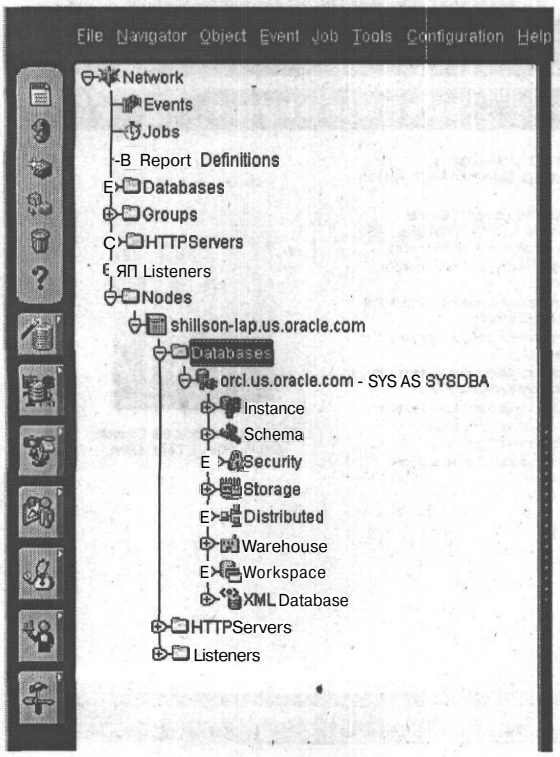


Рис. 7.3. *Enterprise Manager — окно Проводника*

Кроме того, в Oracle 9i Release 2 окно Проводника предоставляет возможность доступа к администрированию базы данных, событиям, созданию отчетов, заданиям и группам. Папка Databases в Проводнике позволяет администрировать экземпляры баз данных, схемы, регулировать безопасность и свойства хранения информации.

На рис. 7.3 окно Проводника показано с раскрытыми предметными областями. В разделе Databases перечислены все базы, независимо от узла, на котором они располагаются. Чтобы увидеть, в какой системе расположена база, нужно раскрыть область узлов (Nodes), и тогда мы увидим, какие базы расположены на каждом конкретном узле. В нашем примере это довольно легко, поскольку узел только один. Если множество баз работают на многих узлах, то вид будет более сложным. Именно здесь становится очень полезным свойство Groups (Группы).

Из окна Проводника можно выполнять многие задачи, связанные с базами данных. Если вы щелкнете по пункту database name, он раскроется и покажет список своих свойств. Когда вы выберете одно из свойств, в правой, информационной панели консоли появится его описание. Например, вы можете запускать и останавливать работу экземпляров баз, щелкнув по пункту Instance (Экземпляр). Вы можете создавать и удалять таблицы, индексы и представления, щелкнув по пункту Schema (Схема). Точно так же вы можете управлять безопасностью, хранением и другими объектами.

Для запуска консоли, соединенной с сервером управления, необходима администраторская учетная запись Enterprise Manager. Если консоль не соединена с сервером управления, это не требуется. В Enterprise Manager есть два типа администраторских учетных записей: обычные администраторы (regular administrator) и супер-администраторы (super administrator). Последние имеют дополнительные привилегии. Учетная запись администратора Enterprise Manager отличается от учетных записей пользователей на базах, которыми вам нужно управлять. На рис. 7.4 супер-администратор создает учетную запись с именем EASYDW для администратора хранилища EASYDW. Дается доступ к системам заданий (job) и событий (event) и, наконец, к базам данных, которыми будет управлять новый администратор. Заметьте, что эта учетная запись не супер-администраторская, поскольку эту опцию мы не указали. Мы можем указать имя пользователя и пароль для любой базы и узла, к которым администратору нужен доступ.

7.2.2. Зачем нужны группы?

Ранее мы говорили о проблемах, связанных с управлением несколькими базами данных, и одним из решений этих проблем является возможность увидеть гра-

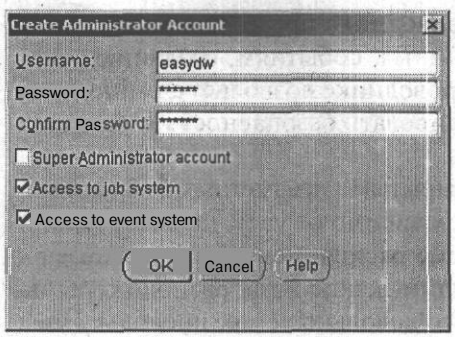


Рис. 7.4. *Создание администраторской учетной записи в Enterprise Manager*

фическое представление среды. Цель группы — дать вам возможность визуально представить среду. Это может быть не вся среда, а только часть ее.

Группа создается из меню Object (Объекты). Вы вводите имя и выбираете, какие объекты из имеющихся будут составлять группу. Вы также можете выбрать фоновый рисунок. Имя группы описывает среду, а фоновый рисунок нужен для того, чтобы накладывать на него объекты.

На рис. 7.5 мы можем видеть нашу группу, названную Easy Shopping Inc., представляющую наше хранилище. Фоном служит карта области Соединенных Штатов, в которой находится хранилище. Согласно рис. 7.6, мы добавили базу данных ORCL и машину shillson_lap.us.oracle.com. Они были помещены на карту, в место, соответствующее их физическому расположению. Конечно, вам решать, куда помещать их.

Вы можете подумать: «Да, все это выглядит красиво, но что здесь есть помимо графического представления системы?». В следующем разделе мы будем описывать события, а когда события соединяются с изображением группы, получается очень мощная среда управления.

Вы можете создать столько групп, сколько вам нужно, и они будут отображаться в каталоге Проводника.

7.2.3. Использование событий

Одной из проблем для каждого, кто управляет базой данных является получение информации о возникновении определенного события. Например, предположим, что экземпляр базы данных неожиданно отключился. Разве не хотели бы вы узнать об этом немедленно, не дожидаясь звонков от пользователей, жалующихся, что доступа к системе больше нет?

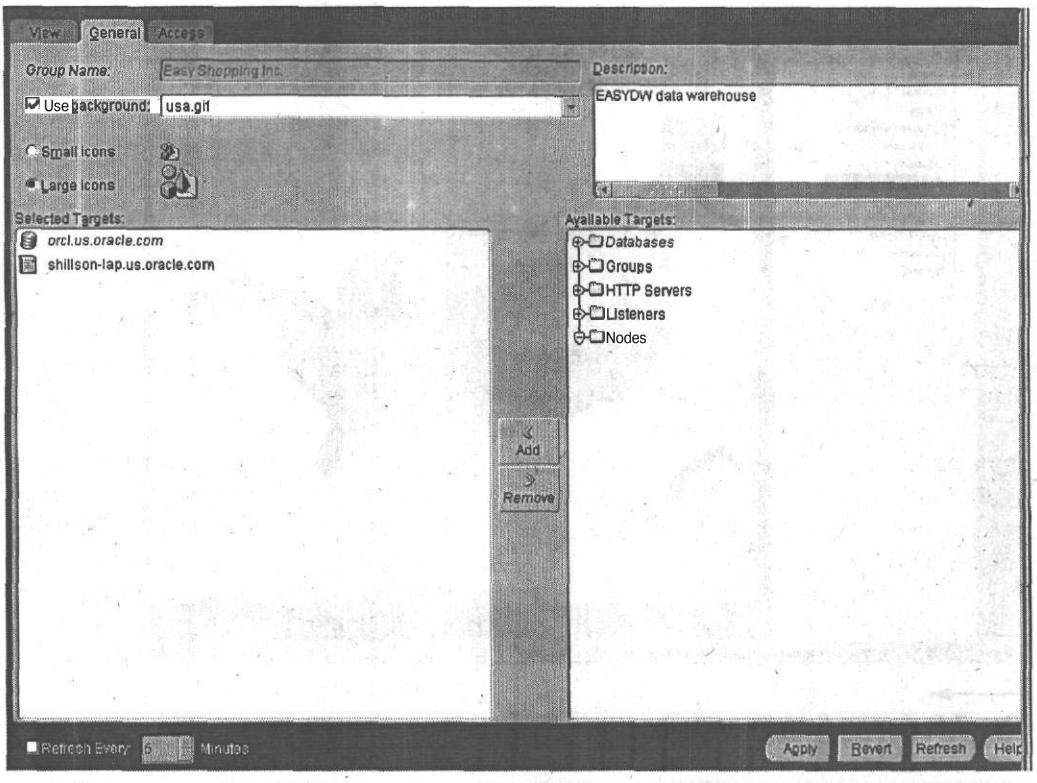


Рис. 7.5. *Создание группы*

Из консоли вы можете определять события, позволяющие следить за хранилищем и получать уведомления в определенных случаях. Событие можно создать, щелкнув мышью по пункту Event (Событие) в панели инструментов, а затем — по пункту Create Event (Создать событие). Появится окно, показанное на рис. 7.7.

Как видно на рис. 7.7, мы дали нашему событию название Warehouse Status (Статус хранилища), *а* также добавили комментарий. Данное конкретное событие является событием базы данных, но мы можем выбирать из раскрывающегося списка в разделе Target Type (Тип цели) и другие типы, например события узлов. Затем мы выбираем базу ORCL из списка справа и, используя кнопки со стрелками, переносим ее в левый список. При необходимости можно также менять частоту отслеживания. Для завершения определения события нужно перейти к другой закладке в верхней части окна.

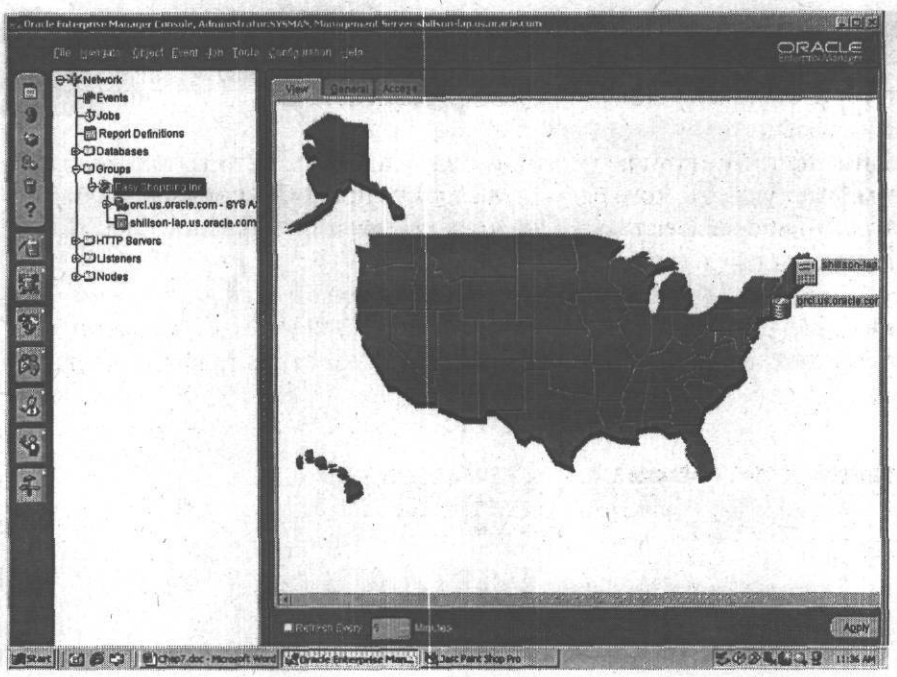


Рис. 7.6. *Группа Easy Shopping Inc*

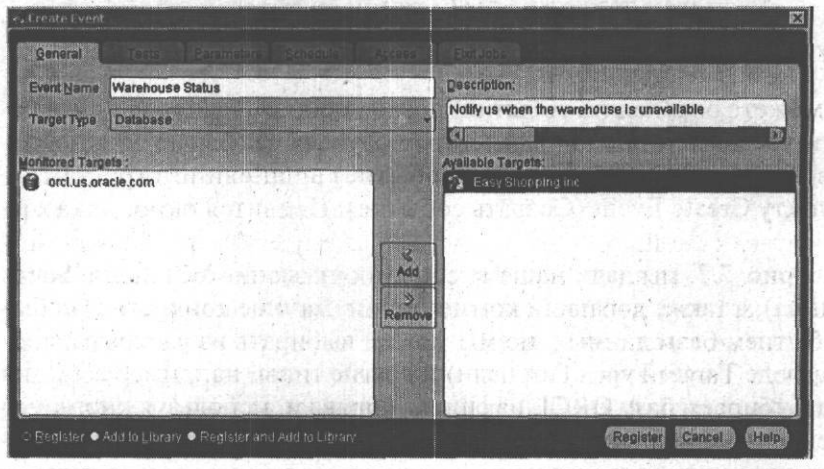


Рис. 7.7. *Создание события*

При переходе к закладке Tests (Тесты) появляется окно, показанное на рис. 7.8, в котором можно выбрать отслеживаемые события. В данном примере мы выбрали событие DatabaseUpDown, чтобы получать сообщение при включении и отключении базы.

Это событие не требует от нас указания каких-либо параметров, так что следующим шагом будет указать, кого следует уведомить при возникновении события. Уведомлять кого-либо необязательно. Вы можете определить событие, и, если база данных попала в какую-то группу, статус события будет регистрироваться в ней.

Например, на рис. 7.6 напротив базы ORLC есть цветной флажок (который, к сожалению, трудно показать в черно-белой книге). Флажок становится зеленым, когда база открыта для использования. Если база закрыта, флажок оказывается красным.

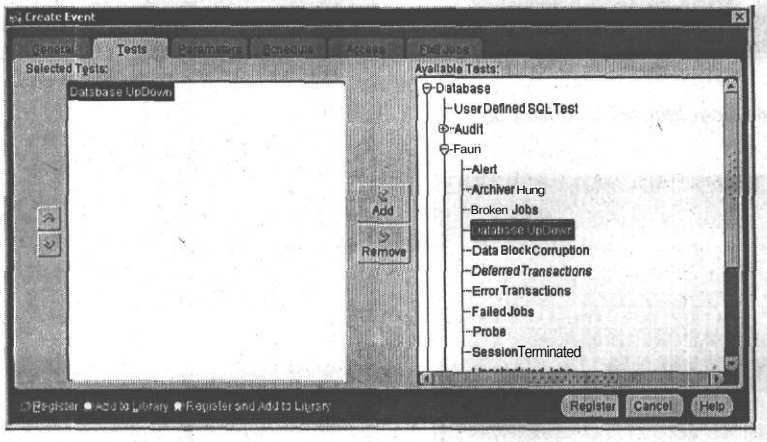


Рис. 7.8. *Выбор отслеживаемых событий*

Это свойство является одной из мощных визуальных возможностей групп, поскольку вы можете взглянуть на картинку и сразу увидеть состояние своей системы. На рис. 7.9 мы указали, что уведомления о возникновении этого события должны получать пользователи SYSMAN и EASYDW. Административный персонал может получать сообщения о событиях по пейджеру или по электронной почте. Обе эти возможности можно сконфигурировать в консоли.

Если выбрать из панели инструментов пункт Configuration (Конфигурирование), а затем перейти к закладке Schedule (Расписание заданий), появится окно, показанное на рис. 7.10.

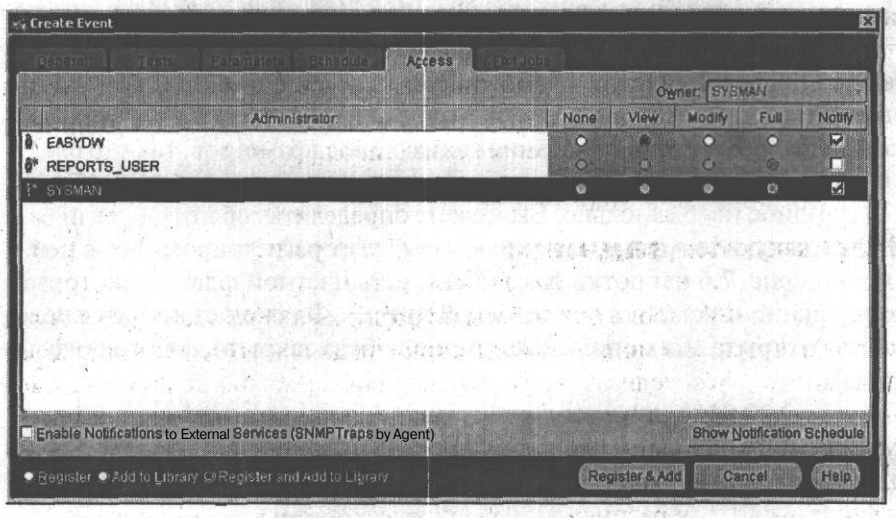


Рис. 7.9. *Уведомление пользователей о событиях*

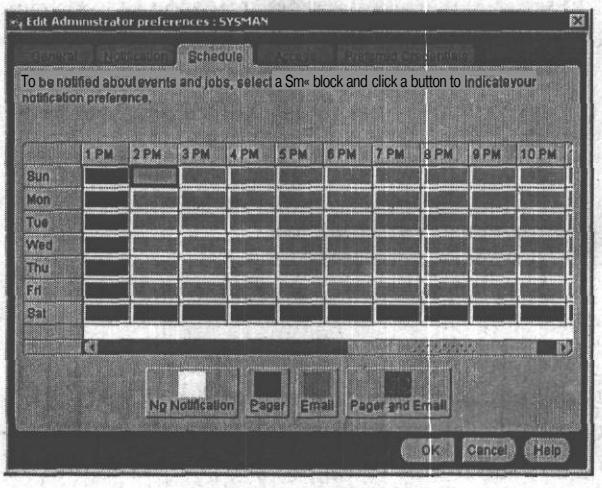


Рис. 7.10. *Расписание уведомлений о событиях*

Опять трудно показать это в черно-белой книге, но здесь вы можете определить, как следует давать уведомления в каждый час дня. Если вы знаете, что вы всегда обедаете в 13:00, то в этот период, а также по субботам, вы указываете, что при

появлении события нужно посылать уведомление на пейджер. А на протяжении рабочего дня предпочтительным будет использование электронной почты.

Здесь мы лишь слегка очертили возможности для работы с событиями. Но мы надеемся, что это даст вам толчок к дальнейшим исследованиям событий и реализации их в своих процедурах управления хранилищем.

7.2.4. Расписание заданий

Хранилища данных требуют большой работы по обслуживанию, поскольку данные постоянно загружаются. Вы можете писать скрипты для вьшолнения этих задач и все время помнить о необходимости их запуска, или же можно использовать Job Scheduler — средство OEM для автоматического запуска заданий в указанное время.

Инструмент работы с заданиями дает возможность определить задания, необходимые для обслуживания хранилища. Эти задания можно поместить в библиотеку и составить в консоли расписание автоматического их вьшолнения.

Чтобы создать задание, выберите в панели инструментов пункт Job (Задание), а потом — Create Job (Создать задание). Появится окно, показанное на рис. 7.11, в котором вы можете подробно описать задание. В данном примере мы создаем задание, которое автоматически отключает хранилище в семь часов вечера. Сначала мы даем заданию название и указываем базу данных, к которой оно должно применяться. Хотя данный пример является заданием базы данных, возможны и другие типы, перечисленные в раскрывающемся списке типов.

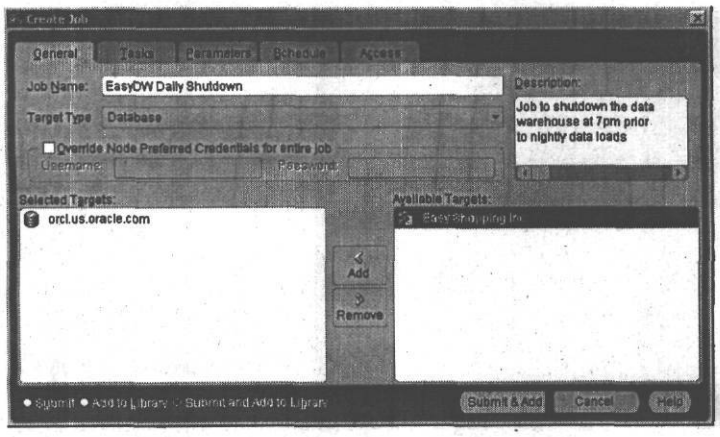


Рис. 7.11. *Создание задания*

Все задания можно сохранять в библиотеке заданий для повторного использования. Поэтому, если вы хотите сохранить задание, сейчас самое время нажать кнопку Submit & Add Job (Утвердить и добавить задание) для записи его в библиотеку. Если вы сделаете это сейчас, вы просто задействуете данную возможность и измените надпись на кнопке Submit на Submit & Add. Пока вы не нажмете кнопку Submit & Add, задание не будет поставлено в очередь на выполнение.

Чтобы перейти к следующему окну, нажмите на закладку Tasks (Действия), и выберите выполняемое действие. Действий, как показано на рис. 7.12, доступно много. В данном примере мы выбрали задание Shutdown Database (Выключить базу данных):

Когда действие выбрано, перейдите к закладке Parameters (Параметры), и вы увидите, нужны ли для данного задания какие-нибудь параметры. На рис. 7.13 видно, что для задания по закрытию базы данных требуется указать, как нужно выполнять закрытие. Мы выбираем режим Immediate (Немедленно).

Также отметьте, что мы имеем здесь возможность указать для выполнения данной операции пользовательскье имя и пароль, отличные от используемых по умолчанию (которые мы установили ранее). Поскольку закрытие базы требует имени пользователя с большими привилегиями, мы берем для выполнения задания имя system.

Определение задания практически завершено. Теперь нужно перейти к закладке Schedule (Расписание) и указать, когда нужно выполнять задание. Как показано на рис. 7.14, мы указали, чтобы выполнение задания запускалось ежедневно, в 7 часов вечера, с понедельника по пятницу, начиная с 3 июля.

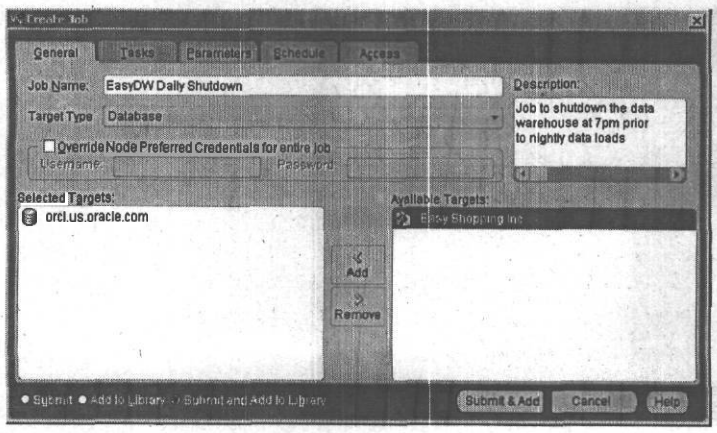


Рис. 7.12. *Выбор действия для задания*

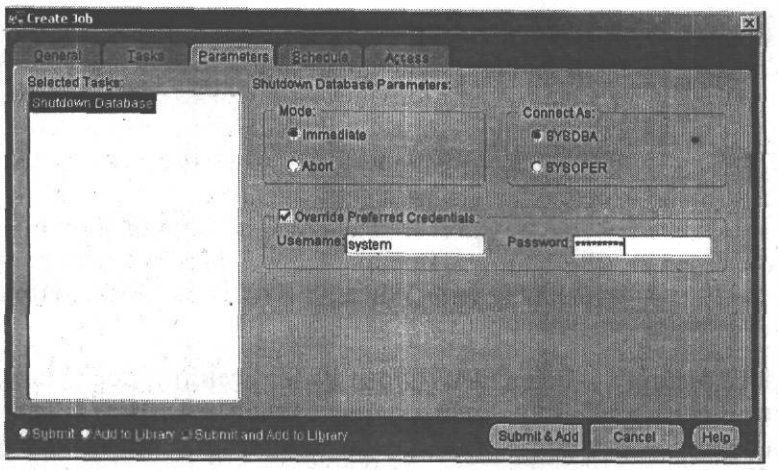


Рис. 7.13. *Параметры задания*

И наконец, перейдите к закладке Access (Доступ) и укажите (если это необходимо), кого следует уведомить о запуске задания. После этого, нажав на кнопку Submit (Утвердить), поставьте задание в очередь на выполнение.

Чтобы увидеть задания, стоящие в очереди, выберите в каталоге Проводника пункт Jobs (Задания). Как только задание было помещено в очередь, вы можете следить за его выполнением, как это показано на рис. 7.15. В примере

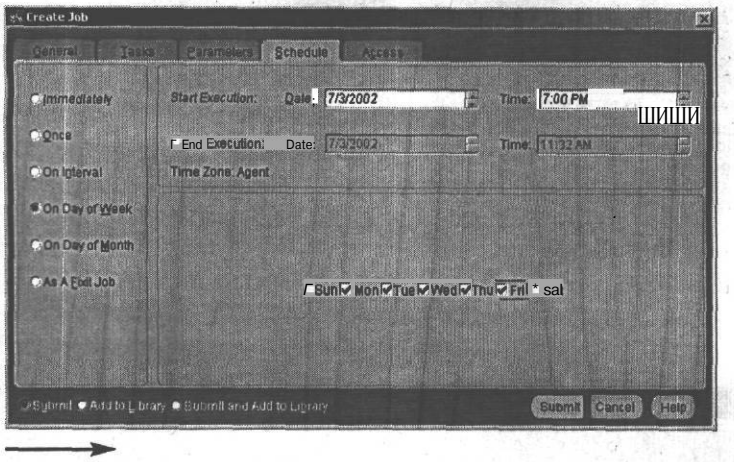


Рис. 7.14. *Расписание выполнения задания*

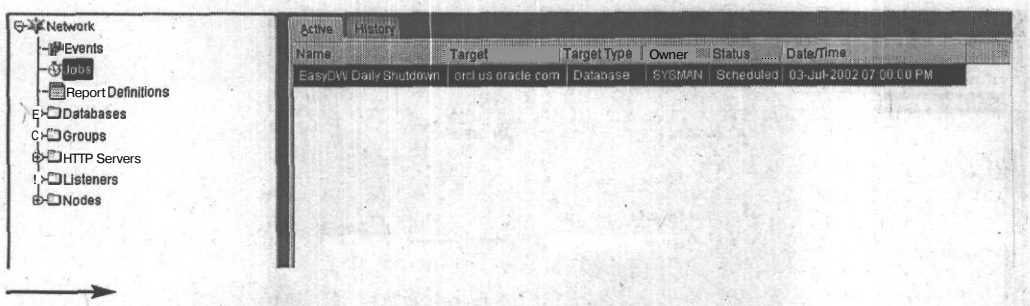


Рис. 7.15. *Наблюдение за выполнением задания*

на этом рисунке статус задания — Scheduled (Стоит в расписании), поскольку задание находится в режиме ожидания. Если задание будет выполняться, статус будет иметь вид Active (Активно). Когда задание завершено, оно исчезает из активного окна. Чтобы увидеть, прошло оно успешно или нет, перейдите к закладке History (Предыстория).

Если сделать двойной щелчок мышью по заданию, появится окно, показанное на рис. 7.16, в котором можно увидеть подробные сведения о задании. На рис. 7.16 видно, что задание было утверждено и запустится в следующий раз 3 июля в 7 часов вечера. .

Чтобы увидеть результат выполнения задания, выберите задание и нажмите кнопку Show Output (Показать результат). На рис. 7.17 показано, что задание

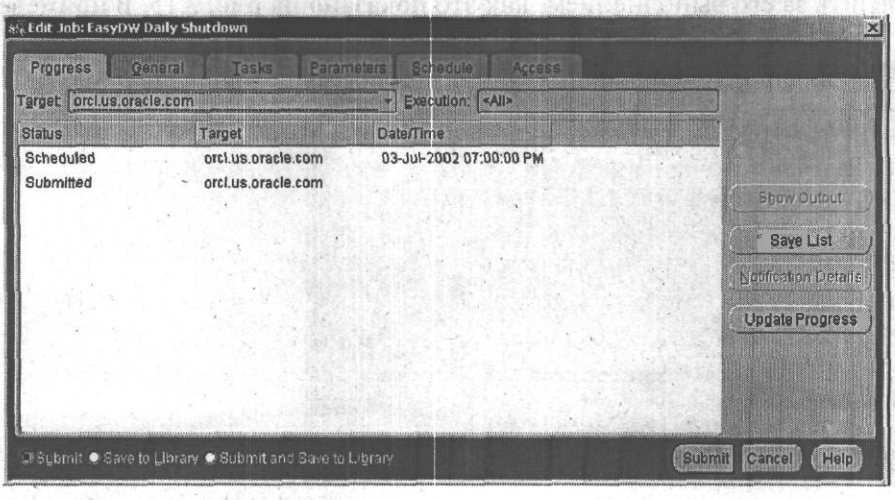


Рис. 7.16. *Информация о задании*

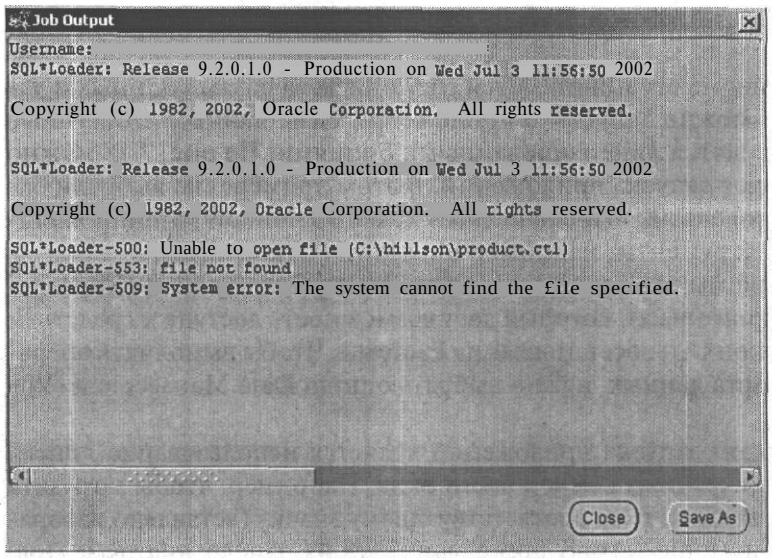


Рис. 7.17. *Результат выполнения задания*

по загрузке данных не было выполнено, поскольку в указанном месте не был найден управляющий файл.

В примерах этого раздела показаны события и задания, относящиеся к базе данных. Кроме этого, задания могут представлять собой задачи операционной системы, которые нужно выполнить для управления хранилищем. Например, проверка того, что файлы перед загрузкой в хранилище скопированы из одной системы в другую.

Количество заданий, которые можно выполнять, очень велико и как только они попадают в библиотеку, мы получаем исчерпывающий набор задач, которые хранятся в безопасности и не теряются.

Библиотеку заданий можно найти, если щелкнуть мышью по пункту Job (Задание) в панели инструментов, а потом выбрать пункт Job Library (Библиотека заданий).

Совет. Не забывайте делать резервную копию базы данных, в которой храниться репозиторий Enterprise Manager, поскольку если в этой базе произойдет авария, вы потеряете все свои задания.

7.2.5. Запуск приложений

Возможности консоли не ограничиваются слежением за базами данных и узлами. Из нее также можно запускать приложения базы данных. Доступные приложения определяются лицензированными опциями. На рис. 7.18 можно видеть один из методов запуска приложений. В этом примере мы выбрали базу данных в окне Проводника. После щелчка правой кнопкой мыши появляются имеющиеся инструменты, например Shutdown (Отключить).

В данном примере мы выбрали инструмент Backup Management (Управление резервным копированием), который дает возможность доступа к средствам создания резервных копий и восстановления системы. Чтобы выполнять операции экспорта и импорта данных, нужно выбрать опцию Data Management (Управление данными).

Еще одним методом запуска приложений является использование линеек инструментов, расположенных в левой части окна. Например, чтобы загрузить SQL*Plus Worksheet, щелкните по соответствующему значку (вставлено изображение значка). Приложение можно также запускать из панели инструментов, раздела Tools (Инструменты).

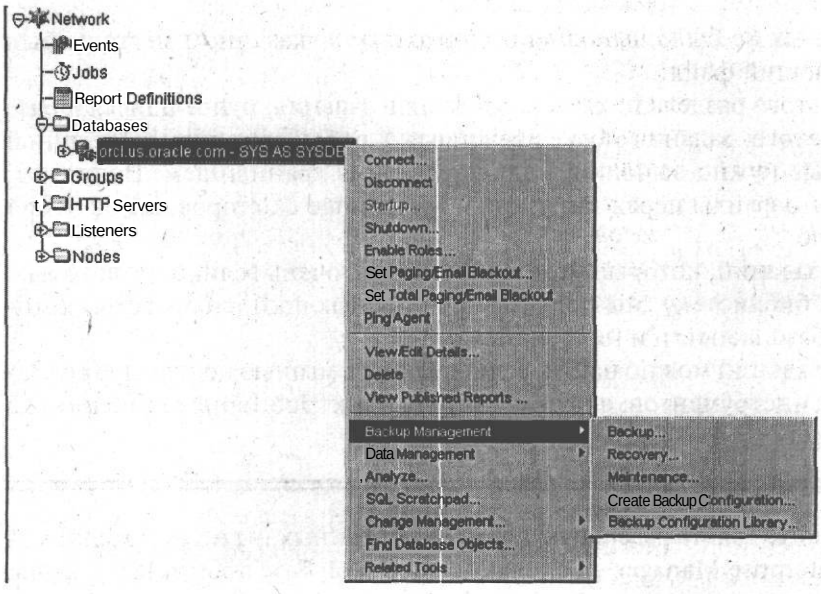


Рис. **7.18.** *Запуск приложений из консоли*

Надеемся, что вы начинаете ценить возможности управления хранилищем с помощью Oracle Enterprise Manager. Далее мы рассмотрим то, как с помощью OEM можно выполнять различные задачи, связанные с управлением.

7.3. Резервное копирование и восстановление системы

Одной из наиболее важных задач по управлению любой базой данных является резервное копирование данных. Это может показаться очевидным, но вы будете удивлены количеством компаний, которые рискуют своим бизнесом, не выполняя резервное копирование достаточно часто или вообще не заботясь об этом.

Резервное копирование хранилища несколько отличается от резервного копирования рабочей системы. Поскольку в хранилища обычно загружаются большие массивы данных, обычно ночью, резервное копирование нужно планировать на это же время. Кроме того, есть дополнительная сложность в том, что мы должны решить, как делать резервную копию.

Почему это является проблемой? Просто, как правило, резервная копия делается для всей базы, но для хранилища такое может оказаться невозможным, особенно если оно имеет очень большие размеры (несколько терабайт). Следовательно, более практичным будет инкрементное резервное копирование и копирование одного или более табличных пространств. Обязательным является аккуратное управление носителями резервных копий, иначе вы можете оказаться не в состоянии восстановить хранилище.

До появления Oracle 8i резервное копирование обычно осуществлялось с помощью утилиты резервного копирования рабочей системы, копирующей все важные файлы. Теперь вы можете использовать Recovery Manager (RMAN). В этом разделе мы обсудим применение этого инструмента.

7.3.1. Резервное копирование

В данном разделе мы обсудим различные методы, которые вы можете использовать для резервного копирования хранилища данных. Однако важно осознавать, что если хранилище очень велико, создание полной резервной копии может оказаться почти невозможным из-за временных и прочих ограничений. Мы обсудим следующие типы резервного копирования:

- создание полных копий;
- инкрементные резервные копии;
- резервные копии табличных пространств.

Создание конфигурации резервного копирования

Выберите базу данных в окне Проводника, щелкните по базе правой кнопкой мыши, выберите Backup Management (Управление резервным копированием), и вы запустите Oracle 9i RMAN Backup Manager. Предлагается список инструментов, в который входит как резервное копирование, так и восстановление. Однако мы начнем с пункта Create Backup Configuration (Создать конфигурацию резервного копирования).

Конфигурация резервного копирования (backup configuration) — это набор заданных по умолчанию параметров, используемых при резервном копировании. В ней определяется, где будет храниться резервная копия, каким будет носитель информации (магнитная лента или диск) и нужно ли вам хранить описание резервной копии в каталоге системы восстановления. Каталог системы восстановления (recovery) описывается позже, в разделе 7.3.2.

Вы можете создавать столько конфигураций резервного копирования, сколько нужно для вашей системы. Целью данного этапа является указание места, в которое будет помещена резервная копия, и имени, которое она получит. На рис. 7.19 показан первый этап работы мастера, в котором мы даем имя конфигурации резервного копирования.

В этом окне вы можете переходить к разным закладкам и указывать различные параметры. Посещение всех закладок не является обязательным, но нужно будет посетить закладку Channels (Каналы), показанную на рис. 7.19. Именно здесь указывается положение и имя файла резервной копии.

И опять же, вы можете создавать столько каналов, сколько вам нужно. Считайте канал тем местом, где вы указываете все физические атрибуты, например, будет ли копия храниться на магнитной ленте или диске.

Когда дело доходит до определения имени файла, есть различные возможности создания уникального имени. В примере на рис. 7.20 мы указали параметр %s, определяющий номер набора копий и параметр %t, определяющий отметку времени создания набора копий.

Еще один полезный параметр — %u, который создает имя, состоящее из 8 символов, включающих в себя номер и время создания резервной копии. Возможность обзора каталогов отсутствует, следовательно, убедитесь в правильности написания.

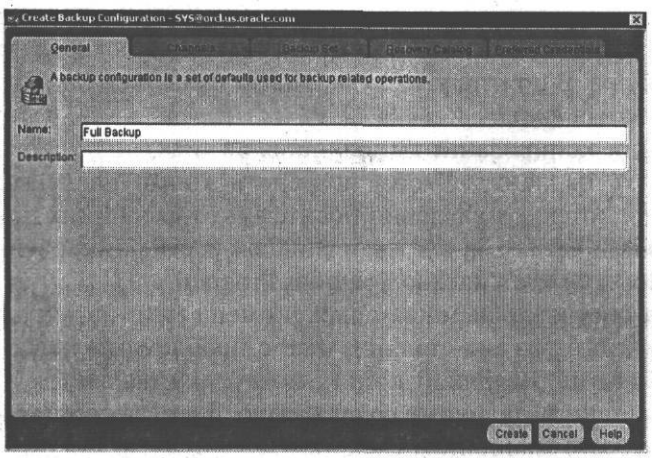


Рис. 7.19. *Создание конфигурации резервного копирования*

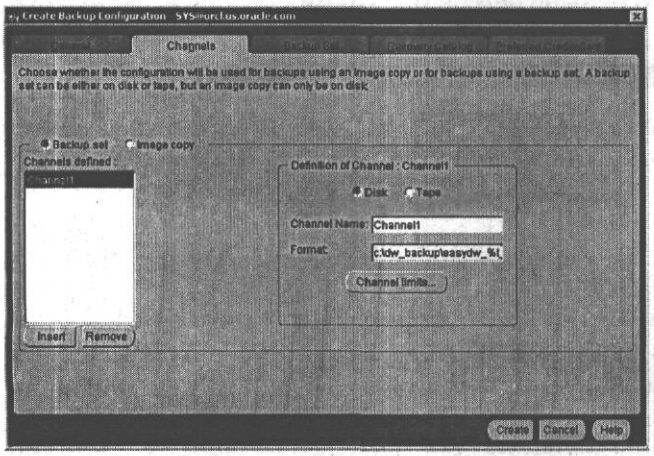


Рис. 7.20. *Выбор каналов в конфигурации резервного копирования*

Совет. Название каталога для хранения резервной копии должно заканчиваться символом \ или / (в зависимости от платформы).

Полный список всех опций можно найти в руководстве Oracle 9i Backup and Recovery Manual. Если вы создаете резервные копии часто, важно осторожно подходить к выбору имен файлов. В противном случае может оказаться так, что вы не сможете распознать нужный файл.

В данном примере резервное копирование выполняется на диск, но в хранилищах приходится копировать на ленту. Чтобы сохранять копии на ленте, RMAN требует установки библиотеки управления носителем от сторонних поставщиков, таких, как Legato или Veritas. За текущим списком сертифицированных поставщиков обращайтесь к Oracle's Backup Solutions Program.

Крайне важно обеспечить соответствие маркировок, помещаемых на ленты, их содержимому. Слишком много раз мы сталкивались с проблемами невозможности восстановления базы из резервной копии, поскольку наклейки на лентах не соответствовали их действительному содержанию. Такая неаккуратность может означать потерю всего хранилища.

Когда бы вы ни проводили полное резервное копирование системы, не выбрасывайте предыдущий набор файлов, поскольку если возникнут проблемы с имеющейся копией, вы потеряете все хранилище. Поэтому перед утилизацией лент записывайте как можно больше резервных копий.

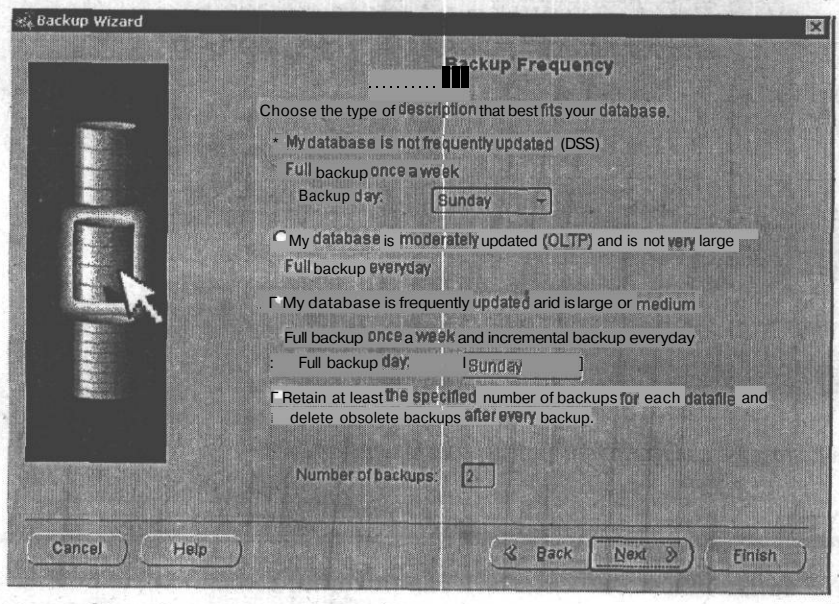


Рис. 7.21. *Частота резервного копирования*

Существует еще несколько закладок, которые можно посетить. Если пользователь данной базы по умолчанию не может войти в систему как SYSDBA, нужно перейти к закладке Preferred Credentials (Привилегированные полномочия) и указать другое имя пользователя и пароль.

Полное резервное копирование

Частоту полных резервных копирований определяет размер хранилища. Даже если хранилище очень велико, нужно периодически выполнять копирования.

Одним из средств, предлагаемых в Oracle Enterprise Manager, является Backup Manager, включающий в себя мастер резервного копирования Backup Wizard. В этом мастере вы можете выбрать уже готовую стратегию резервного копирования или настроить свою собственную.

Готовая стратегия резервного копирования

Первый из представленных здесь примеров показывает, как установить готовую стратегию резервного копирования. Первый вопрос, который задает мастер: как часто нужно выполнять копирование и когда его нужно запускать.

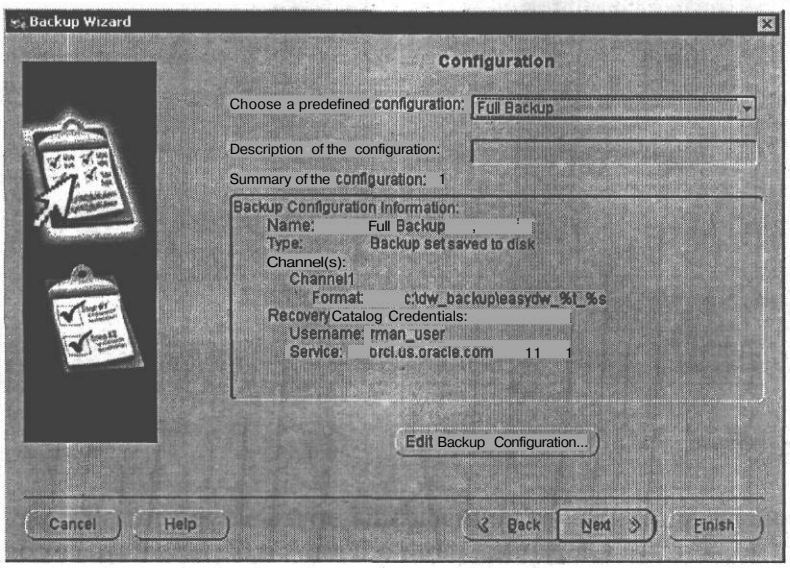


Рис. 7.22. *Выбор конфигурации резервного копирования*

Тем читателям, которые не знакомы с процедурой создания резервных копий, мастер даст совет о том, когда следует выполнять копирование, основываясь на типах прикладных задач, в которых используется данная база. Можно видеть, что для хранилища предлагается выполнять резервное копирование один раз в неделю, в воскресенье, которое, как предполагается, является спокойным днем, подходящим для выполнения этой задачи. Конечно, вы можете изменить и частоту, и день.

В следующем своем окне мастер спрашивает, в какое время нужно выполнять копирование. Обзор возможных альтернатив показан на рис. 7.22. Здесь вы можете указать, какая конфигурация резервного копирования должна использоваться. Конфигурация была создана нами в предыдущем разделе и используется для определения того, где сохраняется резервная копия и записывается ли она на ленту или на диск.

Теперь нам нужно выбрать копируемую базу данных. В примере на рис. 7.23 мы выбрали нашу базу-хранилище EASYDW.

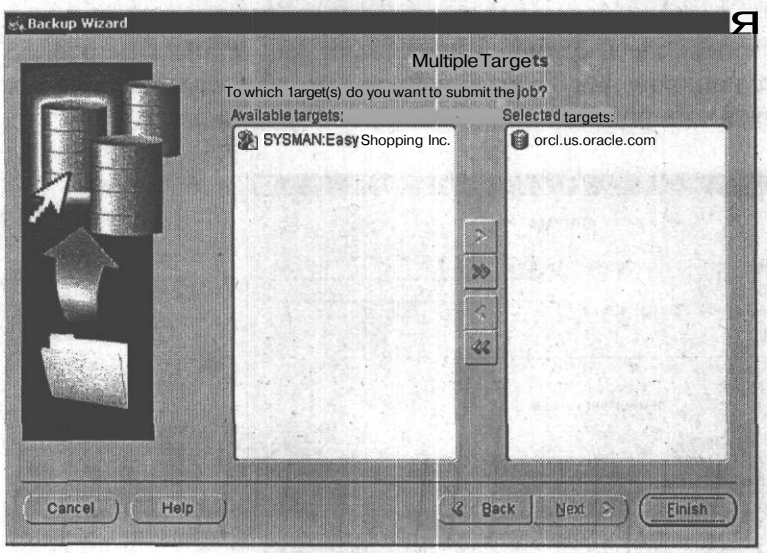


Рис. 7.23. *Выбор базы данных*

Формирование процедуры резервного копирования почти завершено. На рис. 7.24 мы можем просмотреть все выбранные нами опции. Чтобы изменить любое значение, нажмите кнопку Back (Назад), вернитесь в соответствующее окно и внесите нужные изменения.

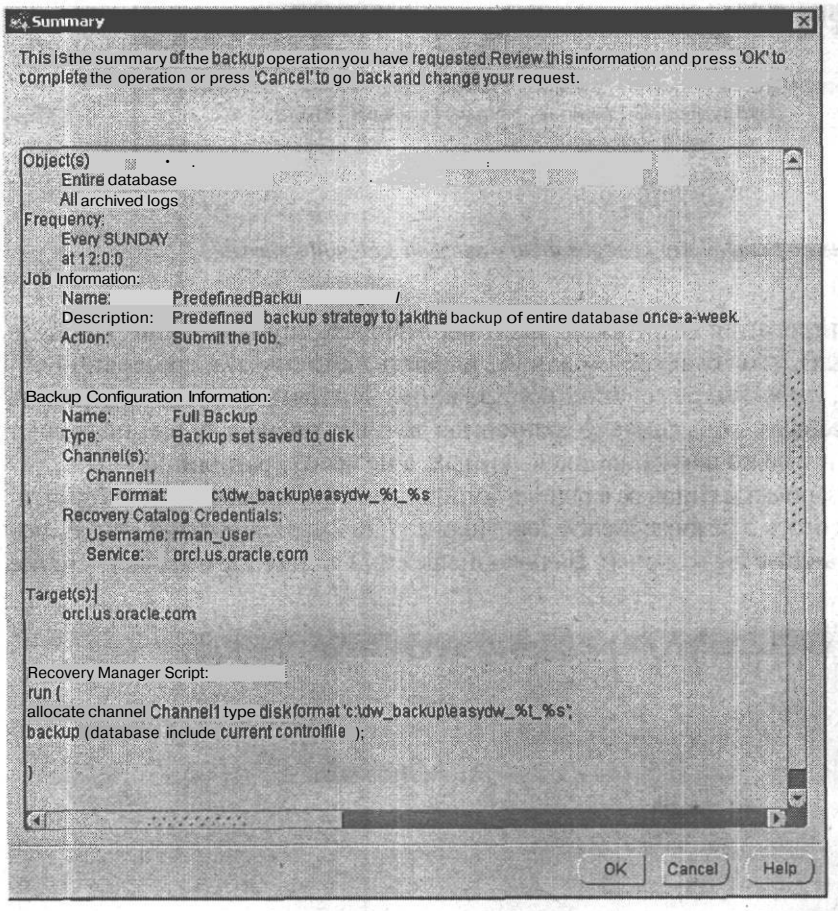


Рис. 7.24. *Обзор процедуры резервного копирования*

Когда определение процедуры резервного копирования завершено, оно помещается в очередь заданий консоли (как показано на рис. 7.25) до того момента, пока не наступит время ее выполнения.

Пользовательская стратегия резервного копирования

Альтернативой готовой стратегии резервного копирования является пользовательская стратегия. Первый вопрос, который задает мастер (рис. 7.26): какой тип

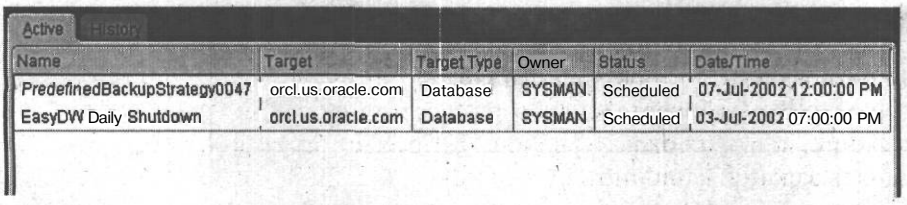


Рис. 7.25. *Задание по резервному копированию в очереди заданий консоли*

резервного копирования вам нужен. Если вы выберете вариант Entire Database (Вся база данных), у вас будет возможность выбрать либо создание резервной копии всей базы, либо инкрементное копирование. Инкрементное копирование может заинтересовать пользователя хранилища данных, поскольку оно позволяет создавать копии только изменившихся данных, а не всего хранилища.

Доступные варианты типов резервного копирования определяются тем, будет ли база данных работать в режиме archive log. На рис. 7.26 имеется больше опций, поскольку режим archive log включен. Включение режима archive log означает, что все

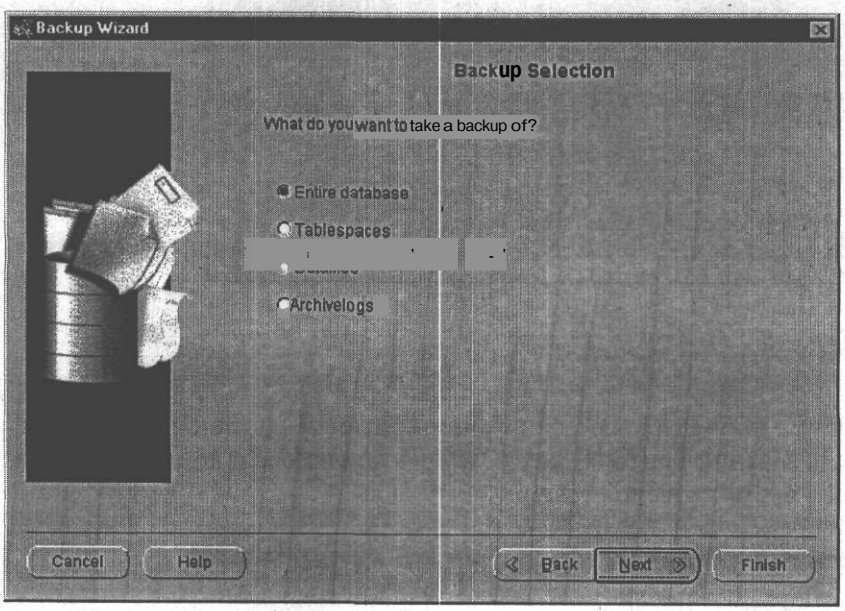


Рис. 7.26. *Типы пользовательского резервного копирования*

изменения, вносимые в базу, будут скопированы в архивный журнал до того, как будет задействован протокол повторного выполнения (redo log).

Этапы пользовательского резервного копирования очень похожи на те, что мы видели ранее. Вы выбираете тип резервного копирования, канал, время запуска и название, как это показано на рис. 7.27. Здесь вы будете иметь возможность включить задание в библиотеку заданий.

Данное задание будет запускаться автоматически, а также будет доступно для использования в будущем, поскольку сохраняется в библиотеке заданий.

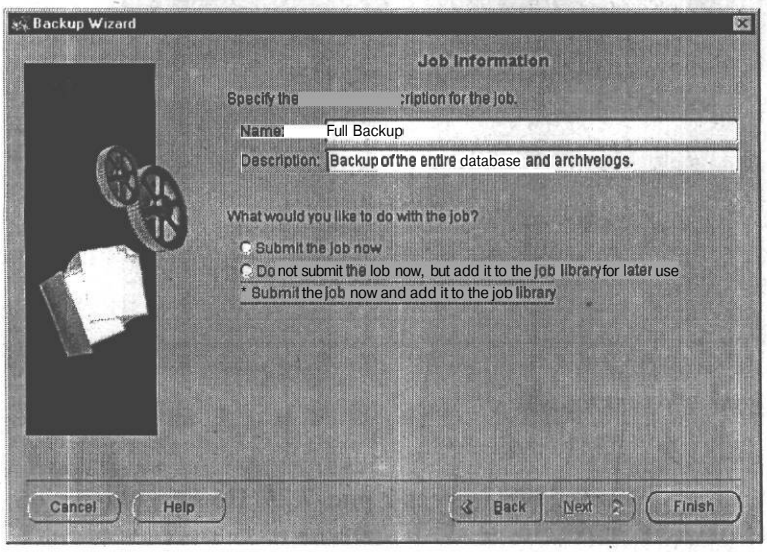


Рис. 7.27. *Включение задания по резервному копированию в библиотеку*

Инкрементное резервное *копирование*

Ранее мы уже говорили, что этот тип резервного копирования очень полезен в хранилищах, поскольку он позволяет делать копии только изменившихся частей базы данных. Окна, которые используются для определения инкрементного резервного копирования, очень похожи на те, которые применяются для полного копирования, за исключением окна, показанного на рис. 7.28.

В этом окне вы должны указать уровень (level) инкрементного копирования. Крайне важно выбрать верный уровень. Иначе вы по ошибке можете выполнить полное копирование.

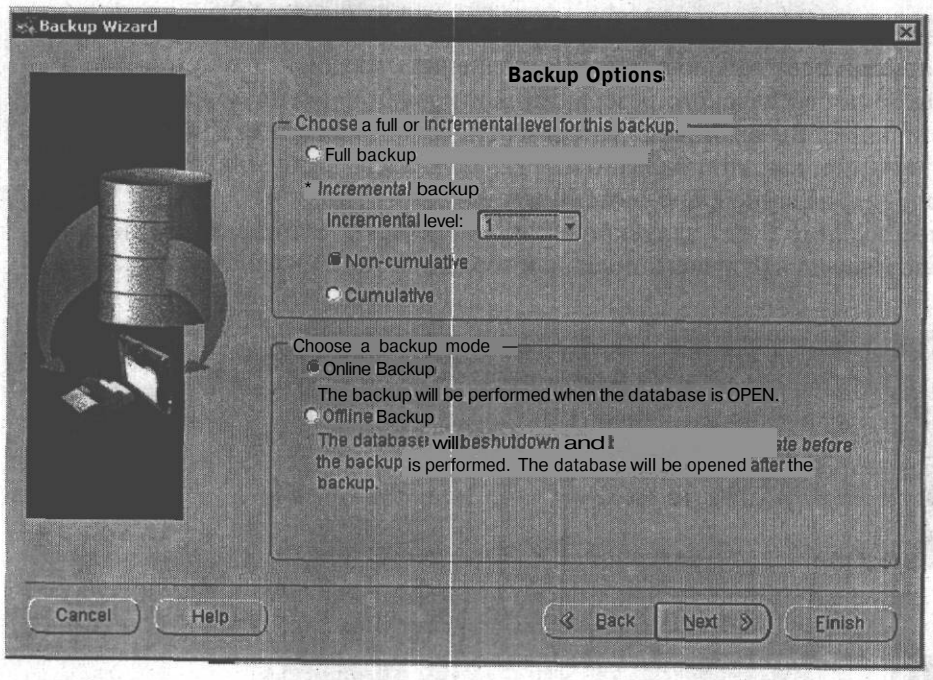


Рис. 7.28. *Уровень инкрементного копирования*

Чтобы понять, как указывать уровень, обратитесь к рис. 7.29. Полное резервное копирование выполняется по воскресеньям. Этот уровень называется уровень О (level 0). В конце рабочего дня в понедельник выполняется инкрементное копирование уровня 2, сохраняющее все изменения, внесенные с воскресенья. Во вторник выполняется еще одно инкрементное копирование уровня 2, содержащее все изменения, внесенные с понедельника. В среду выполняется копирование уровня 1. Поскольку 1 больше, чем 0, но меньше, чем 2, в файл резервной копии попадут все изменения с воскресенья. В четверг проводится копирование уровня 2, так что самой последней резервной копией с уровнем меньше или равным 2 будет копия уровня 1 от среды, и следовательно, эта копия будет содержать все изменения, внесенные со среды.

Такой подход может на первый взгляд показаться сложным, но при использовании автоматического восстановления Recovery Manager сам определит, какие полные и инкрементные резервные копии необходимы для восстановления базы данных.

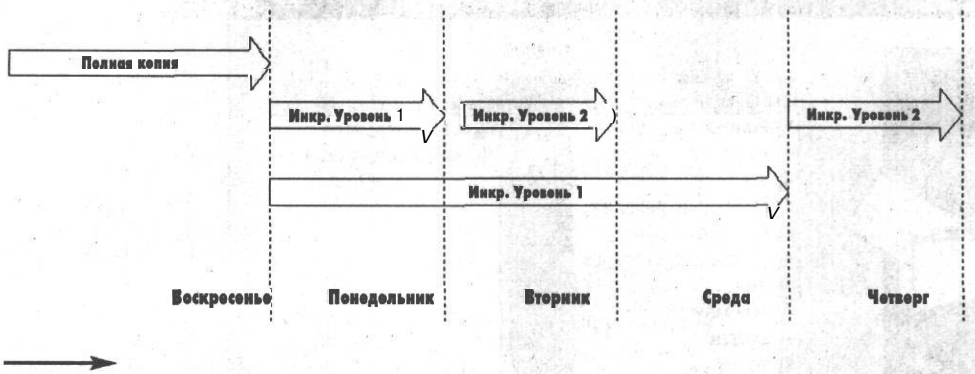


Рис. 7.29. *Выбор инкрементного копирования*

Резервное копирование табличного пространства

Еще один метод резервного копирования базы данных — это копирование табличных пространств (tablespace backup). Этот метод дает возможность сделать резервную копию конкретного табличного пространства, что для хранилища может быть очень удобной стратегией.

На рис. 7.30 мы видим одно из окон мастера Backup Wizard, в котором нужно указать, какие табличные пространства нам нужно копировать. В этом примере мы выбрали два табличных пространства, содержащие данные за январь.

Это хороший пример того, что при первоначальном создании базы следует принимать во внимание управление. Если мы можем сгруппировать обновления базы по датам и если мы знаем, что после определенного момента времени изменения в данное табличное пространство не вносятся, то, если мы сделаем табличное пространство доступным только для чтения и создадим его резервную копию, мы зафиксируем данные в этом табличном пространстве. Поскольку данные никогда не будут меняться, нам и не нужно будет делать резервные копии снова.

Сделать табличное пространство доступным только для чтения очень просто. Вы можете либо ввести команду SQL, показанную в окне текста SQL на рис. 7.31, либо выбрать в каталоге Навигатора пункт Storage (Хранение), указать табличное пространство и поставить галочку напротив пункта Read Only (Только для чтения) в нижней части окна и, наконец, нажать кнопку Apply (Применить). Теперь табличное пространство будет доступно только для чтения. Вы можете в любой момент снова сделать его доступным для чтения/записи.

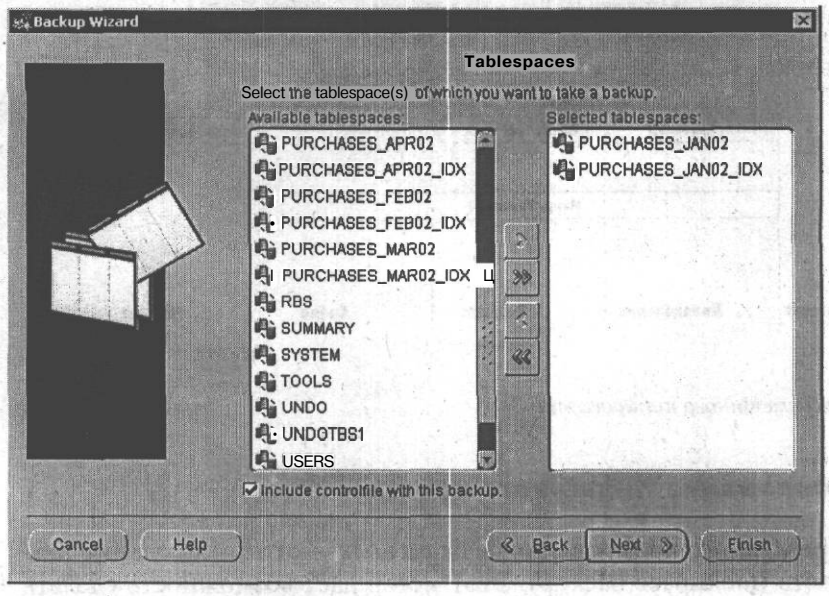


Рис. 7.30. *Выбор табличного пространства для резервного копирования*

Конечно, имеет смысл периодически делать полные резервные копии, просто на случай, если возникнут проблемы с имеющимися файлами копий. Делать только копирование табличных пространств вполне можно, но при этом остается вероятность того, что кто-то случайно перезапишет ленту, содержащую данные за какой-нибудь месяц.

Размеры файлов резервных копий

Когда база данных создается, разработчики всегда знают о ее размерах. Однако эта информация, по-видимому, имеет свойство теряться, когда люди начинают думать о резервном копировании базы. Все склонны считать так: «Здесь 300 Гб, так что для резервного копирования понадобиться *x* минут». При этом люди забывают о том, что файл резервной копии занимает *y* гигабайт места.

В показанных здесь примерах мы использовали для хранения наших резервных копий диски. В реальном хранилище они будут храниться на ленте. Так что не забывайте о том, сколько вам потребуется лент для реализации вашей стратегии резервного копирования. Известно, что иногда ленты просто заканчивались! Наша база данных компании Easy Shopping имеет небольшие

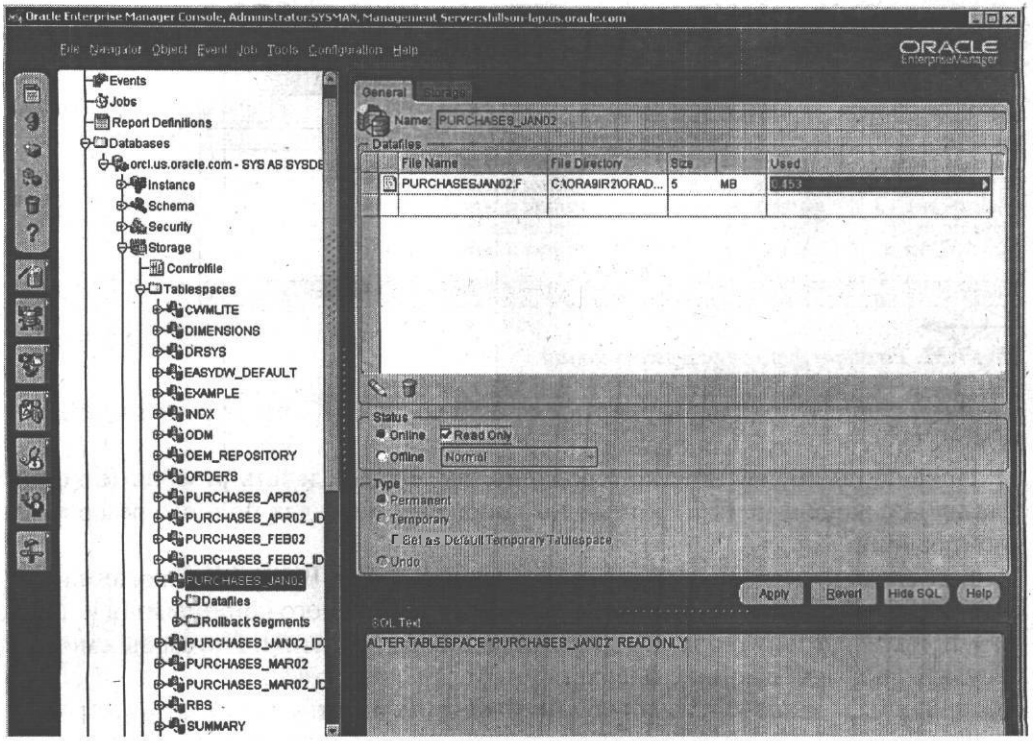


Рис. 7.31. *Как сделать таблицу доступной только для чтения*

размеры и занимает приблизительно 1.33 Гб в системе NT. На рис. 7.32 можно видеть разницу размеров разных файлов резервных копий. .

Полная резервная копия занимает 527 Мб, что представляет собой приблизительно 25% общего размера базы данных. Следовательно, видна существенная экономия места при использовании функций резервного копирования, предлагаемых в Oracle 9i. Если использовать утилиту резервного копирования операционной системы, которая может сжимать или не сжимать файл, то потребуется 1.33 Гб вместо 527 Мб.

Еще меньший размер имеют файлы инкрементных резервных копий и копий табличных пространств, и здесь мы видим большую выгоду применения данной утилиты.

Предполагается, что в базу данных было внесено лишь несколько изменений. Однако, можно видеть, что даже в случае инкрементного резервного копи-

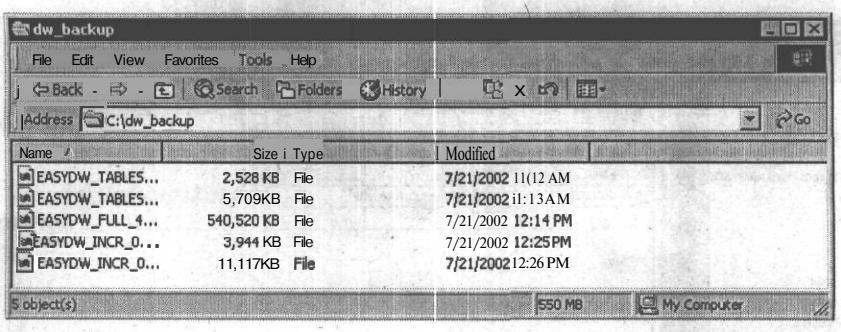


Рис. 7.32. *Размеры файлов резервных копий*

рования, размеры файлов варьируют от 3.85 до 10.8 Мб, что весьма отличается от того, что требуется для полного копирования.

Проводя инкрементное копирование, вы можете делать резервные копии хранилища чаще и не беспокоиться о поиске времени для полного резервного копирования.

Еще одно, последнее напоминание по данной теме. В ходе тестирования хранилища не забудьте протестировать процедуры резервного копирования и получить приблизительную оценку времени для их выполнения. Потом вы сможете обсудить свои требования с операционным отделом, чтобы ваши задачи по управлению были включены в список необходимых работ.

7.3.2. Создание каталога для Recovery Manager

Информацию, описывающую резервное копирование, выполняемое с помощью RMAN, можно хранить в управляющем файле целевой базы или в каталоге восстановления (recovery catalog). Каталог восстановления нужно всегда помещать в базу данных, иную, чем хранилище. Можно сохранить ее в той базе, где хранится репозиторий OEM. Использование каталога восстановления является более безопасным вариантом, чем хранение информации только в управляющем файле, поскольку информация хранится отдельно от базы данных. Это также позволяет хранить информацию обо всех резервных копиях в одном месте.

В ходе этапов создания конфигурации резервного копирования в OEM можно увидеть одну закладку, в которой указывается, нужно ли использовать каталог восстановления. В примере, показанном на рис. 7.33 мы заявляем, что каталог восстановления использовать нужно. Найти его можно в базе данных orcl.us.oracle.com, и для доступа к этой информации необходимо указать имя пользователя и пароль.

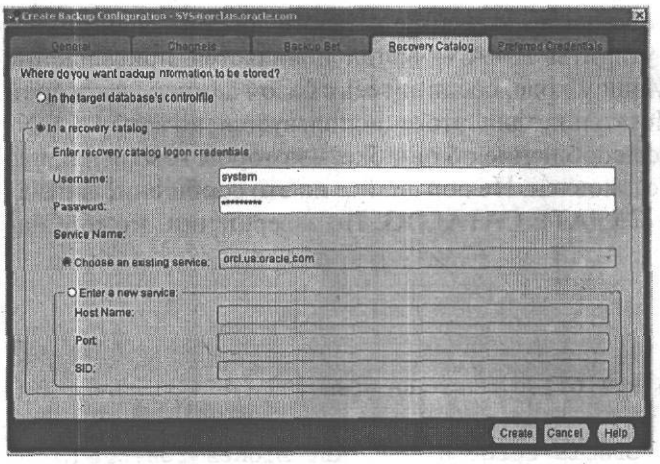


Рис. 7.33. *Каталог восстановления*

Как уже упоминалось ранее, каталог восстановления нужно всегда помещать в иную базу данных, чем та, которая копируется. Мы используем ту же самую базу только для демонстрации. Кроме того, не забудьте, что для каталога восстановления также нужно регулярно проводить резервное копирование, поскольку он хранится в базе данных.

Создание каталога восстановления

Каталог восстановления нужно сохранять в отдельном табличном пространстве. В нашем примере используется табличное прогстранство OEM_REPOSITORY. Нужно создать нового пользователя, которому будет принадлежать схема каталога восстановления в базе данных каталога восстановления. В приводимом примере мы создали пользователя с именем rman_user и предоставили ему соответствующие привилегии.

Создание пользователя

```
SQL> CREATE USER rman_user IDENTIFIED BY rman_user
     TEMPORARY TABLESPACE temp
     DEFAULT TABLESPACE oem_repository;

SQL> GRANT recovery_catalog_owner TO rman_user;
SQL> GRANT connect,resource TO rman_user;
```

Следующий шаг — это, собственно, создание каталога восстановления. Это достигается путем запуска утилиты RMAN. Когда появляется приглашение RMAN, как показано в следующем коде, соединитесь с базой данных, в которой будет храниться каталог. Поскольку для этого используется команда CONNECT CATALOG, появляется сообщение об ошибке, в котором говорится, что каталог восстановления не установлен. Не обращайте на это сообщение внимания. Затем подайте команду CREATE CATALOG. По завершении, можете начинать использовать каталог.

```
Rman

Recovery Manager: Release 9.2.0.1.0 — Production

Copyright <c> 1995, 2002, Oracle Corporation. All rights reserved.

RMAN> CONNECT CATALOG rman_user/rman_user;
Connected to recovery catalog database
Recovery catalog is not installed

RMAN> CREATE CATALOG TABLESPACE oem_repository;
Recovery catalog created
```

Перед тем как базу данных можно будет включать в каталог, ее нужно зарегистрировать. В противном случае вы можете увидеть сообщение Rman-20001 target database not found in recovery catalog (Rman-20001 Целевая база данных не найдена в каталоге восстановления). Увидев это сообщение, не беспокойтесь.

Чтобы зарегистрировать базу данных, нужно соединиться в одном сеансе и с базой, в которой храниться каталог восстановления, и с целевой базой, для которой выполняется резервное копирование, как это показано в приведенном примере.

```
RMAN> CONNECT TARGET easydw/easdydw@orcl;
Connected to target database: ORCL <DBID=9915458574>
RMAN> CONNECT CATALOG rman_user/rman_user@orcl;
Connected to recovery catalog database
```

Когда целевая база будет зарегистрирована, RMAN заполнит таблицы каталога информацией из управляющего файла. Примеры сохраняемой здесь информации — идентификатор базы, ее имя, табличные пространства, файлы данных, протоколы

повторного выполнения и архивные журналы. Выходная информация показывает полную ресинхронизацию (full resynch), означающую, что таблицы каталога обновлены самой последней информацией из управляющего файла.

```
RMAN> REGISTER DATABASE;
Database registered in recovery catalog
Starting full resynch of recovery catalog
Full resynch complete
```

Как только база данных зарегистрирована в каталоге восстановления, каталог можно использовать для получения информации о собранных резервных копиях.

Чтобы увидеть, какие базы данных зарегистрированы в каталоге восстановления, используйте команду LIST INCARNATION. В приводимом примере зарегистрированной является одна база данных. Если зарегистрировано несколько баз, все они будут перечислены здесь.

```
RMAN> LIST INCARNATION;
List incarnation;
List of Database Incarnations

DB Key  Inc Key  DB Name  DB ID       CUR  Reset SCN  Reset Time
------  -------  -------- ----------  ---  ---------  ----------
1       2        ORCL     9915458574  YES  190578     05-JUN-02
```

Запустив утилиту RMAN, вы можете сделать много всего и кроме резервного копирования базы. Чтобы получить информацию о табличных пространствах базы данных, для которых нужно создать резервные копии, используйте команду REPORT SCHEMA, как показано на рис. 7.34.

Команду LIST можно использовать для получения различной информации о резервном копировании. В примере на рис. 7.35 показана команда LIST BACKUP, которая показывает файлы данных, скопированные в ходе операции полного резервного копирования, проведенной 7 июля, и хранимые в файле C:\DW_BACKUP\EASYDW_466605312_2.

Еще одна полезная команда — REPORT. Вы можете обнаруживать ни к чему не относящиеся резервные копии, устаревшие копии и те копии, для которых нужны архивные журналы за несколько дней. Устаревшие резервные копии более не нужны либо потому, что они избыточны, либо потому, что они не требуются для восстановления. Ни к чему не относящиеся копии не нужны для восстановления базы, поскольку они относятся к предыдущей версии (incarnation) базы

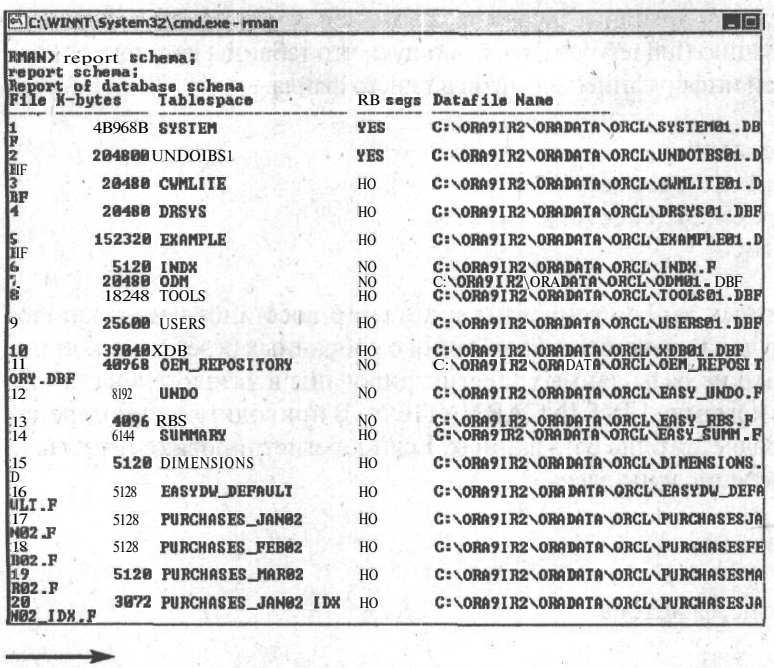


Рис. 7.34. *Информация о схеме RMANс целевой базой данных*

данных. На рис. 7.36 показана информация о файлах, для чьего восстановления требуется более двух дней архивного журнала. Кроме этого, у команды REPORT есть еще множество опций, и мы предлагаем обратиться к руководству Oracle 9i Backup and Recovery Manual за подробной информацией.

Хотя все это было лишь кратким знакомством с каталогом восстановления, мы надеемся, что вы начали видеть некоторые из преимуществ его использования, особенно возможность видеть имеющиеся резервные копии. Позже вы увидите, как с помощью этой информации можно выполнять автоматическое восстановление, хотя в нашем хранилище мы предпочитаем делать это вручную.

7.3.3. Восстановление

Задача восстановления базы данных из резервной копии, как правило, внушает страх большинству администраторов. Они всегда боятся, что восстановление может не сработать и они останутся вообще без базы. К сожалению, пока не будут протестированы все резервные копии, вы не сможете быть уверены в работоспо-

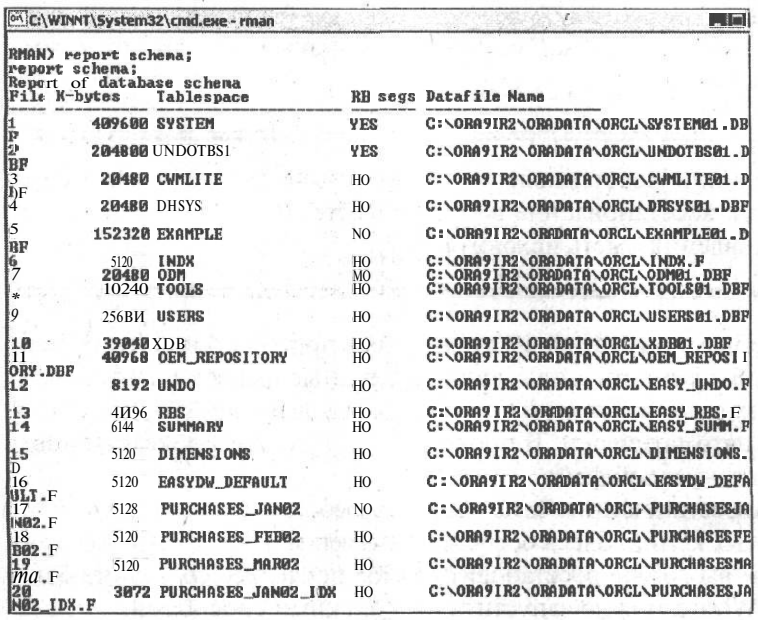


Рис. 7.35. *Вывод содержимого резервной копии в RMAN*

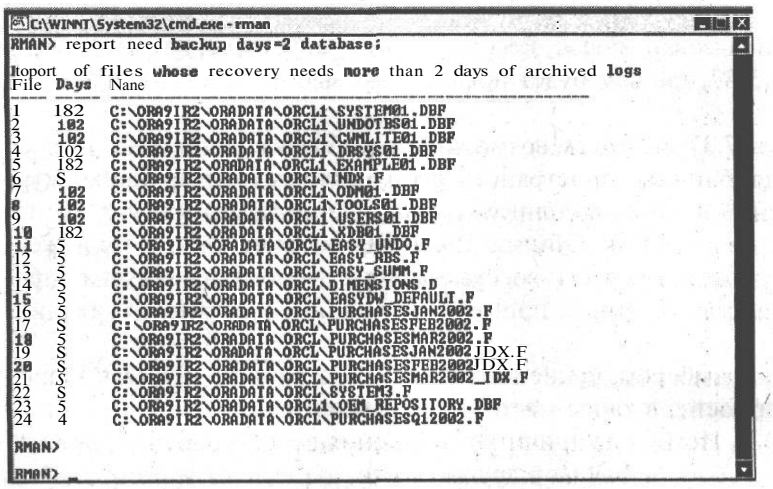


Рис. 7.36. *RMAN — информация о файлах, нуждающихся в резервном копировании*

собности файлов. Если есть такая возможность, хорошо бы периодически восстанавливать базу из копии в тестовой системе, чтобы знать, что файлы и сама процедура работоспособны.

Совет. Если вы выполняете восстановление после серьезной проблемы в базе, попробуйте выполнить восстановление в другом месте, чтобы в случае безрезультатности восстановления иметь исходную базу.

Одной их первых проблем, с которой вы столкнетесь при восстановлении базы данных, является необходимость знать, какие резервные файлы использовать. Если вы используете каталог восстановления, тогда все очень просто, поскольку каталог выяснит это автоматически. В противном случае вам придется проверять записи и выявлять нужный файл.

Не забывайте, что восстановление базы может потребовать нескольких файлов полных и инкрементных копий. Здесь особенно полезен мастер RMAN Recovery Wizard, особенно если вы хотите восстановить самое последнее состояние базы.

Мастер Recovery Wizard можно запустить несколькими способами:

- выбрав его в пункте Tools (Инструменты) в панели инструментов;
- с помощью правой кнопки мыши, выбрав базу данных в окне Проводника;
- начав со значка инструментов базы данных.

Какой бы ни был использован способ, Recovery Wizard запустится, появится окно, показанное на рис. 7.37, где вам будет предложено выбрать необходимый тип восстановления.

В примере на рис. 7.37 мы имеем возможность выбрать восстановление всей базы или отдельных табличных пространств, файлов данных или архивных журналов. Восстановление можно выполнить к определенному моменту времени, например к 11:30 в понедельник. Обычно такой тип восстановления не нужен, однако вы можете захотеть провести восстановление до момента запуска определенного задания в базе. В данном примере мы решили восстановить табличное пространство.

После того как мы выберем, какие табличные пространства нам нужно восстановить, мастер спросит, в какое место мы хотим помещать восстанавливаемые файлы (рис. 7.38). Исходя из причин выполнения восстановления, вам может понадобиться поместить файлы в другое место, и здесь вы можете указать его. Еще одним преимуществом является то, что в этом окне вам предоставляется возможность проверить восстанавливаемые файлы.

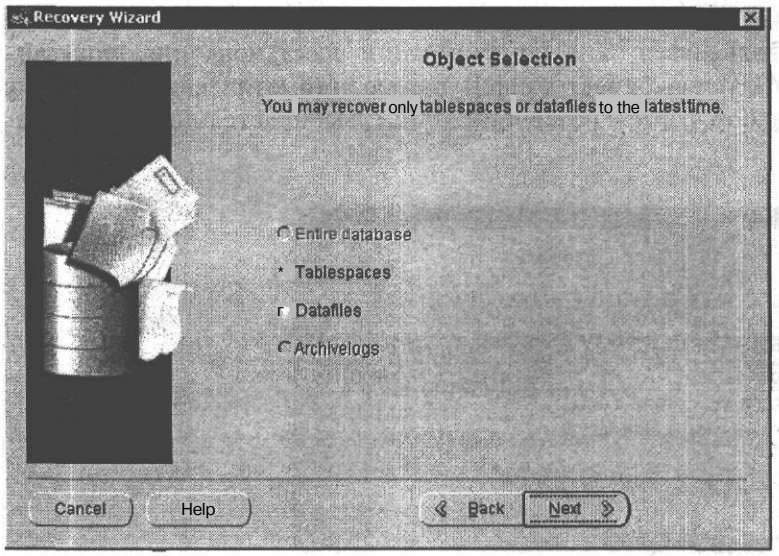


Рис. 7.37. *Выбор типа восстановления*

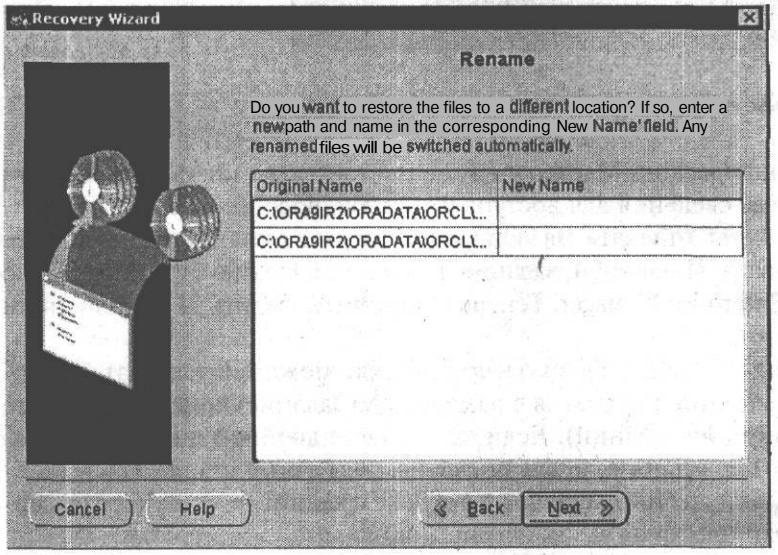


Рис. 7.38. *Местоположение файлов данных*

Следующий этап показывает, почему вам нужно знать свою процедуру восстановления. Появляется окно конфигурации восстановления, показанное на рис. 7.39. Мастер выбирает конфигурацию автоматически. Поскольку существует лишь набор файлов резервной копии, выбирается полная резервная копия.

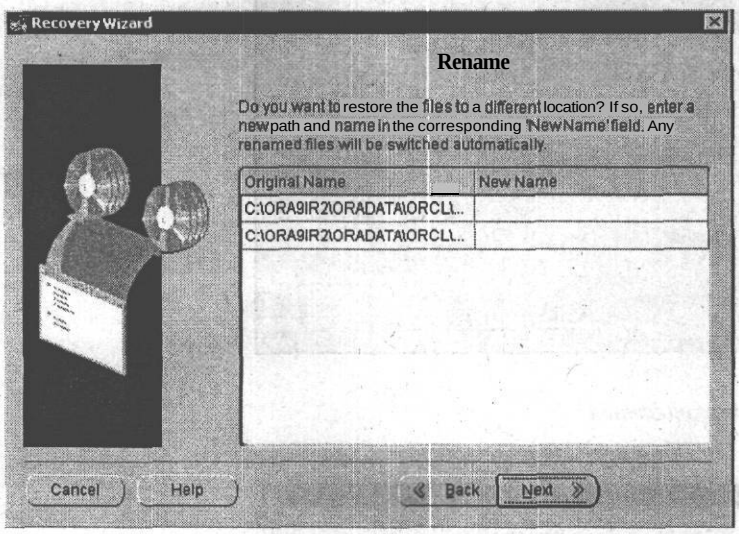


Рис. 7.39. *Конфигурация восстановления*

С помощью этой информации Мастер узнает о местоположении файлов резервной копии и получает сведения для доступа к каталогу восстановления.

Теперь нужно будет отвечать на вопросы в других окнах, определяя, что нужно восстанавливать. И наконец, задание по восстановлению будет помещено в очередь заданий Enterprise Manager. Теперь вы сможете следить за вышолнением задания из консоли.

Когда выполнение задания будет завершено, вы можете проверить его результат, просмотрев протокол задания в разделе Job (Задание) консоли, в пункте Job History (Предыстория заданий). Если вы сделаете двойной щелчок мышью по интересующему вас заданию, будут показаны различные стадии его выполнения. Если вы сделаете двойной щелчок по строке с надписью completed (завершено), будет показан протокол задания (рис. 7.40).

В этом примере мы видим лишь часть протокола, но заметьте, что, поскольку мы использовали каталог восстановления, утилита автоматически подобрала

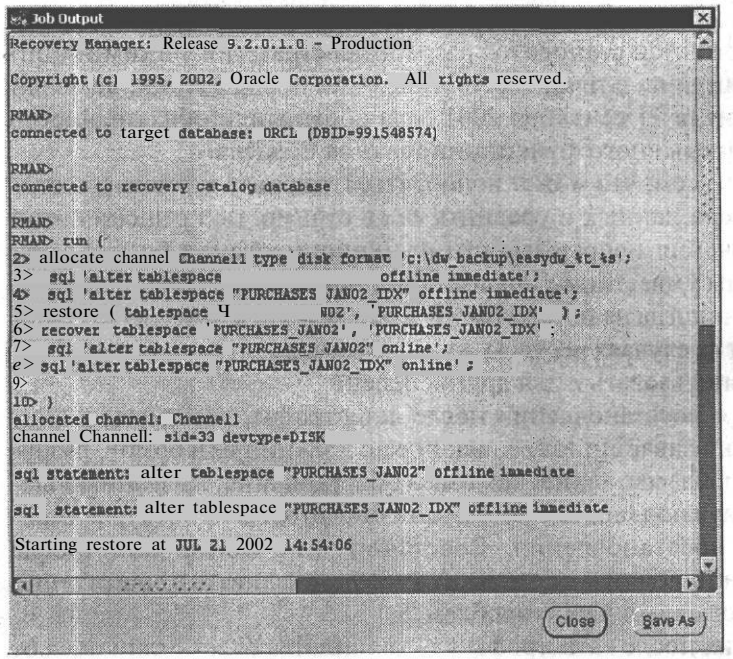


Рис. 7.40. *Протокол восстановления*

необходимые файлы резервной копии. Хотя мы показали здесь процесс восстановления табличного пространства, процедура практически идентична и для файла данных, и для всей базы.

Мы надеемся, что этот раздел дал вам некоторое представление о том, как выполнять резервное копирование и восстановление хранилища Oracle 9i. Эта тема столь широка, что мы настоятельно рекомендуем обратиться к руководству Oracle 9i Release 2 Backup and Recovery Manual за подробной информацией о возможностях, а также и за многими другими сведениями о том, как проектировать, создавать и выполнять операции по резервному копированию и восстановлению баз данных.

7.4. Восстановление после катастрофы

По мере того как хранилище становится все более операционным по природе и по мере того как возникает необходимость доступа к нему 24 часа в сутки,

даже временные простои начинают представлять опасность для бизнеса. Как и в OLTP-системе, критичной становится разработка стратегии минимизации времени простоев хранилища и плана восстановления после аварии, ведущей к потере данных. События 11 сентября 2001 года обозначили обязательность наличия плана для непрерывного функционирования бизнеса.

Важно подумать обо всем, что может испортиться, оценить возможную стоимость простоев и потери данных и сравнить ее со стоимостью решения этой проблемы/Для обеспечения непрерывности функционирования бизнеса при катастрофах необходимы инвестиции, аналогичные покупке страхового полиса. Надеемся, что вам это никогда не понадобится, но если и это случится, то решение у вас будет. Во многих случаях ресурсы, вкладываемые в повышение доступности, также можно использовать и для других целей.

Когда мы говорим о восстановлении после катастрофы, мы имеем в виду именно катастрофические аварии, такие, как пожар в здании или бомбу, разрушившую его. Другие проблемы, такие, как отказ оборудования, вызванный выходом из строя диска, нужно решать с помощью стандартных процедур резервного копирования и восстановления. Восстановление после катастрофы отличается тем, что компьютерные системы нужно устанавливать в совершенно новом месте, что создает целый набор проблем.

План восстановления после катастрофы для хранилища будет отличаться от плана восстановления системы обработки транзакций. Одна из главных проблем хранилища — его абсолютные размеры. Мы надеемся, что по меньшей мере один набор лент с недавними резервными копями хранится где-нибудь подальше, а те, которые хранятся там же, хранятся в огнеупорном месте. Когда вы знаете, что у вас есть набор лент для восстановления базы данных, нужно принять во внимание в процедурах восстановления размеры базы и необходимое аппаратное обеспечение.

Будет ли хранилище восстанавливаться в новом, временном месте, зависит от важности его для бизнеса. Может быть, вы решите, что бизнес сможет обойтись без хранилища до восстановления его в обычном местоположении. Принимать такое решение следует с величайшей осторожностью, поскольку возврат к нормальной работе может занять значительно больше времени, чем вы думаете. Существует вероятность, что в случае крупной аварии, особенно касающейся всего здания, возврат к нормальной работе займет много месяцев.

Когда будет принято решение о создании плана восстановления после катастрофы, для реализации этого плана можно использовать Oracle Data Guard.

7.4.1. Oracle Data Guard

Oracle Data Guard создает и поддерживает в рабочем состоянии запасную базу данных (standby), располагающуюся в другом месте. Если основная база данных становится недоступной, приложения можно перезапустить для работы с запасной базой. Запасная база первоначально создается из резервной копии основной базы. По мере того как в основную базу вносятся изменения, создающийся протокол повторного выполнения (redo) передается в запасную базу, где он запускается, поддерживая, таким образом, базы в синхронизованном состоянии. Data Guard можно сконфигурировать так, чтобы обеспечить отсутствие потерь данных в случае аварии. Его также можно использовать для проведения плановых операций по обслуживанию, таких, как обновление аппаратуры или операционной системы. И основная, и запасная системы могут использовать реальные кластеры приложений. Существуют два типа запасных баз данных: физические и логические.

Физические запасные базы данных имеются во многих версиях баз, и в настоящее время используются для защиты OLTP-систем и рабочих информационных хранилищ. Поскольку физическую запасную базу нельзя открывать во время операций по обслуживанию, такую базу нельзя использовать для других целей. Логические запасные базы впервые предложены в Oracle 9i Release 2 для решения этой проблемы.

В Oracle 9i Release 2 запасная база может быть либо физической, либо логической:

- Физическая запасная база — это идентичная, поблочная копия основной базы. Она синхронизируется с основной базой путем применения к запасной базе протокола повторного выполнения основной базы (такой процесс называется redo apply) с использованием того же самого процесса, который используется для восстановления. Когда в запасной базе не проводится процесс восстановления, к ней возможен доступ только для чтения.
- Логическая запасная база содержит логически те же самые данные, что и основная база, но ее физическая структура может быть иной. Она синхронизируется с основной базой путем перевода данных протокола повторного выполнения в код SQL, а затем выполнения этих команд (такой процесс называется SQL apply). Данные можно читать в то же самое время, когда с помощью команд SQL вносятся изменения в базу, однако данные нельзя модифицировать. В логической запасной базе можно также добавлять индексы и материализованные представления для повышения производительности обработки запросов. Можно добавлять и дополнительные таблицы. Эти таблицы можно обновлять в любое время. В момент записи в логической запасной базе существует ряд ограничений по типам данных, типам таблиц и операциям.

Конфигурация Data Guard может слагаться из одной основной базы данных и до девяти физических и логических баз. Типичная конфигурация показана на рис. 7.41. Основная база данных передает протокол повторного выполнения и в физическую и в логическую запасную базу, которые могут иметь то же самое или иное местоположение. Физическая запасная база используется для восстановления после катастрофы, а логическая — в первую очередь для создания отчетов, но ее также можно использовать и для восстановления.

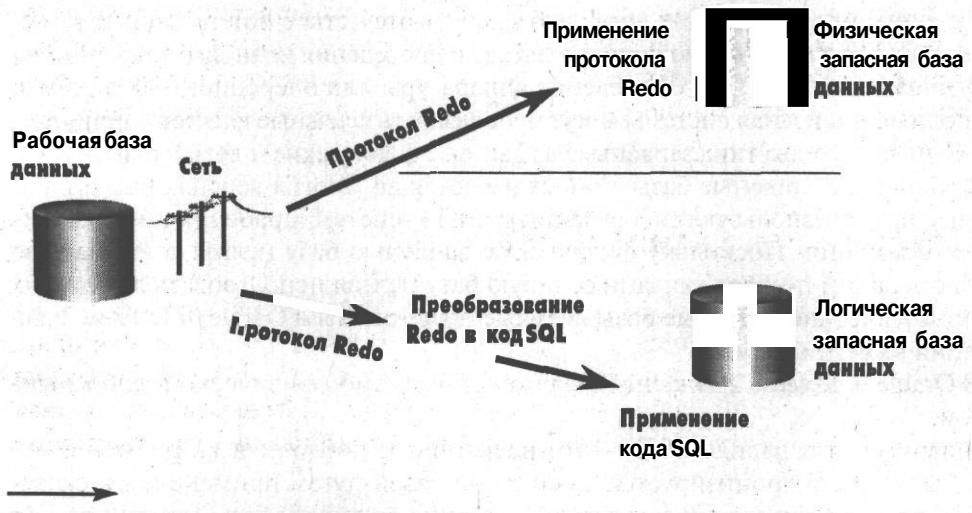


Рис. 7.41. *Конфигурация Data Guard*

7.4.2. Хранилище данных и план восстановления после катастрофы

Хранилище данных можно включить в план восстановления после катастрофы (Enterprise disaster recovery plan) несколькими способами.

- Можно использовать логическую запасную базу в качестве источника данных для хранилища. Поскольку она содержит копию данных рабочей системы, ее можно использовать в процессе извлечения-преобразования-загрузки (ETL) данных для хранилища.
- Можно использовать логическую запасную базу в качестве рабочего информационного хранилища (operational data store). Затем данные можно переме-

стить в промежуточную область другой системы, преобразовать в схему звезды, и они готовы для выполнения запросов.

- Логическая запасная база может содержать таблицы хранилища. Таблицы, обслуживаемые с помощью применения кода SQL (SQL apply), должны иметь ту же логическую структуру, что и таблицы OLTP-системы, но для выполнения преобразований и создания схемы звезды в запасной логической базе можно создавать дополнительные таблицы. Модифицировать можно все таблицы, не обслуживаемые с помощью применения кода SQL.

В конечном счете, стратегия восстановления после катастрофы будет определяться важностью хранилища для бизнеса. Мы бегло познакомились с тем, каким образом следует включать хранилище данных в план восстановления после катастрофы и какие возможности, предлагаемые Oracle, позволяют реализовать этот план.

7.5. Реорганизация хранилища

Реорганизация базы данных, будь это хранилище или система обработки транзакций, является задачей весьма нелегкой. К сожалению, в случае хранилищ временные затраты, которые необходимы для реорганизации, могут стать весьма серьезной проблемой из-за больших объемов данных. Следовательно, полная реорганизация не должна быть необходимой, хотя небольшие реорганизации могут понадобиться.

7.5.1. Зачем проводить реорганизацию?

Потребность в реорганизации может возникнуть в силу многих причин, например:

- изменились потребности бизнеса;
- обнаружен недостаток в построении базы;
- проведение архивации данных;
- правительство поменяло правила;
- после поглощения другой компании потребовалась интеграция компьютерных систем.

Изменения требований бизнеса почти невозможно предугадать, так что одним из решений проблемы может оказаться создание тематического хранилища (data mart) вместо полной реорганизации всего хранилища.

Крупный недостаток построения базы можно преодолеть, если тщательно изучить дизайн и, если это необходимо, пригласить опытного разработчика хранилищ со стороны. Одна из проблем хранилищ состоит в том, что дизайн, который поначалу кажется хорошим, в перспективе может оказаться неподходящим из-за больших объемов данных. Следовательно, старайтесь сводить такие проблемы к минимуму, используя такие методы, как секционирование.

Реорганизация может быть плановой. Если вы хотите поддерживать более-менее постоянный размер хранилища и не допускать его бесконечного роста, можно хранить данные только за несколько лет. Когда новые данные добавляются, старые архивируются и удаляются. Такая операция называется сдвигом рамки (rolling window), и она подробно обсуждается в следующем разделе.

7.5.2. Обслуживание разделов

Некоторые необходимые для хранилища структурные изменения можно осуществить с помощью операций над разделами. В этом разделе мы рассмотрим некоторые наиболее распространенные операции над разделами, в том числе следующие:

- добавление и удаление разделов;
- обмен разделами;
- разделение и слияние разделов;
- сращивание разделов;
- отсечение раздела;
- перемещение раздела.

Добавление и удаление разделов и выполнение операций сдвига рамки

При первоначальном создании хранилища у вас может совсем не быть данных предыстории для его заполнения или иметься лишь небольшой их объем. Следовательно, в течение первых 18–24 месяцев вы спокойно добавляете новые данные ежемесячно, пока не исчерпаете возможности системы. Теперь, когда хранилище заполнено, каждый месяц вы делаете копию старых данных и удаляете их, чтобы освободить место для новых. Без секционирования вам придется сканировать данные и вручную удалять их, что, конечно, занимает много времени. Более быстрым вариантом решения этой проблемы является удаление раздела, содержащего старые данные, и создание нового раздела для новых данных, как это показано на рис. 7.42.



Рис. 7.42. *Операции с разделами*

Такой метод применим, только если данные секционированы по дате. Поэтому, если вы решите использовать другую схему секционирования, например по коду, то такой тип операций станет невозможным.

Команды SQL, используемые для выполнения задачи, соответствующей рис. 7.42, приведены ниже. Во-первых, удаляется старый раздел, содержащий данные о продажах за май 1997.

```
SQL> ALTER TABLE easydw.purchases
   DROP PARTITION purchases_may97;
```

Затем создается новый раздел для данных о продажах за апрель 2002 года. Первый этап — создать табличное пространство, в котором будут располагаться данные.

```
SQL> CREATE TABLESPACE purchases_apr2002
   datafile 'E:\EASYDW\PURCHASESapr2002.f'
   size 5m reuse autoextend on default storage
   (initial 16k next 16k pctincrease 0 maxextents unlimited);
```

Следующий этап — изменить определение таблицы, включив в нее ссылку на только что созданное табличное пространство. К таблице добавлен раздел PURCHASES_APR2002.

```
SQL> ALTER TABLE easydw.purchases
   ADD PARTITION purchases_apr2002
     values less than (TO_DATE('30-04-2002', 'DD-MM-YYYY'))
     pctfree 0 pctused 99
     storage (initial 64k next 16k pctincrease 0)
     tablespace purchases_apr2002 ;
```

Обмен разделами

После того как к таблице был добавлен новый раздел, в него можно загружать данные. Методы загрузки данных были описаны в главе 5. Если данные уже находятся в таблице базы данных, то самым быстрым способом переноса их в новый раздел будет метод обмена разделами (exchange partition), который используется для переноса данных из несекционированной таблицы в секционированную. Обмен разделами также можно применять для преобразования раздела в несекционированную таблицу или для преобразования типов секционированных таблиц. В приведенном фрагменте кода показано перемещение данных из таблицы APR_ORDERS в раздел PURCHASES_APR2002 таблицы EASYDW.PURCHASES.

```
SQL> ALTER TABLE easydw.purchases
     EXCHANGE PARTITION purchases_apr2002 WITH TABLE apr_orders;
```

Слияние разделов

Вы можете слить (merge) разделы в один или разрезать (split) раздел, если он становится слишком велик. Данные можно объединить в новом разделе или в уже существующем.

Одним из методов, который часто используют разработчики баз, является хранение данных за первые шесть месяцев в помесячных разделах, а затем — в поквартальных. С помощью опции MERGE PARTITION данные, как показано в приводимом фрагменте кода, можно легко перенести в новый раздел.

Если мы хотим объединить данные с января по март 2002 года в квартальном разделе, мы можем провести операцию слияния. Во-первых, создается новое табличное пространство для хранения данных о покупках за квартал Q1.

```
SQL> CREATE TABLESPACE purchases_q1_2002
    DATAFILE 'c:\ora9ir2\oradata\orcl\PURCHASESQ12002.f' SIZE 5M
    REUSE AUTOEXTEND ON
    DEFAULT STORAGE
    (INITIAL 64K NEXT 64K PCTINCREASE 0 MAXEXTENTS UNLIMITED);
```

Разделы сливаются по два за раз. Если таблица секционирована по диапазону, сливать можно только смежные разделы. Мы можем слить разделы за январь и февраль или за февраль и март, но не можем слить разделы за январь и март.

Новый раздел наследует верхнюю границу двух объединенных разделов. Следовательно, в квартальный раздел Q1 первыми должны сливаться два раздела с наивысшим диапазоном значений (февраль и март). В приводимом коде два раздела сливаются и сохраняются в только что созданном табличном пространстве. Раздел PURCHASES_Q12002 автоматически добавляется в таблицу purchases. Разделы PURCHASES_FEB2002 и PURCHASES_MAR2002 автоматически удаляются из таблицы.

```
SQL> ALTER TABLE purchases
   MERGE PARTITIONS purchases_feb2002,purchases_mar2002
   INTO PARTITION purchases_q12002
   TABLESPACE  purchases_q1_2002
```

После этого с разделом PURCHASES_Q12002 сливается раздел с меньшим диапазоном значений — PURCHASES_JAN2002. В этом фрагменте иллюстрируется слияние в уже существующем разделе.

```
SQL>ALTER TABLE purchases
   MERGE PARTITIONS purchases_jan2002,purchases_q12002
   INTO PARTITION purchases_q12002
   TABLESPACE purchases_q1_2002
```

После этой операции таблица purchases будет содержать один раздел, как это показано на рис. 7.43. Наивысшим значением для таблицы PURCHASES_Q12002 будет 31 марта 2002 года.

При слиянии разделов сливаются и сами данные, и индексы. Разделы индекса за январь, февраль и март автоматически удаляются и заменяются новым разделом индекса за первый квартал. В следующем запросе показан новый раздел индексов PURCHASE_PRODUCT_INDEX и PURCHASE_TIME_INDEX, с именем PURCHASES_Q12002.

```
SQL> SELECT INDEX_NAME, PARTITION_NAME, STATUS
FROM USER_IND_PARTITIONS;
INDEX_NAME                PARTITION_NAME       STATUS
------------------------  -----------------    -------
PURCHASE_PRODUCT_INDEX    PURCHASES_Q12002     UNUSABLE
PURCHASE_TIME_INDEX       PURCHASES_Q12002     UNUSABLE
```

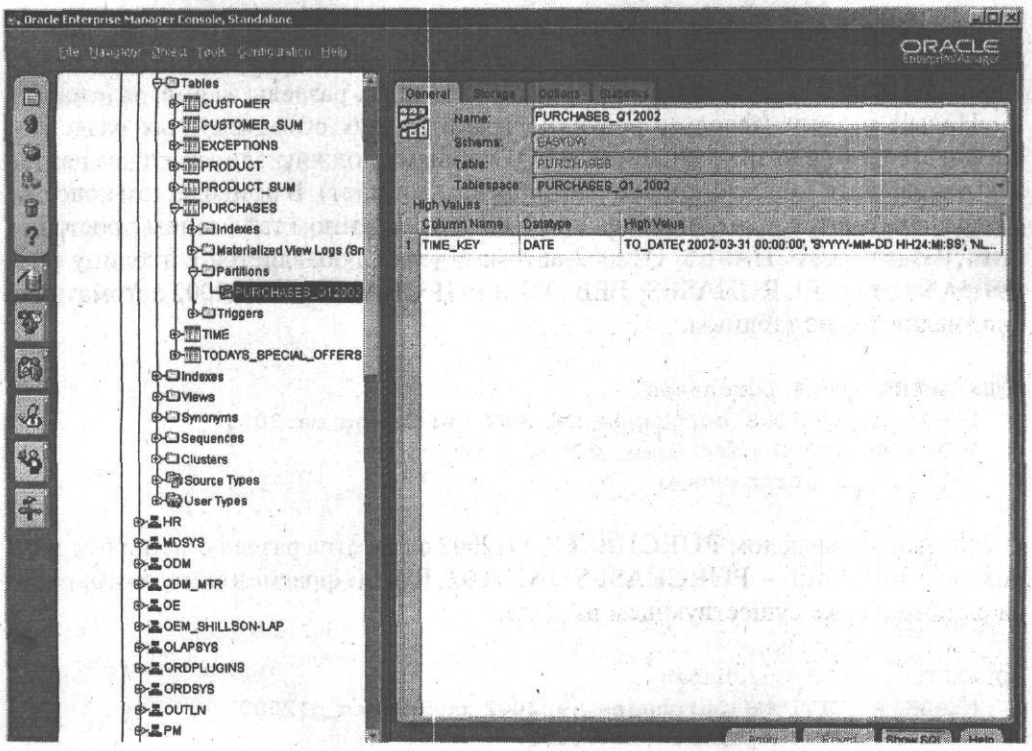


Рис. 7.43. *Слияние разделов*

Новые разделы индексов являются непригодными (unusable) и должны быть перестроены, как это показано в приводимом примере.

```
SQL> ALTER INDEX purchase_product_index
REBUILD PARTITION purchases_q12002
SQL> ALTER INDEX purchase_time_index
REBUILD PARTITION purchases_q12002
```

Разрезание разделов

Если раздел становится слишком велик, его может понадобиться разрезать для того, чтобы его можно было обслуживать за меньшие периоды времени или чтобы разделить нагрузку на каналы ввода-вывода между несколькими устройствами.

Раздел можно разделить на два новых раздела. Если таблица purchases была изначально секционирована по кварталам, а объем продаж существенно превысил ожидаемые значения, что привело к появлению очень крупных разделов, раздел можно разделить на три помесячных раздела. Сначала создаются табличные пространства для разделов PURCHASES_JAN2002, PURCHASES_FEB2002 и PURCHASES_MAR2002.

В следующем примере все строки со значением PURCHASE_DATE (даты покупки) меньшей или равной 31 января 2002 года отделяются в раздел PURCHASES_JAN2002. Оставшиеся строки остаются в разделе PURCHASES_Q12002. Данные за январь будут храниться в табличном пространстве PURCHASES_JAN2002.

```
SQL> ALTER TABLE purchases
   SPLIT PARTITION purchases_q12002
   AT (TO_DATE('31-JAN-2002','dd-mon-yyyy'))
   INTO (PARTITION purchases_jan2002
     TABLESPACE purchases_jan2002,
     PARTITION purchases_q12002)
```

После операции разделения появляются два раздела. Теперь оставшиеся строки раздела PURCHASES_Q12002 разделяются на мартовский и февральский разделы.

```
SQL> ALTER TABLE purchases
   SPLIT PARTITION purchases_q12002
   AT (TO_DATE('28-FEB-2002','dd-mon-yyyy'))
   INTO (PARTITION purchases_feb2002
     TABLESPACE purchases_feb2002,
     PARTITION purchases_mar2002
     TABLESPACE purchases_mar2002)
```

Данные перераспределяются, и раздел PURCHASES_Q12002, как можно видеть на рис. 7.44, автоматически удаляется.

Все разделы локальных индексов, соответствующих разделу PURCHASES_Q12002, удаляются. На их месте возникают разделы локальных индексов для новых разделов таблицы.

```
SQL> SELECT INDEX_NAME, PARTITION_NAME, STATUS
    FROM USER_IND_PARTITIONS;
```

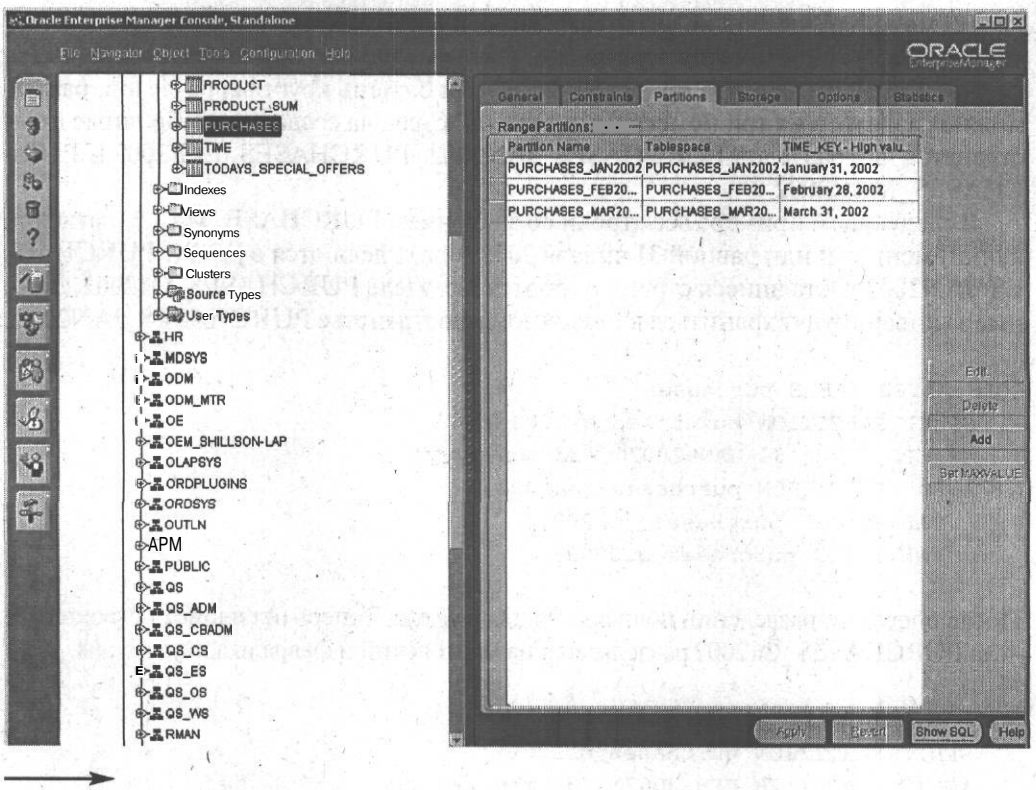


Рис. 7.44. *Разрезание раздела*

```
INDEX_NAME                PARTITION_NAME      STATUS
------------------------  ------------------  -------
PURCHASE_PRODUCT_INDEX    PURCHASES_JAN2002   USABLE
PURCHASE_PRODUCT_INDEX    PURCHASES_FEB2002   UNUSABLE
PURCHASE_PRODUCT_INDEX    PURCHASES_MAR2002   USABLE
PURCHASE_TIME_INDEX       PURCHASES_JAN2002   USABLE
PURCHASE_TIME_INDEX       PURCHASES_FEB2002   UNUSABLE
PURCHASE_TIME_INDEX       PURCHASES_MAR2002   USABLE
```

Все непригодные индексы нужно перестроить.

```
SQL> ALTER INDEX PURCHASE_PRODUCT_INDEX
     REBUILD PARTITION purchases_feb2002
```

```
SQL> ALTER INDEX purchase_time_index
     REBUILD PARTITION purchases_feb2002
```

Сращивание хэш-разделов

Линейные и списочные разделы можно сливать (merge), но хэш-разделы — нельзя. Их нужно сращивать (coalesce). Вместо того чтобы выявлять, в каком разделе хранится строка, сравнивая значение ключа секционирования с критерием секционирования таблицы, как это делалось для линейных и списочных разделов, здесь раздел определяется путем применения хэш-функции.

Слияние разделов уменьшает их количество на единицу. Когда меняется число разделов, для перераспределения данных нужно применять хэш-функцию. Сращивание разделов делает именно это.

В следующем примере показано создание таблицы с хэш-секционированием, имеющей 10 разделов.

```
CREATE TABLE easydw.hash_purchases
 (product_id varchar2(8),
  time_key date,
  customer_id varchar2(10) ,
  purchase_date date,
  purchase_time number(4,0),
  purchase_price number(6,2),
  shipping_charge number(5,2),
  today_special_offer varchar2(1))
PARTITION BY HASH(product_id)
PARTITIONS 10;
```

Чтобы уменьшить количество разделов на единицу, подайте следующую команду:

```
SQL> ALTER TABLE easydw.hash_purchases COALESCE PARTITION;
```

В результате один из разделов исчезнет.

Кроме того, таблицу с хэш-секционированием нельзя разрезать, если она становится слишком большой. Чтобы увеличить количество разделов в таблице с хэш-секционированием, измените таблицу, введя туда новый раздел.

Отсечение разделов

Иногда нам бывает нужно удалить все строки раздела. Например, мы можем работать только с данными за 18 месяцев, и один раз в месяц нам приходится удалять старые данные. Вместо того чтобы удалять строки по отдельности, можно использовать команду TRUNCATE PARTITION, которая быстро удалит данные.

Чтобы удалить из раздела все строки, но не удалять сам раздел, используйте команду TRUNCATE. Это намного быстрее, чем удалять каждую строку отдельно. Все локальные индексы для раздела, например EASYDW.PURCHASE_TIME_INDEX, также отсекаются. До отсечения в разделе PURCHASES_JAN2002 было 7498 строк.

```
SQL> SELECT COUNT(*) FROM purchases
    WHERE time_key BETWEEN TO_DATE('01-JAN-2002') AND
                           TO_DATE('31-JAN-2002')

COUNT(*)
--------
    7498


SQL> ALTER TABLE PURCHASES
     TRUNCATE PARTITION purchases_jan2002;
```

После отсечения раздела все ряды удалены.

```
SQL> SELECT COUNT(*) FROM purchases
    WHERE time_key BETWEEN TO_DATE('01-JAN-2002') AND
                           TO_DATE('31-JAN-2002')
COUNT(*)
----------
         0
```

Перемещение разделов

Раздел можно переместить из одного табличного пространства в другое. Если в табличном пространстве EASYDW_DEFAULT или в каком-нибудь другом был создан раздел для информации о покупках за январь и этот раздел нужно перенести в табличное пространство PURCHASES_JAN2002, можно использовать нижеприведенную команду. При указании условия NOLOGGING операция не

будет протоколироваться, что повысит производительность. После выполнения операций с опцией NOLOGGING не забывайте делать резервную копию, поскольку иначе в случае сбоя вы не сможете провести восстановление.

```
SQL> ALTER TABLE purchases MOVE PARTITION purchases_jan2002
     TABLESPACE purchases_jan2002 NOLOGGING
```

Разделы облегчают управление

Большую часть операций по обслуживанию, которые можно выполнять на уровне таблиц, можно также выполнять над одним разделом или подразделом таблицы. Можно делать резервные копии разделов и подразделов, экспортировать их, восстанавливать и возвращать к предыдущему состоянию, не оказывая при этом влияния на другие разделы и подразделы. Чтобы лучше осуществлять независимое управление разделами и подразделами, их следует размещать в отдельных табличных пространствах. Каждое табличное пространство нужно хранить на одном или более отдельном устройстве.

Система управления сводными данными (Summary Management) и оптимизатор запросов используют тот факт, что данные секционированы при выборе оптимальной стратегии обработки запроса или обновления материализованного представления. Система слежения за изменениями разделов (Partition Change Tracking, PCT) для материализованных представлений, которую мы обсуждали в главе 4, отслеживает, какие разделы подверглись модификациям после операции над разделами, и пересчитывает при обновлении материализованного представления только эту его часть. Слежение за изменениями разделов также повышает возможности переписывания запросов под материализованное представление, так как переписываются запросы с использованием только тех разделов материализованного представления, которые не являются устаревшими. Если таблица секционирована, оптимизатор запросов может использовать метод, называемый отбрасыванием разделов (partition elimination) или отсечением разделов (partition pruning), и определить, можно ли ответить на данный запрос, прочитав только некоторые конкретные разделы.

Совет. В примерах, показанных в этом разделе, операции по обслуживанию разделов осуществляются над таблицами, но не забывайте, что разделы также используются для индексов, так что вам придется создавать и обслуживать соответствующие разделы индексов. Кроме того, в данных примерах мы используем разделы, но точно так же можно использовать подразделы.

7.5.3. Изменения индекса

Возможно, что одним из меняющихся аспектов проекта будут индексы. Будут создаваться новые и изменяться уже существующие. Существует склонность создавать в хранилищах много индексов, поскольку система обновляется нечасто. Однако вам следует подумать о том, какое влияние все эти индексы окажут на время загрузки данных.

Если по какой-то причине индекс нужно перестроить, мы предлагаем использовать оператор ALTER INDEX REBUILD, что даст большую производительность, чем удаление и воссоздание индекса.

Многие операции по обслуживанию секционированных таблиц делают непригодными к использованию глобальные индексы, и они помечаются как UNUSABLE. Вам придется перестроить весь глобальный индекс или, если он секционирован, перестроить все его разделы. Чтобы избежать этого, в Oracle 9i Release 2 вы можете включить в оператор ALTER TABLE этой операции условие UPDATE GLOBAL INDEXES. Это условие заставляет базу Oracle обновить глобальный индекс в момент выполнения команды DDL, связанной с обслуживанием таблицы.

Совет. Если данные будут секционированы, вы сможете проводить операции по обслуживанию над отдельными разделами индекса, а не над всем индексом.

7.6. Обновление хранилища

Чрезвычайно важная управленческая задача — это обновление хранилища самыми последними данными. Эта задача обычно выполняется ночью, а данные заключаются в пакеты. Как описывалось выше, часто до того как данные можно будет загрузить в хранилище, их нужно очистить и преобразовать. В главе 5 описываются различные методы, используемые для загрузки новых данных в хранилище.

На всем протяжении этой книги мы активно применяли Oracle Enterprise Manager для управления многими нашими задачами и контроля за ними. В зависимости от сложности задач, а также от зависимостей данных, вы можете предпочесть использовать свои собственные методы. Но если вы этого не хотите, вы можете поместить задание в очередь заданий консоли и на следующий день прийти и проверить, все ли идет хорошо. Например, не так уж необычно полу-

чать данные из разных источников в разное время. Если между данными существуют зависимости, то вам, вероятно, придется управлять ими своими собственными управленческими методами.

Когда данные загружены в хранилище, нужно обновить материализованные представления. Это также может занять значительное время, зависящее от числа материализованных представлений.

В OEM вы можете создать задание по обновлению материализованных представлений. Вы можете обновить все материализованные представления, запустив процедуру dbms_mview.refresh_all_mviews. Кроме того, можно использовать опцию dbms_mview.refresh_dependent, чтобы обновлялись только те материализованные представления, которые зависят от определенных таблиц. Или же можно вызвать процедуру обновления и указать обновляемые материализованные представления. Подробно проведение обновлений материализованных представлений было описано в главе 4.

7.7. Сбор статистики оптимизатора

Отсутствие статистики оптимизатора или устаревшие статистические данные часто являются причиной неоптимальной производительности обработки запросов. Оптимизатор, работающий на основе стоимости, использует для определения стоимости пути доступа такую статистику, как число элементов (cardinality) таблицы, число возможных значений столбца и распределение данных. Стоимость является мерой того, сколько памяти, ресурсов процессора и каналов ввода-вывода потребуется для выполнения запроса. Чтобы эффективно использовать оптимизатор, работающий на основе стоимости, нужно собрать статистику числа элементов (cardinality) и распределения данных для каждой таблицы, индекса и материализованного представления.

Статистика собирается с помощью пакета DBMS_STATS. (Заметьте, что в предыдущих версиях Oracle использовалась команда ANALYZE, которая теперь исключена). Статистические данные можно собирать либо путем чтения всех строк, либо путем проведения оценки на основе чтения только небольшой выборки строк или блоков. В пакете DBMS_STATS предлагаются процедуры для сбора статистики уровня базы данных, схемы или таблицы, а также раздела таблицы.

Чтобы собрать статистику для всех таблиц и индексов схемы EASYDW, выполните такой запрос.

```
SQL> EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS ('EASYDW');
```

Если вас интересует только конкретная таблица, например, PURCHASES, вы можете использовать следующее выражение. При этом будет собрана статистика по таблице, ее столбцам и индексам.

```
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('EASYDW',
                                            'PURCHASES');
```

Увидеть статистику по таблице и столбцу можно, подав запрос к словарным представлениям user_tables и user_tab_cols следующим образом.

```
— статистика по таблице
SQL> SELECT NUM_ROWS, BLOCKS, AVG_ROW_LEN, LAST_ANALYZED
FROM USER_TABLES WHERE TABLE_NAME = 'PURCHASES';

NUM_ROWS      BLOCKS        AVG_ROW_LEN     LAST_ANAL
---------     ----------    -----------     ---------
94113         623           46              07-AUG-02

-- column statistics
SQL> SELECT NUM_DISTINCT, NUM_NULLS, AVG_COL_LEN
FROM USER_TAB_COLS WHERE TABLE_NAME = 'PURCHASES';

COLUMN_NAME              NUM_DISTINCT    NUM_NULLS     AVG_COL_LEN
----------------         ------------    ---------     -----------
PRODUCT_ID                        162            0               7
TIME_KEY                           20            0               8
CUSTOMER_ID                        40            0               9
PURCHASE_DATE                      11            0               8
PURCHASE_TIME                     492            0               4
PURCHASE_PRICE                   1032            0               5
SHIPPING_CHARGE                     3            0               4
TODAY_SPECIAL_OFFER                 2            0               2
```

Чтобы собрать статистику по индексу CUSTOMER_GENDER_INDEX, используйте такой запрос:

```
SQL> EXECUTE DBMS_STATS.GATHER_INDEX_STATS('easydw',
                                'customer_gender_index');
```

Статистику нужно собирать после загрузки данных, а также всегда, когда внесенные изменения могут повлиять на распределение данных, что обеспечит наличие у оптимизатора самых свежих сведений для принятия решений. При использовании секционированных таблиц анализ статистики следует проводить только для самых новых разделов и подразделов, в которые были добавлены строки. Например, если в таблицу PURCHASES был добавлен новый раздел с данными июля 2002 года, статистику для этого нового раздела можно собрать следующим образом.

```
SQL> ALTER TABLE purchases ADD PARTITION purchases_jul2002
     values less than (TO_DATE('31-07-2002', 'DD-MM-YYYY'));

SQL> INSERT /*+APPEND */ INTO purchases ...

SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS('easydw', 'purchases',
                  'purchases_jul2002');
```

Для автоматизации сбора статистики по таблице, включите мониторинг DML-операций в данной таблице с помощью ключевого слова MONITORING в команде CREATE TABLE или ALTER TABLE. После этого используйте опцию GATHER STALE процедуры GATHER_DATABASE_STATS для сбора статистики только для тех таблиц, чье содержимое существенно изменилось. Эту процедуру можно вызывать из периодически запускаемого задания, чтобы статистические сведения оставались свежими.

Чтобы включить автоматический мониторинг DML в таблице PURCHASES, подайте следующий запрос:

```
SQL> ALTER TABLE purchases MONITORING;
```

Кроме этого, мониторинг можно включить для всей схемы или базы данных с помощью процедур ALTER_SCHEMA_TAB_MONITORING или ALTER_DATABASE_TAB_MONITORING. Например, чтобы включить мониторинг DML для всей схемы EASYDW выполните такую команду:

```
SQL> EXECUTE DBMS_STATS.ALTER_SCHEMA_TAB_MONITORING('easydw', TRUE);
```

Чтобы собрать статистику только по тем таблицам, в которых имеется значительное число изменений, выполните такую команду:

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS ('easydw',
                    options=>'GATHER STALE');
```

7.7.1. Сбор системных статистических данных

В Oracle 9i оптимизатор, работающий на основе стоимости, был усовершенствован, и при определении плана выполнения он принимает в расчет статистические данные системы. Системная статистика, т. е. скорость процессора и количество каналов ввода-вывода, дает информацию о доступности ресурсов системы. Это позволяет оптимизатору принимать лучшие решения в условиях конкуренции нескольких пользователей за один ресурс. Системную статистику можно собрать с помощью процедуры DBMS_STATS.GET_SYSTEM_STATS. Как и для любой другой статистики, важно, чтобы оптимизатор получил точную картину загрузки системы. Статистику можно собирать для разных периодов в течение дня, например в нормальные рабочие часы и после завершения рабочего дня. Статистику можно сохранить с помощью процедуры EXPORT_SYSTEM_STATS, чтобы позже импортировать в базу с помощью процедуры IMPORT_SYSTEM_STATS.

7.7.2. Динамический подбор

В Oracle 9i Release 2 введена новая возможность, называющаяся динамический подбор (dynamic sampling), который используется при отсутствии статистических данных по таблице. При динамическом подборе оптимизатор автоматически собирает статистику путем выполнения пробного извлечения данных перед оптимизацией запроса. Динамический подбор полезен при выполнении запросов к таблицам, для которых нет статистических данных или они слишком устарели. Чтобы задействовать динамический подбор, нужно установить инициализационный параметр OPTIMIZER_DYNAMIC_SAMPLING в значение большее, чем 1. Оптимизатор будет выполнять динамический подбор, если в запросе присутствуют более одной таблицы, и некоторые из этих таблиц не имеют статистических данных или индексов. Заметим, что динамический подбор создает определенную нагрузку на ресурсы в ходе процесса оптимизации и его нужно использовать с осторожностью. В хранилищах данных будет хорошей практикой всегда собирать и поддерживать в свежем состоянии статистику для всех таблиц в ходе обслуживающих процедур.

7.8. Параллельные управленческие задачи

На всем протяжении этой книги мы подразумеваем, что типичный большой размер хранилищ означает, что выполнение задач занимает много времени. Мы рассмотрели пока несколько методов повышения производительности, но есть

один очень полезный, который стоит упомянут еще раз: возможность запускать операции параллельно.

Как правило, один процесс выполняет одну инструкцию SQL, но представьте, какой прирост производительности мы получили бы, если бы смогли распределить эту инструкцию между процессами, выполняющимися одновременно. Если вы используете Oracle 9i Enterprise Edition, эта возможность у вас есть.

Параллельное выполнение наиболее эффективно для операций, осуществляющих доступ к большим объемам данных, таких, как запросы, создание индексов, вставка массивов данных, обновления, удаления, агрегирование и перенос данных. С каждой новой версией в базе Oracle повышается количество операций и команд, которые можно запускать параллельно. На момент написания этой книги, параллельно может работать большинство операций:

- Параллельная обработка запросов и подзапросов в операторах SELECT.
- Параллельное выполнение команд DLL, в том числе:
  - CREATE TABLE AS SELECT, CREATE INDEX и ALTER INDEX REBUILD;
  - для секционированных таблиц: ALTER TABLE MOVE, SPLIT, COALESCE;
  - для секционированных индексов: ALTER INDEX REBUILD или SPLIT.
- Параллельные операции DML — INSERT, UPDATE, DELETE, многотабличная вставка.
- SQL*Loader и внешние таблицы.

Для оптимальной производительности параллельных операций требуется точная статистика. В главе 3 обсуждалось использование параллелизма для повышения производительности запросов, а в главе 5 были приведены примеры параллельной загрузки данных. Похожие методы используются и для параллельного выполнения операторов DDL.

7.9. Настройка хранилища

Не следует забывать, что хранилище данных нужно настраивать, как и любую другую базу данных, хотя настройку можно проводить и по следующим причинам:

- повышение производительности загрузки;
- повышение производительности обработки запроса;
- повышение производительности обновления данных;
- повышение производительности выполнения задач по обслуживанию.

Когда люди думают о настройке, первая их мысль, вероятно, будет о повышении производительности обработки запросов. Для решения этой проблемы используются традиционные методы, такие, как создание индексов и размещение данных в буферном кэше, чтобы их можно было быстрее прочесть. В хранилищах данных индекс остается чрезвычайно важным элементом, но размещение данных в буферном кэше вряд ли поможет, поскольку поиск в хранилищах ведется по огромным объемам данных.

Поэтому одним из лучших методов повышения производительности обработки запросов является создание материализованных представлений. Если запрос используется несколько раз и если создано материализованное представление, в котором заранее вычислен ответ на запрос, оптимизатор при обработке такого запроса возьмет результаты из материализованного представления, а не из основных таблиц. Если таблица велика, это даст очень значительный прирост производительности обработки.

7.9.1. Инструменты для настройки

Существует множество средств для настройки базы данных Oracle, как от самой компании Oracle, так и от других производителей. Если вы приобретете дополнительный настроечный пакет для Oracle Enterprise Manager, у вас окажутся следующие инструменты.

- Oracle Expert — создает дизайн базы данных (только для NT-систем).
- SQL Analyze — выявляет и настраивает интенсивно использующие ресурсы операторы SQL, предоставляет графический вариант EXPLAIN PLAN (только для NT).
- Tablespace Map — выявляет проблемы с управлением дисковым пространством.
- Reorg Wizard — устраняет проблемы использования дискового пространства, изменяет размер и местоположение объектов.
- Outline Management — управляет сохраненными планами обработки запросов.

Еще несколько полезных средств настройки содержаться в диагностическом пакете (Diagnostic Pack).

- Performance Manager — показывает, как используется база данных, включая ресурсы процессора, памяти и ввода-вывода.
- TopSessions — показывает использование базы данных пользователем.
- Capacity Planner — собирает базовую статистику по производительности.

Есть несколько программ-консультантов, которые дадут вам советы по улучшению производительности.

- Summary Advisor — дает рекомендации по материализованным представлениям (поставляется с версией Enterprise).
- Index Tuning Wizard — дает рекомендации по индексам (только для NT).
- Buffer Cache Advisor — определяет оптимальный объем буферного кэша и разделяемого пула.

Когда база данных используется в качестве хранилища, забывать о ее настройке просто непростительно. Если вы знакомы с настройкой OLTP-систем, то здесь методы будут немного отличаться, хотя конечный результат будет тот же — повышение производительности обработки запросов.

Настройка — это своего рода «черная магия», и невозможно на нескольких страницах рассказать, как следует настраивать хранилище. Вместо этого мы предлагаем вам познакомиться с имеющимися средствами настройки и определить, какую информацию вы бы хотели получить. Например, насколько были задействованы каналы ввода-вывода при выполнении данного запроса. После этого вы поймете, как получить эту информацию от средства настройки (если это возможно).

Испробуйте этот метод в контролируемой среде на небольшой системе, перед тем как пытаться анализировать рабочую систему. Продолжайте практиковаться, чтобы чувствовать себя более уверенно, а потом, когда вам действительно придется проводить настройку, это уже будет не так сложно. Существует масса книг по настройке баз Oracle, и они должны дать вам полезные советы по настройке хранилища.

7.9.2. Стабильность плана

Одним из значительных преимуществ использования реляционной базы данных является то, что вам не нужно при составлении запроса указывать, каким образом надо осуществлять доступ к данным. Наиболее эффективный план обработки запроса определяет оптимизатор запросов. Как правило, оптимизатор выбирает наилучшую стратегию, но могут возникнуть проблемы после инсталляции новых версий программного обеспечения, или когда меняется статистика по числу элементов (cardinality) базы, поскольку в этом случае оптимизатор может выбрать другую стратегию. Следовательно, существует риск, что выполнение запроса может вдруг занять намного больше времени. Одним из решений этой проблемы является использование инструкций (hints), но это подразумевает внесение изменений в код, что не всегда возможно.

Для преодоления этой проблемы в Oracle 8i было введено новое свойство, называемое стабильностью плана (plan stability). Это свойство позволяет создавать сохраняемый план обработки запроса (outline) для конкретного оператора SQL или же для всех операторов. Когда стабильность плана задействована, каждый запрос сравнивается с каждым запросом в сохраненном плане. Если запрос в точности совпадает с запросом, содержащимся в сохраненном плане, тогда используется стратегия данного плана.

Создание сохраненного плана

Сохраненный план (outline) состоит из набора инструкций (hints), представляющих собой план обработки запроса для SQL-выражения. Эти инструкции похожи на те, которые вы можете добавлять в запрос для того, чтобы повлиять на выбор оптимизатором стратегии. Сохраненные планы можно создавать либо для всех SQL-выражений, либо для некоторых конкретных.

Сохраненные планы можно группировать в категории, чтобы легче было организовывать их и управлять ими. Чтобы создать сохраненный план для всех SQL-выражений, установите параметр CREATE_STORED_OUTLINES в true или присвойте ему имя категории, как это показано в следующем примере, где именем категории является EASYDW_CAT.

```
SQL>ALTER SESSION SET CREATE_STORED_OUTLINES = EASYDW_CAT
```

После этого сохраненные планы будут автоматически создаваться для каждого выражения SQL. Этот способ легкий, но при этом может получиться большое количество сохраненных планов. Этот параметр можно включать при запуске приложения, а выключать при завершении его, установив в false параметр CREATE_STORED_OUTLINES.

Кроме этого, можно создавать сохраненные планы для конкретных выражений с помощью оператора CREATE OUTLINE или средства Outline Management в Oracle Enterprise Manager, как это показано на рис. 7.45.

Как было показано выше, с помощью инструкции CREATE_STORED_OUTLINES были автоматически созданы три сохраненных плана с именами, начинающимися с SYS_OUTLINE. Заметьте, как планы создаются для каждого SQL-выражения, включая те, которые, по всей видимости, и не нужны, например SELECT * FROM USER_OUTLINES.

Сохраненный план с именем SUM_OF_SALES, показанный на рис. 7.45, был создан для конкретного SQL-выражения, после того, как мы выбрали пункт Create (Создать) из меню Outlines (Сохраненные планы). Код SQL для этого плана показан в текстовом окне.

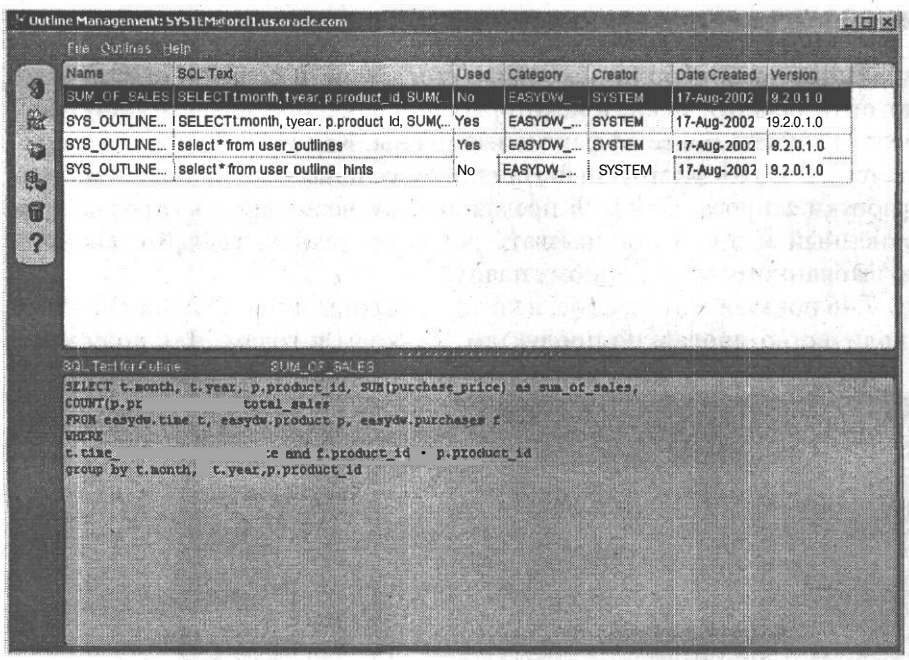


Рис. 7.45. *Outline Management*

Использование сохраненного плана

Чтобы использовать сохраненный план, присвойте параметру USE_STORED_OUTLINE значение true или имя категории. Когда база Oracle компилирует инструкцию SQL, она выясняет, имеется ли сохраненный план, содержащий точно такой же текст, какой имеется в запросе. Если такой обнаруживается, тогда при выполнении запроса используется стратегия этого плана.

```
SQL> ALTER SESSION SET USE_STORED_OUTLINES = easydw_cat
```

В примере на рис. 7.45 два сохраненных плана были использованы как минимум однажды, а два еще не были использованы. Чтобы отключить использование сохраненных планов, присвойте параметру USE_STORED_OUTLINE значение false.

Редактирование сохраненного плана

Хотя оптимизатор, работающий на основе стоимости, в большинстве случаев генерирует оптимальный план, в некоторых случаях с помощью ручной настройки можно получать более эффективные планы. Как уже говорилось выше, вы можете использовать инструкции (hints) или, если вы хотите изменить стратегию обработки запроса, Oracle 9i предлагает вам возможность отредактировать сохраненный план. Чтобы вызвать редактор планов, сделайте двойной щелчок по выбранному сохраненному плану.

На рис. 7.46 показан SQL-запрос, в котором вычисляется суммарный объем и общее количество продаж по продуктам, месяцам и годам. Для получения

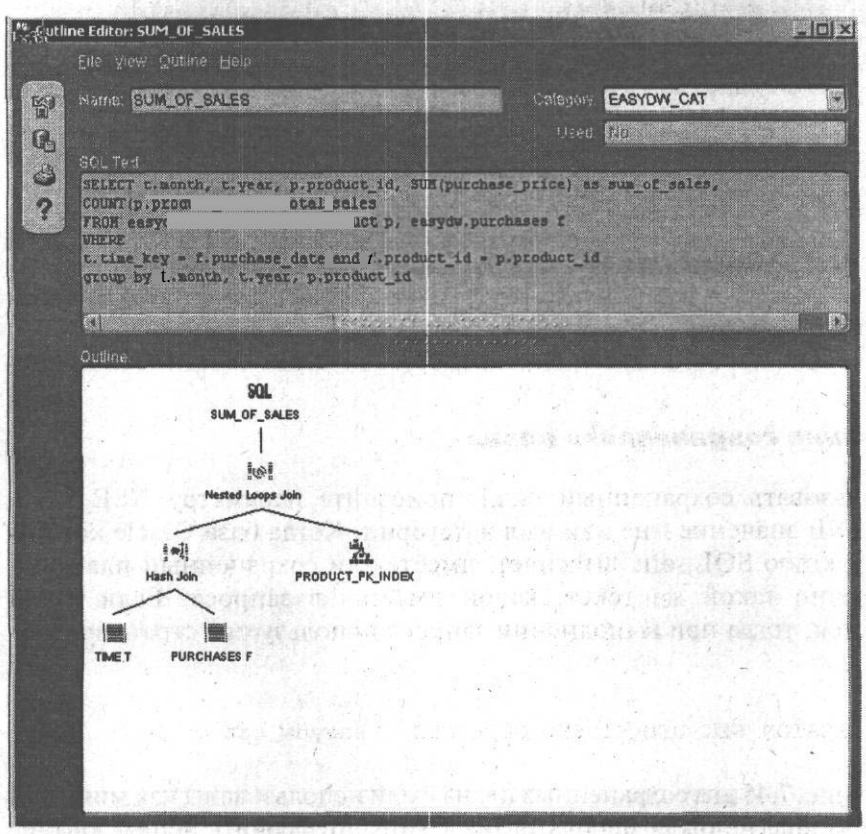


Рис. 7.46. *Исходный сохраненный план*

результата нужно соединить таблицы TIME (время), PRODUCT (продукты) и PURCHASES (покупки). Выбранная оптимизатором стратегия показана в графическом формате. Таблица PURCHASES соединяется с таблицей TIME с помощью хэш-соединения. Полученный результат соединяется с таблицей PRODUCT соединением вложенных контуров (nested loops join) с помощью индекса на основе первичного ключа PRODUCT_PK_INDEX.

Этот запрос также может быть обработан при изменении последовательности соединения: сначала соединить таблицу TIME с таблицей PRODUCT, а затем соединить полученный результат с таблицей PURCHASES.

Последовательность соединений можно изменять в меню Outlines (Сохраненные планы). На рис. 7.47 показано, как можно выбрать узел и перемещать его вниз или вверх. В приводимом примере, узел PURCHASES был перемещен вниз.

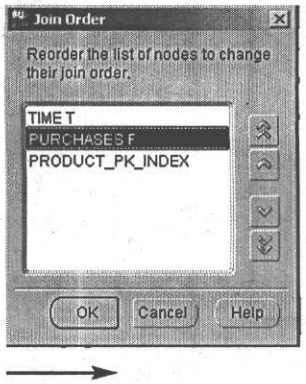


Рис. 7.47. *Изменение порядка соединений*

Теперь у нас есть новая стратегия, показанная на рис. 7.48. Таблица TIME будет соединена с таблицей PRODUCT соединением вложенных контуров с помощью первичного ключа таблицы PRODUCT. Результат будет соединен с таблицей PURCHASES с помощью хэш-соединения.

Вы можете проверить соединения и запустить запрос, оценивая, будет ли он выполняться быстрее (рис. 7.48). Сравните истекшее время, затраты времени процессора и другие статистические показатели с исходной стратегией и определите, сделало ли ваше редактирование план лучше (или хуже).

Если вы не используете Oracle Enterprise Manager, вы можете редактировать сохраненные планы с помощью пакета DBMS_OUTLN_EDIT. Чтобы увидеть созданные пользовательские планы, изучите представление USER_OUTLINES, показанное в следующем фрагменте кода.

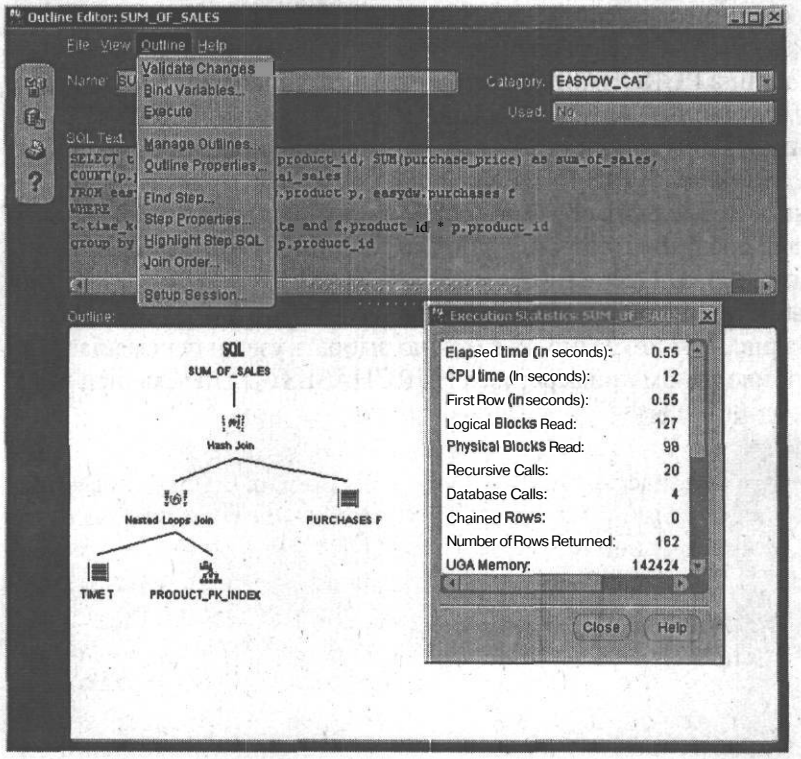


Рис. 7.48. *Выполнение запроса с отредактированным планом*

```
SQL> SELECT NAME, USED, SQL_TEXT FROM USER_OUTLINES;

NAME                           USED     SQL_TEXT
------------------------------ -------- ------------------------------------
SYS_OUTLINE_020817142          USED     SELECT t.month, t.year, p.product_id
SYS_OUTLINE_020817142          USED     select * from user_outlines
SYS_OUTLINE_020817142          UNUSED   select * from user_outline_hints
SUM_OF_SALES                   UNUSED   SELECT t.month, t.year, p.product_id
```

По умолчанию сохраненные планы записываются в таблицы OL$, OL$HINTS и OL$NODES табличного пространства SYSTEM схемы OUTLN. Вы, вероятно, замечали эту схему в вашей базе, начиная с версии Oracle 8i. Если сохраненных планов много, они могут занять в табличном пространстве SYSTEM слишком

много места, и их можно перенести. Последовательность действий для этого см. в документации Oracle.

В заключение скажем, что если вас заботит изменение стратегии обработки запроса, особенно при обновлении программного обеспечения, подумайте о создании нескольких сохраненных планов для наиболее важных запросов. Когда только они созданы и записаны в базу, использовать их не обязательно. Их можно держать на всякий случай и задействовать при необходимости. Сохраненные планы также полезны при создании приложений, размещаемых в нескольких местах, чтобы обеспечивать одинаковую стратегию работы оптимизатора даже при различных количествах элементов (cardinality) в данных.

7.10. Обеспечение безопасности

Не следует забывать о безопасности хранилища. Во многих отношениях безопасность для данных в хранилищах должна быть даже выше, чем для данных в рабочих системах, поскольку ценность этих данных для ваших конкурентов может быть огромной. Представьте, что один из ваших пользователей начал зарабатывать деньги, делая отчеты по вашим данным, или что служащий ушел из компании, прихватив данные из хранилища. Если вы смеетесь над последней фразой, пожалуйста, не делайте этого, хотя такое случается крайне редко, оно все же случается.

Степень безопасности хранилища зависит от того, что вы считаете подходящим для своей системы. В некоторых местах предпочитают открывать хранилища для запросов только для чтения, а возможность выполнять управленческие задачи, такие как создание сводок, предоставлять только группе администрирования.

В главе 2 мы уже видели, как можно защищать отдельные таблицы, явно указывая, кто может иметь к ним доступ с помощью команд GRANT и REVOKE.

Еще один метод — это создание роли, назначение этой роли определенных привилегий, а затем предоставление этой роли пользователю. Роли можно создавать на SQL или с помощью Oracle Security Manager.

В примере на рис. 7.49 создается роль с названием EASY_USER, в которой мы указываем таблицы базы, к которым такой пользователь может иметь доступ. Такая система безопасности не ограничивается только таблицами. Ее можно применять к широкому диапазону объектов базы данных. Для тех, кто не желает использовать OEM, в нижней части рис. 7.50, в окне кода SQL, приведены команды, реализующие данную роль.

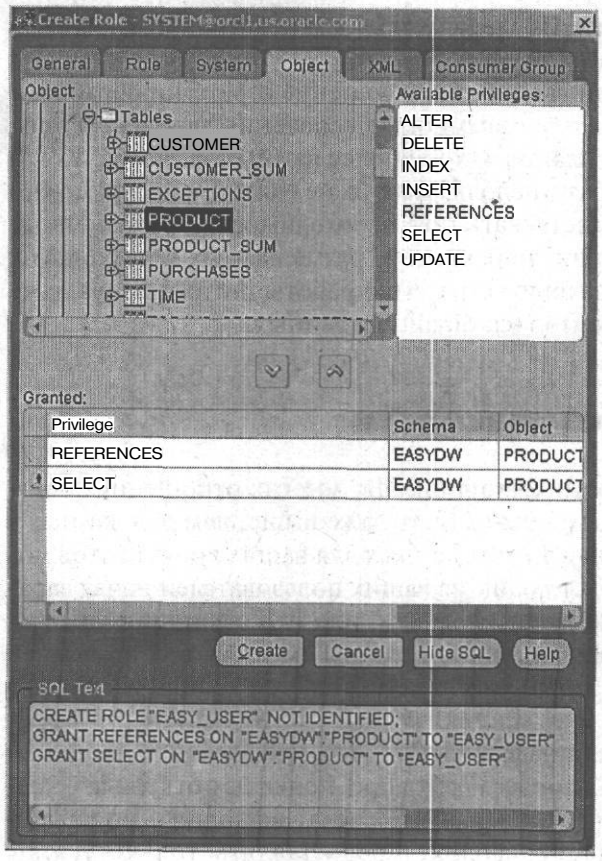


Рис. 7.49. *Создание роли*

Когда роль определена, ее можно назначить пользователю, как это показано на рис. 7.50. Роли — это чрезвычайно сильная возможность Oracle, которая может сэкономить массу времени, поскольку не придется давать отдельным пользователям массу специфических привилегий.

Виртуальная частная база данных

Приложения, которые выносятся в Интернет и делают информацию доступной для клиентов, поставщиков, служащих и прочих пользователей должны обеспечивать очень тонкий контроль доступа. Он-лайновая банковская система должна

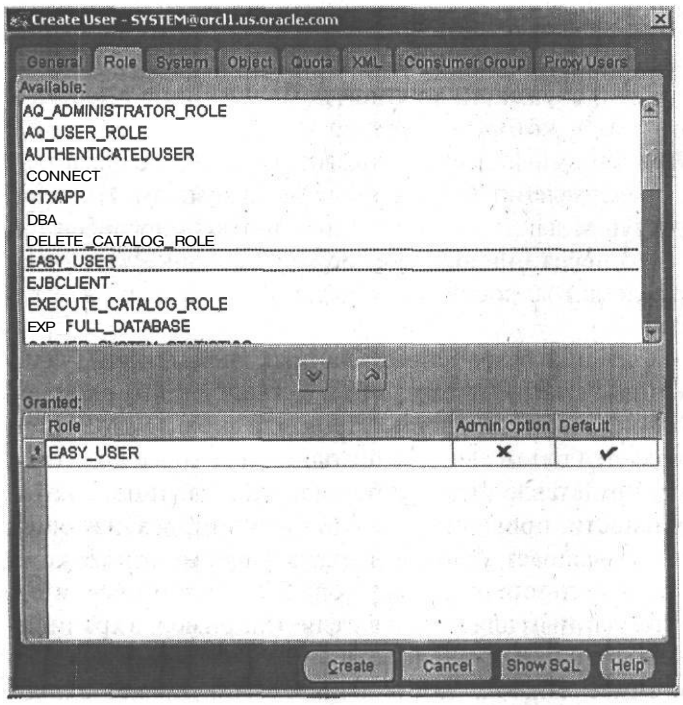


Рис. 7.50. *Назначение роли пользователю*

делать так, чтобы клиенты видели транзакции только для своего счета, и ничьего больше. Система учета кадров может позволять служащим просмотреть запись о себе, изменить свой воинский статус, адрес и номер телефона, но не зарплату. То же самое приложение может предоставлять управленческому персоналу возможность читать и изменять записи всех работающих на них служащих.

По мере того как компании растут и концентрируются на своих основных задачах, они могут привлекать к прочим задачам, таким, как кадровое обеспечение, работа с клиентами и платежные операции, другие компании. При разработке приложения, обеспечивающего размещение информации, данные для каждой компании следует держать отдельно и защищенными друг от друга.

По традиции, контроль доступа осуществляется на уровне объектов. Политика безопасности данных определяет, какие пользователи к каким объектам схемы имеют доступ и какие действия разрешено проводить с каждым объектом. Например, пользователь может читать данные из таблицы, но не может вставлять, изменять или удалять строки. Если вы хотите дать пользователю доступ к огра-

ниченной части строк таблицы, можно создать представление и дать пользователю доступ к этому представлению. Если у вас есть приложение с системой самообслуживания, в которой каждый служащий крупной компании имеет доступ к тем строкам базы кадрового учета, которые содержат его собственные персональные данные, то каждый служащий должен иметь свое собственное представление (view). Однако число представлений быстро стало бы огромным. Если вы, например, хотите давать доступ к данным о служащих только определенной группе пользователей и только через приложение, осуществляющее кадровый учет, и не давать его для проведения запросов, то представления не дадут вам такой возможности.

Еще одним способом реализации безопасности данных является виртуальная частная база данных (Virtual Private Database, VPD) с тонким контролем доступа. VPD обеспечивает безопасность на уровне строк для всех приложений. Виртуальная частная база создается путем связывания одной или более политики безопасности с таблицей или представлением. Любой доступ к таблице, с которой связана политика безопасности, приводит к вызову функции, реализующей данную политику. Функция возвращает условие доступа в форме инструкции WHERE, которая добавляется к пользовательскому коду SQL, динамически изменяя, таким образом, доступ к данным для пользователя. Например, в хранилище EASYDW мы можем позволить каждому клиенту видеть в режиме он-лайн информацию о сделанных ранее заказах. Когда пользователь подает запрос SELECT * FROM PURCHASES, функция добавит в условие WHERE идентификатор покупателя (customer_id), что даст в результате такой запрос:

```
SELECT * FROM PURCHASES WHERE customer_id = 'AB123459'
```

Часто бывает нужно контролировать доступ, основываясь на нескольких атрибутах пользователя, таких, как код профессии, подразделение, местоположение, и по тому, является ли пользователь клиентом или партнером. Для этого создается контекст приложения (application context). Контекст приложения связывается с сеансом работы пользователя при его входе в базу данных. Каждое приложение может иметь свой контекст, имеющий свои атрибуты.

После создание контекста приложения создаются функции на PL/SQL, которые реализуют политики безопасности. Функция определяет, какое условие WHERE вернуть, основываясь на контексте приложения пользователя.

Для управления политиками безопасности и для применения их к соответствующим таблицам используется пакет на PL/SQL, называющийся DBMS_RLS. С помощью этого пакета вы можете добавлять, удалять, включать, отключать и обновлять создаваемые политики безопасности.

Многие из приложений Oracle используют VPD для обеспечения тонкого контроля доступа.

7.11. Мониторинг дискового пространства

Очень важной для администратора базы данных является информация о том, сколько свободного места имеется для хранилища. Одним из методов, позволяющих избежать исчерпания свободного места является создание саморасширяющихся (autoextend) файлов данных и установка для них неограниченного числа расширений. Однако, это не поможет, если место на диске действительно закончится. Следовательно, в своем повседневном слежении за состоянием хранилища вы должны проверять наличие свободного места. Делать это можно с помощью Storage Manager, входящего в OEM, или же можно просто подавать запросы к метаданным базы Oracle. На рис. 7.51 показан пример использования места на диске для экземпляра базы ORCL.

Name	Type	Extent Management	Size (M)	Used (M)	Used %
CWMLITE	PERMANENT	LOCAL	20.000	9.375	46.88
DIMENSIONS	PERMANENT	LOCAL	5.000	0.141	2.81
DRSYS	PERMANENT	LOCAL	20.000	9.688	48.44
EASYDW_DEFAULT	PERMANENT	LOCAL	5.000	0.109	2.19
EXAMPLE	PERMANENT	LOCAL	148.750	148.625	99.92
INDX	PERMANENT	LOCAL	5.000	0.406	8.13
ODM	PERMANENT	LOCAL	20.000	9.313	46.56
OEM_REPOSITORY	PERMANENT	LOCAL	35.008	32.008	91.43
PURCHASES_FEB2002	PERMANENT	LOCAL	5.000	0.563	11.25
PURCHASES_FEB2002_IDX	PERMANENT	LOCAL	3.000	0.063	2.08
PURCHASES_JAN2002	PERMANENT	LOCAL	5.000	0.125	2.50
PURCHASES_JAN2002_IDX	PERMANENT	LOCAL	3.000	0.063	2.08
PURCHASES_MAR2002	PERMANENT	LOCAL	5.000	0.188	3.75
PURCHASES_MAR2002_IDX	PERMANENT	LOCAL	1.000	0.063	6.25
PURCHASES_Q1_2002	PERMANENT	LOCAL	5.000	0.125	2.50
RBS	PERMANENT	LOCAL	4.000	0.063	1.56
SUMMARY	PERMANENT	LOCAL	6.000	0.094	1.56
SYSTEM	PERMANENT	LOCAL	405.000	398.500	98.40
TOOLS	PERMANENT	LOCAL	10.000	6.063	60.63
UNDO	PERMANENT	LOCAL	8.000	0.063	0.78
UNDOTBS1	UNDO	LOCAL	200.000	16.500	8.25
USERS	PERMANENT	LOCAL	25.000	0.063	0.25
XDB	PERMANENT	LOCAL	38.125	37.938	99.51
TEMP	TEMPORARY	LOCAL	40.000	12.000	30.00

Рис. 7.51. *Использование места на диске*

Совет. Добавьте в OEM событие, уведомляющее администратора базы о приближении к пределам имеющегося свободного места.

В базе Oracle существует довольно много системных таблиц, содержащих полезную информацию о базе данных. Подробное описание всех этих таблиц и содержащейся в них информации можно найти в документации к базе данных Oracle.

7.11.1. Автоматизированное управление дисковым пространством

Введенные усовершенствования упростили процесс управления дисковым пространством. Табличные пространства с локальным управлением, появившиеся в Oracle 8i, устранили необходимость в реорганизации табличных пространств для устранения фрагментации. В Oracle 9i табличные пространства можно создавать с условием SEGMENT SPACE MANAGEMENT AUTO, что приводит к автоматическому управлению свободным пространством в пределах сегмента базы данных, такого, как таблица или индекс. Автоматическое управление отменами (undo) устраняет необходимость вручную управлять используемыми при этом сегментами. В Oracle 9i для создания и удаления файлов, необходимых для данных, он-лайновых протоколов и управляющих файлов можно использовать «файлы управляемые Oracle» (Oracle managed files, OMF), что устраняет необходимость в ручном управлении ими. Все, что от вас требуется, — это указать местоположение для этих файлов.

7.11.2. Возобновляемое выделение места

Длительные операции по обновлению или добавлению данных в базу могут прерваться при исчерпании свободного места. Повторное выполнение этой процедуры может занимать много времени, особенно если ошибка возникла на завершающих этапах. Новая возможность Oracle 9i — возобновляемые выражения — делают возможным вмешательство и исправление ошибки в ходе выполнения операции. Когда проблема, вызвавшая ошибку, исправлена, операция автоматически возобновляется.

Выполнение возобновляемого выражения приостанавливается при возникновении следующих ошибок:

- исчерпание свободного места;
- достижение максимального числа расширений;
- исчерпание квоты на дисковое пространство для пользователя.

Возобновляемыми операциями могут быть:

- операторы SELECT при исчерпании места для хранения временных данных (для сортировки);
- операторы DML — INSERT, UPDATE и DELETE;
- импорт/Экспорт;
- SQL*Loader;
- операторы DDL - CREATE TABLE ... AS SELECT, CREATE INDEX, ALTER INDEX ... REBUILD, ALTER TABLE ... MOVE PARTITION, ALTER TABLE ... SPLIT PARTITION, ALTER INDEX ... REBUILD PARTITION, ALTER INDEX ... SPLIT PARTITION, CREATE MATERIALIZED VIEW и CREATE MATERIALIZED VIEW LOG.

Для сеанса необходимо включить режим возобновления. При желании, вы можете указать лимит ожидания (timeout). Если условие, вызвавшее ошибку, не было устранено за этот период, операция отменяется. Также вы можете указать имя, которое поможет идентифицировать отложенный сеанс в представлениях USER_RESUMABLE или DBA_RESUMABLE. В следующем примере период ожидания устанавливается равным 3600 секунд (1 час), а возобновляемым выражениям данного сеанса присваивается имя data warehouse load.

```
SQL> ALTER SESSION ENABLE RESUMABLE
     TIMEOUT .3600 NAME 'data warehouse load';
```

Для SQL*Loader и утилит импорта/экспорта предлагаются те же параметры, что и для командной строки.

Когда возобновляемая операция откладывается, сообщение об ошибке посылается в протокол предупреждений. Также система генерирует системное событие AFTER SUSPEND. Пользователь может зарегистрировать триггеры для данного события как на уровне базы, так и на уровне схемы. Триггеры можно использовать для того, чтобы уведомить администратора базы о прерывании операции, чтобы можно было произвести корректирующие действия.

Информацию о статусе возобновляемых выражений можно увидеть в представлениях DBA_RESUMABLE и USER_RESUMABLE, как это показано в приведенном примере. Загрузка данных в хранилище была отложена, поскольку оказалось недостаточно свободного места для расширения PURCHASE_PRODUCT_INDEX.

```
SQL> SELECT STATUS, NAME, ERROR_MSG FROM DBA_RESUMABLE

STATUS        NAME          ERROR_MSG
---------     -----         ---------
SUSPENDED     Data          ORA-01683: unable to extend index
              Warehouse     EASYDW.PURCHASE_PRODUCT_INDEX
              Load          partition PURCHASES_FEB02 by 2 in
                            tablespace INDX
```

Когда выполнение выражения откладывается, сеанс, вызвавший данное выражение, переводится в состояние ожидания, а в представление V$SESSION_WAIT добавляется строка, как показано в следующем примере.

```
SQL> SELECT EVENT, STATE FROM V$SESSION_WAIT;

EVENT                                                    STATE
-------------------------------------------------------  --------------
statement suspended, wait error to be cleared            WAITING
```

Когда свободное место добавлено, сеанс автоматически возобновляется. Если этого не происходит в течение периода времени ожидания, возникает ошибка.

7.12. Другие проблемы управления

Мы рассмотрели здесь несколько управленческих задач. Описанные задачи никоим образом не являются исчерпывающим списком, и вполне могут найтись такие, которые не упомянуты здесь, но применимы в ваших условиях. Здесь мы упомянем еще несколько дополнительных задач, которые вы можете принять во внимание.

7.12.1. Создание тестовой системы

Когда людям предлагают создать тестовую систему, весьма часто многие из них поднимают руки вверх и говорят: «Невозможно». Тем не менее, лучше на мгновение остановиться и подумать, какие последствия будет иметь для бизнеса отсутствие тестовой системы. Если тестовая система имеется, ее можно использовать для множества задач, таких как:

- тестирование новых версий программного обеспечения;
- определение времени загрузки данных;
- оценка времени выполнения задач по управлению базой;
- отработка задач по управлению;
- тестирование скриптов перед выполнением их в рабочей системе;
- оценка влияния задач по обслуживанию (например, перестройки индекса);
- определение времени обработки запроса.

Многие думают, что тестовая система должна быть идентична рабочей системе, но в случае хранилища это почти невозможно. Следовательно, все что нам требуется, — это маломасштабная версия, являющаяся образцом реального хранилища. В идеале, получаемые в тестовой системе показатели должны легко масштабироваться, чтобы можно было определять эффект на рабочую систему.

Данные в тестовом хранилище должны, по возможности, быть образцом реальных данных. Может случиться так, что для получения желаемого эффекта вам потребуется извлечь данные из основного хранилища и загрузить их в тестовое.

Теперь мы обсудим различные способы использования тестовой системы.

7.12.2. Тестирование нового программного обеспечения

Когда база данных становится критичным компонентом бизнеса, и получаемая из нее информация служит для принятия критических решений, никто не захочет ставить бизнес под угрозу, устанавливая новое программное обеспечение, которое может привести к появлению проблем. Поэтому, если у вас будет тестовая система, вы можете проверить, остаются ли теми же самыми все важные части используемого программного обеспечения. Диапазон элементов, которые нужно проверить, может быть очень широким. Например, нужно будет удостовериться в том, что для всех запросов оптимизатор использует ту же самую стратегию, что и ранее. Если стратегия изменилась, нужно будет создать сохраненные планы, чтобы обеспечить соответствие времени обработки запроса.

Тем не менее, после установки новой версии Oracle, нужно всегда убедиться в том, что при использовании сохраненных планов производительность не упадет. Производительность может вырасти после изменения оптимизатора, поэтому всегда поверяйте текущую стратегию.

Еще одна важная проверка — это убедиться в том, что те ключевые свойства, которые вы применяете, функционируют, как и ранее. Например, если вы пол-

ностью полагаетесь при обслуживании данных на операции над разделами, убедитесь в том, что они работают также.

Используемые утилиты также нужно проверить, чтобы в них не оказалось изменений, которые могли бы изменить их поведение.

Если создать скрипт, выполняющий все эти необходимые задачи, тогда при каждом обновлении программного обеспечения вам нужно будет лишь запустить его и проверить результаты его работы. Хотя поначалу построение таких скриптов может потребовать значительных усилий, но после этого их можно будет использовать многократно, и вы будете знать, что все, что является для вас важным, будет проверено.

7.12.3. Определение времени загрузки данных

Тестовая система дает идеальную возможность определить время загрузки данных и отработать процесс очистки данных, который может оказаться необходим. Вообще, самым быстрым способом загрузки данных в хранилище Oracle 9i является использование утилиты SQL*Loader в режиме прямого пути. Однако вы можете сравнить производительность этого метода с использованием внешних таблиц, особенно если вам необходимо также проводить преобразования.

Совет. Не забывайте проверять протоколы заданий SQL*Loader в случае возникновения любых проблем по ходу работы (например, невключенные ограничения).

7.12.4. Оценка/отработка задач по управлению

Теперь существует идеальная возможность отработать и апробировать все управленческие задачи перед выполнением их в работающей системе. Когда нужно будет тестировать процессы резервного копирования и восстановления, сэкономить время вам поможет использование сначала небольшой базы, а затем, когда вы будете уверены в правильной работе всех процедур, можно будет переходить и к полноразмерной.

7.12.5. Определение времени обработки запроса

Тестовая система дает идеальную возможность увидеть хранилище данных в использовании до того, как пользователи получат к нему доступ. Вы можете столкнуться с тем, что некоторые запросы, даже при ограниченном числе пользователей, обрабатываются непроизводительно. Следовательно, у вас будет время для выявления причины плохой производительности и для решения проблемы путем, например, добавления нового индекса или создания материализованного представления для этого запроса.

В тестовой системе вы вряд ли увидите воспроизведение использования основного хранилища. Тем не менее, проблемы будут выявляться, и легче исправлять их сейчас, чем когда пользователи будут давить на вас, требуя скорейших отчетов.

7.13. Заключение

В этой главе мы рассмотрели некоторые задачи, необходимые для управления хранилищем, и использование Oracle Enterprise Manager, упрощающего управление. Обсуждались повседневные задачи, такие, как резервное копирование, сбор статистики оптимизатора, слежение за использованием дискового пространства и обновление материализованных представлений. Был предложен метод периодической реорганизации с помощью операций над разделами. Освещены важные вопросы разработки тестовой системы и плана непрерывного функционирования бизнеса.

Это была длинная глава, но мы надеемся, что вы начали хорошо разбираться в тех задачах, которые составляют управление базой данных. Многие из описанных здесь задач также применимы к традиционной OLTP-базе. Различие задач определяется именно большими размерами хранилища. Помните, что лучше взяться за управление сейчас, чем вообще никогда. Отсутствие резервного копирования или слежения за использованием свободного места может привести к катастрофическим результатам, обычно как раз тогда, когда времени совсем не будет хватать.

8

Хранилища информации и Web

8.1. Общий обзор

Начав читать эту главу, вы можете удивиться: какое отношение имеет Web к хранилищам информации? Одной из проблем является то, что Интернет или Web — это термин, который часто используют, но у которого есть несколько интерпретаций. Чтобы разделить внутренние сети Web и Всемирную паутину (World Wide Web), для описания глобальных соединений компьютеров с общим доступом используется термин Интернет, а для описания часто располагающихся «позади» Интернет сетей отдельных компаний, которые доступны только для членов соответствующих организаций — термин Интранет. Именно Интранет и интересует нас прежде всего. Большинству из нас, когда мы впервые задумываемся о Web, в голову, вероятно, придет поиск продукта данной компании и размещение заказа, но какое отношение это имеет к хранилищу? Вас, наверное, удивит то, насколько важна может быть сеть Web для хранилища информации.

8.1.1. Интернет и Интранет

В настоящее время большинство компаний имеют свое представительство во Всемирной паутине, но многие люди могут и не представлять, что внутри компании может быть огромный внутренний web-сайт, который используется только служащими и через который проходит весь бизнес. Сама компания Oracle — хороший пример фирмы, использующей Интранет для общения и внутренних деловых операций. Например, когда служащим Oracle требуется новый портативный компьютер или мобильный телефон, они входят во внутреннюю систему снабжения и совершают покупку, точно так же, как если бы они делали это через Интернет. В некоторых странах информация о налоговых отчислениях и годовые налоговые

декларации доступны только в режиме он-лайн. Многочисленные отчеты, сведения о состоянии и спецификации продуктов доступны с помощью такого браузера, как Internet Explorer через web-системы. Следовательно, служащий компании Oracle может легко и быстро получить доступ к информации, независимо от того, в какой точке мира он находится, если он соединится с сетью компании. Что же компания размещает в своей сети Интранет? Число вариантов, конечно, бесконечно, но вот несколько идей:

- пресс-релизы;
- информацию о продуктах;
- программное обеспечение для скачивания;
- информацию о налоговых отчислениях и HR-системы;
- системы для совершения покупок;
- телефонные справочники;
- информацию о продажах;
- исторические сведения;
- тенденции и анализ;
- разного типа отчеты и документы.

На первый взгляд, этот перечень выглядит как описание типичного Интернет-сайта, но последние пункты — информация о продажах, исторические сведения, тенденции и анализ отличают этот сайт от обычного и показывают роль, которую тут играет хранилище информации.

Перед тем как мы рассмотрим способы использования сети Интранет, нужно сказать, что предметом беспокойства многих людей при публикации данных в Web, даже если сеть внутренняя, является вопрос безопасности данных. Как правило, внутренние сети компаний не соединены с Интернет и доступ к ним возможен только из офиса компании, через коммутируемый канал или, вне офиса компании, через безопасное VPN-соединение. Следовательно, информация будет в безопасности и неуязвима для хакеров. Конечно, если вас волнует безопасность внутри вашей организации, информацию можно защитить и так, чтобы важные сведения мог просматривать только определенный персонал.

Если принять, что наши данные в безопасности, каковы могут быть основания размещения хранилища данных в сети Интранет компании? Ну, например, без Интранет, если менеджеру был необходим отчет, скажем, о продажах за этот месяц, он должен был запросить его, а затем ждать, пока ему принесут его в напечатанном виде.

Интранет все изменил, поскольку теперь можно создавать стандартные отчеты и публиковать их в Интранет, и менеджеры могут просматривать эти отчеты

в удобное для них время. Отчеты можно легко сделать доступными сразу после их создания для любого служащего, неважно, где он находится. Больше не нужно ждать, пока отчет придет по внутренней электронной почте. Если вы сегодня не в офисе, а работаете в отеле, дома или в каком-нибудь другом месте, вы все равно можете просмотреть отчет. Вы можете даже конвертировать отчет в формат электронной таблицы, чтобы его можно было скачать и работать с ним вручную.

Хотя отчеты можно запускать или просто публиковать в Web, в идеале, наши пользователи должны иметь возможность создать отчет, используя такие средства, как Discoverer Viewer и Discoverer Plus, которые были описаны в главе 6. При использовании такого подхода пользователи смогут не только получать отчеты в удобное для них время, но они также не будут ограничены рамками стандартного отчета, поскольку в настоящее время такие средства, как Discoverer, до некоторой степени допускают проведение настройки отчета. Можно, например, вывести информацию только о продажах в южном регионе или только сведения о магазинах, объем продаж в которых составил менее $10000.

Мы надеемся, что вы начинаете понимать, как Интранет может способствовать решению проблем, связанных с публикацией и распределением, и как он может предоставить пользователям механизм доступа к самой свежей информации. Предположим, например, что у вас есть статичный отчет. Вы можете автоматически обновлять этот отчет каждый час и публиковать его в Web или же обновлять его, когда поступает запрос от пользователя. Работа с отчетами в хранилищах в режиме он-лайн традиционно считается невозможной, поскольку обработка отчета может занимать часы. Однако теперь, когда ес'ть такие возможности, как Summary Management, описанный в главе 4, тот запрос, обработка которого обычно занимала часы, может быть выполнен за минуты и секунды. Поэтому запрос, использующий эти данные, может обрабатываться чаще.

Теперь, когда мы имеем представление о возможностях сети Web, мы расскажем о нескольких разных способах ее взаимодействия с хранилищем данных.

Web становится платформой для размещения всей бизнес-информации с помощью таких средств, как Oracle Discoverer и Oracle Reports.

Web может служить источником данных для хранилища при отслеживании активности web-сайта — т. е. маршрутов перемещения пользователей по web-узлу.

Web становится платформой, в которой можно настраивать приложения с помощью Oracle Personalization на основе данных хранилища.

8.1.2. Программное обеспечение для хранилищ от Oracle

Компания Oracle предлагает несколько продуктов, которые могут помочь вам опубликовать данные из хранилища в Интранет или Интернет (в зависимости от того, как используется ваш сайт), или, используя информацию о предыдущих посещениях сайта данным пользователем, предложить ему сведения, соответствующие его интересам. К интересующим нас компонентам относятся следующие:

- Oracle Reports;
- Oracle 9i Application Server, в который входят:
  - Discoverer;
  - Portal;
  - Clickstream Intelligence;
  - Personalization.

Теперь мы кратко опишем каждый из этих продуктов и посмотрим, как они могут помочь нам в работе с хранилищем.

8.2. Oracle 9i Application Server

Мы надеемся, что вы начинаете ценить то, как Web может значительно изменить способ использования и обработки информации в хранилище. Использование Oracle 9i Application Server для запуска всех приложений, связанных с бизнес-информацией, не является необходимым условием. Однако, применяя его, вы получаете полную среду для работы с электронной бизнес-информацией.

В одном продукте Oracle 9i Application Server (9iAS) вам дается вся технология, необходимая для разработки и реализации порталов для электронной коммерции, web-служб и приложений, связанных с транзакциями. В 9iAS имеется поддержка Java, XML и промышленных стандартов web-служб. Его можно использовать для доступа к данным как в базах Oracle, так и в других базах.

В 9iAS входит Oracle HTTP Server, основанный на Apache Web Server. Этот сервер уже сконфигурирован и готов к немедленному использованию. Имеется также среда J2EE (Java 2 Enterprise Edition), представляющая собой платформу, разработанную для создания приложений, предназначенных для бизнеса масштаба предприятия. С помощью технологий J2EE вы можете создавать приложения, используя серверные страницы (server page), сервлеты (servlets) и Enterprise JavaBeans. Java сервлет — это программа, которая, получив информацию от клиента, динамически извлекает информацию из базы данных, генерирует ответ,

который может быть в формате HTML или XML, и отсылает его обратно клиенту. Страницы JavaServer помогают разрабатывать сервлеты, позволяя разработчику создавать и обслуживать динамические web-страницы, отделяя динамическое содержание от оформления страницы. Пакет Enterprise JavaBeans позволяет разработчику инкапсулировать бизнес-логику, что избавляет от необходимости писать код этих функций.

Из Oracle 9iAS вы можете вести разработку, используя Java, XML, Perl и PL/SQL. Oracle 9iAS, по сути, состоит из следующих продуктов (не обязательно инсталлировать их все):

- Oracle HTTP Server;
- Oracle 9iAS Reports Services;
- Oracle 9iAS Portal;
- Oracle 9iAS Discoverer;
- Oracle 9iAS Clickstream Intelligence;
- Oracle 9iAS Personalization;
- Oracle 9iAS Forms Services;
- Oracle 9iAS Wireless;
- Oracle 9iAS Web Cache;
- Oracle 9iAS Unified Messaging;
- Oracle 9iAS Security;
- Oracle Internet Directory;
- Oracle 9iAS Containers for J2EE;
- Oracle Enterprise Manager.

Теперь мы посмотрим, какие компоненты могут потребоваться нам для того, чтобы разместить наше хранилище в Web. Существует много способов конфигурирования web-сайта, и их обсуждение находится за рамками этой книга, но, как правило, web-сервер работает не на той машине, на которой находится хранилище.

8.2.1. Зачем устанавливать портал

Когда Интернет только начинал становиться популярным, компании говорили о своем web-сайте, но теперь говорят о портале. Портал — это web-сайт, предлагающий массу информации и множество услуг. Если web-сайт посвящен конкретной теме, например, медицинский сайт WebMD (http://www.webmd.com), его называют вертикальным порталом. Однако большая часть порталов охватывает широкий диапазон тем, например, такой сайт, как Yahoo (http://www.yahoo.com). Такие сайты называют горизонтальными порталами.

Портал предлагает доступ ко всему своему богатству информации, и как правило, если вы зарегистрируетесь, вы увидите информацию, настроенную в соответствии с вашими интересами. Например, если вы посетите сайт Oracle, показанный на рис. 8.1 (http://www.myoracle.com), вы увидите страницу, совершенно иную, чем та, которую увидит служащий Oracle.

Служащему Oracle будет предложена внутренняя информация компании и доступ к широкому диапазону внутренних систем. Обычному же посетителю web-сайта будет предложено зарегистрироваться в Мире Oracle, а также будут представлены последние новости. Поскольку сайт является порталом, посетители могут настраивать данную страницу, чтобы видеть только ту информацию, которая их интересует.

Так почему же портал так важен для нашего хранилища данных? Многие компании в настоящее время понимают, что им нужен портал, а не web-сайт, поскольку внутренний портал значительно облегчает работу служащих, которые могут иметь настраиваемый доступ к необходимой информации, что повышает эффективность их работы. Поэтому подумайте об использовании портала для

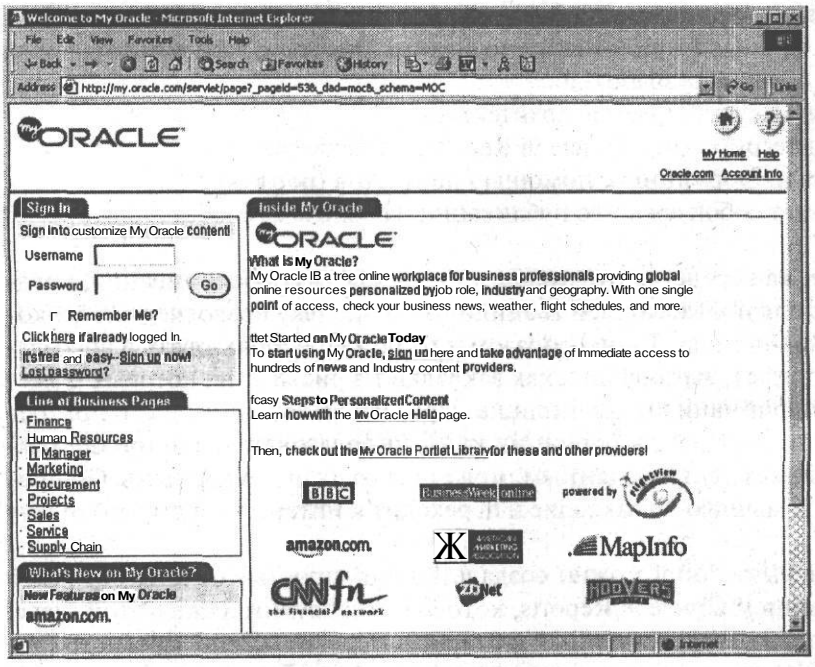


Рис. 8.1. *Регистрация на портале Oracle*

доставки информации от хранилища к его пользователям. В 9iAS есть два продукта, которые делают это возможным: Portal и Personalization.

Oracle 9iAS Portal

В Oracle 9i Application Server входит Oracle 9iAS Portal, который представляет собой программу, позволяющую легко и быстро создать портал. Поскольку 9iAS Portal использует единый механизм аутентификации, являющийся частью Oracle 9iAS, в нем предлагается также механизм контроля доступа к информации.

Разрабатывать приложения с помощью 9iAS Portal очень легко, даже если у вас нет опыта разработки. В него входят несколько мастеров, которые можно использовать для быстрого создания отчетов и диаграмм, которые будет видеть пользователь. Существуют также мастера, которые помогут составить запрос для извлечения данных из базы. Вы также можете составить его на SQL самостоятельно. Преимущества использования 9iAS Portal следующие.

- Это приложение имеет расширяемую основу, интегрирующую все ресурсы Web.
- В нем используется единый механизм аутентификации 9iAS.
- Он предоставляет возможность быстрого доступа к информации.
- С помощью мастеров приложения создаются быстро и при минимальных познаниях в области разработки.
- Используется масштабируемая архитектура.
- 9iAS Portal интегрирован с Oracle 9i Reports и Discoverer.
- Он публикует информацию с помощью портлетов (portlets).
- Он представляет собой средство публикации на основе самообслуживания.

Это приложение, как средство публикации на основе самообслуживания, представляет интерес для пользователей хранилища, поскольку позволяет опубликовать данные из хранилища. Таким образом, его можно использовать для создания портлетов (portlet), выглядящих как закладки на рис. 8.2, на которых может публиковаться информация из хранилища. При использовании такого подхода к публикации и получению информации каждый пользователь внутри организации, который имеет доступ к данным, может легко их просматривать. Он просто делает щелчок мышью по закладке и переходит к интересующему его отчету или графику.

Хотя в Oracle 9iAS Portal можно создавать лишь простые отчеты, он тесно связан с приложением Oracle 9i Reports, которое мы обсудим ниже в этой главе. С помощью мастера отчет, созданный в Oracle 9i Reports, можно зарегистрировать в Oracle 9iAS Portal и опубликовать в виде портлета. Также и простые отчеты, созданные в Portal Reports, можно конвертировать в отчет Oracle 9i Reports.

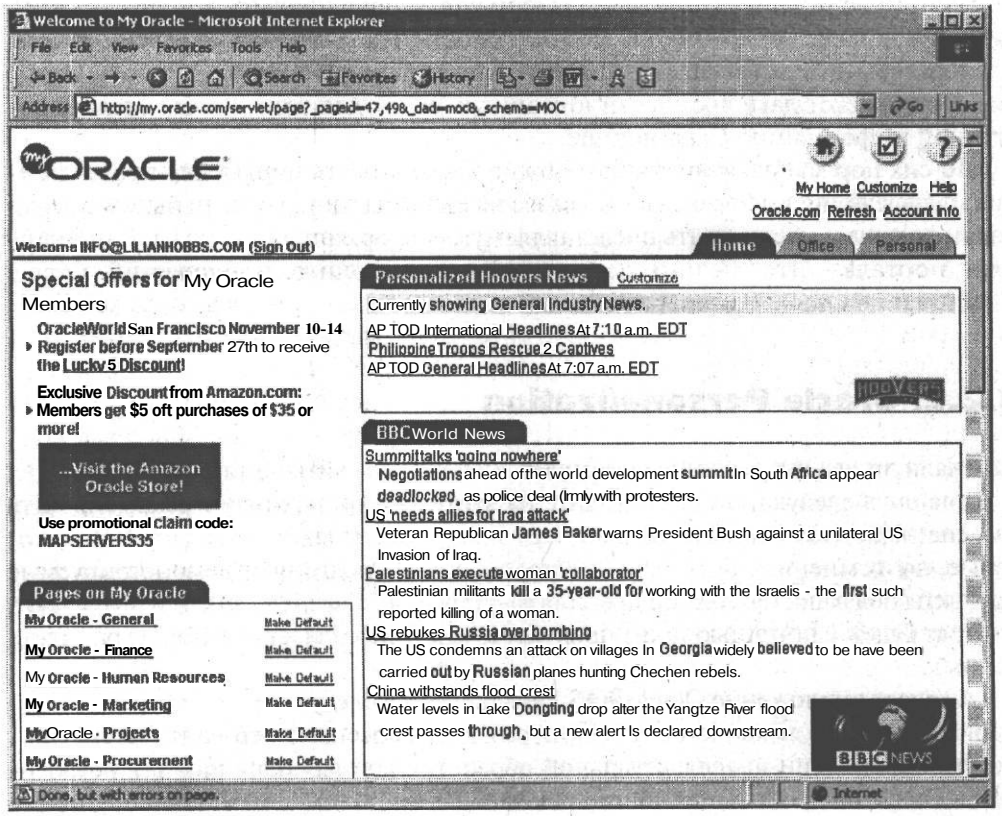


Рис. 8.2. *Портал Oracle — стандартный вид*

Тем не менее, поскольку в Oracle 9iAS Portal можно создавать лишь простые отчеты и графики, большинство людей предпочитает использовать мощные возможности Oracle 9i Reports и преобразовывать полученные отчеты в портлеты.

С помощью настраиваемых страниц Oracle 9iAS Portal пользователи могут осуществлять значительный контроль над тем, как создается и выглядит отчет. Например, если отчет был настроен так, что принимает параметры, например дату и код магазина, пользователи могут запросить интересующие их данные, например информацию только по магазинам в своем регионе за последние несколько недель, и увидеть в портале свою версию отчета. Другой пользователь может запросить другие данные и увидеть свою версию.

После помещения в портал содержимое отчета не обязательно остается ста-

тическим. Его можно автоматически обновлять и публиковать заново с новыми данными, и даже возможно при необходимости иметь несколько версий отчета. Следовательно, можно видеть, что тесная интеграция Oracle 9iAS Portal и Oracle 9i Reports может дать чрезвычайно мощную среду для того, чтобы сделать доступной информацию в хранилище.

До сих пор мы изучали то, как можно использовать портал для представления и извлечения информации, но разве не было бы здорово, если бы мы смогли пойти дальше и настраивать представляемую информацию под каждого посетителя портала? Это можно осуществить с помощью приложений Oracle Clickstream и Oracle Personalization.

8.2.2. Oracle Personalization

Замечали ли вы, что, когда вы посещаете некоторые сайты, они запоминают вас, и когда вы в следующий раз заходите на этот сайт, он начинают рекомендовать вам специфические продукты, дают вам знать о специальных предложениях, которые, по их мнению, будут вам интересны, и даже подгоняют демонстрируемые продукты под ваши нужды? Если вы когда-нибудь хотели знать, как они это делают, то ответ будет: с помощью таких продуктов, как Oracle Personalization и хранилищ данных.

А делает приложение Oracle 9iAS Personalization следующее: оно берет информацию о том, что делает клиент всякий раз, когда посещает web-сайт. Затем, с помощью технологии интеллектуальной обработки данных (data mining), встроенной в Oracle9iR2, оно дает клиенту рекомендации по тем темам, которые, по его мнению, этого клиента интересуют. С помощью Oracle Personalization можно ответить на следующие вопросы:

- Какой продукт, скорее всего, приобретут пользователи?
- Какие еще продукты можно предложить им при покупке данного товара?
- Какой рейтинг пользователь может поставить продукту?
- Какие товары из списка самых продаваемых, скорее всего, купят пользователи?

И снова мы используем информацию из нашего хранилища не только на благо клиента, но и, надеемся, на благо бизнеса, как в финансовом аспекте, так и в смысле качества работы с клиентами.

8.2.3. Слежение за маршрутами перемещений по web-узлу и хранилище информации

По традиции, когда люди думают о хранилище, они подразумевают массу исторических данных (например, все телефонные звонки, сделанные за последний год, все проданные продукты, весь персонал, принятый на работу за последние десять лет). Но теперь нам следует искать массу информации на web-сайте, причем информации, идеальной для записи в хранилище. Эти данные могут представлять собой активность за очень короткий промежуток времени.

Однако, даже если эти данные собраны за короткий период, их может быть очень много, поскольку с помощью данных о маршрутах перемещений (clickstream) мы следим за тем, что делают посетители нашего web-сайта. Следовательно, если сегодня наш сайт посетили 250 тысяч человек и все они просмотрели по пять страниц, сохранять придется очень много информации. Так что еще одной областью, в которой хранилище данных связывается с Интернет, является сохранение данных о маршрутах перемещений, т. е. о том, что делали люди, посещая наш web-сайт.

Когда вы занимаетесь традиционным бизнесом, связанным, например, со стройматериалами, вы можете наблюдать за своими покупателями, когда они изучают ваш магазин. Вы можете видеть, как они входят, берут товар, кладут его на место и выходят. Или они могут подойти прямо к товару, а затем — к кассе, или же ходят взад-вперед по проходу и смотрят, что у вас есть, и только потом идут к кассе. Как понять, что делают покупатели, в электронном бизнесе, когда они, скорее всего, находятся дома или в офисе, возможно, за тысячи миль от вас?

Такие продукты, как Oracle Clickstream, входящий в состав Oracle 9iAS, отслеживают, что делают люди, когда они посещают ваш web-сайт. Это приложение интегрировано с Oracle 9i Warehouse Builder и Oracle 9iAS Discoverer, что дает полную систему слежения для систем электронной коммерции. Теперь вы можете определять, какие страницы посещают пользователи, что они кладут в корзину для покупок и ждут ли они загрузки страниц.

Все это богатство информации можно использовать для совершенствования бизнеса. А разве есть место для хранения и анализа этой информации лучше, чем хранилище? Почему хранилище? Просто если ваша фирма — это широко известная авиакомпания, сайт которой еженедельно посещает 600 тысяч человек, вы накопите массу данных. И эти данные надо записывать, если их нужно хранить в течение хоть какого-то периода времени. В прошлом в хранилища записывалась лишь информация о только что случившемся событии, например покупке товара. Теперь пора понимать, что в хранилище можно хранить и другие типы данных.

К этому моменту вы должны начать разбираться в том, какую важную роль Web может играть в доставке информации, содержащейся в хранилище, к тем людям, которым она нужна. Теперь мы рассмотрим некоторые инструменты, которые позволят увидеть эту информацию в браузере.

8.2.4. Отчеты Oracle Discoverer в Web

Существует множество инструментов, которые можно использовать для извлечения информации из хранилища, и одним из очень популярных является Discoverer, который предлагает компаниям механизм легкого доступа к данным хранилища и интеграции этой информации с Oracle 9iAS Portal, для того чтобы ее можно было просматривать в браузере. Также вы можете использовать Oracle Discoverer Plus или Viewer, которые входят в Oracle 9iAS, и они также позволяют запускать отчеты, которые мы создавали в главе 6, через браузер.

Использование Discoverer Viewer и Discoverer Plus

На рис. 8.3 мы можем видеть один из таких отчетов, который был запущен и выведен с помощью Discoverer Viewer через браузер. В этой рабочей книге у нас есть три отчета: Total Sales by Customer (Объемы продаж по покупателям), Best Customers (Самые лучшие покупатели) и Most Popular Products (Наиболее популярные продукты). Каждый представлен в виде активной ссылки на закладке в левой части экрана. Следовательно, пользователям очень просто выбрать нужный отчет.

Хотя для отображения данных мы используем браузер, отчет не обязательно должен быть статичным, поскольку мы можем осуществлять взаимодействие с тем, кто запустил создание отчета. Если выбран отчет Most Popular Products, появляется окно, показанное на рис. 8.4, в котором предлагается ввести критерий, определяющий популярность продукта. В данном примере мы говорим, что нас интересуют любые продукты, которых было продано более 100 штук.

Discoverer Viewer предлагает простой способ извлечения информации в браузере. На рис. 8.3 возможны некоторые настройки отчета, но при использовании Viewer пользователь может только читать данные, а изменять их невозможно.

Распространение отчетов в Web

Discoverer Viewer или Plus, по сути, запускают отчет через браузер, но что делать, если нам нужно просто опубликовать отчет в Web, чтобы его можно было

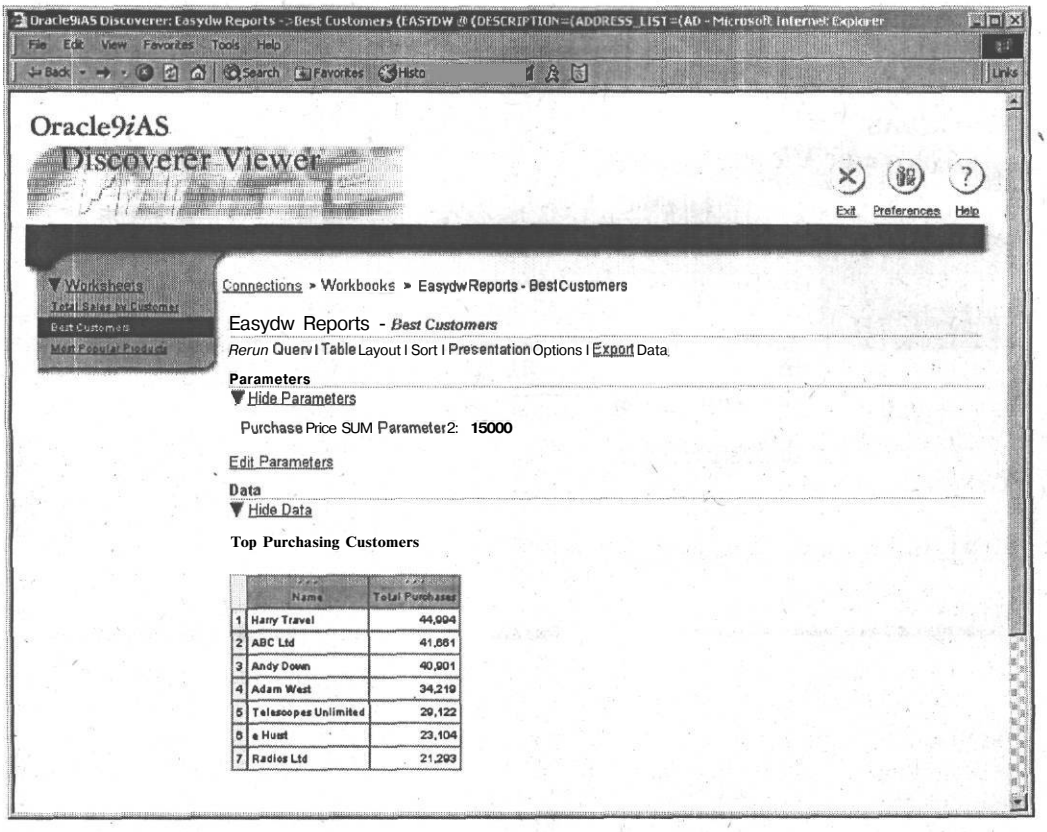


Рис. 8.3. *Отчет Discoverer Viewer в браузере*

просматривать как статичный документ? Это также возможно, и, если обратиться к рабочей книге, в которой мы первый раз создавали отчеты в главе 6, то мы увидим, что это делалось при помощи не-Интернет-версии Discoverer. Однако в Discoverer Plus мы можем принимать такие отчеты и разделять их с другими пользователями, экспортируя рабочие книги в несколько форматов, в том числе:

- XML для Oracle Reports;
- Excel;
- HTML (отчеты можно будет посылать по e-mail;
- Формат Comma-Separated Values (CSV).

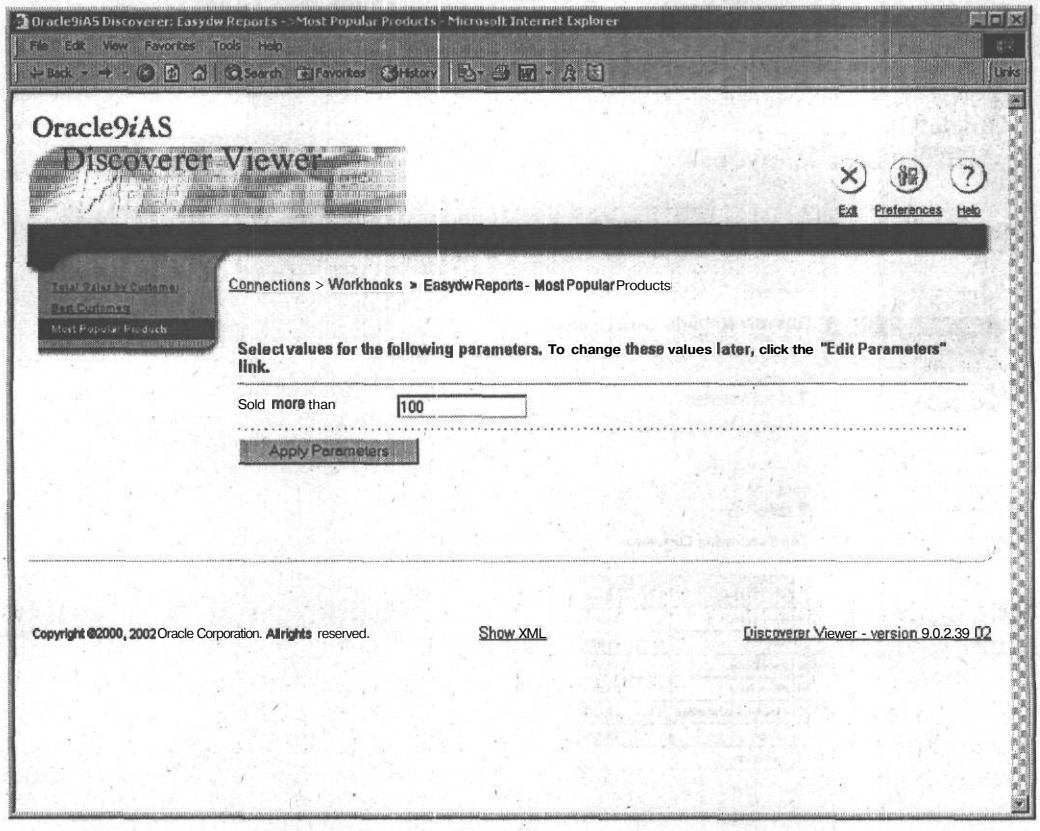


Рис. 8.4. *Discoverer Viewer — указание критерия для отчета*

Процесс экспортирования рабочего листа или книги благодаря мастеру экспорта (Export Wizard) весьма понятен и начинается с меню File (Файл). Просто выберите те рабочие листы, которые вы хотите отправить в Oracle Reports, и укажите, нужна ли только таблица или таблица вместе с графиком. После этого нужно указать формат отчета. Имеющиеся варианты можно видеть в окне, показанном на рис. 8.5, в котором есть раскрывающийся список форматов. Чтобы отчеты Discoverer можно было опубликовать в Web, их нужно экспортировать в формат Oracle Reports.

Теперь нам осталось только указать местоположение отчета. В последнем окне, которое мы не стали изображать в этой книге, просто задается вопрос, нужно ли осуществлять экспорт под контролем (что даст возможность испра-

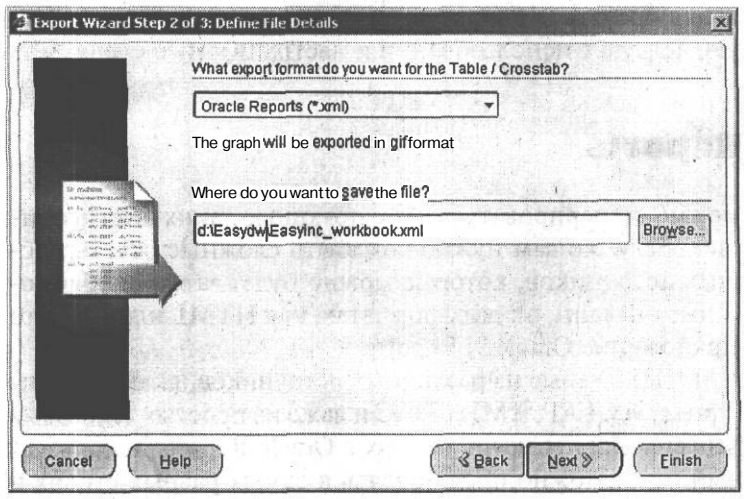


Рис. 8.5. *Discoverer Plus — Export Wizard — определение формата выводимого отчета*

вить проблемы, которые могут появиться) или без контроля. Нажмите кнопку Finish (Готово) и отчет для Oracle Reports готов.

Отчеты из Discoverer можно также интегрировать в Oracle 9iAS Portal, что обеспечивает чрезвычайно эффективный способ доставки данных к конечным пользователям. Достигается это тем, что Oracle 9iAS Discoverer является поставщиком портлетов для Oracle 9iAS Portal. Существует два типа портлетов, которые может поставлять Discoverer:

- список портлетов рабочих книг;
- портлеты рабочих листов, которые являются отдельными рабочими листами.

Одним из действительно приятных свойств является то, что при разработке страницы для портала на страницу можно помещать столько портлетов, сколько нужно. Следовательно, на одной странице портала может находиться множество рабочих листов и графиков Discoverer. Когда пользователь обращается к такой странице, он может или выбрать необходимый рабочий лист, щелкнув по нему мышкой, или рабочие листы и графики могут появиться на странице автоматически.

Нашему конечному пользователю не нужно даже беспокоиться о свежести данных, полученных из портала, поскольку при разработке портлета можно указать частоту его обновлений. Притом что пользователи имеют возможность настраивать портлет Discoverer в соответствии со своими требованиями, приме-

нение Oracle 9iAS Portal для представления отчетов Discoverer и графиков обеспечивает дружественную, легкую в использовании и настраиваемую среду.

8.3. Oracle 9i Reports

Выше мы видели, как можно сгенерировать отчет с помощью таких инструментов, как Oracle Discoverer. Если же вам нужно создавать сложные отчеты, используя данные из разных источников, которые можно будет затем опубликовать на бумаге или в режиме он-лайн, в таких форматах, как HTML или PDF, то примите во внимание приложение Oracle 9i Reports.

Oracle 9i Reports принимает данные из различных источников, включая базы данных Oracle, Oracle Express, 9i OLAP, XML, JDBC и даже из простых текстовых файлов. Например, Discoverer для передачи данных в Oracle Reports использует формат XML. Созданный отчет можно представлять в самых разных стилях и улучшать, используя такие виды диаграмм, как секторная диаграмма или гистограмма. Каждый отчет может базироваться на заранее созданном шаблоне (template). Для создания собственных шаблонов вы можете использовать редактор Template Editor. Это означает, что вы можете определить стандартный внешний вид и добавить, например, логотип компании. Как только отчет создан, его можно отправлять в самые разные пункты назначения, например в файл, на принтер, в электронное письмо или в Oracle Portal.

Oracle Reports состоит из нескольких компонентов:

- Report Builder — инструмент для создания отчетов с использованием Мастеров;
- службы Oracle 9iAS Reports:
  - Reports converter (Преобразователь отчетов), позволяющий менять формат отчета;
  - Queue Manager (Менеджер очереди).

Давайте теперь рассмотрим некоторые типы отчетов, которые можно создавать с помощью Oracle Reports.

8.3.1. Создание отчета с помощью Report Builder

Отчет можно либо создавать вручную, либо с помощью мастера Report Wizard, который проведет вас через этапы создания отчета. На рис. 8.6 изображено одно из первых окон, в котором нам предлагается выбрать тип раскладки (layout). Поскольку наш отчет будет публиковаться в Интранет и распространяться в бумажном

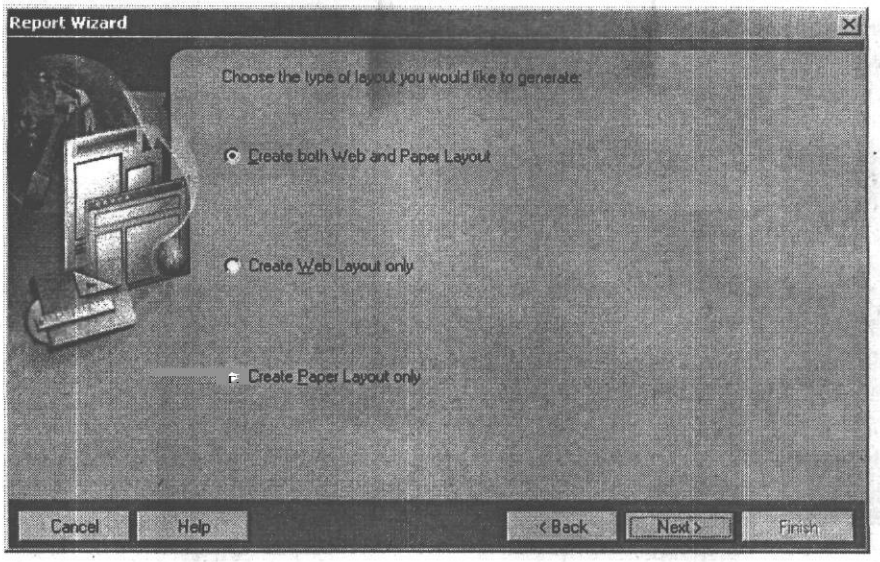


Рис. 8.6. *Oracle Reports — Выбор ситуации*

варианте, мы выбрали пункт Create both Web and Paper Layout (Создать раскладки как для Web, так и для бумажного варианта). Это означает, что один отчет, если это необходимо, может иметь два формата: один для бумаги, а другой — для Web.

Нашим следующим шагом будет выбор нужного типа отчета. На рис. 8.7 видно несколько вариантов выбора. Предварительный набросок стиля представлен рядом переключателем. Заголовок этого отчета будет таким: «Sales by Customer» (Объемы продаж по покупателям), — а данные будут представляться в табличном формате. Позже мы увидим отчеты, использующие матричный вариант формата (Matrix).

На рис. 8.8 можно видеть способы, которые Oracle Reports позволяет использовать для запроса к источнику данных. В данном примере мы решили использовать SQL, но так же просто сделать запрос и к Oracle Express или JDBC.

Когда мы делаем запрос к базе данных, мы можем использовать три метода создания кода SQL (рис. 8.9). Если вы знаете SQL, вы можете написать код вручную или импортировать его из файла. В примере, показанном на рис. 8.9, выражение SQL было создано с помощью SQL*Plus, записано в файл и импортировано в Report Builder.

Тем пользователям, которые не знакомы с SQL, мы рекомендуем использовать Query Builder, который делает составление выражений SQL очень легким

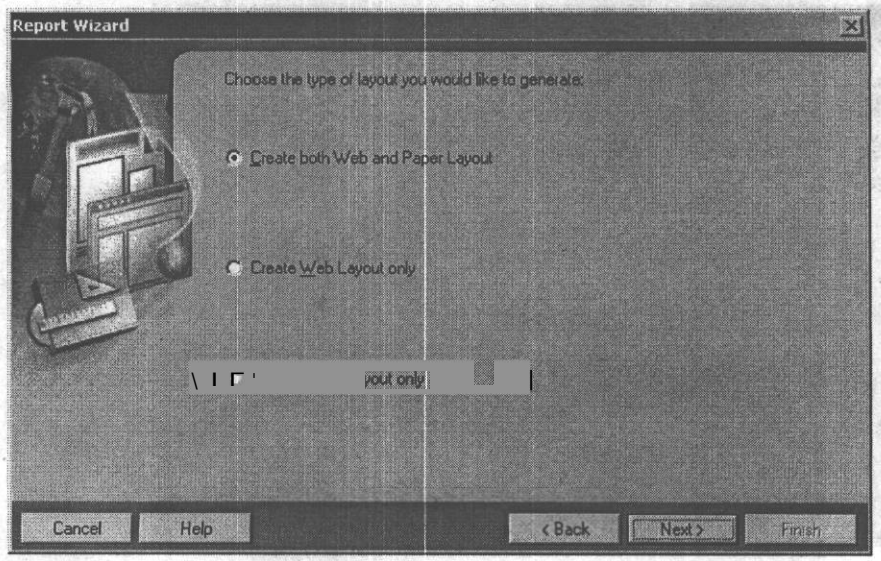


Рис. 8.7. *Выбор стиля отчета и его заголовка*

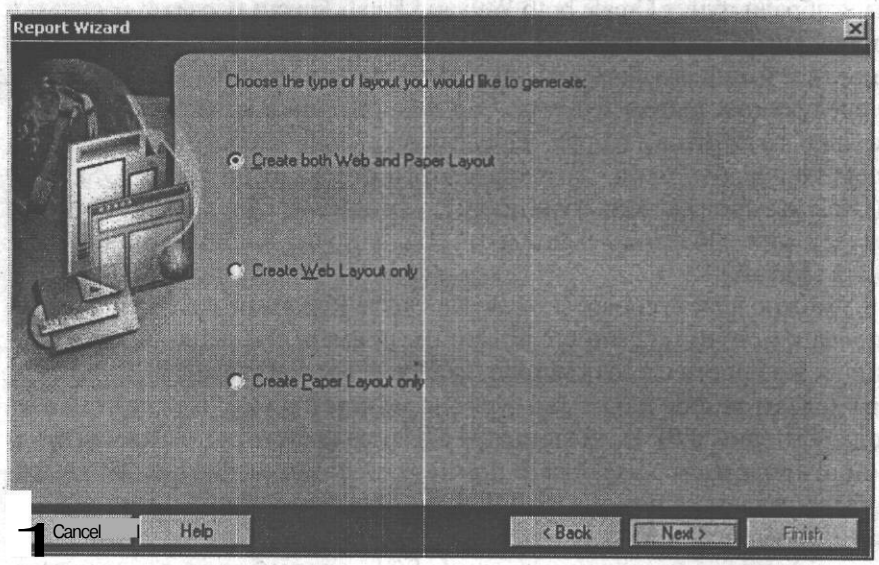


Рис. 8.8. *Файлы Oracle — выбор источника данных*

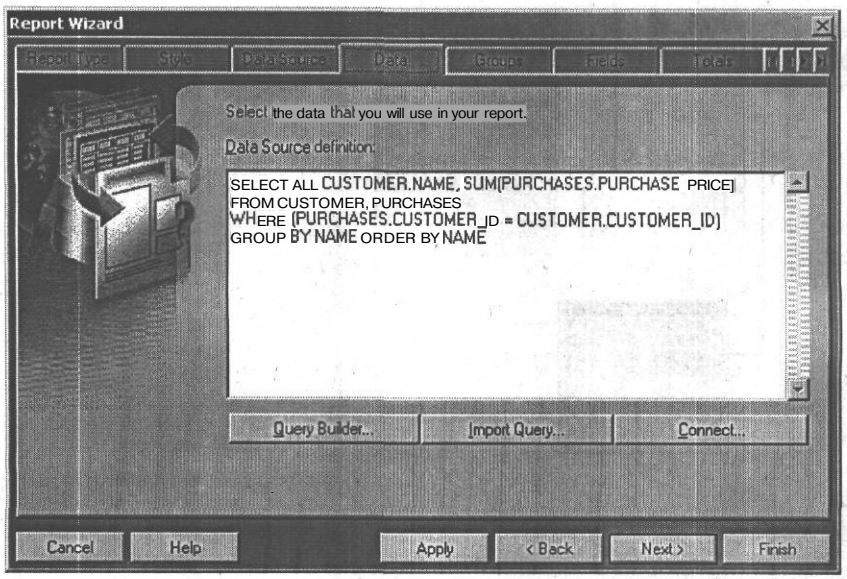


Рис. 8.9. *Oracle Reports — составление запроса на SQL*

делом и, без сомнения, сэкономит вам массу времени. Все, что нужно делать — это выбирать таблицы, включаемые в отчет, а затем отмечать те элементы, которые должны отображаться. Oracle Reports автоматически определяет способы соединений данных и создает необходимый код SQL, когда вы покидаете Query Builder. На рис. 8.10 мы выбрали таблицы CUSTOMER, PRODUCT и PURCHASES и запрашиваем из этих таблиц только четыре столбца. Заметьте, как Oracle Reports определяет способы соединения таблицы PURCHASES с таблицами CUSTOMER и PRODUCT.

Теперь Oracle Reports проверит наш код SQL, а потом нам будет предложено указать тип группировки. Отчет с группировкой может содержать один или более подуровень, но на рис. 8.11 он будет только один — по имени (NAME).

Затем выбираются выводимые столбцы отчета — можно выбрать все столбцы или какую-то часть их. В примере на рис. 8.12 мы будем выводить имя покупателя и суммарный объем покупок для покупателя.

Теперь (рис. 8.13) мы можем указать, требуется ли нам вычисление каких-либо итоговых величин по данным. Мы видим, что Oracle Reports предлагает нам кнопки для создания запроса на сумму (sum), среднее (average), количество (count), минимум (minimum), максимум (maximum) и % от общего числа (% total). В нашем примере нам нужна только сумма по столбцу purchase_price.

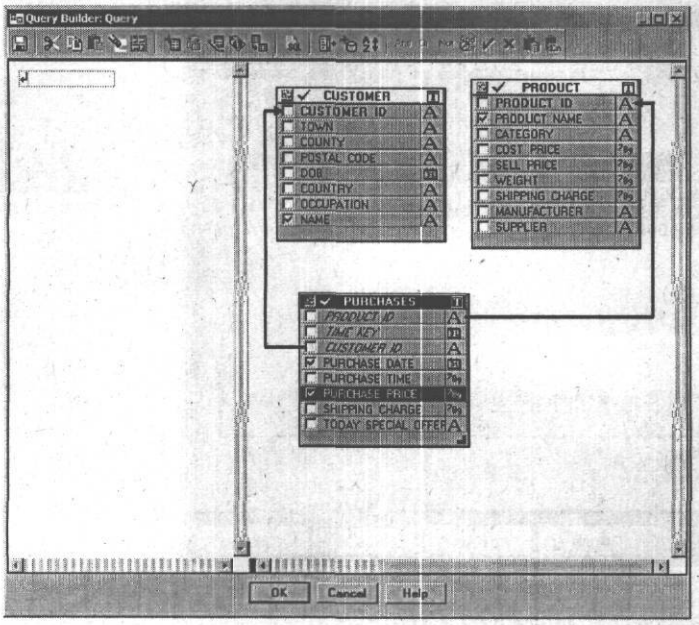


Рис. 8.10. *Oracle Reports — использование Query Builder*

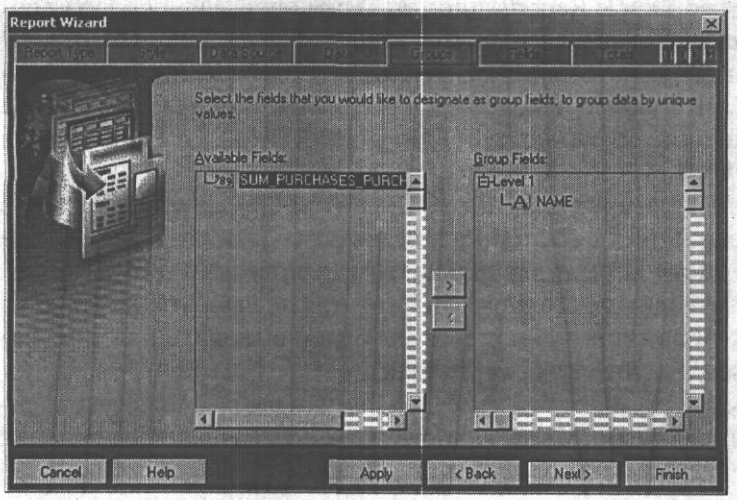


Рис. 8.11. *Oracle Reports — указание уровней группировки*

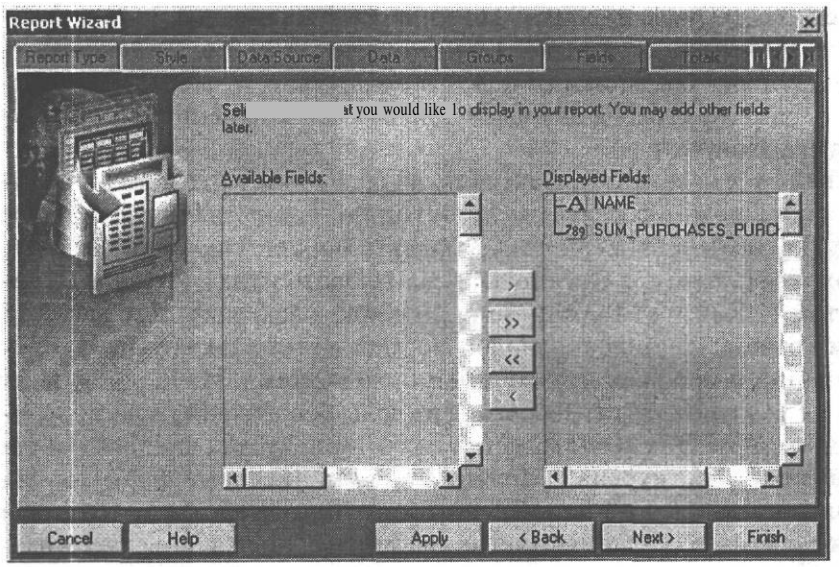


Рис. 8.12. *Oracle Reports — Выбор полей отчета*

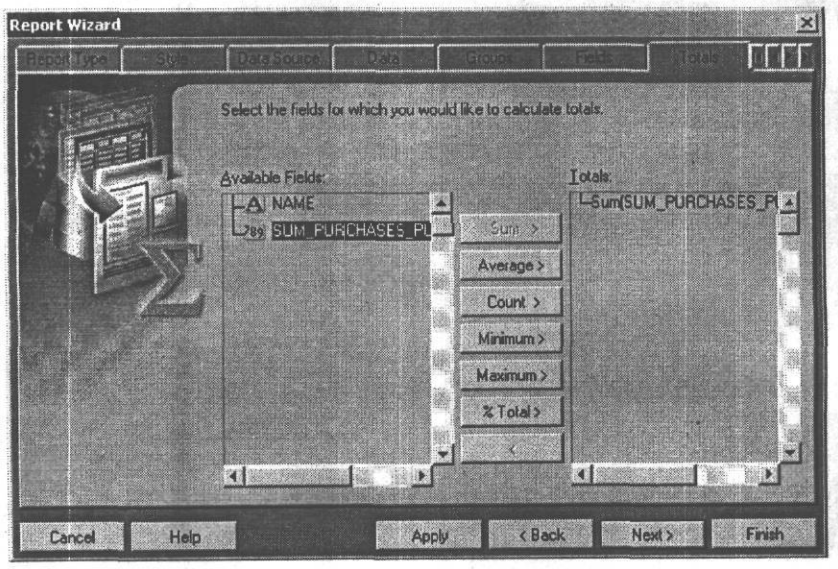


Рис. 8.13. *Oracle Reports — вычисление итоговых значений*

Приятное свойство Oracle Reports — это возможность указать ширину столбцов данных и их заголовки. В окне, показанном на рис. 8.14, мы можем указать ширину столбцов перед созданием отчета. Следовательно, у нас есть возможность увеличить ширину столбца для имен покупателей и изменить его заголовок на «Customer» (Покупатель).

Внешний вид отчета можно определить, выбрав один из готовых шаблонов. Шаблон также можно создать самостоятельно. Когда закончите, жмите кнопку Finish (Готово), и вы вернетесь в Reports Builder. Оказавшись здесь, вы можете сохранить отчет, а для его запуска выбрать пункт Program (Программа), а потом — Run Web Layout (Запустить web-раскладку). На рис. 8.15 показан пример отчета. Он имеет черновой формат, до того как его снабдят логотипами компаний и прочими необходимыми элементами. Из Report Builder вы можете также изменять отчет, добавляя символы обозначения валют, запятые и десятичные точки. Также вы можете менять гарнитуры и шрифты текста (жирный, подчеркнутый, курсив). Можно изменять числовые форматы для представления денежных единиц, добавлять логотипы, графику, заголовки и сноски. Данное средство создания отчетов действительно является очень разносторонним.

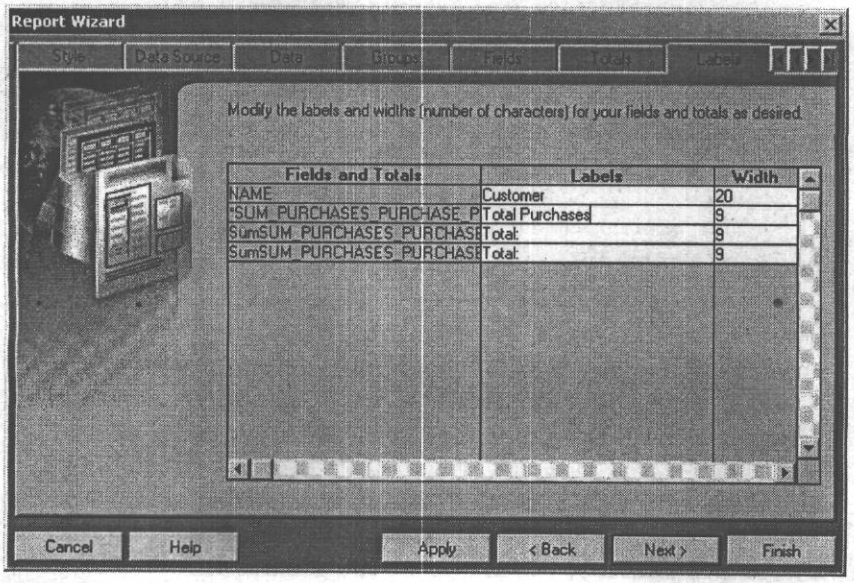


Рис. 8.14. *Oracle Reports — определение ширины столбцов*

8.3.2. Еще несколько примеров отчетов Oracle Reports

Отчет, показанный на рис. 8.15, — это очень простой отчет и не очень привлекательный, но он иллюстрирует процесс создания отчета. Давайте теперь рассмотрим еще несколько примеров отчетов, которые мы создали с помощью Oracle Reports по данным из нашего хранилища EasyDW.

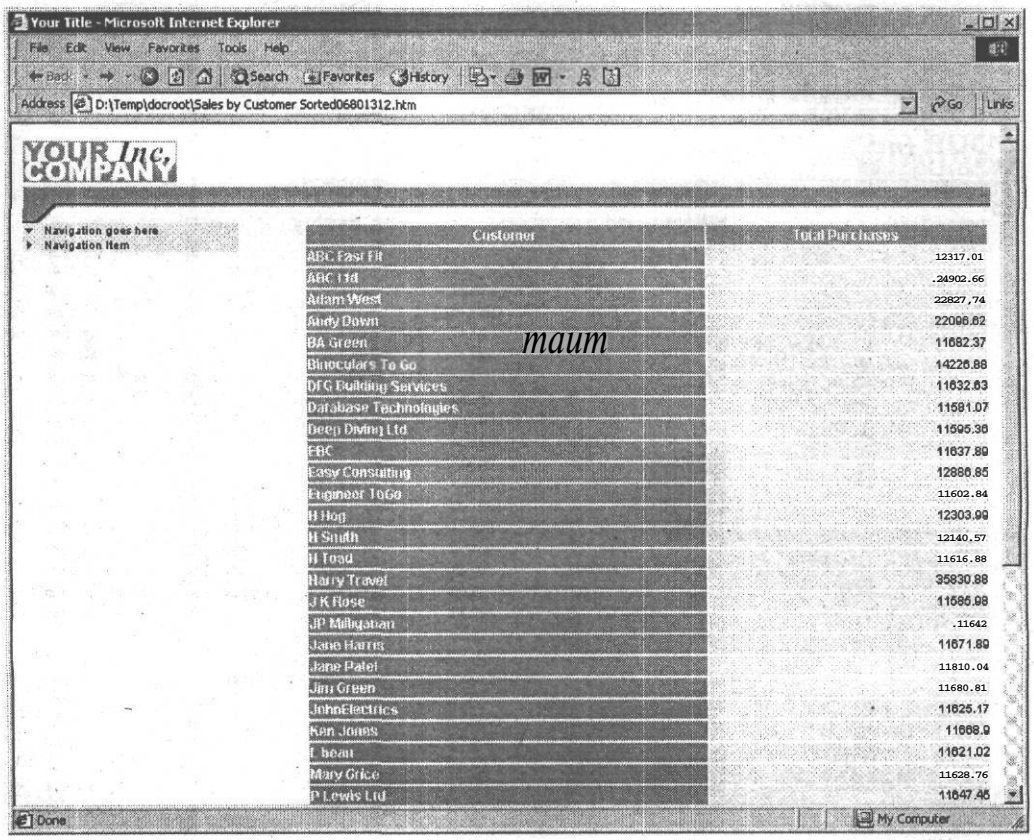
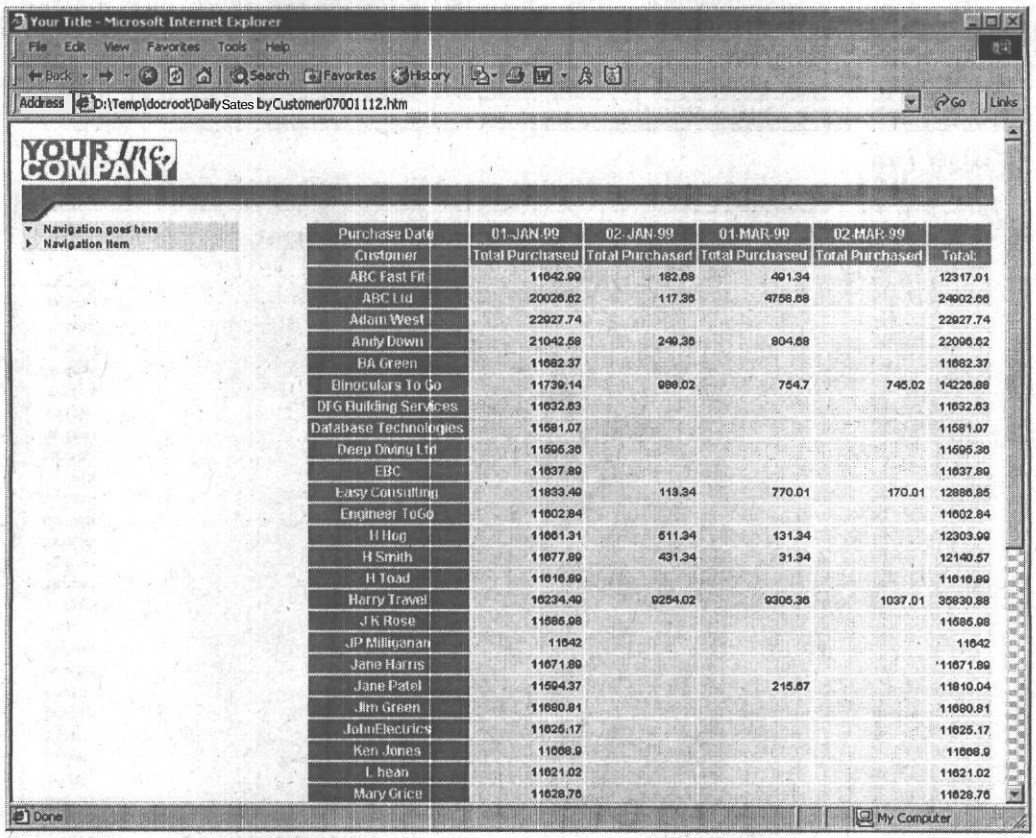


Рис. 8.15. *Oracle Reports — готовый отчет*

Матричный отчет

На рис. 8.16 мы видим пример отчета, созданный с помощью матричной раскладки, на котором показано, сколько всего наши покупатели потратили денег по дням. Глядя на этот отчет, мы видим, что стимулирование сбыта 1 января дало нам большую прибыль, но после этого бизнес пошел на спад, и некоторые покупатели не делали у нас покупок в течение нескольких следующих месяцев.

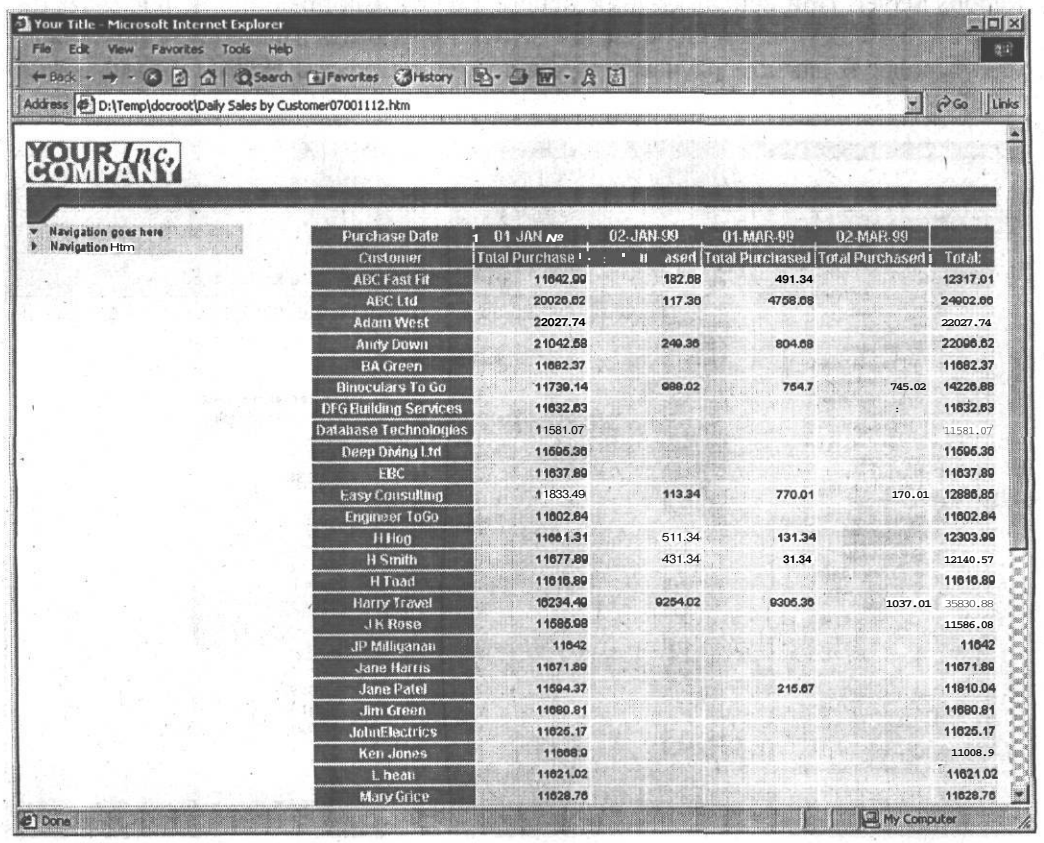


Purchase Date	01-JAN-99	02-JAN-99	01-MAR-99	02-MAR-99	
Customer	Total Purchased	Total Purchased	Total Purchased	Total Purchased	Total:
ABC Fast Fit	11642.99	182.68	491.34		12317.01
ABC Ltd	20026.62	117.36	4758.68		24902.66
Adam West	22927.74				22927.74
Andy Down	21042.58	249.36	804.68		22096.62
BA Green	11682.37				11682.37
Binoculars To Go	11739.14	989.02	754.7	745.02	14226.88
DFG Building Services	11632.63				11632.63
Database Technologies	11581.07				11581.07
Deep Diving Ltd	11595.36				11595.36
EBC	11637.89				11637.89
Easy Consulting	11833.49	119.34	770.01	170.01	12886.85
Engineer ToGo	11602.84				11602.84
H Hog	11661.31	511.34	131.34		12303.99
H Smith	11677.99	431.34	31.34		12140.67
H Toad	11616.89				11616.89
Harry Travel	16234.49	9254.02	9305.36	1037.01	35830.88
J K Rose	11586.98				11586.98
JP Milliganan	11642				11642
Jane Harris	11671.89				11671.89
Jane Patel	11594.37		215.67		11810.04
Jim Green	11680.81				11680.81
JohnElectrics	11625.17				11625.17
Ken Jones	11668.9				11668.9
L hean	11621.02				11621.02
Mary Grice	11628.76				11628.76

Рис. 8.16. *Oracle Reports — матричный отчет*

Отчеты с условиями

Обширные возможности Oracle Reports становятся очевидны, когда мы видим, что в отчете можно определить условие, при выполнении которого производится выделение данных цветом. На рис. 8.17 мы решили выделять красным цветом всех покупателей, которые потратили менее 250$. Мы можем даже пойти еще дальше и указать несколько условий, с выделением разными цветами. Покупатели, тратящие менее $250, обозначаются красным цветом, до $1000 — желтым, а более $1000 — черным. К сожалению, мы не можем показать это цветом, так что красные цифры будут обозначаться курсивом.



Purchase Date	01 JAN №	02-JAN-99	01-MAR-99	02-MAR-99	
Customer	Total Purchase	ased	Total Purchased	Total Purchased	Total:
ABC Fast Fit	11642.99	182.68	491.34		12317.01
ABC Ltd	20026.62	117.36	4758.68		24902.66
Adam West	22027.74				22027.74
Andy Down	21042.58	249.36	804.68		22096.62
BA Green	11682.37				11682.37
Binoculars To Go	11739.14	986.02	754.7	745.02	14226.88
DFG Building Services	11632.63				11632.63
Database Technologies	11581.07				11581.07
Deep Diving Ltd	11595.36				11595.36
EBC	11637.89				11637.89
Easy Consulting	11833.49	113.34	770.01	170.01	12886.85
Engineer ToGo	11602.84				11602.84
H Hog	11661.31	511.34	131.34		12303.99
H Smith	11677.89	431.34	31.34		12140.57
H Toad	11616.89				11616.89
Harry Travel	16234.49	9254.02	9305.36	1037.01	35830.88
J K Rose	11586.08				11586.08
JP Milliganan	11642				11642
Jane Harris	11671.89				11671.89
Jane Patel	11594.37		215.67		11810.04
Jim Green	11680.81				11680.81
JohnElectrics	11625.17				11625.17
Ken Jones	11608.9				11008.9
L bean	11621.02				11621.02
Mary Grice	11628.76				11628.76

Рис. 8.17. *Отчет с условием*

8.3.3. Создание отчета для Web

Все, что мы пока видели, можно использовать при публикации отчетов в Web. Одним из приятных свойств Oracle Reports является то, что вы можете разработать отчет одного формата для Web, и другого формата — для печати на бумаге. Мы можем видеть эти два формата в окне Object Navigator (Проводника по объектам), показанном на рис. 8.18.

Опубликовать отчет в Web можно разными путями, например, сделать его доступным для Oracle 9iAS Portal в виде портлета или преобразовать отчет в документ HTML. Перевести документ в формат HTML или PDF можно, выбрав пункт Generate (Сгенерировать) из меню File (Файл), или с помощью Oracle Reports Server. При использовании Reports Builder, как показано на рис. 8.18, нажмите File (Файл), затем Generate to File (Сгенерировать в файл) и укажите формат вывода. В данном случае мы решили использовать HTML с нумерацией

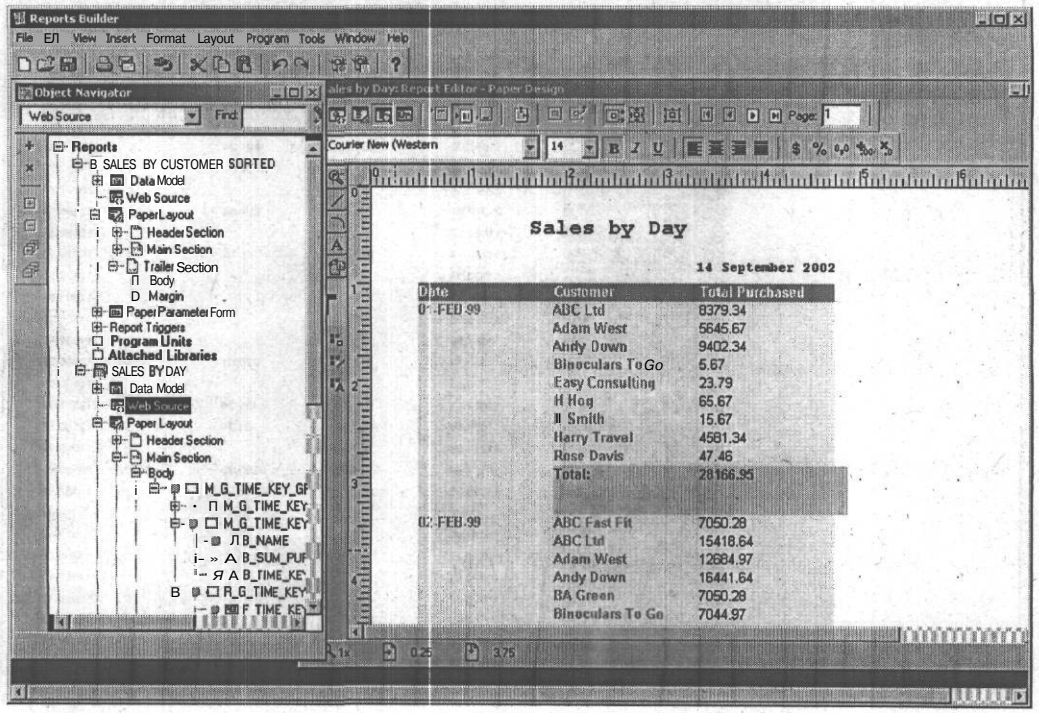


Рис. 8.18. *Oracle Reports- Navigator и Designer*

страниц, но точно так же мы могли выбрать и PDF. Отчет переводится в формат HTML и сохраняется.

На рис. 8.19 мы можем видеть, как наш отчет выглядит в браузере, а выглядит он точно так же, как когда мы разрабатывали его в окне на рис. 8.18. Отметьте, что мы добавили в отчет дату, изменили формат, внешний вид (раскладку) и шрифты заголовка, удалили логотип, выделили итоговое значение красным цветом и добавили несколько пустых строк ниже итога.

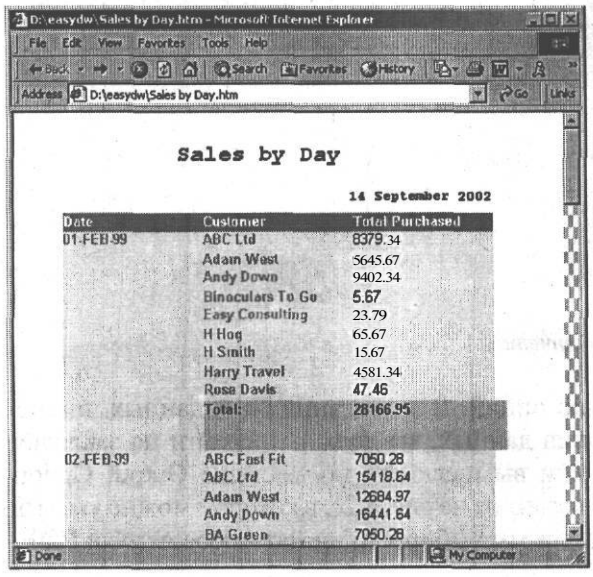


Рис. 8.19. *Отчет Oracle Reports в web-формате*

Oracle Reports Server

Если запущен Oracle 9iAS Reports Server, отчеты можно экспортировать в формат времени выполнения (run-time format) и поместить в очередь, после чего они преобразуются в один из следующих форматов:

- ASCII;
- HyperText Markup Language (HTML);
- HTML Cascading Style Sheets (CSS);
- Adobe Portable Document Format (PDF);
- XML.

В Oracle Reports Queue Manager (менеджере очереди Oracle Reports) определяется новое задание и появляется окно, показанное на рис. 8.20. В нем вы можете указать преобразуемый отчет, в нашем примере — Sales_by_month, а затем — имя выходного файла (здесь — sales_by_month.pdf, поскольку мы создаем файл Adobe Acrobat).

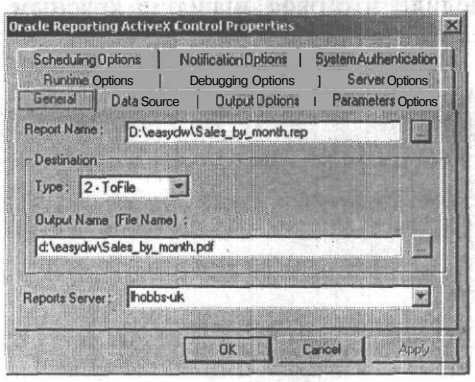


Рис. 8.20. *Oracle Reports Server — выбор отчета*

Как видно, можно указать несколько опций. Для указания базы данных, имени пользователя и пароля для извлечения данных, вы должны перейти на закладку Data Source (Источник данных). Если вы щелкнете по закладке Output Option (Формат вывода), появится окно, показанное на рис. 8.21, в котором можно указать формат выходного файла. Именно здесь мы выбираем формат Adobe Acrobat PDF.

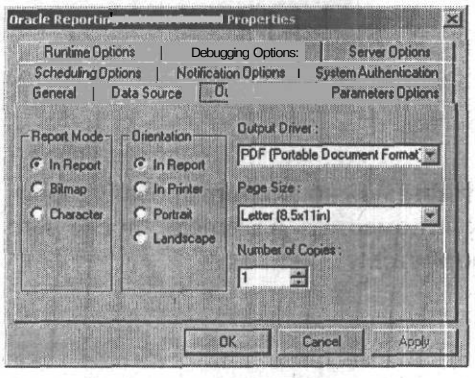


Рис. 8.21. *Oracle Reports Server — определение формата вывода*

При выборе форматов PDF и HTML, вы имеете возможность просматривать отчет в Web-браузере. Кроме того, Oracle Reports позволяет выводить отчет

- в файл;
- на принтер;
- в Oracle Portal;
- в e-mail;
- в кэш (для прямого вывода в браузер).

Преимущество использования Oracle Reports Server состоит в том, что отчет можно поставить в расписание и обновлять на регулярной основе. На рис. 8.22 мы перешли к закладке Scheduling Options (Параметры расписания), в которой нам было предложено указать проведение ежечасного обновления. Как только отчет создан, его можно распространить или разместить для чтения пользователями в портале или файлах Oracle.

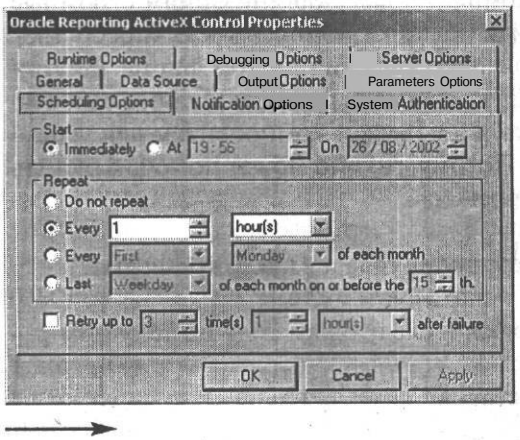


Рис. 8.22. *Oracle Reports Server — расписание для отчета*

Когда мы закончим с созданием задания, оно автоматически будет помещено в очередь отчетов. Когда придет время его запуска, будут проведены соответствующие преобразования. На рис. 8.23 мы видим наш отчет с именем Sales_by_month, выполнение которого прошло успешно, и был сгенерирован файл PDF.

Теперь, имея отчет, мы можем распространять его по e-mail, в файлах Oracle или через портал. Мы надеемся, вы согласитесь, что намного легче использовать имеющиеся инструменты для упрощения и ускорения выполнения данной задачи.

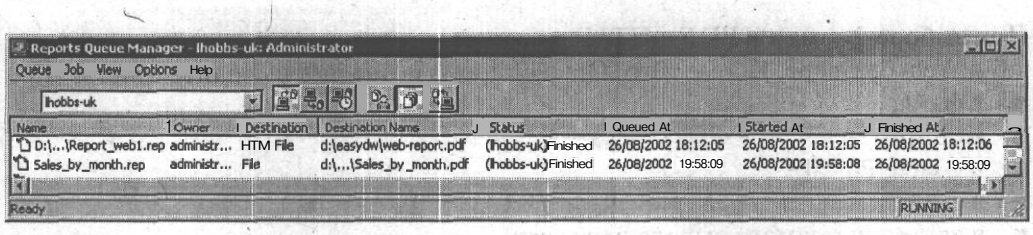


Рис. 8.23. *Oracle reports Server — очередь заданий*

8.4. Хранилища данных и информационные системы для электронной коммерции

Интернет и сети Интранет отдельных компаний дают организациям огромные возможности. Для бизнеса они могут найти новых клиентов, а также способствовать тому, что клиент может отдать свои предпочтения другой фирме. Внутри компаний они предоставляют собой гибкую, удобную в использовании рабочую среду. Хранилище данных может оказаться ключевым компонентом для успеха вашего электронного бизнеса. Почему? Потому что оно может помочь ответить на такие важные для успеха бизнеса вопросы, как:

- Какие продукты покупают люди?
- Когда продаются эти продукты: в какое время года и в какое время дня?
- В какое время есть возможность отключать систему?
- Как можно предсказать объемы продаж в следующем году по текущим результатам?
- Во что нам обойдется бесплатная доставка?

Мы надеемся, что эта глава дала вам представление о том, как можно использовать Web совместно с хранилищем данных. Существует множество способов его применения, и мы лишь слегка коснулись поверхности в вопросе имеющихся средств и методов. Кроме того, Oracle непрерывно совершенствует свои инструменты, стремясь облегчить создание web-приложений, и новые информационные сообщения поступают постоянно, поэтому старайтесь посещать сайт http://www.oracle.com и получать самую свежую информацию. Если вам никогда не приходилось настраивать web-сайт, попробуйте себя в этом с помощью сайта http://myoracle.com.

9

Oracle 9i OLAP и интеллектуальная обработка информации

Ни одна книга о хранилищах не будет полной без обсуждения двух их важных областей применения: OLAP и интеллектуальной обработки информации (data mining). В Oracle 9i эти функции интегрированы в сервер базы данных и имеются как необязательное дополнение версии Enterprise Edition. В данной главе мы обсудим эти функции.

9.1. Oracle OLAP

Аналитическая обработка в режиме он-лайн (Online Analytical Processing, OLAP) используется при обработке запросов, которые дают информацию о бизнесе. Как правило, при запросах OLAP выполняются сложные вычисления, такие, как сравнение по периодам времени, выявление наиболее продаваемых продуктов, анализ временных рядов и прогнозирование будущих продаж. Данные анализируются по многим измерениям, таким, как продукт, географический регион и время. Поэтому такой тип данных часто называют многомерными данными.

ROLAP и MOLAP

С тех пор как зародилась индустрия OLAP, продолжается дискуссия о том, как лучше хранить многомерные данные. Одна школа выступает за хранение данных в таблицах реляционных баз, известных своей способностью масштабироваться и хранить большие объемы данных. Такой подход называется Relational OLAP (реляционная OLAP) или ROLAP. В этом случае анализ выполняется с помощью SQL-запросов. Все свойства SQL, которые мы обсуждали в этой книге, будут применимы и для Relational OLAP. Другая школа считает, что обработку многомерных данных нужно выполнять с использованием специализированного формата

хранения, который называется multidimensional database (многомерная база данных, MDDB) и который специально создан для быстрой обработки запросов OLAP. Этот подход называют Multidimensional OLAP (многомерная OLAP) или MOLAP. Многие производители программного обеспечения предлагают возможности MOLAP, и в том числе они имеются в продукте Oracle Express Server. Преимущество таких продуктов в том, что пользователи могут очень легко получать доступ к данным без составления сложных запросов. Однако главной проблемой такого подхода является то, что даю проведения анализа, данные нужно перенести из реляционной базы, в которой они, как правило, хранятся, в многомерную базу. Это означает, что данные дублируются и возможна их десинхронизация. Более того, продукты с технологией MOLAP, которые концентрируются, в первую очередь, на анализе, не обеспечивают такого уровня безопасности и надежности, как реляционные базы. Исходя из сказанного, оба механизма имеют свои преимущества, и верный выбор между ними будет зависеть от конкретной области применения. По сути, до появления Oracle 9i, в большинстве случаев приходилось вкладывать деньги в обе технологии: в реляционную базу для простого анализа и составления отчетов и в специализированный продукт для более сложного анализа и планирования бизнеса. Oracle 9i устраняет необходимость иметь два отдельных аналитических продукта, позволяя выполнять и то, и другое в сервере базы данных Oracle.

Oracle 9i OLAP

На протяжении несколько лет, начиная с появления Oracle 8i, компания Oracle включает функцию OLAP в базу данных, поддерживая реляционную OLAP. Аналитические функции и операторы агрегирования CUBE, ROLLUP и GROUPING SET, которые обсуждались в главе 3, позволяют пользователям выполнять сложные OLAP-вычисления с помощью SQL. Материализованные представления и переписывание запросов позволяют заранее агрегировать данные, чтобы ответы на запросы можно было давать быстрее.

В версии Oracle 9i Release 2 Oracle теперь также поддерживает многомерную OLAP прямо в базе данных. Это осуществляется с помощью функции OLAP Option, которая входит в Oracle 9i Database, Enterprise Edition. С помощью OLAP Option база может хранить данные в многомерном формате в так называемой аналитической рабочей области (analytical workspace). Более того, в базу данных встроена полнофункциональная система многомерных вычислений. Это означает, что у вас есть все аналитические возможности, которые дает традиционный продукт MOLAP, но при использовании Oracle 9i OLAP Option вы также пользуетесь дополнительными преимуществами безопасности, управляемости

и надежности, которые дает система управления базой. Таким образом, Oracle 9i Release можно назвать реляционно-многомерной OLAP-базой!

С появлением Oracle 9i OLAP Option вычислительные возможности продукта MOLAP от Oracle, Express Server, теперь доступны в самой базе данных. Существующие базы данных Oracle Express Server можно перенести в аналитические рабочие области базы данных. Вы можете создавать приложения Java для OLAP, используя интерфейсы OLAP API и компоненты, называемые BI Beans, которые встроены в JDeveloper. Это аналогично среде разработки приложений Oracle Objects, которая поддерживалась в Oracle Express, за исключением того, что такой способ гораздо более мощный. В будущем такие продукты, как Oracle Financial Analyzer и Oracle Sales Analyzer, которые используют Express, могут быть заменены приложениями, использующими данную возможность Oracle 9i OLAP.

Oracle 9i OLAP Option и Discoverer

Вы можете захотеть узнать, зачем нужна такая программа, как OLAP Option, когда есть такие средства, как Discoverer. Discoverer и OLAP Option выполняют дополняющие функции. Discoverer — это пакетное приложение, служащее для создания специальных запросов и отчетов. Оно разработано для конечных пользователей, которые могут не знать языка SQL. Однако в некоторых случаях и некоторым пользователям могут потребоваться более сложные аналитические функции, такие, как финансовое прогнозирование, бюджетные ассигнования и моделирование. Приложение OLAP Option предназначено именно для этих профессиональных аналитических нужд. Более того, OLAP Option также предоставляет среду для разработки OLAP-приложений. С помощью BI Beans и OLAP API вы можете писать собственные приложения, соответствующие требованиям вашего бизнеса.

Мы подробно обсуждали свойства обработки SQL-запросов в базах Oracle на всем протяжении этой книги. В данной главе мы сконцентрируемся на многомерной аналитической модели, предлагаемой в Oracle Option. Для тех кто ранее работал с Oracle Express, мы будем везде, где возможно, указывать на эквивалентные для этих двух продуктов концепции.

Давайте начнем с рассмотрения тех областей, где применение OLAP Option наиболее полезно.

9.1.1 Аналитические приложения

Когда хранилище данных создано, для бизнеса может понадобиться множество приложений, связанных с бизнес-информацией. Некоторые из них могут быть

связаны со специализированными запросами и отчетами, например Discoverer или Reports, которые мы обсудили в главе 6. Однако бизнесу также могут понадобиться несколько более мощных аналитических приложений, например, для планирования спроса, прогнозирования сбыта, составления бюджета корпорации и финансового моделирования. Такие приложения не только анализируют имеющиеся данные, но и генерируют новые. Oracle OLAP Option очень подходит как раз для таких приложений, поскольку обеспечивает хранение многомерных данных и предлагает аналитическую модель. Давайте коротко рассмотрим некоторые из операций, часто выполняемых такими приложениями.

Аналитические вычисления и агрегирование

Для OLAP-приложений часто требуется агрегирование и анализ, связанный с ранжированием (например, «горячая десятка продуктов»), вычислением временных рядов (например, скользящего среднего) и междустрочные вычисления (например, сравнение по периодам). Отметьте, что такие вычисления можно теперь выполнять на SQL, как это описывалось в главе 3, хотя по традиции такие вычисления проводятся с помощью средств многомерного анализа.

Прогнозирование

Прогнозирование (forecasting), как говорит само название, — это количественное предсказание, основанное на имеющейся предыстории, например прогнозирование продаж в следующем квартале на основе данных по прошлому году. Для этих целей используются мощные статистические алгоритмы, такие, как линейная и нелинейная регрессия, одинарное или двойное экспоненциальное сглаживание и метод Холта-Винтерса.

Распределение

Распределение (allocation), также называемое обратной агрегацией, используется для разделения количественных значений, например бюджета или квоты, на несколько частей. Распределение — важная часть приложений для бизнес-планирования. Для таких приложений требуется распределить бюджет по иерархической структуре или среди конкретных членов организации. Весовые коэффициенты для распределения могут определяться пользовательскими параметрами или аналитическими функциями.

Финансовые расчеты

Такие вычисления, как определение процентов или графика выплат, можно легко осуществлять в табличной среде.

Моделирование

Моделирование включает в себя описание количественных величин с помощью набора уравнений. Затем модель можно использовать для вычисления других величин, подставляя данные в эти уравнения. В уравнение может входить предполагаемый порядок зависимостей данных, и с их помощью можно вычислять новые значения для таблиц измерений и таблиц фактов. Например, в простую программу моделирования могут входить такие уравнения:

```
Прибыль = Выручка — Стоимость
Выручка = КаналПродаж('Интернет') + КаналПродаж('Розница')
РозничнаяПродажа('Одежда') = РозничнаяПродажа('Мужская одежда') +
                                    РозничнаяПродажа('Женская одежда')
РозничнаяПродажа(квартал) = lag(РозничнаяПродажа,квартал),1) * 1.1
...
```

Анализ возможных вариантов

Анализ возможных вариантов (what-if analysis) или управление сценариями — это очень важный аспект профессиональных аналитических приложений. Сюда входит анализ данных при гипотетическом сценарии развития событий с целью определения возможного их влияния на бизнес. Например, во что обойдется компании закрытие некоторых магазинов розничной торговли и запуск он-лайнового магазина? Какое влияние на выручку окажет внесение изменений в организацию продаж? Для анализа возможных вариантов требуется транзакционная модель, отличающаяся от того, что предлагает традиционный SQL. Пользователи должны иметь возможность локализованно изменять структуру и содержание данных, не делая эти данные видимыми для всей базы. Более того, изменения могут быть временными, и пользователь должен имеет возможность восстановить предыдущее состояние данных. Можно считать такие действия долгосрочным сеансом, в котором вы выполняете последовательность DDL или DML-операций и вносите и отменяете изменения. Видимы эти изменения только для вашего сеанса.

В Oracle 9i OLAP все эти операции поддерживаются с помощью 9i OLAP Option.

9.1.2. Архитектура Oracle OLAP

Теперь мы опишем различные компоненты, которые образуют Oracle 9i OLAP, а затем обсудим использование каждого из них.

Совет. В архитектуре Oracle OLAP с появлением версии 9i Release 2 произошли существенные изменения. Все примеры и функциональные возможности, описываемые в этой главе, относятся к версии 9i Release 2.

На рис. 9.1 показана архитектура Oracle 9i OLAP. Она состоит из следующих компонентов:

- база данных Oracle 9i и система обработки SQL;
- аналитические рабочие области;
- система вычислений OLAP;
- командный язык OLAP DML;
- табличные функции SQL;
- каталог OLAP;
- OLAP API;
- BI Beans.

База данных Oracle 9i

База данных Oracle 9i служит источником всех данных для OLAP Option. Данные могут храниться в реляционных таблицах (как правило, по схеме звезды или снежинки) или в многомерном формате в аналитических рабочих областях внутри базы Oracle. Объединение всех типов данных в одном месте дает много преимуществ.

- Управлять всеми данными можно с помощью одного приложения — Oracle Enterprise Manager.
- База данных предлагает единый механизм безопасности для всех типов данных.
- Данные не нужно перемещать в отдельную базу, и поэтому уменьшается задержка перед тем, как данные станут доступными для анализа.

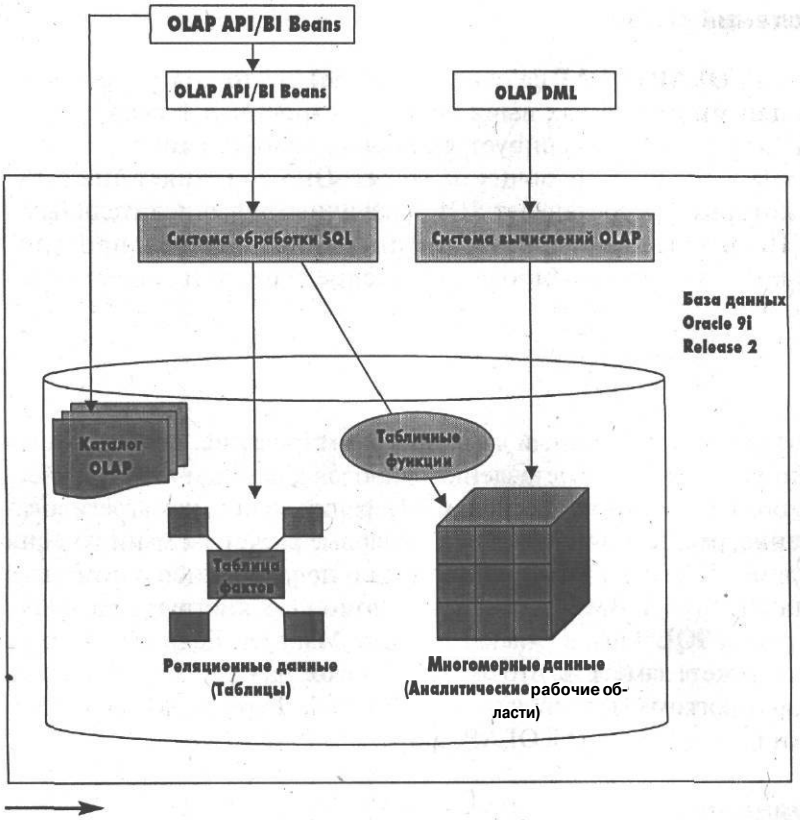


Рис. 9.1. *Архитектура Oracle 9i Release 2*

- Повышается надежность и доступность данных благодаря таким приложениям, как Real Application Clusters и Data Guard.

Аналитическиерабочие области

Аналитическая рабочая область — это место хранения данных в многомерном формате в базе Oracle. Как и любой другой объект базы данных, аналитическая рабочая область имеет конкретного владельца. Однако в отличие от других объектов аналитическая рабочая область может быть постоянной или временной (т. е. на время анализа). Аналитические рабочие области используются, когда требуется применить мощные аналитические возможности системы вычислений OLAP.

Система вычислений OLAP

Система вычислений OLAP (OLAP calculation engine) — это полнофункциональная система для многомерных вычислений, встроенная в сервер базы данных Oracle 9i. Эта система оперирует данными, хранящимися в многомерной форме в аналитических рабочих областях. Она дополняет аналитические свойства, которые предоставляет SQL, различными вычислительными возможностями. Некоторые из распространенных типов анализа, проводимых с ее помощью, — это прогнозирование, распределение и моделирование.

OLAP DML

OLAP DML — это простой, но мощный язык программирования, используемый для загрузки, манипулирования и составления запросов к данным в аналитических рабочих областях. Он предлагает несколько операций, таких, как агрегирование, прогнозирование, регрессионный анализ, числовые расчеты и манипуляции с временными рядами. Команды OLAP DML можно подавать либо с помощью OLAP API, пакета PL/SQL DBMS_AW, либо с помощью инструмента OLAP Worksheet (похожего на SQL*Plus) в Oracle Enterprise Manager. Если вы знакомы с Oracle Express, вы можете заметить, что он очень похож на язык SPL. Фактически, существующие программы, написанные для Oracle Express, должны с небольшими изменениями работать и в OLAP DML.

Табличные функции

В главе 5 мы видели, как можно использовать табличные функции для выполнения различных ETL-операций (т. е. операций извлечения-преобразования-загрузки) в хранилище. Табличные функции могут выполнять любые виды вычислений, но выходные данные их представляют собой набор строк. Поэтому их можно использовать в SQL-запросах, как если бы они были таблицей базы. Благодаря этой своей возможности, эти функции дают механизм доступа к многомерным данным с помощью SQL. Если у вас есть табличная функция, дающая доступ к аналитической табличной области с помощью OLAP DML, приложения могут использовать ее как таблицу в обычном SQL-запросе. Oracle предлагает табличную функцию, называемую OLAP_TABLE, которая дает доступ с помощью SQL к аналитической рабочей области. Это делает Oracle OLAP доступным для пользователей, незнакомых с OLAP DML.

Каталог OLAP

Каталог OLAP (OLAP Catalog) описывает логическую модель метаданных для многомерной информации с помощью измерений, уровней, иерархий, атрибутов, вычислений и кубов. Вы должны быть знакомы с терминами «измерения», «уровни», «атрибуты» и «иерархии» объекта-измерения SQL, которые мы обсуждали в главе 4. Мера (measure) — это то же самое, что факт в таблице фактов, например purchase_price. Как правило, ее используют при агрегировании. Куб (cube) определяет, каким образом рассчитываются меры по разным измерениям. Позже в данной главе мы опишем, каким образом вы можете создавать такие вещи с помощью Oracle Enterprise Manager. Заметим, что каталог OLAP просто определяет метаданные, но не помещает данные в куб. Если вы хотите использовать OLAP API, BI Beans или создавать аналитические рабочие области с помощью пакетов, поставляемых Oracle, вы должны создать метаданные в каталоге OLAP.

OLAP API

OLAP API — это набор интерфейсов Oracle OLAP для программирования на Java. OLAP API позволяет разработчикам приложений писать программы для проведения вычислений, отбора многомерных данных и навигации по ним. Поскольку эти интерфейсы основаны на Java, они обеспечивают переносимую на разные платформы, объектно-ориентированную основу для разработки OLAP-приложений. Чтобы объектами можно было манипулировать с помощью OLAP API, их нужно сначала определить в каталоге OLAP.

BI Beans

BI Beans — это компоненты, специально разработанные для быстрого создания OLAP-приложений. Вы можете создавать BI Beans с помощью простых Мастеров в JDeveloper и хранить их на постоянной основе в базе данных. С помощью BI Beans можно выполнять различные операции, такие, как соединение с базой данных, создание аналитических вычислений и отображение их в различных графических и табличных форматах. Затем их можно использовать в приложениях Java и JSP, в которых требуются аналитические функции. Для доступа к данным BI Beans используют интерфейсы OLAP API.

Теперь мы подробно рассмотрим каждый из этих компонентов.

9.1.3. Аналитические рабочие области

Аналитические рабочие области позволяют хранить данные в многомерной форме. Как и реляционные таблицы, аналитическая табличная область принадлежит к конкретной схеме и использует для хранения данных табличную область базы Oracle. Вы можете использовать аналитические рабочие области для хранения данных, используемых в таких вычислениях, как прогнозирование и распределение.

Совет Существующие базы данных Oracle Express также можно переносить в аналитические рабочие области базы Oracle.

Аналитические рабочие области могут быть постоянными или временными, в зависимости от ваших нужд. Если вам нужно произвести вычисления, но не нужно сохранять результаты, вы можете удалить изменения вместе с аналитической рабочей областью по завершению сеанса. Временные аналитические рабочие области часто используются при анализе возможных вариантов, когда нужно испробовать несколько разных гипотетических сценариев, не делая изменения постоянными. В отличие от реляционных таблиц, в которых изменения, например добавление столбца, внесенные с помощью DDL, автоматически становятся видимыми для всей базы, все изменения, вносимые в аналитические рабочие области, являются локальными для сеанса до тех пор, пока противоположное не будет указано явно.

Также вы можете хранить в аналитической рабочей области агрегированные данные, во многом так же, как и в материализованном представлении, однако в этом случае вы будете отвечать за обновление этих данных. Механизмы инкрементного обновления для данных в аналитических рабочих областях недоступны.

Теперь давайте рассмотрим физическую модель хранения данных, используемую в аналитических рабочих областях, и чем она отличается от реляционной модели.

Концепции

На протяжении этой книги мы описывали, каким образом можно создавать хранилище, использующее реляционные таблицы, и управлять им. В нашем реляционном мире данные, как правило, хранятся в схеме звезды или снежинки. Таблица

фактов хранит различные количественные показатели, которые нам нужно анализировать по отношению к каждому измерению. Объекты-измерения используются для определения иерархических взаимоотношений между различными столбцами таблиц-измерений. Такие таблицы измерения часто называют уровневыми измерениями (level-based dimensions). Для того чтобы произвести вычисление, запрос должен соединить таблицу фактов и измерений с помощью определенного предиката.

При многомерном формате данных, таком, как в аналитических рабочих областях, нет таблиц или столбцов. Вместо этого есть три главных типа сущностей.

- Измерение. В отличие от объекта-измерения SQL, который выражает взаимоотношения между столбцами, здесь измерение соответствует списку значений. Такие измерения называются величинными измерениями (value dimensions). Например, измерение-город может состоять из значений Бостон, Лондон и Сан-Франциско. Географическое измерение может состоять из значений Мир, Соединенные Штаты, Массачусетс, Нью-Хэмпшир, Великобритания и Лондон. Таким образом, отсутствует иерархия, которую подразумевает измерение. Чтобы указать взаимоотношения между разными значениями измерения, вы должны создать связь (relation).
- Связь. Связь заключает в себе отношение между значением в одном измерении и другим значением в том же или ином измерении. Заметьте, что в связях может объявляться взаимоотношение между двумя значениями в одном и том же измерении. Это называется «связь с самим собой» (self-relation). Как иерархия описывает взаимоотношения в уровневом измерении, также связи описывают взаимоотношения в величинном измерении.

Можно считать взаимоотношение, определяемое связью, идентичным таблице измерения родитель-потомок в реляционной схеме. На рис. 9.2 показаны различия между уровневой таблицей измерения и таблицей родитель-потомок в реляционном мире.

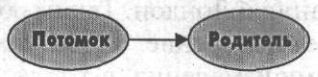
В уровневом измерении каждый уровень хранится в отдельном столбце. Например, могут быть столбцы для города, штата, и региона. С другой стороны, в измерении родитель-потомок все значения хранятся в столбце-потомке, а родительский столбец содержит соответствующее значение-родитель. Предположим, например, что столбец-потомок содержит значения Boston, San Francisco и MA. Для каждого значения в столбце-потомке будет соответствующее значение в столбце-родителе. Для значения-потомка Boston родительским значением будет MA . В отличие от уровневой таблицы-измерения, в которой значения верхних уровней повторяются для каждого значения нижнего уровня, в измерении родитель-потомок каждое отношение сохраняется только один раз, что автоматически обеспечивает правильность данных.

Город	Штат	Регион
Boston	MA	Северо-восток
Waltham	MA	Северо-восток
Burlington	VT	Северо-восток
San Francisco	CA	Запад
Son Jose	CA	Запад



Уровневое измерение

Потомок	Родитель
Boston	MA
Waltham	MA
Burlington	VT
San Francisco	CA
San Jose	CA
MA	Северо-восток
VT	Северо-восток
CA	Запад
Северо-восток	N/A
Запад	N/A



Измерение Родитель-Потомок

Рис. 9.2. *Таблицы уровневого измерения и измерения родитель-потомок*

- **Переменная:** Переменная используется для хранения данных, и она эквивалентна таблице фактов. Переменная определяется относительно специфического набора измерений. На рис. 9.3 показана концептуальная схема хранения данных в переменной. В этом примере, переменная Sales имеет измерения географическое и временное. Вы можете запросить значение переменной для любого значения из измерений, для которых она определена. Например, значение переменной Sales для соединенных Штатов (United States) для года 2002 равно $3102. Заметим, что это похоже на электронную таблицу, в которой вы можете получить значение любой ячейки, просто указав строку и столбец.

У такого многомерного формата есть несколько преимуществ:

- Он повышает ссылочную целостность. Например, если переменная определена по измерениям Покупатели и Время, каждая ячейка будет иметь уникальное значение для данного покупателя и времени. Кроме того, взаимоотношения между значениями измерений сохраняются только один раз, и вы застрахованы от таких несоответствий, как Бостон, штат Массачусетс и Бостон, штат Калифорния.

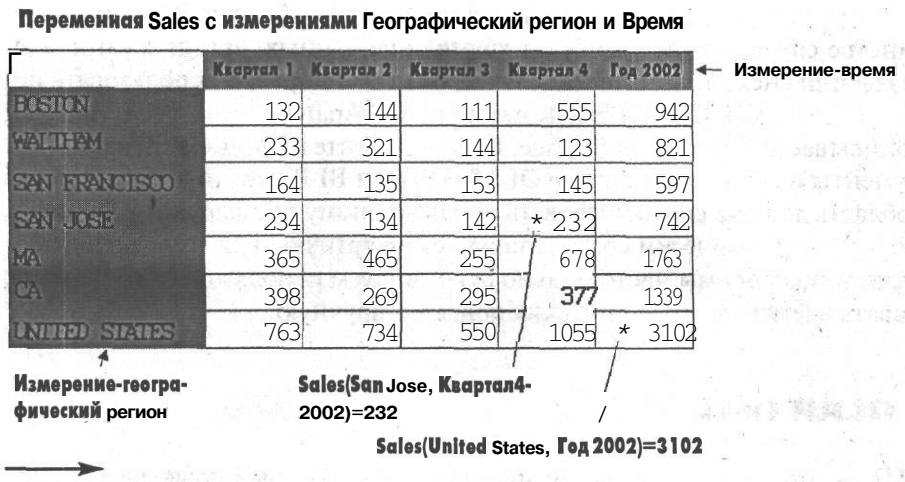


Рис. 9.3. *Концептуальная схема переменной*

- В измерении существует определенный порядок строк, который определяется при его создании. В SQL для определения порядка получаемых данных вы должны явным образом указать условие ORDER BY.
- Пользователи не должны указывать для получения ответов способ соединения таблиц фактов и измерений. Они просто запрашивают отчет по переменной для определенных значений измерений, как в электронной таблице. Однако, в отличие от электронной таблицы, здесь вы не ограничены двумя измерениями.
- Данные можно выдавать приложению полностью вычисленными. Когда администратор базы установит аналитическую рабочую область с различными вычислениями, пользователям приложений не придется описывать в запросе, как нужно проводить вычисления. Им нужно просто указать, какие из имеющихся вычислений им нужны, а система вычислений сама позаботится о деталях расчетов. Вычисление может представлять собой сложную аналитическую функцию, формулу или агрегат. Данные могут рассчитываться заранее для повышения производительности или подсчитываться «на лету». Однако приложению не требуется знать, как это делается.

На этом этапе вы можете захотеть узнать, что нужно для создания таких аналитических рабочих областей и для составления запросов к ним. Для манипулирования данными, хранимыми в аналитических рабочих областях, используется язык OLAP DML. С помощью этого языка можно создавать измерения и пере-

менные и загружать данные в аналитические рабочие области. Тем не менее, в большинстве случаев у вас уже будет хранилище данных, организованное по схеме звезды или снежинки. В таком случае вы можете просто использовать пакет CWM2_OLAP_AW_CREATEили приложение Analytic Workspace Manager, которое описывается ниже в этой главе. Если вы хотите использовать какие-либо инструменты от Oracle, например OLAP API или BI Beans, то аналитическая рабочая область должна соответствовать определенному стандартному формату. Данный пакет автоматически создаст такую стандартную аналитическую рабочую область, и поэтому мы настоятельно рекомендуем использовать эти пакеты, а не создавать элементы аналитических областей вручную.

9.1.4. OLAP DML

OLAP DML — это очень простой, но мощный язык, который позволяет составлять разнообразные вычисления и выводить отчеты подобно электронным таблицам, по данным, хранящимся в аналитической рабочей области. Он предлагает функции для прогнозирования, распределения, агрегирования, статистического анализа и финансовых расчетов.

Выполнять команды OLAP DML можно с помощью OLAP Worksheet в Oracle Enterprise Manager, с помощью пакета PL/SQL DBMS_AW или с использованием класса SPLExecutor в OLAP API.

Чтобы найти OLAP Worksheet в Oracle Enterprise Manager, раскройте узел Database (База данных) в панели Проводника, затем — Warehouse (Хранилище) и OLAP, а потом Object Menu (Меню объектов). Это простое приложение (напоминающее SQL*Plus), которое позволяет выполнять команды OLAP DML, а также редактировать, компилировать и запускать программы, написанные на языке OLAP DML. На рис. 9.4 показано приложение OLAP Worksheet. Вы запускаете команду в нижней части окна, и результат появляется в верхней части. В примере, показанном на рис. 9.4, мы подаем запрос к измерению с именем OLAP_TIME_KEY.

Совет. OLAP DML весьма отличается от SQL и совершенно не зависит от него. Однако в OLAP Worksheet есть режим SQL, в котором можно подавать обычные команды SQL, как в SQL*Plus. И наоборот, вы можете подавать команды DML в SQL*Plus с помощью пакета DBMS_AW.

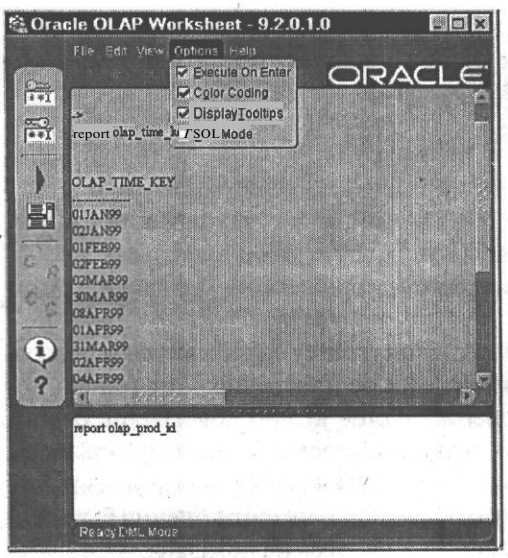


Рис. 9.4. *OLAP Worksheet в Oracle Enterprise Manager*

Перед тем как вы сможете подавать команды OLAP DML, вы должны создать новую или задействовать существующую аналитическую рабочую область. Ниже приведена команда AW CREATE, которая создает аналитическую рабочую область с именем EASYDWOLAP. С ней связывается табличная область OLAPTBSP.

```
AW CREATE EASYDWOLAP TABLESPACE olaptbsp
```

В нескольких следующих разделах мы покажем некоторые примеры использования OLAP DML с целью проиллюстрировать типы вычислений, которые можно выполнять при помощи аналитических рабочих областей. Однако эти примеры не являются руководством по OLAP DML. За этим вам нужно обращаться к документации Oracle. В OLAP Worksheet также есть великолепная справочная система, в которой на примерах описываются команды OLAP DML.

Измерения и Переменные

Измерения и переменные являются основными элементами многомерного формата. Мы вкратце покажем, как они определяются, чтобы вы могли понимать примеры, приводимые в последующих разделах.

В первом примере мы определяем эквивалент измерения с данными о покупателях — customer с иерархией customer_id, town и county (идентификатор, город и округ).

```
DEFINE cust_id DIMENSION TEXT
DEFINE cust_town DIMENSION TEXT
DEFINE cust_county DIMENSION TEXT
DEFINE customers DIMENSION CONCAT (cust_county cust_town cust_id)
DEFINE customers.parents RELATION customers <customers>
```

Каждый уровень иерархии — cust_id, cust_town и cust_county определяется в виде измерения. Команда CONCAT позволяет создать единое измерение — customers, которое содержит значения из всех трех измерений — cust_id, cust_town и cust_county. Выражение customers.parent RELATION — это связь с самим собой (self-relation), показывающее взаимоотношения типа родитель-потомок между различными значениями в измерении customers. Заметьте, что реальные взаимоотношения будут определяться при загрузке данных в измерения, которая здесь не показана.

Следующая группа команд определяет измерение-время, очень похожее на предыдущее измерение. Есть три отдельных измерения — time_id, time_month и time_year (идентификатор, месяц, год), соответствующие разным уровням, и связь timedim.parents, устанавливающая взаимоотношения родитель-потомок между значениями разных измерений.

```
DEFINE time_id DIMENSION TEXT
DEFINE time_month DIMENSION NUMBER(2)
DEFINE time_year DIMENSION NUMBER(4)
DEFINE timedim DIMENSION CONCAT (time_year time_month time_id)
DEFINE timedim.parents RELATION timedim <timedim>
```

Затем мы определим составное измерение (composite dimension), которое указывает, какие измерения будут использоваться вместе. Это полезно, когда в данных есть не все комбинации значений customers и timedim. В данном случае составное измерение будет хранить только важные пары <customers.timedim>.

```
DEFINE dims COMPOSITE <customers timedim>
```

Составное измерение будет использоваться для связывания с измерениями наших переменных, которые мы определяем в следующем примере. Как говорилось ранее, переменная (variable) похожа на таблицу фактов и должна быть связа-

на с одним или более доступным измерением. Мы определим две переменные — purchase_price и ship_charge, связанные с составным измерением DIMS, которое мы определили ранее.

```
DEFINE purchase_price
   VARIABLE NUMBER (6,2) <dims <customers timedim»

DEFINE ship_charge
   VARIABLE NUMBER (6,2) <dims <customers timedim»
```

Мы будем использовать эти измерения и переменные в следующих разделах.

Вывод данных с помощью OLAP DML

Мы можем выводить данные из переменной по ее измерениям командой REPORT. Команда REPORT возвращает ответ, сходный с электронной таблицей, в которой одно измерение соответствует строкам, а другое — столбцам. Можно ограничить запрашиваемые данные одним значением или набором значений, используя команду LIMIT. Это похоже на выбор данных с условием WHERE в SQL, однако в SQL критерий выбора определяется запросом, а команда LIMIT в OLAP DML существует столько, сколько вы будете соединены с аналитической рабочей областью.

В следующем примере мы делаем запрос к переменной purchase_price по измерению cust_county и по месяцам с января по март. Ключевое слово ACROSS означает, что значения измерения time_month будут выводиться в столбцах. Заметьте, что система вычислений автоматически выявит задействованные измерения с помощью определения переменной.

```
LIMIT customers to cust_county
LIMIT time_month to 1 to 3

REPORT across time_month : purchase_price

                          ----------PURCHASE_PRICE----------
                          ----------TIME_MONTH----------
CUSTOMERS                        1           2           3
---------------------     ----------  ----------  --------
<CUST_COUNTY: Hants>       1,976.03    1,924.90    3,286.77
<CUST_COUNTY: London>      2,335.90    2,555.28    3,122.70
<CUST_COUNTY: Mersey>      2,335.90    2,555.28    3,122.70
```

Используя переменные, которые были определены ранее, можно также вычислять другие переменные. В следующем примере определяется переменная с именем totalsales, в которой вычисляется сумма purchase_price и shipping_charge.

```
DEFINE TOTALSALES VARIABLE NUMBER (6,2) <DIMS <CUSTOMERS TIMEDIM>>
```

Затем мы вычисляем эту переменную для самых низких уровней измерений customers и timedim.

```
LIMIT timedim TO time_id
LIMIT customers TO cust_id
ACROSS CUSTOMERS TIMEDIM
    DO 'TOTALSALES = PURCHASE_PRICE + SHIP_CHARGE'
```

Теперь мы можем вывести эту новую переменную. В следующем примере мы запрашиваем значение totalsales по первым трем месяцам и по каждому покупателю. Отметьте, что нам не надо указывать в запросе, как вычислять totalsales. Здесь становится очевидной мощь OLAP DML.

```
LIMIT CUSTOMERS TO CUST_ID
LIMIT TIME_MONTH TO 1 TO 3

REPORT across time_month : totalsales

                          ----------TOTALSALES---------
                          ----------TIME_MONTH----------
CUSTOMERS                        1          2          3
-------------------      ---------  ----------  -------
<CUST_ID: AB123456>           32.51      134.24     143.46
<CUST_ID: AB123457>           65.02      188.42      65.02
...
```

Агрегирование данных

Вы можете использовать OLAP DML для проведения агрегирования. Давайте предположим, что мы загрузили в переменную purchase_price данные, соответствующие значениям cust_id и time_id. Теперь нам нужно вычислить значения для более высоких уровней, таких, как cust_county и time_month с помощью агрегирования. Для агрегирования данных мы должны определить карту агреги-

рования (aggregation map) с помощью команды AGGMAP. Карта агрегирования дает системе вычислений инструкции по проведению агрегирования: какую агрегационную функцию применять, какие уровни агрегировать и какие агрегаты вычислять заранее. Когда у администратора базы будет определена карта агрегирования, ее можно будет использовать, и не зная подробностей того, как производится агрегация.

В следующей команде AGGMAP показано, что нам нужно вычислить агрегаты по измерению time_id и измерению cust_id. Команда AGGMAP указывает отношения, определяющие то, каким образом происходит агрегирование уровней.

```
DEFINE easyagg AGGMAP
AGGMAP
RELATION timedim.parents PRECOMPUTE (ALL)
RELATION customers.parents PRECOMPUTE (ALL)
END
```

Теперь вы можете выводить данные, используя предыдущую карту.

```
>report across time_month : aggregate(ship_charge using easyagg)
                          ----AGGREGATE(SHIP_CHARGE-------
                          -------USING EASYAGG1)---------
                          ---------------TIME_MONTH-------------
CUSTOMERS                        1          2          3
-----------------------   ----------  ---------- ---------
<CUST_TOWN: Eastleigh>          2.95       7.45      11.95
<CUST_TOWN: Soton>             10.40      14.90      23.90
<CUST_TOWN: Chandlers>          2.95       5.90      11.95
<CUST_TOWN: Soton2>             2.95       5.90      11.95
<CUST_TOWN: Soton3>             2.95       5.90       2.95
...
```

В приведенном выше примере мы указали предварительно вычислять все агрегаты, но мы можем быть и более изощренными и задать предварительное вычисление только некоторых уровней или только некоторых значений.

Прогнозирование

Одной из обычных операций, выполняемых с помощью OLAP DML, является прогнозирование (forecasting). Это одна из многих профессиональных аналитических

операций, которую можно выполнить в аналитических рабочих областях на OLAP DML, но нельзя выполнить на SQL. Чтобы предсказать количественное значение, например величину продаж, мы должны выполнить следующие шаги:

- определить переменные для хранения результатов прогнозирования;
- определить параметры прогнозирования;
- выполнить прогнозирование.

Мы покажем очень простой пример прогнозирования продаж в следующие 6 месяцев на основе данных по предыдущим 6 месяцам.

Первый этап — это определить переменную с именем SALES_FORECAST, в которой будет храниться результат прогнозирования. Заметьте, что мы опять связали с этой переменной составное измерение DIMS.

```
DEFINE SALES_FORECAST VARIABLE NUMBER(6,2) <DIMS <TIMEDIM CUSTDIM>>
```

Затем мы ограничиваем измерение timedim уровнем time_month, а измерение customers — уровнем cust_id. Это означает, что прогноз будет вычисляться с использованием месяцев из измерения-времени и всех значений cust_id из измерения customers.

```
LIMIT TIMEDIM TO TIME_MONTH
LIMIT CUSTOMERS TO CUST_ID
```

Чтобы указать параметры и запустить прогнозирование, мы должны создать дескриптор (handle), который будет использоваться в последующих командах. Дескриптор с именем sf_handle получается путем вызова команды FCOPEN.

```
DEFINE sf_handle VARIABLE INTEGER;

sf_handle = FCOPEN('EasyDWSalesForecast')
```

Затем мы укажем параметры прогнозирования с помощью команды FCSET. Для прогнозирования мы используем автоматический метод. 6 временных периодов (месяцев) будут считаться данными предыстории, и при прогнозировании будет использоваться параметр периодичности 2.

```
FCSET sf_handle method 'automatic' histperiods 6 periodicity 2
```

И, наконец, мы запускаем прогнозирование с помощью команды FCEXEC. Мы должны указать имя измерения-времени, а также переменную, которая содер-

жит данные, используемые для прогнозирования, в нашем случае — purchase_price. Результаты помещаются в переменную sales_forecast, которую мы определили ранее.

```
FCEXEC sf_handle TIME timedim INTO sales_forecast purchase_price
```

В конце мы закрываем дескриптор.

```
FCCLOSE sf_handle
```

Теперь вы можете использовать команду AGGMAP, которую мы обсуждали ранее, для агрегирования данных прогноза в более высокие уровни для последующего вывода. Предположим, что мы уже сделали это, и мы можем теперь вывести переменную sales_forecast, как это делалось ранее, и увидеть результаты прогнозирования для месяцев с 9 по 11.

```
LIMIT time_month to 9 to 11
LIMIT customers to cust_id

REPORT across time_month: sales_forecast

                                 ----------SALES_FORECAST----------
                                 ----------TIME_MONTH----------
CUSTOMERS                          9           10           11
---------------------------    ----------  ----------  ----------
<CUST_ID: AB123456>            232.51      334.24      343.46
<CUST_ID: AB123457>            165.02      188.42      265.02
...
```

Создание пользовательских вычислений

При помощи OLAP DML вы можете также определять формулы для выполнения вычислений с помощью простых арифметических и аналитических функций. Когда формула будет определена, пользователи смогут ссылаться на нее при создании отчета, как на любую другую заранее созданную переменную. Мы опять должны подчеркнуть, что конечному пользователю не нужно знать, как выполняются эти вычисления и какие аналитические функции используются.

В следующем примере мы определяем простую формулу для итоговой стоимости (totalsales), складывая purchase_price (цену продукта) и ship_charge (стоимость доставки).

```
DEFINE TOTALSALES FORMULA DECIMAL <TIMEDIM CUSTOMERS>
EQ purchase_price + ship_charge
```

В следующем примере выводятся значения totalsales для 4, 5 и 6 месяцев.

```
LIMIT time_month TO 4 to 6
LIMIT timedim TO time_month

REPORT across cust_dim : totalsales
                         TOTALSALES
                      CUST_COUNTY
TIMEDIM                  Hants       LondonC       Mersey
---------------      ---------     ---------   -----------
<TIME_MONTH:  4>      6,376.96      7,687.26      7,687.26
<TIME_MONTH:  5>      4,493.81      5,363.68      5,363.68
<TIME_MONTH:  6>     13,749.80     16,663.60     16,663.60
```

В следующем примере определяется более сложная формула для вычисления изменений в продажах, произошедших за определенный период времени относительно предыдущего периода с помощью функции LAGDIF. Ключевое слово NOSTATUS показывает, что функция LAGDIF будет вычисляться по всем значениям в измерении timedim, невзирая на имеющиеся условия LIMIT.

```
DEFINE SALES_PREV_MONTH FORMULA
DECIMAL <TIMEDIM CUSTOMERS>
EQ LAGDIF(PURCHASE_PRICE,1,TIMEDIM,NOSTATUS)
```

Выводим данные *с* теми же ограничениями, что и для предыдущего примера.

```
REPORT across cust_dim: sales_prev_month
                      SALES_PREV_MONTH
                      CUST_COUNTY
TIMEDIM                  Hants        LondonC         Mersey
---------------      ---------    -----------    -----------
<TIME_MONTH:  4>      3,885.18      4,481.96       4,481.96
<TIME_MONTH:  5>     -1,883.15     -2,323.58      -2,323.58
<TIME_MONTH:  6>      9,254.98     11,298.91      11,298.91
```

В данном разделе мы предложили вам быстрый, но широкий обзор OLAP DML и аналитических рабочих областей. Мы увидели, как данные хранятся, агрегируются, вычисляются и выводятся в многомерном формате. Мы лишь слегка коснулись поверхности богатых возможностей OLAP DML, но надеемся, что вы получили определенное представление о простоте и мощи этого языка.

Помимо интерактивной подачи команд OLAP DML в OLAP Worksheet, вы также можете использовать их в приложениях, как это показывается в следующем разделе.

9.1.5. Доступ к аналитическим рабочим областям из приложения

Oracle OLAP предоставляет приложениям несколько способов доступа к многомерным данным в аналитической рабочей области. Эти способы объединены на рис. 9.5. Приложение на PL/SQL, использующее OLAP DML, может осуществлять

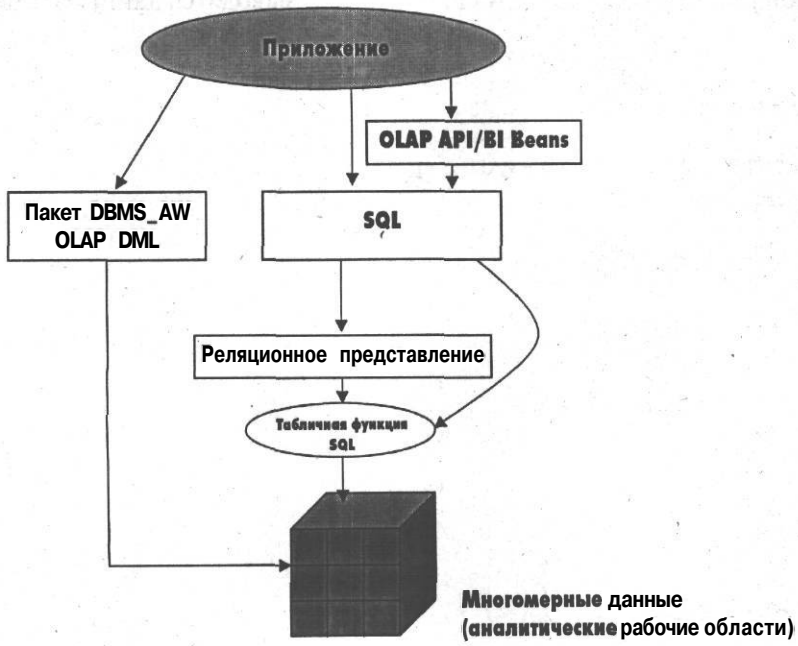


Рис. 9.5. *Доступ приложений к аналитическим рабочим областям*

доступ к аналитической рабочей области с помощью пакета DBMS_AW PL/SQL. Программы на Java могут получать доступ через OLAP API при условии, что определены метаданные каталога OLAP. Приложения на SQL также могут осуществлять доступ к данным путем определения табличных функций, которые будут переводить данные в строки. При необходимости поверх табличных функций можно определить реляционные представления, чтобы еще больше инкапсулировать доступ к аналитической рабочей области.

Теперь мы коротко рассмотрим каждый из этих механизмов.

Пакет DBMS_AW

Пакет DBMS_AW предоставляет функции, которые позволяют выполнять команды и программы на OLAP DML в программах на PL/SQL или SQL*Plus.

Процедуру EXECUTE можно использовать для выполнения одной или более команд OLAP DML, а ее выходные данные можно вывести на экран с помощью пакета DBMS_OUTPUT. В следующем примере производится соединение с аналитической рабочей областью EASYDWOLAP и выводятся данные измерения cust_town.

```
SET SERVEROUTPUT ON;
BEGIN
  DBMS_AW.EXECUTE ( ' aw attach easydwolap
        report cust_town');
END;
/

CUST_TOWN
--------------
Eastleigh
Soton
Chandlers
Soton2
Soton3
Soton4
Soton5
...
```

Доступ к многомерным данным из SQL

Приложения могут осуществлять доступ к многомерным данным, хранимым в аналитических рабочих областях, из SQL при помощи табличных функций SQL. База Oracle предоставляет для этого табличную функцию, называющуюся OLAP_TABLE, но вы также можете писать свои собственные табличные функции. Например, предположим, что нам нужно извлечь данные из определенной ранее аналитической рабочей области EASYDWOLAP. Мы хотим получить значения purchase_price по измерениям customer и time. Чтобы использовать функцию OLAP_TABLE, как и любую другую табличную функцию, которые описаны в главе 5, нам нужно сначала определить тип (TYPE), который описывает возвращаемые строки и тип, описывающий таблицу, чьи строки являются объектами первого типа.

В нашем примере мы определяем тип purchase_price_type следующим образом.

```
CREATE TYPE purchase_price_type AS OBJECT
 (cust_id VARCHAR2 (80),
  time_id VARCHAR2 (80),
  purchase_price NUMBER) ;
/
```

Затем мы определяем тип purchase_price_tbl, который описывает таблицу, строки которой имеют тип purchase_price_type.

```
CREATE TYPE purchase_price_tbl AS TABLE OF purchase_price_type;
/
```

Теперь мы можем получить значения purchase_price по нужным нам измерениям с помощью функции OLAP_TABLE.

```
SELECT * FROM TABLE(OLAP_TABLE(
 'easydwolap duration session', 'purchase_price_tbl', '',
 'measure purchase_price from purchase_price
  dimension cust_id from cust_id
  dimension time_id from time_id'));
```

Функция OLAP_TABLE принимает в качестве параметров имя аналитической рабочей области, ее длину (easydwolap duration session), тип результата (pur-

chase_pice_tbl) и команды OLAP DML для получения результата, заносимого в разные столбцы. Также есть необязательный ограничительный параметр, который позволяет получать только часть данных.

Теперь вы можете создать реляционное представление по данной табличной функции, чтобы приложения могли получать данные даже без необходимости знать что-либо о табличных функциях и OLAP DML.

```
CREATE VIEW purchase_price_vw
as
SELECT * FROM TABLE(OLAP_TABLE(
'easydwolap duration session', 'purchase_price_tbl', '',
'measure purchase_price from purchase_price
 dimension cust_id from cust_id
 dimension time_id from time_id'));
```

Теперь к этому представлению можно обращаться, как к любой другой реляционной таблице. Например, в следующем примере мы агрегируем purchase_price с помощью функции SUM.

```
SELECT time_id, SUM(purchase_price) as purchases
FROM purchase_price_vw
GROUP BY time_id;
TIME_ID       SUM(PURCHASE_PRICE)
---------     -------------------
01JAN2002     19859.46
01FEB2002     19188.96
01APR2002     44660.79
...
```

Этот пример очень простой, но вы можете с помощью табличных функций решать весьма изощренные задачи. При использовании данной возможности, приложение может совершенно ничего не знать о том, хранятся ли данные, с которыми оно работает, в реляционном или многомерном формате.

OLAP API

Интерфейс OLAP API — это набор классов языка Java, которые можно использовать для разработки OLAP-приложений. Он очень хорошо подходит для создания тонких клиентов, доступ к которым может осуществляться с помощью

web-браузера. OLAP API использует в запросах к данным многомерную модель. Тем не менее, он внутренне транслирует эти запросы на язык SQL. Чтобы осуществлять доступ к объектам с помощью этого интерфейса, вам нужно иметь объекты в каталоге OLAP, который описывается в следующем разделе. Основные данные могут храниться либо в реляционных таблицах, либо в аналитических рабочих областях, инкапсулированных в реляционных представлениях.

Подробное описание OLAP API не входит в задачи этой книги. Вместо этого мы проиллюстрируем использование BI Beans — компонентов, которые можно многократно применять и которые для доступа к данным используют OLAP API. Эти компоненты можно включать в пользовательские OLAP-приложения.

В следующем разделе мы рассмотрим каталог OLAP, который определяет метаданные, необходимые для работы таких средств, как OLAP API, BI Beans и Analytical Workspace Manager Application.

9.1.6. Каталог OLAP

Каталог OLAP хранит метаданные, которые определяют логическую модель данных. Если вашему приложению нужно осуществить доступ к данным с помощью OLAP API, они должны быть определены в каталоге OLAP.

Метаданные OLAP включают в себя сущности следующих типов:

- **Измерения.** Измерения (dimensions) используются для определения взаимосвязей, например иерархий данных. Измерения состоят из уровней, иерархий и атрибутов уровней. Кроме того, вы также можете указывать некоторые атрибуты измерения, например его описание.
- **Меры.** Мера (measure) — это количественный показатель, используемый в вычислениях, например стоимость покупки или цена товара.
- **Папки мер.** Папка мер (measure folder), которая также называется каталогом, — это удобное место для совместно хранения связанных между собой мер.
- Куб. Куб (cube) определяет, как меры будут агрегироваться по одному или более измерению. Если говорить терминами реляционной базы, куб определяет способ соединения таблиц фактов и измерений. Куб также показывает, какие иерархии в измерениях будут использоваться для вычисления агрегатов.

Каталог OLAP требует, чтобы вы связали эти элементы с реляционными таблицами и столбцами. Следовательно, если у вас есть данные, хранящиеся в аналитических рабочих областях, вы должны инкапсулировать доступ к этим данным в реляционных представлениях, как это описывалось выше.

Метаданные OLAP можно определять двумя способами:

- с помощью пакетов CWM или CWM2;
- с помощью Oracle Enterprise Manager.

Пакет CWM соответствует первой версии Common Warehouse Metadata (Общих Метаданных Хранилища), которая также называется CWMLite. Этот пакет поддерживает традиционные измерения, создаваемые по схеме звезды или снежинки. Измерения, которые определяются с помощью CWM API в каталоге OLAP, помимо метаданных OLAP имеют в своей основе объект-измерение SQL (который описывается в главе 4). Средства разработки хранилищ, такие, как Oracle Warehouse Builder, также позволяют автоматически сгенерировать метаданные, соответствующие спецификации CWM.

CWM2 — это вторая версия CWM, которая имеет дополнительные возможности, такие, как измерения типа родитель-потомок, разорванные (ragged) иерархии (когда не все данные иерархии имеют один и тот же самый низкий уровень), картирование одинаковых значений на разных уровнях в разных иерархиях и значения NULL в столбцах уровней.

На момент написания этой книги Oracle Enterprise Manager обеспечивает графическим интерфейсом только создание метаданных версии CWM. Если ваши данные хранятся в реляционных таблицах и соответствуют схеме звезды или снежинки, вам должно быть достаточно и этого.

Теперь мы проиллюстрируем использование Oracle Enterprise Manager для генерации метаданных OLAP. Интерфейсы CWM API очень просты в использовании, и общий ход событий для них аналогичен использованию мастеров Enterpriser Manager, поэтому мы не будем обсуждать их в этой книге.

Создание *метаданных OLAP в Oracle Enterprise Manager*

Чтобы найти функции OLAP в Oracle Enterprise Manager, раскройте узел Database (База данных), потом узел Warehouse (Хранилище), а потом узел OLAP. В Oracle Enterprise Manager можно использовать мастер создания измерения и куба. Кроме того, вы можете создавать папки мер.

Если у вас есть объекты-измерения SQL, созданные для использования при переписывании запросов (которое описывается в главе 4), вы автоматически увидите их в пункте Dimensions (Измерения). Если вы щелкнете по одному из них мышкой, вы можете увидеть окно, в котором говорится, что созданы еще не все метаданные, необходимые для OLAP. Сейчас вы можете продолжить работу мастера. Будут сгенерированы несколько вызовов CWM API, как это показано на рис. 9.6. Будьте уверены, это действие нисколько не повредит вашему существующему объекту-изме-

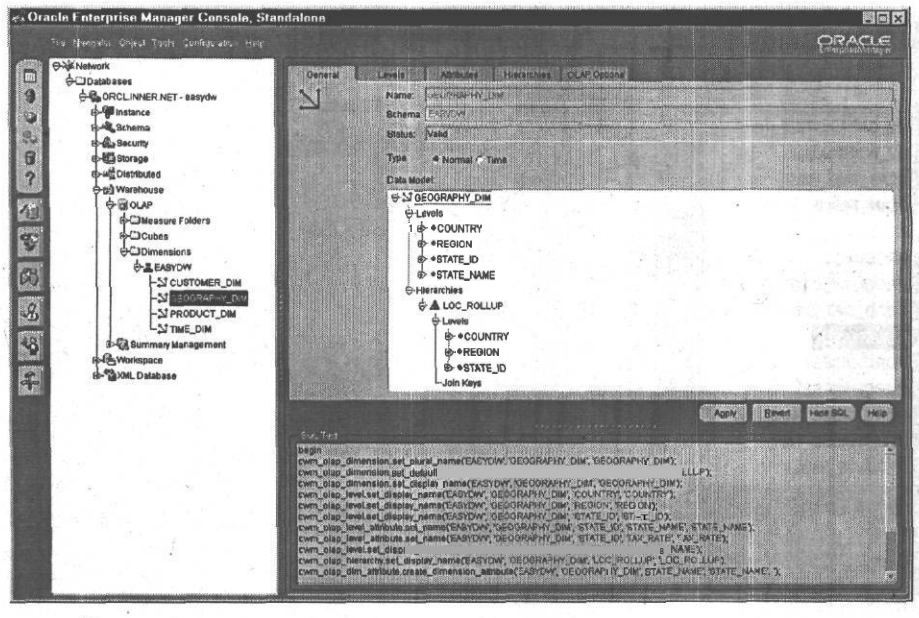


Рис. 9.6. *Измерения для OLAP в Oracle Enterprise Manager*

рению. Как только вы нажмете кнопку Apply (Применить), будут созданы метаданные CWM. Теперь вы можете использовать данное измерение при определении объекта-куба, как это описывается ниже.

Совет. В закладке OLAP Options (рис. 9.6) вы можете дать описательные названия измерениям, уровням и иерархиям. Мы рекомендуем вам ввести всю эту информацию, если вы планируете использовать мастер BI Beans Wizard.

Oracle Enterprise Manager позволяет определить метаданные куба которые связаны с вашими измерениями и мерами. Мы пройдем через некоторые этапы создания куба при помощи мастера Cube Wizard. Мастер можно найти в меню Object (Объект), показанном на рис. 9.6.

Сначала вам будет предложено ввести имя куба, и выбрать схему, в которую его нужно поместить (окно не показано). Затем вам будет предложено выбрать таблицу фактов, которая содержит меры, сохраняемые в кубе (рис. 9.7). В данном примере мы выбрали таблицу EASYDW.PURCHASES.

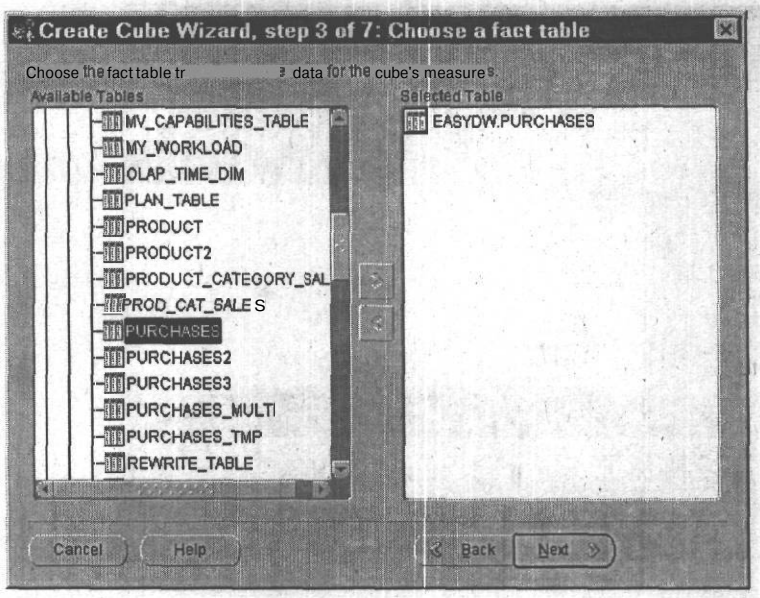


Рис. **9.7.** *Cube Wizard— выбор таблицы фактов*

После этого вам нужно будет указать измерения, которые вы хотите включить в куб (рис. 9.8). В этом примере у нас есть два измерения — CUSTOMER_DIM и OLAP_TIME_DIM, которые, как предполагается, были созданы ранее.

В следующем окне вам будет предложено определить, каким образом измерения соединяются с таблицей фактов. Вам нужно определить это отношение для каждого измерения куба, перед тем как можно будет двигаться дальше. Также вы должны определить иерархию по умолчанию, по которой куб будет вычислять меры. На рис. 9.9 мы указали, что уровень CUSTOMER в измерении CUSTOMER_DIM соединяется с таблицей фактов с помощью столбца customer_id.

Следующий шаг, который показан на рис. 9.10 — это определение мер, которые будут храниться в кубе. Вы должны дать каждой мере имя и показать, какому столбцу таблицы фактов она соответствует. По умолчанию в список мер включаются все столбцы таблицы фактов, имеющие числовой тип данных. Заметьте, что по умолчанию для каждой меры принят метод агрегирования SUM. И, наконец, вам будут показаны итоговые данные по кубу (рис. 9.11). Вы можете нажать копку Show SQL (Показать код SQL) и вы увидите интерфейсы CWM API, которые будут использованы для генерации куба.

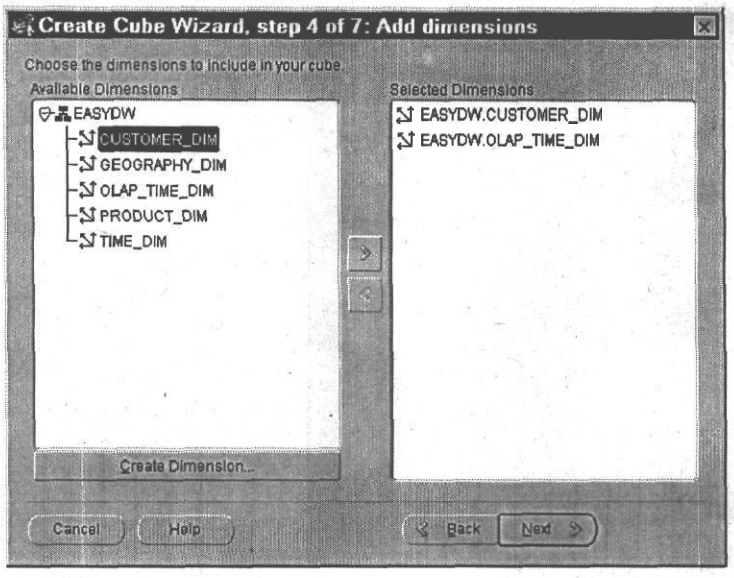


Рис. 9.8. *Cube Wizard — выбор измерений*

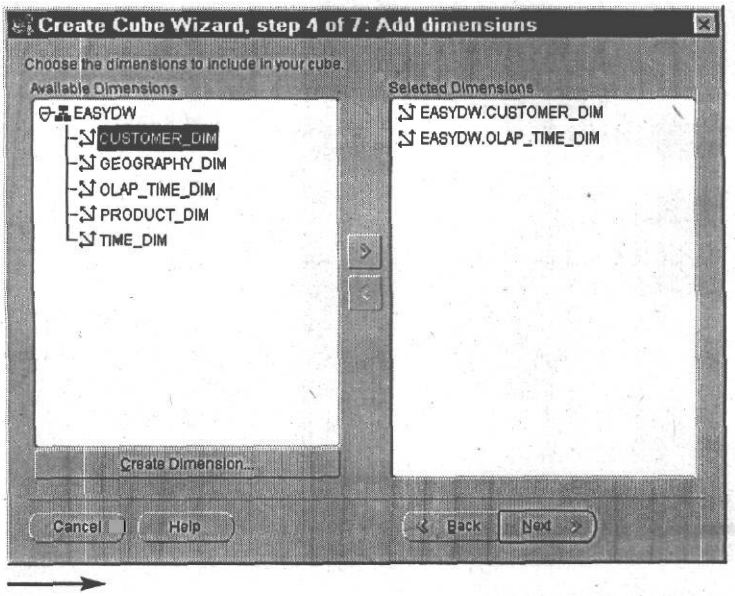


Рис. 9.9. *Cube Wizard — Соединение измерений и таблиц фактов*

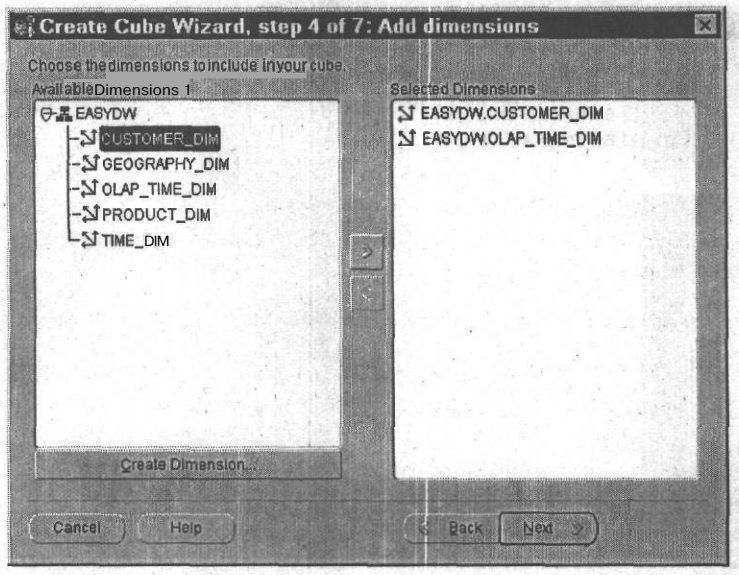


Рис. 9.10. *Cube Wizard — указание мер*

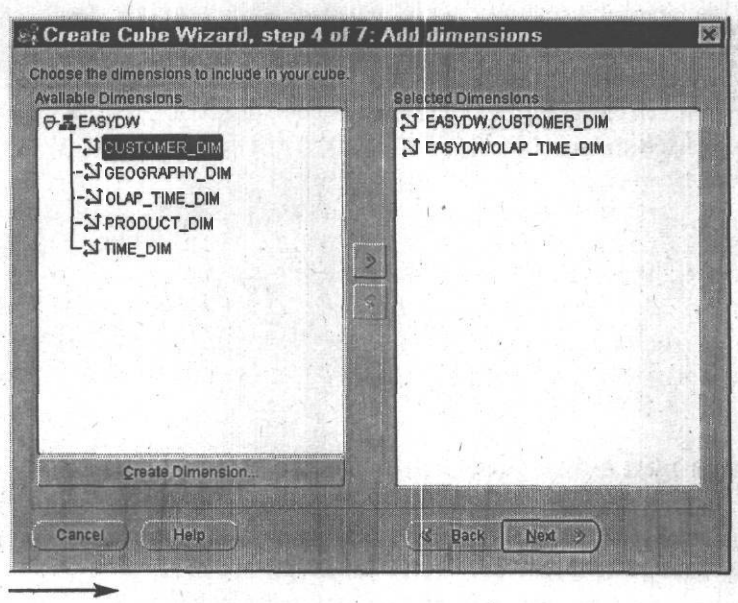


Рис. 9.11. *Cube Wizard — окно с итоговой информацией*

Когда вы нажмете кнопку Finish (Готово) в окне на рис. 9.11, куб будет создан. Если вы решили вызвать OLAP Summary Advisor, отметив его в нижней части окна, показанного на рис. 9.11, он будет запущен. OLAP Summary Advisor особым образом использует Summary Advisor, о котором мы рассказывали в главе 4, и генерирует материализованные представления для оптимизации куба. Однако с этими материализованными представлениями ассоциируются дополнительные метаданные, так что их нельзя путать с материализованными представлениями, используемыми для других целей. В своем приложении вам следует использовать соответствующую версию Summary Advisor.

Совет. До OLAP Summary Advisor можно добраться, щелкнув правой кнопкой мыши по объекту-кубу в Oracle Enterprise Manager. До Summary Advisor, описанного в главе 4, можно добраться, развернув узел Summary Management (Управление сводными данными).

Когда куб создан, вы можете редактировать любые его свойства, щелкнув по кубу в разделе OLAP Management правой кнопкой мыши. Также вы можете просматривать данные в кубе с помощью приложения Cube Viewer в Oracle Enterprise Manager. Если вы выберете куб, который желаете посмотреть, и щелкнете по нему правой кнопкой, то во всплывающем меню вы найдете пункт Cube Viewer. Приложение Cube Viewer отображает результаты запроса в формате таблиц с перекрестными ссылками, как это показано на Рисунке 9.12. В данном примере мы выводим меру purchase_price по всем продуктам в округе Hants. Можно спускаться вниз по уровням данных, щелкая по клеточкам в заголовочных ячейках.

Отчет, результаты которого нужно отобразить, можно создать или изменить с помощью Query Builder в меню File (Файл). Этот Query Builder очень похож на тот, который используется в BI Beans, и который мы обсудим позже. Однако, в отличие от BI Beans, где вы можете различными способами настраивать отчет, Cube Viewer — это простое приложение для внимательного изучения данных и удобный способ удостовериться в том, что метаданные OLAP были определены верно.

9.1.7. Средства создания аналитических рабочих областей

Весьма вероятно, что многие пользователи столкнутся с ситуацией, когда реляционное хранилище уже есть и нужно создать аналитическую рабочую область

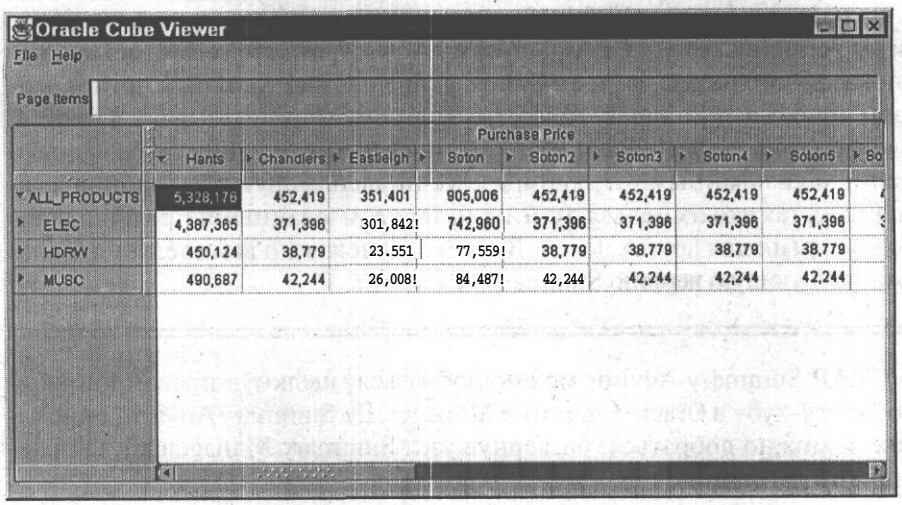


Рис. 9.12. *Cube Viewer*

для некоторых специальных свойств, таких, как прогнозирование и анализ возможных вариантов. Oracle предлагает несколько простых инструментов для создания аналитической рабочей области из схемы звезды или снежинки. Эти инструменты обеспечивают создание аналитических рабочих областей таким образом, чтобы его могли использовать все аналитические инструменты анализа от Oracle, такие как OLAP API и BI Beans.

В данном разделе мы коротко опишем два средства, предназначенные для этой цели:

- CWM2_OLAP_AW_CREATE, пакет PL/SQL;
- приложение Analytic Workspace Manager.

Чтобы можно было создать аналитическую рабочую область с помощью любого из этих инструментов, ваши данные должны храниться в схеме звезды или снежинки. Кроме того, в каталоге OLAP должны быть определены метаданные CUBE и DIMENSION.

Пакет CWM2_OLAP_AW_CREATE позволяет:

- определять измерения в аналитической рабочей области, соответствующие измерению в метаданных OLAP, с помощью процедуры AW_DIMENSION_CREATE;
- определять переменные, соответствующие кубу в метаданных OLAP с помощью процедуры AW_CUBE_CREATE;

- загружать данные в измерение и переменные в аналитической рабочей области из таблиц фактов и измерений с помощью разнообразных процедур загрузки;
- создавать реляционные представления поверх аналитической рабочей области с помощью процедур AW_DIMENSION_CREATE_ACCESS и AW_CUBE_CREATE_ACCESS. Это позволяет осуществлять доступ к аналитической рабочей области с помощью SQL.

Кроме того, Oracle предлагает приложение под названием Analytic Workspace Manager, которое позволяет делать то же самое с помощью графического интерфейса. Далее мы продемонстрируем некоторые из его возможностей.

Совет. Приложение Analytic Workspace Manager можно бесплатно получить с сайта Oracle Technology Network (http://otn.oracle.com). Оно не входит в состав инсталляционного компакт диска базы данных Oracle.

Приложение Analytic Workspace Manager позволяет создавать и редактировать измерения и переменные в аналитической рабочей области. В примере на рис. 9.13 мы редактируем переменную с именем COST.

Кроме того, Analytic Workspace Manager позволяет просматривать кубы и измерения в каталоге OLAP. В него также входит мастер для создания аналитической рабочей области из куба, определенного в каталоге OLAP (Analytic Workspace Creation Wizard). На рис. 9.14 показано первое окно мастера, в котором вы можете выбрать нужный вам куб. В этом примере мы выбрали куб PURCHASES, предполагая, что он был определен ранее.

Затем вам будет предложено указать имя аналитической рабочей области и выбрать табличную область, в которой она будет храниться, как это показано на рис. 9.15.

Наконец, если вы хотите использовать данную аналитическую рабочую область в OLAP API и BI Beans, вы можете включить эту опцию, как показано на рис. 9.16. При этом будут сгенерированы различные скрипты для создания реляционных представлений поверх элементов аналитической рабочей области. Вы можете указать директорию, в которой скрипты будут храниться, и запустить их, когда вам будет удобно.

Мы настойчиво рекомендуем использовать для создания аналитических рабочих областей эти средства, а не создавать их вручную, что может оказаться весьма затруднительно.

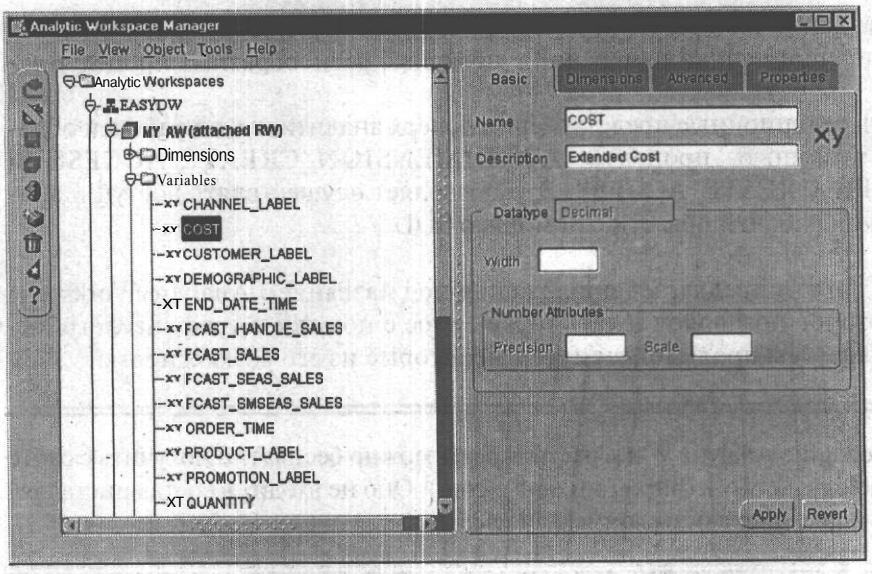


Рис. 9.13. *Analytic Workspace Manager*

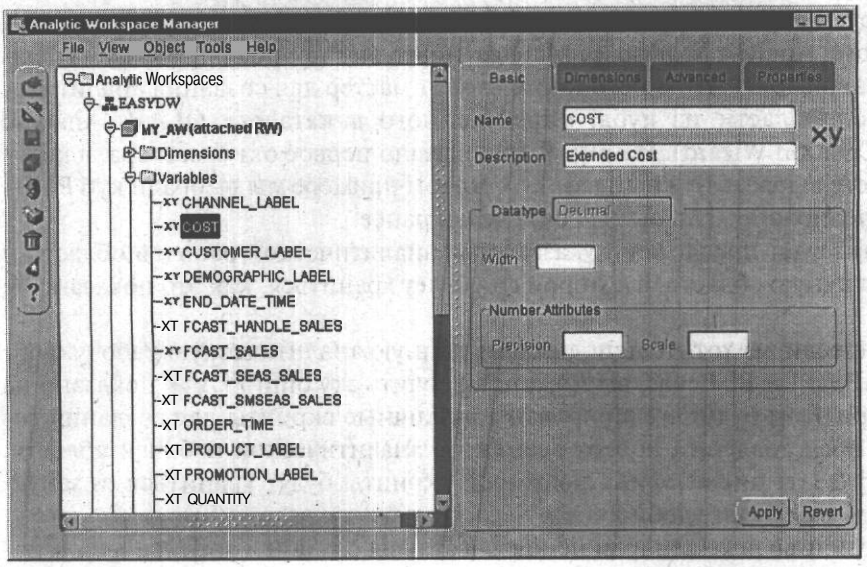


Рис. 9.14. *Мастер создания аналитической рабочей области — Выбор куба*

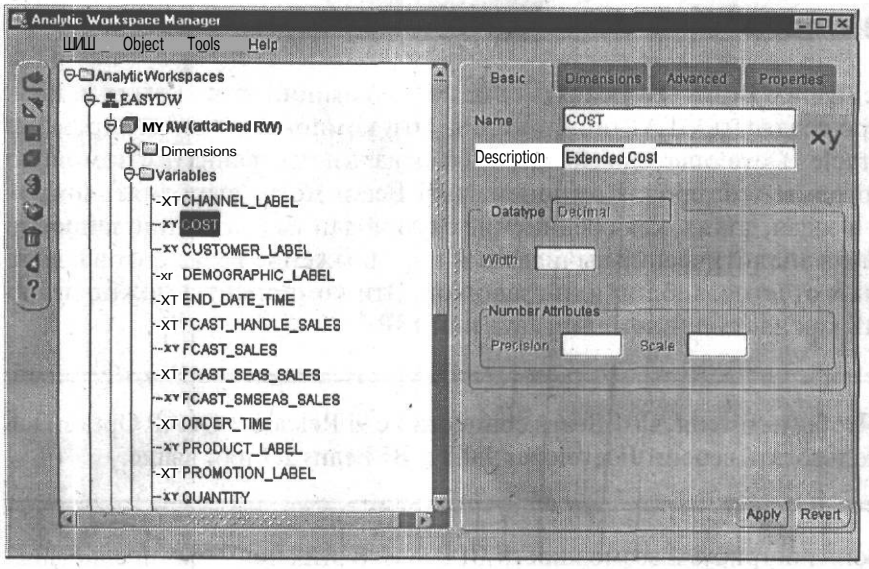


Рис. 9.15. *Мастер создания аналитической рабочей области — выбор аналитической рабочей области.*

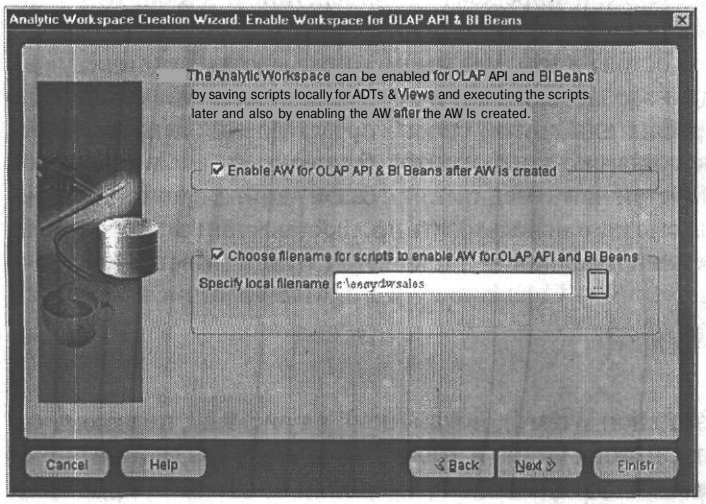


Рис. 9.16. *Мастер создания аналитической рабочей области — Использовать рабочую область в OLAP API и BI Beans*

9.1.8. BI Beans

Business Intelligence Beans (BI Beans) — это набор компонентов, который позволяет быстро создавать OLAP-приложения. Эти компоненты интегрированы в продукт Oracle JDeveloper, где их можно создавать и настраивать с помощью простых программ-мастеров. Компоненты BI Beans могут выполнять широкий диапазон задач, таких, как соединение с базой данных, создание запросов для проведения аналитических вычислений и отображение результатов в виде очень мощных отчетов, таблиц или графиков. Эти компоненты можно легко использовать как часть приложения Java или JSP.

Замечание. Чтобы применять BI Beans совместно с 9i Release 2 OLAP Option, вы должны использовать версии JDeveloper 9.0.3 и BI Beans 2.7 или выше.

Далее мы проиллюстрируем возможности BI Beans. В этих примерах предполагается, что был определен куб с именем PURCHASE_DATA, имеющий два измерения — PRODUCT и CUSTOMERS и две меры — purchase_price и shipping_charge, которые объединены в папке мер с именем Purchase Measures. Описательные атрибуты, которые указываются при определении измерений и кубов, будут использоваться при отображении мер и измерений в мастерах.

Новый аспект BI Beans состоит в том, что в момент разработки компонента вы имеете полный доступ к данным и, таким образом, сразу можете увидеть, как будет выглядеть итоговый график или другой способ представления данных. Перед тем как вы приступите к созданию компонентов бизнес-информации (business intelligence, BI), вы должны установить соединение с базой данных с помощью компонента Designer. Затем вы сможете использовать при составлении запросов Query Builder, при создании отчетов и графиков — Presentation Wizard (Мастер презентаций), а при создании вычислений — Calculation Wizard (Мастер вычислений).

Создание отчетов

Мастер презентаций (Presentation Wizard), показанный на рис. 9.17, позволяет создавать отчеты в форматах Crosstab, Graph и Table. Сначала мы создадим отчет об итоговых значениях продаж в округе Hants по различным категориям продуктов. Пока мы будем настраивать этот отчет, вы получите представление о возможностях BI Beans. В примере на рис. 9.17 мы решили создать отчет в формате Crosstab (перекрестная таблица) с именем EasyDWSalesAnalysis.

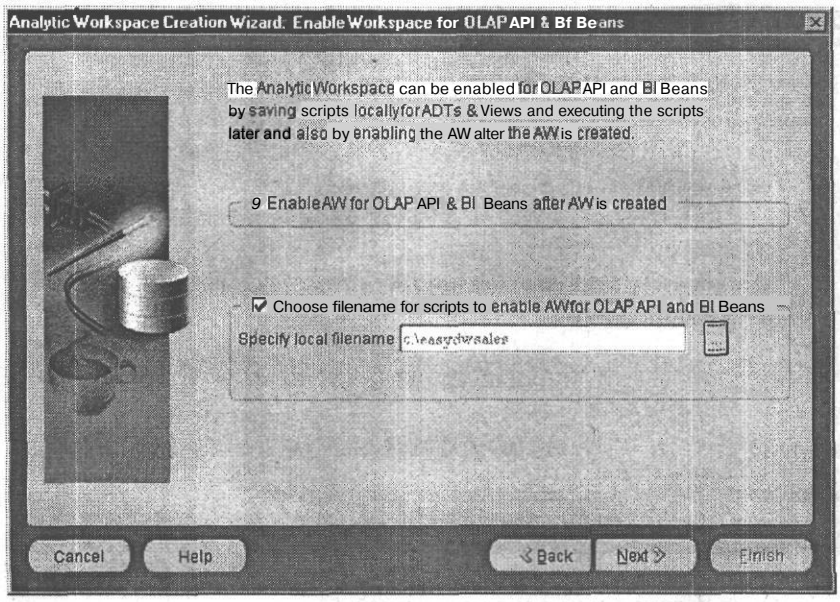


Рис. 9.17. *Мастер создания новой презентации (New Presentation Wizard)*

На следующем этапе, показанном на рис. 9.18, нам нужно выбрать, какие меры мы хотим иметь в отчете. Переключатель Automatically add/remove dimensions включен и, следовательно, измерения, связанные с мерой, будут автоматически включены в отчет. Заметим, что некоторые из окон Presentation wizard также используются в Query Builder Wizard, поскольку в конечном итоге презентация — это просто визуальное представление запроса.

Следующий шаг — определить, как эти измерения будут расположены в отчете. В формате crosstab вы можете располагать данные в двух измерениях — в столбцах и строках и, при желании, в третьем, страничном измерении. Мастер выберет для измерений расположение по умолчанию. Вы можете изменять расположение простым перетаскиванием, перемещая измерения в желаемую строку, столбец или страницу. В примере на рис. 9.19 у нас есть только два измерения, и поэтому мы будем располагать покупателей (customers) в столбцах, а продукты (product) в строках, а страничное измерение использовать не будем.

Затем мы выбираем, какие уровни или значения измерения включать в отчет (рис. 9.20). В нашем примере мы выбрали только округ Hants. Заметьте, что даже в среде разработки мастер показывает реальные значения измерения customers.

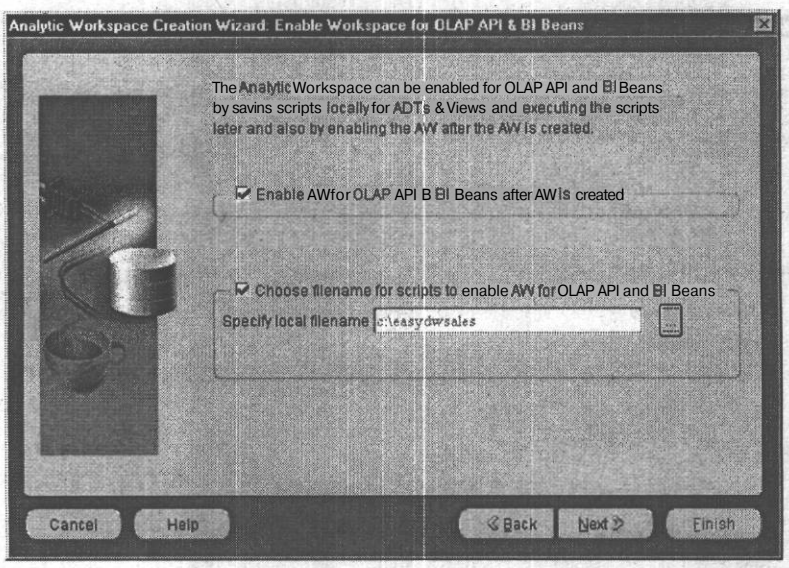


Рис. 9.18. *New Presentation Wizard— выбор мер и измерений*

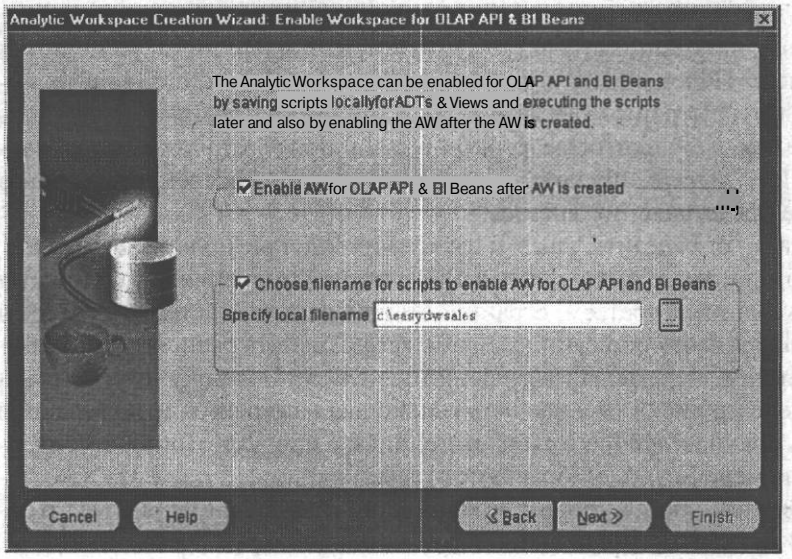


Рис. 9.19. *Расположение перекрестной таблицы*

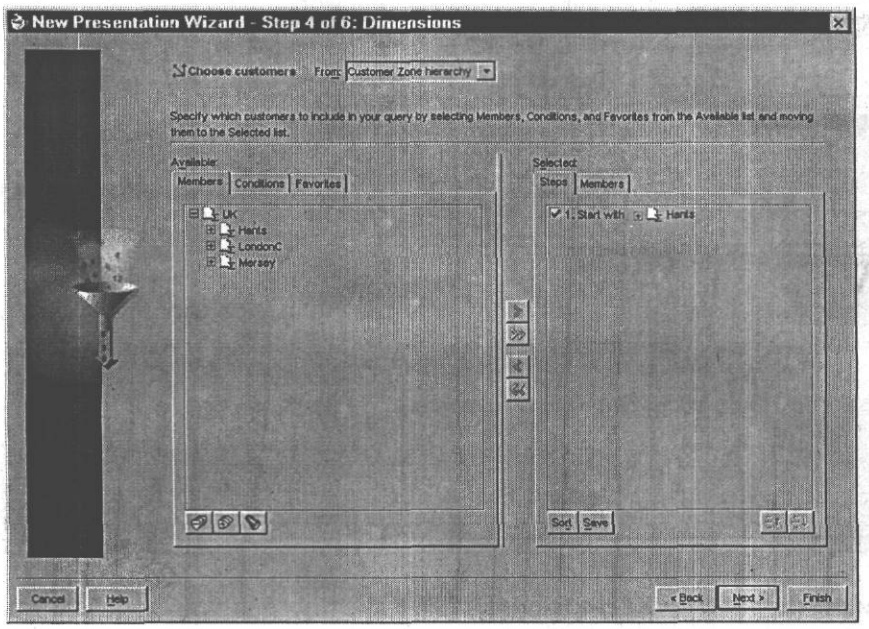


Рис. 9.20. *Выбор членов измерения*

В правой части рис. 9.20 имеется панель Selected (Выбранное) с двумя закладками — Steps (Шаги) и Members (Члены). В закладке Members показаны значения, выбранные в конкретном измерении. В закладке Steps показан способ определения списка членов. В данном примере у нас есть только один член — округ Hants. Позже мы обсудим, как можно добавлять члены в соответствии с определенными условиями.

Мы могли бы продолжить и настроить измерение PRODUCT, но давайте просто нажмем кнопку Finish (Готово), а настройку выполним позже. Получившийся отчет будет выглядеть так, как показано на рис. 9.21. Вы можете захотеть узнать, что же особенного в этом отчете. Вы заметите, что в заголовочных ячейках ALL_PRODUCTS, ELEC, HDRW и MUSC есть небольшие стрелки. Если мы щелкнем мышкой по ячейке ELECT и развернем ее, компонент автоматически подаст запрос к данным и перейдет к уровню отдельных продуктов. Фактически, мы раскрыли ячейку ALL_PRODUCTS, и перешли к отдельным категориям. Если вы включите такой отчет в JSP-приложение, пользователи получат интерактивный отчет, в котором они смогут переходить вверх и вниз по уровням иерархии. В измерении customers мы выбрали в качестве верхнего уровня

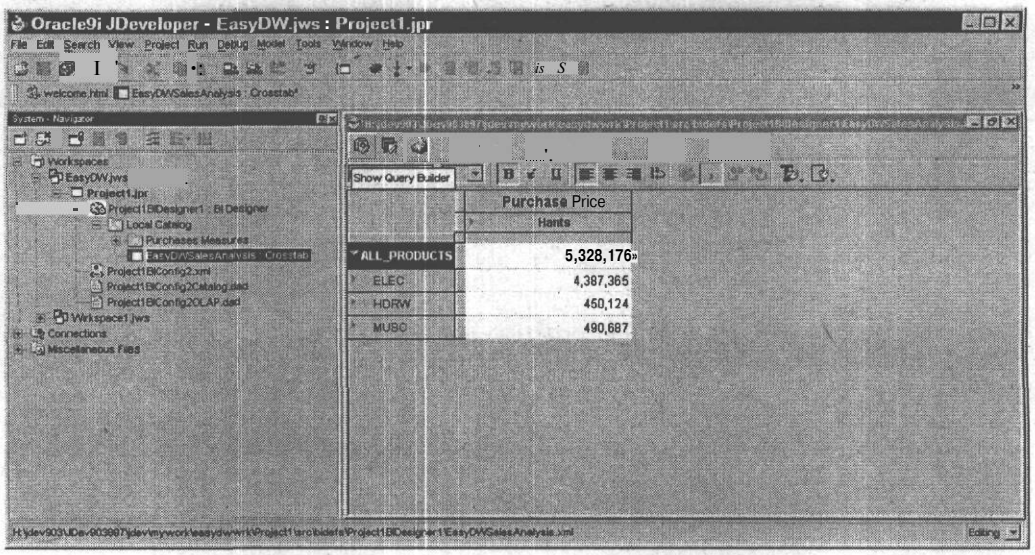


Рис. 9.21. *Аналитический отчет по продажам EASYDW*

округ Hants, и это означает, что мы можем переходить к городам и конкретным покупателям в этом округе.

Теперь давайте посмотрим, как можно редактировать и настраивать такой отчет. На рис. 9.21 прямо над окном с отчетом есть панель инструментов. Если мы нажмем на первую кнопку в этой панели (с контекстной подсказкой Show Query Builder — Показать Query Builder), мы вызовем Мастер запросов, показанный на рис. 9.22, в котором мы можем редактировать любые аспекты отчета.

Ранее мы упоминали закладки Steps (Шаги) и Members (Члены). Теперь давайте обсудим закладку Conditions (Условия). Каждое измерение в отчете можно настраивать, включая в него члены в соответствии с определенным условием. Мастер автоматически предложит вам некоторые полезные условия, подходящие для данных. В примере на рис. 9.22 мы выбрали «Top 10 based on Purchase Price» («Первые 10 по стоимости покупки»). Шаги (steps) показывают, что мы начинаем с ALL_PRODUCTS, а затем добавляем первые десять продуктов. Таким образом, отчет будет состоять из данных в ALL_PRODUCTS (в которых можно двигаться вниз по иерархии), а также первых десяти продуктов, показанных на рис. 9.23.

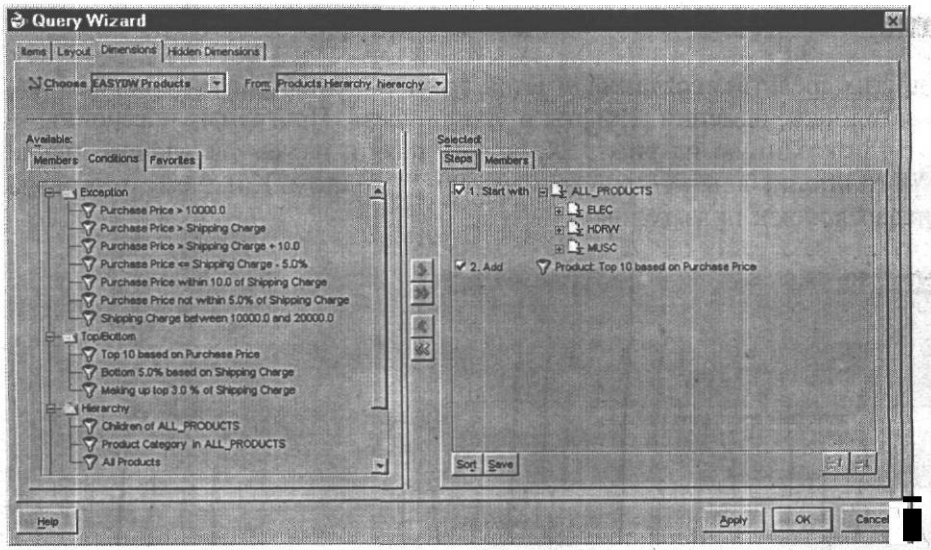


Рис. 9.22. *Настройка содержимого отчета Мастером запросов (Query Wizard)— закладка Conditions (Условия)*

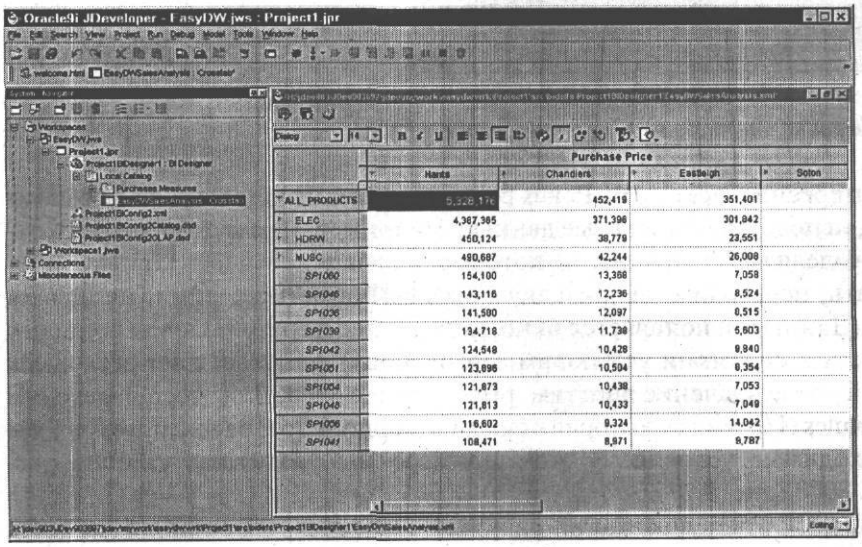


Рис. 9.23. *Аналитический отчет EASYDWc первыми десятью продуктами*

Форматирование, управляемое данными

Как и в других средствах создания отчетов, вы можете настроить данный отчет, добавив заголовки, изменив шрифты и указав цвета. Программу настройки — Customizer, показанную на рис. 9.24, можно найти, нажав следующую после Query Wizard кнопку в панели инструментов над отчетом. К нашему отчету мы добавили заголовок и подзаголовок.

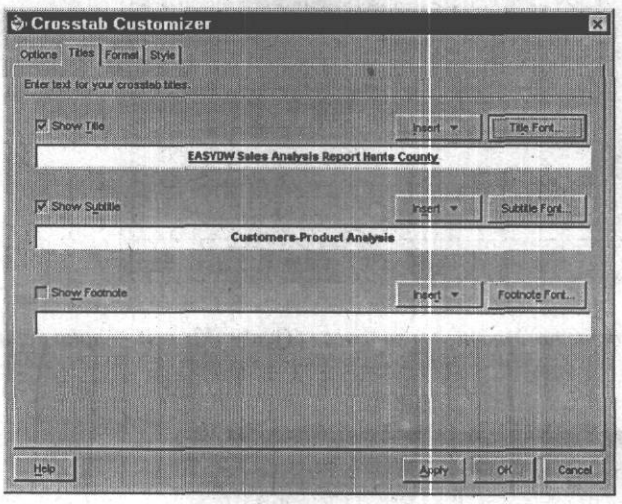


Рис. 9.24. *Настройка внешнего вида перекрестной таблицы*

Еще один интересный аспект BI Beans состоит в том, что вы можете форматировать отчет, используя конкретные данные. Например, нам нужно, чтобы в отчете цветом выделялись те продукты, которые продаются не очень хорошо. Чтобы это сделать, переходим на закладку Format (Форматирование), нажимаем кнопку Data (Данные) и появляется окно, показанное на рис. 9.25. Мы создадим формат с именем «Losers» и установим серый (Gray) фоновый цвет всех ячеек отчета, где итоговое значение purchase_price меньше 8000. Для этого переходим на закладку Rules (Правила), которая показана на рис. 9.26 и редактируем ее, добавляя необходимые условия. В этом примере мы указываем условие purchase_price < 8000 (рис. 9.27).

Получившийся отчет показан на рис. 9.28. Надеемся, вы оценили легкость создания очень красивых и мощных отчетов. Хотя мы и показываем отчеты в JDeveloper, вы видите их в точности так, как они будут выглядеть в приложении.

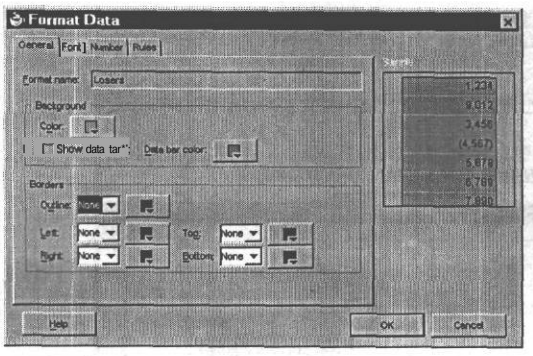


Рис. 9.25. *Форматирование, управляемое данными*

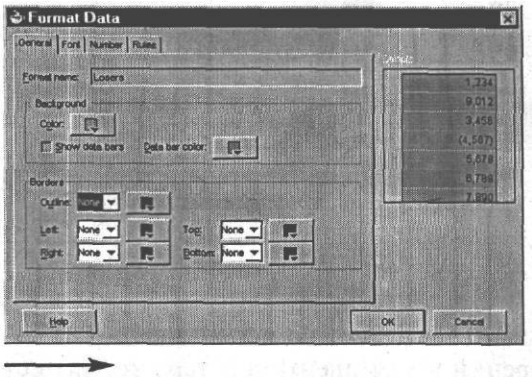


Рис. 9.26. *Определение правил для форматирования, управляемого данными*

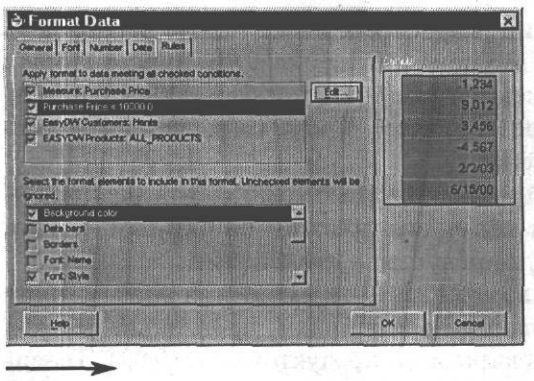


Рис. 9.27. *Указание условия по значению* ***purchase_price***

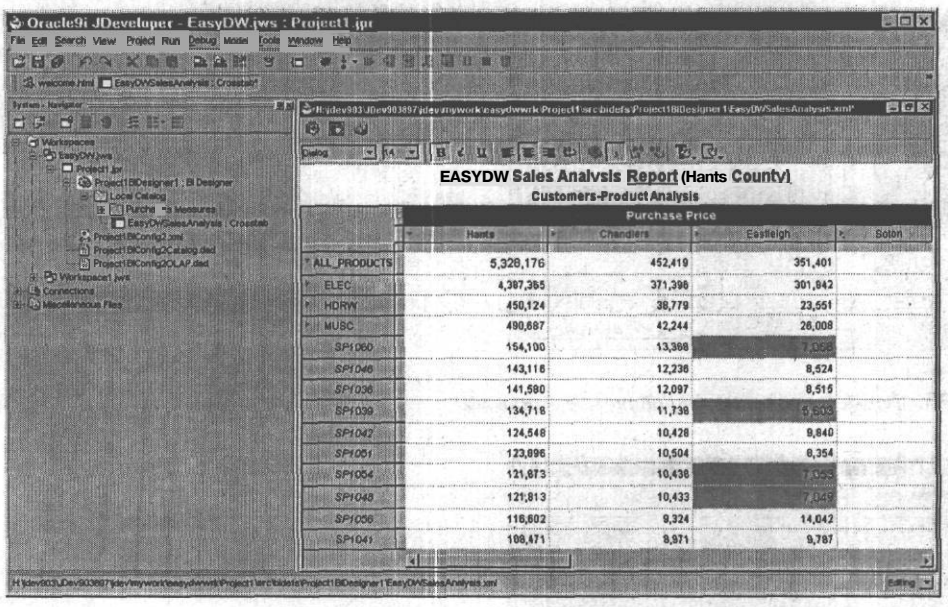


Рис. 9.28. *Аналитический отчет EASYDW с выделением плохо продаваемых продуктов*

Графики

Мастера презентаций (Presentation Wizard) также можно использовать для создания графиков. Этапы выбора измерений и мер идентичны тем, что мы использовали для перекрестной таблицы. На рис. 9.29 показаны различные типы графиков.

Как и в случае перекрестных таблиц, в графиках также можно располагать данные в нескольких измерениях, как это показано на рис. 9.30. Вместо строк и столбцов в гистограммах (bar graph) имеются столбики, показывающие одно измерение, и группы, отражающие второе измерение. Кроме того, третьими измерением может быть страничное измерение.

На рис. 9.31 показан график, созданный с помощью компонента для презентаций (presentation bean), на котором мы видим продажи по округам. В этом примере мы сделали округа группами, а продукты — столбиками.

Отметьте, что в этом графике есть возможность перемещения вниз по уровням иерархии. Если вы щелкните мышкой по столбикам, произойдет переход на более низкий уровень, т.е. уровень категорий продуктов (рис. 9.32). Теперь вы сможете увидеть наиболее популярные категории продуктов в каждом округе.

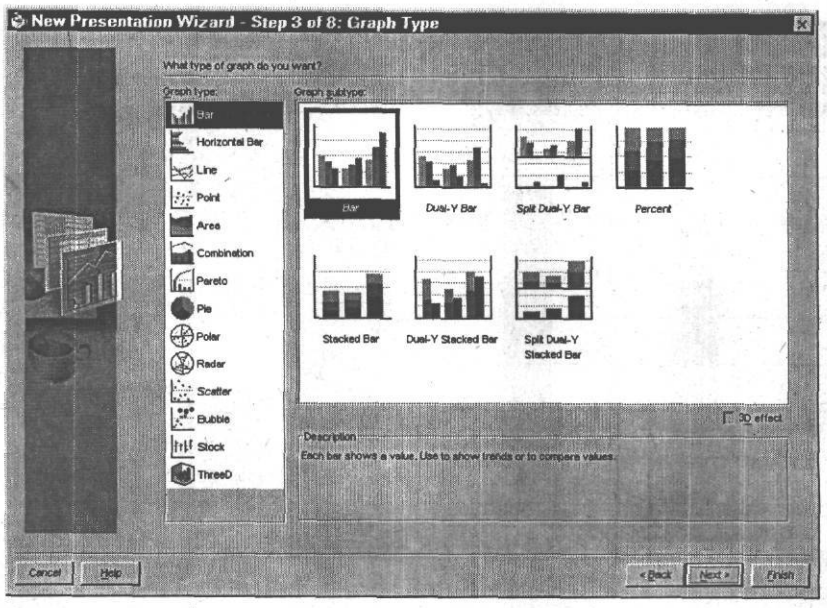


Рис. 9.29. *Графики*

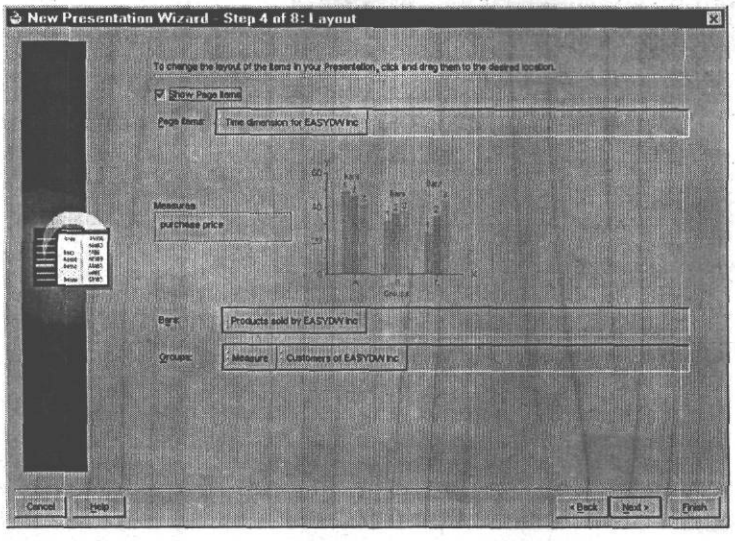


Рис. 9.30. *Определение внешнего вида графика*

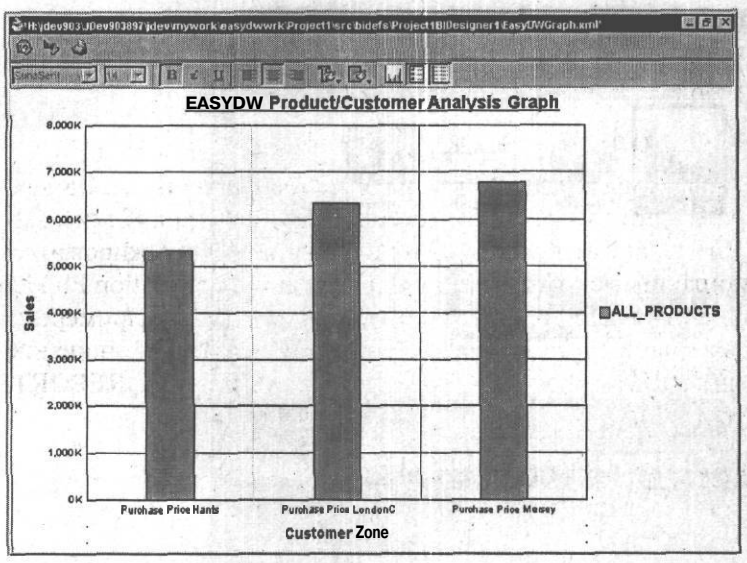


Рис. 9.31. *Диаграмма продаж продуктов по округам*

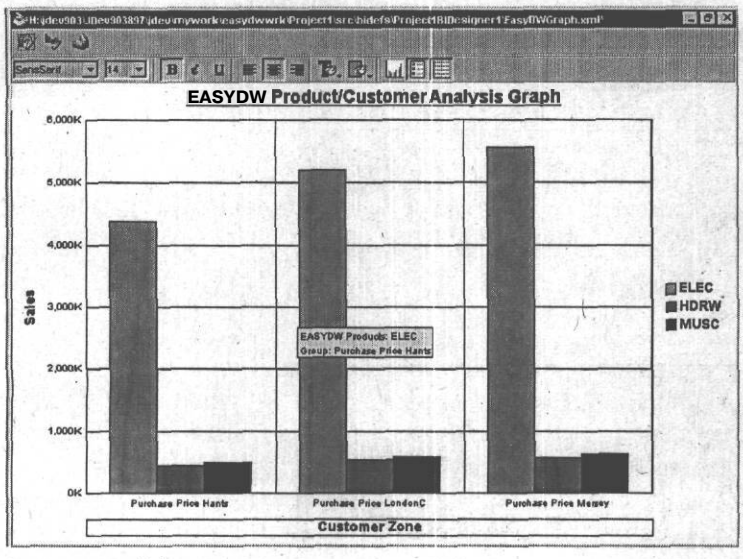


Рис. 9.32. *Перемещение по уровням иерархии в графике*

График можно настраивать по данным. Здесь, например, мы выбрали свой цвет для каждой категории.

Вычисления

В приводимых пока примерах мы создавали отчеты по основным данным. Однако большинству OLAP-приложений требуется производить вычисления. К этим вычислениям могут относиться аналитические функции, например ранжирование, отчетные агрегаты, накопленные суммы и скользящие средние. Calculation Builder позволяет создавать такие вычисления, используя меры вашего куба. В примере на рис. 9.33 мы создаем разделяемое вычисление, которое рассчитывает отношение количественного значения к итоговому (вызов функции RATIO_TO_REPORT, которая обсуждалась в главе 3).

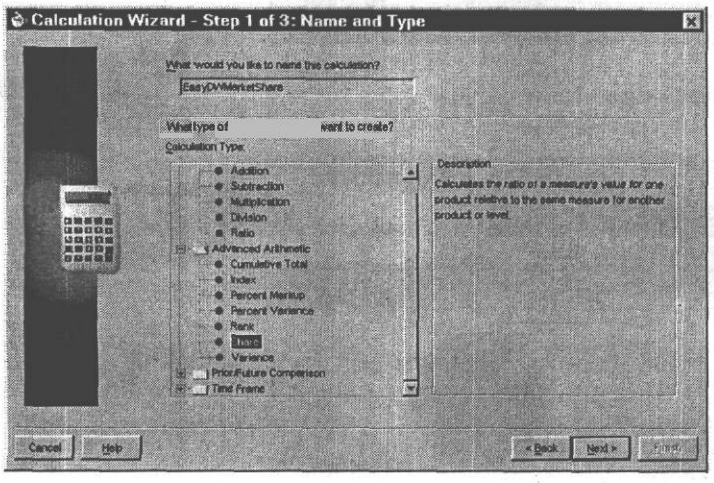


Рис. 9.33. *Создание вычисления*

После нажатия кнопки Next (Далее) появляется окно, показанное на рис. 9.3,4, в котором вы можете указать параметры вычисления. В данном примере мы указали меру purchase_price и определили, что отношение к итоговому значению будет вычисляться для каждого уровня иерархии продуктов. Так мы сможем найти для каждой категории продуктов ее долю на рынке.

Затем эти вычисления можно будет включать в представления данных т. е. в графики и отчеты. Например, на рис. 9.35 мы изобразили секторную диаграмму продаж для каждой категории и каждого округа.

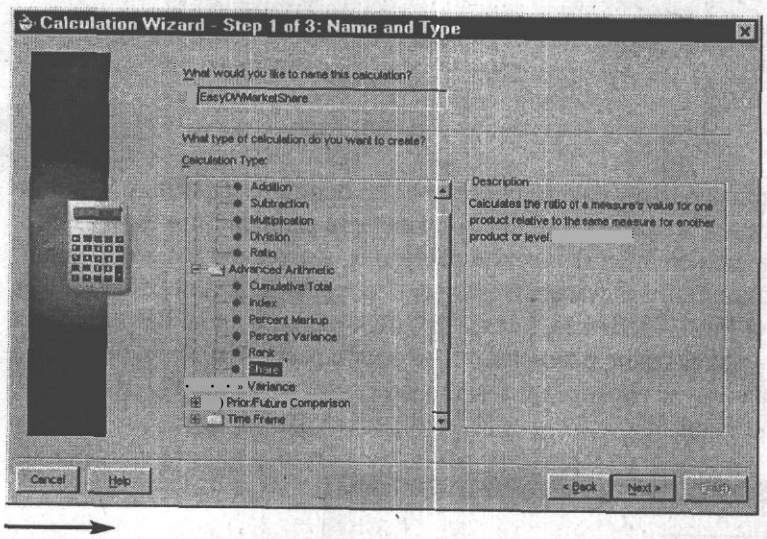


Рис. 9.34. *Указание параметров вычисления*

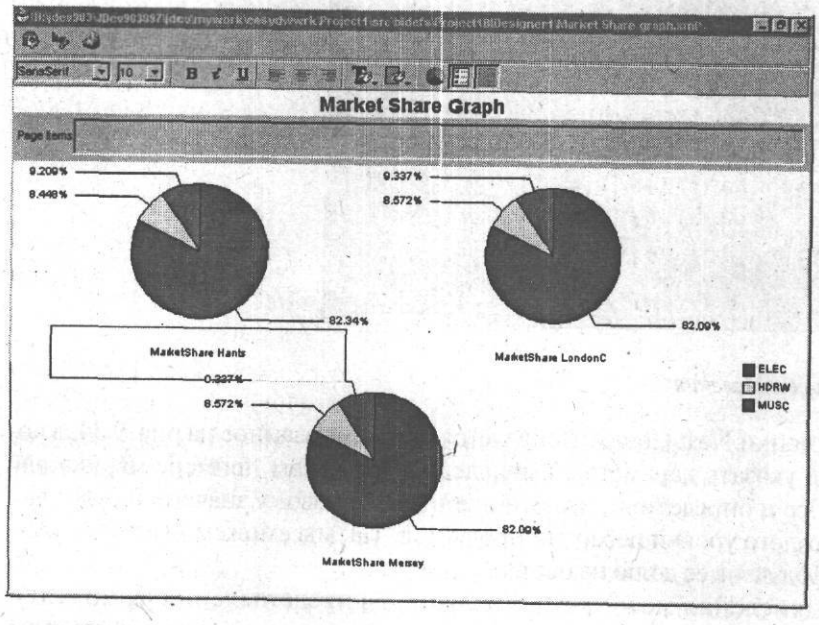


Рис. 9.35. *Графическое представление результатов вычисления*

9.1.9. Заключение

В данном разделе мы дали вам общий обзор компонентов и возможностей Oracle 9i OLAP. Независимо от того, храните вы свои данные в реляционном или многомерном формате, база данных Oracle предоставляет вам все инструменты и методы, необходимые для продвинутых приложений для бизнес-анализа.

9.2. Oracle Data Mining (Интеллектуальная обработка данных)

Интеллектуальная обработка данных — это процесс поиска скрытых закономерностей и взаимоотношений в данных для облегчения принятия решений в бизнесе.

Когда вы ведете свой бизнес, вы можете сталкиваться с различными формами данных. Операционные данные сообщают вам повседневную информацию о ваших клиентах, например: «Что купил Джон Доу 5 августа 2002 года?». В хранилище данные организуются в соответствии с заранее установленными связями с помощью таблиц фактов и измерений. Вы можете использовать средства OLAP для поиска в вашем бизнесе различных фактов. Сколько людей купили электронную технику в вашем магазине и сколько в среднем они потратили? Сколько покупателей электронных товаров были инженерами? Все это — вопросы, основанные на имеющихся данных.

Однако вам могут понадобиться ответы на такие вопросы, как:

- Какие из ваших клиентов, скорее всего, купят персональный видеомагнитофон?
- В каком городе лучше всего начать продажи нового продукта?
- Какое стимулирование спроса лучше всего проводить в Рождество?

Здесь вам нужен определенный прогнозирующий взгляд на данные. Интеллектуальная обработка (data mining) использует различные статистические методы, а также методы обучения машин, и выявляет тенденции и шаблоны поведения. Иногда вы можете знать, что ищете, в других случаях у вас может не быть четкого определения того, что процесс обработки должен выявить. Вы говорите: «Вот мои данные, найди мне в них что-нибудь интересное». Эта проблема как раз для интеллектуальной обработки — найти иголку в стоге сена, только вы можете еще и не знать, как выглядит иголка! Одним из самых больших преимуществ хранилища является то, что данные очищены, объединены и хорошо подходят для интеллектуальной обработки.

Интеллектуальную обработку можно использовать для многих целей, например для улучшения расположения магазинов, обнаружения мошенников, стимулирования заказов по почте, уменьшения текучести клиентуры и настройки Web-сайта под пользователя. Также ее можно использовать в науках о жизни, например для поиска тех, кто наиболее подвержен определенным заболеваниям.

Опция Oracle 9i Data Mining

Oracle Data Mining — это опциональный компонент Oracle 9i Enterprise Edition, который встраивает в сервер функцию интеллектуальной обработки данных. При интеллектуальной обработке обрабатываются большие объемы данных. Часто оказывается необходима их предварительная обработка и перевод в формат, наиболее подходящий для соответствующих алгоритмов. Включаясь в базу данных, алгоритмы интеллектуальной обработки могут пользоваться преимуществами масштабируемости базы Oracle и таких свойств, как секционирование, сжатие и параллельное выполнение для повышения своей производительности.

Oracle Data Mining — это Java API, который поддерживает правила ассоциаций, классификации и кластеризации. Компания Oracle является главным инициатором создания спецификации интеллектуальной обработки под названием Java Data Mining Specification (JSR-73). В будущих версиях базы Oracle интеллектуальная обработка будет соответствовать этому стандарту, когда он достигнет статуса эталона.

Oracle Data Mining не следует путать с продуктом Darwin, также известным как Oracle Data Mining Suite. Этот продукт выполнял интеллектуальную обработку вне базы данных и сейчас не поддерживается. Darwin имел графический интерфейс с системой «покажи и щелкни» (point-and-click), но было чрезвычайно сложно, если вообще возможно, интегрировать с его помощью интеллектуальную обработку в приложение. С другой стороны, Oracle Data Mining позволяет встраивать интеллектуальную обработку прямо в логику работы приложения, а не извлекать из приложения данные для анализа.

В отличие от прочего материала этой книги, алгоритмы интеллектуальной обработки требуют определенной математической подготовки, в частности, по статистике и теории вероятностей. В данном разделе мы дадим общий обзор интеллектуальной обработки данных, не вдаваясь в технические подробности алгоритмов. Мы опишем задачи и кратко проиллюстрируем применение Oracle Data Mining Java API. Чтобы читать примеры, приведенные в конце этой главы, вы должны иметь базовое представление о языке программирования Java. Однако, если вам нужен только общий обзор, вы можете спокойно пропустить эти примеры.

9.2.1. Методики Oracle Data Mining

Проблемы, решаемые с помощью интеллектуальной обработки данных, можно разделить на две категории. В некоторых ситуациях у вас имеется определенное представление о том, что вы ищете. Например, вас интересуют клиенты, которые могут приобрести цифровую камеру. Это называется направляемое или контролируемое обучение. В других случаях вы даете возможность процессу обработки найти что-нибудь интересное, например высокий процент аварий на определенном перекрестке. Это называется ненаправляемым или неконтролируемым обучением.

Oracle Data Mining поддерживает разнообразные методы анализа данных, которые можно разделить на следующие категории: ,

- ассоциация
- кластеризация
- классификация

Мы рассмотрим их чуть подробнее.

Правила ассоциаций

Вы, вероятно, знакомы с такой возможностью Amazon.com: «Здравствуйте, Джейн! У нас есть для вас несколько новых рекомендаций». При такой настройке web-сайта на пользователя используется метод интеллектуальной обработки данных, называющийся «правила ассоциаций» (association rules). Обработка с правилами ассоциаций ищет те элементы, которые часто появляются вместе. Например, может быть обнаружено, что большая часть клиентов, купивших книгу «Властелин колец», также покупают книгу «Хоббит». Так что, если вы покупаете «Властелина колец», система может порекомендовать вам «Хоббита».

Интеллектуальная обработка с помощью правил ассоциаций часто также называется анализом потребительской корзины. При посещении своего местного бакалейного магазина вы можете обнаружить, что в отделе морепродуктов рядом с рыбой лежат лимоны и винный соус. Это объясняется тем, что было выявлено, что 80% людей, покупающих рыбу, также покупают лимоны, которые к ней хорошо подходят. Большинство таких магазинов предлагают карточки постоянных покупателей. Запоминая количество комбинаций товаров, купленных одним человеком, они могут лучше расположить продукты на полках и даже посылать вам купоны на такие же или похожие товары к следующему вашему визиту.

Интеллектуальная обработка по правилам ассоциаций включает в себя оценку того, сколько раз определенная группа товаров появлялась вместе. В этом случае

вам не обязательно знать, какие комбинации вы ищете, поэтому такой анализ попадает в категорию неконтролируемого или ненаправляемого обучения. Алгоритм ассоциаций приходит к правилу вида «А подразумевает В». Для пользователя алгоритма интерес представляют два количественных показателя: спрос (support) и доверительность (confidence).

- **Спрос** — это процентная доля транзакций, в которых элементы А и В появлялись вместе. Он помогает выявить частые комбинации (например, покупка рыбы или покупка рыбы и лимонов).
- **Доверительность** — это процентная доля тех транзакций, имеющих элемент А, которые также имеют элемент В (т. е. сколько раз покупка рыбы сопровождалась покупкой лимонов).

Чтобы использовать правила ассоциаций, пользователь должен указать уровень спроса и доверительности, который он считает интересным. Например, предположим, что мы указали спрос 20% и доверительность 70%. Если мы обнаруживаем, что в 50% всех транзакций участвует рыба, а в 40% всех транзакций участвуют рыба и лимоны, то комбинация «рыба, лимоны» считается интересной, поскольку превышает минимальный уровень спроса — 20%. В этом примере комбинация «рыба, лимоны» имеет доверительность (40/50)×100 = 80%, что укладывается в минимум критерия доверительности. Следовательно, будет выведено правило ассоциации «рыба подразумевает лимон».

Для осмысленности решений в бизнесе важно учитывать и спрос, и доверительность. Рассмотрим альтернативный пример с регионом, в котором рыба непопулярна из-за высокой концентрации ртути. Может быть так, что рыба участвует только в 5% транзакций. Элемент «рыба» не будет иметь достаточно спроса, и следовательно, правило «рыба подразумевает лимоны» не окажет существенного влияния на наши продажи.

На рис. 9.36 показаны типичные выходные данные интеллектуальной обработки с помощью правил ассоциаций, проведенной в Oracle 9i Discoverer. Первые два правила говорят, что люди часто покупают с рыбой (fish) лимоны (lemons) или вино (wine).

Кластеризация

Кластеризация — это метод, используемый для разделения большого набора данных со многими атрибутами на небольшое число тесно упакованных групп. Такие группы не очень очевидны для человеческого глаза из-за большого числа атрибутов. Предположим, например, что у нас есть данные переписи населения, включающие несколько атрибутов: возраст, занятие, наличие болезней и т. п.

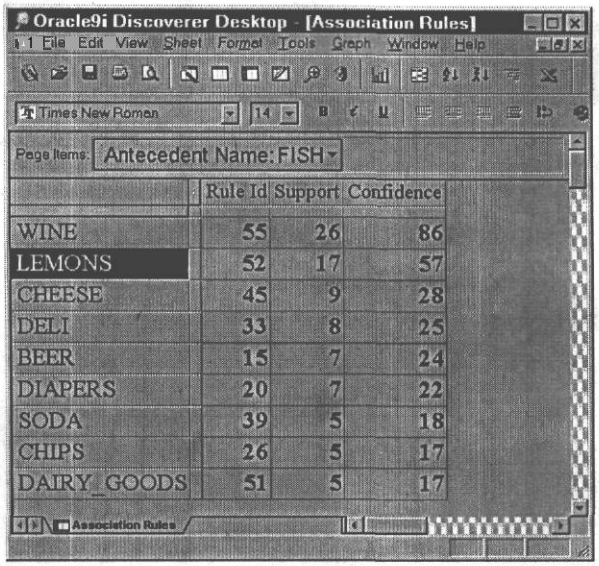


Рис. 9.36. *Правила ассоциаций*

Кластеризуя эти данные, вы можете обнаружить, что есть несколько районов, в которых превалирует определенное заболевание, что, возможно, указывает на загрязнение воды в этих районах. Поскольку у нас нет определенного представления о том, что мы можем найти, это еще один пример неконтролируемого обучения.

Группы, сгенерированные хорошим кластеризующим алгоритмом таковы, что все элементы одной группы очень похожи друг на друга и значительно отличаются от элементов Других групп. рис. 9.37 иллюстрирует концепцию кластеризации данных. На этом рисунке два атрибута расположены на графике рассеяния (scatter plot), чтобы подчеркнуть кластеры. Для всех элементов одного кластера значения этих двух атрибутов имеют большее сходство, по сравнению со значениями в других кластерах. В этом примере кластеры стали очевидны при простом рисовании графика. На практике часто невозможно таким образом визуализировать кластеры, поскольку измерений может быть больше, чем два или три! Например, в биоинформатике могут встречаться тысячи измерений! В таком случае для выявления кластеров используются математические кластеризующие алгоритмы.

Одной из областей применения кластеризации является сегментирование рынка. Предположим, что вы крупный розничный продавец, торгующий широким диапазоном продуктов от мыла до дисководов. Используя кластеризацию,

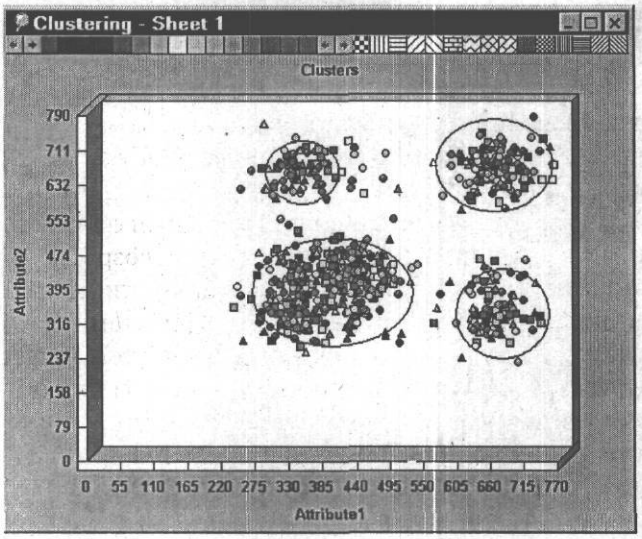


Рис. 9.37. *Кластеризация*

вы сможете сегментировать свою базу данных о клиентах на основе демографических данных или сведений о прошлых предпочтениях в покупках. Это позволит вам формировать стратегию рекламной компании для каждого сегмента, чтобы наилучшим образом обслуживать те сегменты, в которых ожидается наибольшая выгода.

Кластеризация проводится следующим образом: сначала анализируется небольшая часть данных, и определяются кластеры. Когда кластеры определены, анализируются оставшиеся данные, и каждый отдельный элемент вводится с определенной вероятностью в определенный кластер.

Классификация

Предположим, вам нужно нацелить рекламную кампанию новой цифровой камеры и вы хотите узнать, кто из ваших клиентов, скорее всего, купит такую камеру. Классификация — это метод интеллектуальной обработки данных, подходящий как раз для такого случая. При классификации данные делятся на два или более четко определенных класса. В отличие от кластеризации, где вы не знаете, какие будут сформированы группы, при классификации вы точно знаете, что представляет собой каждая группа. В приведенном выше примере две группы —

это покупатели, которые, скорее всего, купят камеру, и покупатели, которые, скорее всего, не будут ее покупать. Эта ситуация — пример контролируемого обучения.

При классификации сначала проводится анализ небольшой части данных и строится модель. Например, вы анализируете реальные данные о людях, которые покупали цифровые камеры, за определенный период времени. Данные, используемые для построения модели, называются формирующими данными (build data). Модель будет учитывать различные факторы, такие, как возраст, доходы и область занятий, которые, как известно, влияют на предпочтения людей при покупках. Эти факторы называются предсказательными атрибутами (predictor attribute). Предсказанный результат называют целевым атрибутом (target attribute), и его значения (т. е. будет человек покупать камеру или нет) называются категориями или классами. Когда модель сформирована, ее можно применять к другим данным и получать прогноз. Этот процесс называется применением модели (model apply) или подсчетом (scoring). В нашем примере мы используем модель для предсказания того, будет ли данный клиент покупать цифровую камеру.

В нашем примере целевой атрибут имеет два значения: будет покупать цифровую камеру и не будет покупать цифровую камеру. Вы также можете использовать классификацию для предсказания атрибутов при более чем двух значениях. Например, является ли риск того, что человек откажется платить, низким, средним или высоким.

Классификацию часто используют для создания профилей клиентов. Например, когда вы определили, какие клиенты, возможно, купят цифровую камеру, вы можете составить их профиль по занятиям, как это показано на рис. 9.38. по этому графику вы можете видеть, что большая часть вероятных покупателей — это инженеры и администраторы. Так что вы сможете нацеливать свою рекламу более точно на этих клиентов, уменьшая тем самым свои затраты.

Чтобы классификация хорошо работала, формирующие данные должны содержать достаточное число примеров для каждой целевой категории, в противном случае результаты будут неточны. Иными словами, формирующие данные должны включать сведения о достаточном числе людей, купивших в прошлом цифровую камеру, и достаточно о тех, кто ее не покупал.

Тестирование модели классификации

Интеллектуальная обработка с помощью классификации обычно имеет в своем составе фазу тестирования, в которой проверяется, насколько хороша модель. Для этого тестируются данные, для которых результат известен, и выясняется,

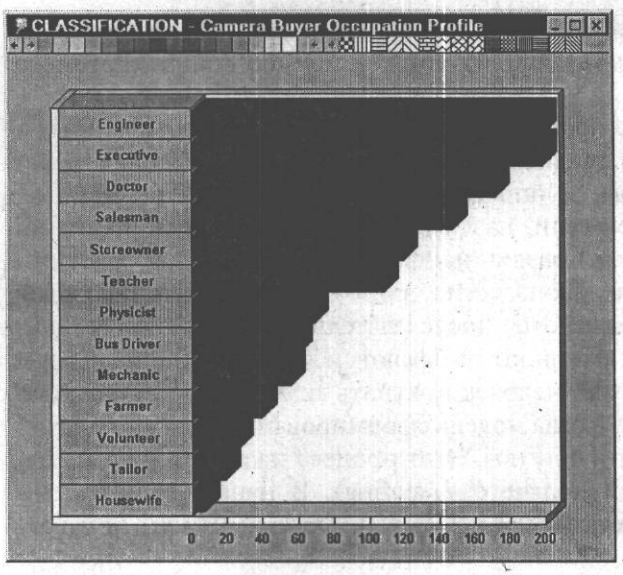


Рис. 9.38. *Классификация*

насколько предсказания моделей соответствуют этому результату. Например, можно взять данные о клиентах, которые покупали цифровые камеры раньше, и проверить по ним предсказания, даваемые моделью.

Тестирование модели включает в себя вычисление структуры, известной как матрица ошибок (confusion matrix). Матрица ошибок говорит, сколько раз предсказание модели совпадало с реальными данными, и сколько раз не совпадало. Столбцы соответствуют предсказанным значениям, а строки — реальным. Например, на рис. 9.39 модель оказалась права 555 + 45 = 600 раз и неправа 12 + 8 = 20 раз. Это показывает, что модель очень хорошая.

Вычисление подъема

Еще один количественный показатель эффективности модели — это подъем (lift). Чтобы понять, что это такое, рассмотрим следующий пример. Предположим, что у нас есть база сведений о 100 тысячах семей и в среднем около 2.5% из них (2500 клиентов) отвечают на любую рекламу. Мы хотим поступать умно и направлять рекламу только на тех покупателей, которые, скорее всего, на нее ответят. Имея хорошую модель классификации, мы сможем выявить большую часть из этих 2500 отвечающих, нацеливая рекламу на мень-

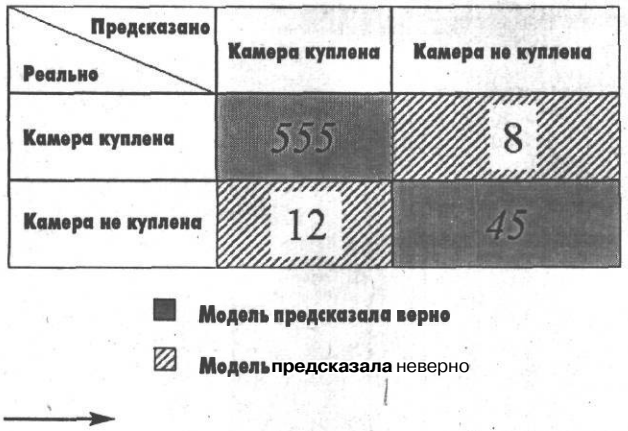


Рис. 9.39. *Матрица ошибок*

шее число клиентов, чем 100 тысяч. При данной процентной доле клиентов, на которых нацеливается реклама, подъем — это отношение числа отвечающих на нее, полученное с помощью модели, к числу отвечающих на нее, полученному без модели.

Подъем вычисляется следующим образом. Модель классификации применяется к реальному набору данных о клиентах, где мы знаем, кто отвечал на рекламу, а кто — нет. Затем клиенты сортируются по вероятности ответа, как она предсказывается моделью, и клиенты с наибольшей вероятностью ответа помещаются впереди. Этот отсортированный список делится на десять одинаковых групп, называемых децилями. Для каждого дециля вычисляется количество клиентов, которые в реальности ответили на рекламу. Если модель хороша, большая часть ответивших попадет в первые несколько децилей, поскольку предсказано, что они являются наиболее вероятными отвечающими.

Если вы нарисуете график, где децили с первого по десятый будут расположены на оси X, а количество реально ответивших клиентов — на оси Y, вы, скорее всего, получите кривую, похожую на ту, что изображена на рис. 9.40. Эта кривая говорит нам, что чтобы получить 70% отвечающих, рекламу можно направлять только на клиентов в первых трех децилях (30% всех семей). С другой стороны, прямая линия на этом рисунке соответствует случайному ответу, когда предсказывается, что каждый может ответить с равной вероятностью. Если вы, не имея модели, направите рекламу на 30% семей, вы получите 30% вероятных отвечающих. Чтобы получить 70% отвечающих, нужно нацелить рекламу на 70% клиентов! Чем выше кривая от прямой линии в первых 1-2 децилях, тем лучше подъем модели.

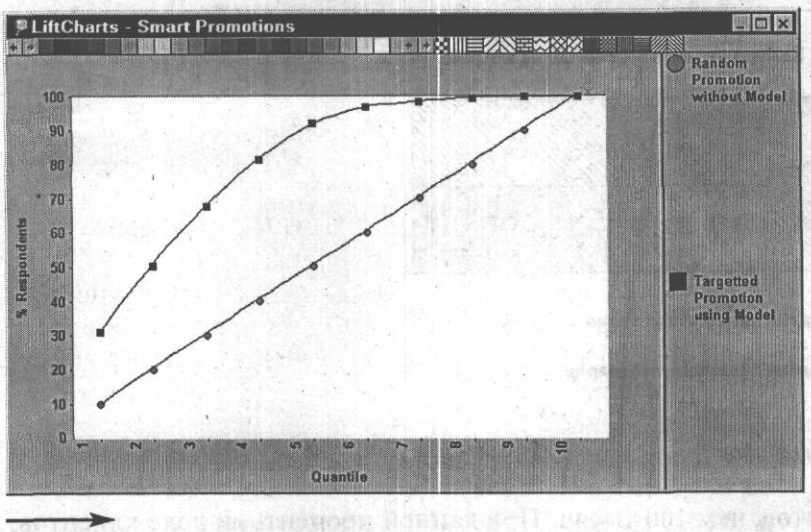


Рис. 9.40. *Использование анализа подъема для направленной рекламы*

И матрица ошибок, и подъем — часто применяемые методы определения точности модели классификации. В Oracle Data Mining предлагаются API, позволяющие вычислять эти показатели.

9.2.2. Алгоритмы интеллектуальной обработки данных Oracle

Теперь, когда мы изучили методы, используемые при интеллектуальном анализе, мы коротко рассмотрим различные алгоритмы, которые предлагает Oracle для каждого из этих методов.

Правила ассоциаций

Алгоритм, который используется в Oracle Data Mining для правил ассоциаций, называется Apriori. Пользователь указывает минимальные значения спроса и доверительности. Сначала алгоритм определяет отдельные элементы, которые встречаются часто и укладываются в минимальный спрос, например «рыба». Затем определяются пары элементов, укладывающихся в минимальный спрос, так чтобы по меньшей мере один из элементов пары встречался часто сам по себе, например «рыба» и «лимоны». Процесс повторяется, находится все больше комбинаций, пока количество

их не заканчивается. Когда все наборы элементов найдены, выявляются те из них, которые удовлетворяют критерию доверительности, указанному пользователем. Такие пары выводятся как правила ассоциаций.

Классификация

Существует два главных алгоритма классификации: Naive Bayes и Adaptive Bayes Network.

Алгоритм Naive Bayes (NB) основан на теореме вероятности, известной как теорема Байеса, и он предполагает, что каждый атрибут является независимым от других. Интересное свойство алгоритма NB состоит в том, что строить и осуществлять перекрестную проверку модели можно с помощью одних и тех же данных. Лучше всего алгоритм работает при небольшом числе предсказательных атрибутов (менее 200).

Алгоритм Adaptive Bayes Network (ABN) создает модель в форме дерева решений, пример которого показан на рис. 9.41. По этому дереву можно видеть, что камеры, скорее всего, будут покупать мужчины в возрасте от 15 до 35 лет и женщины после 26. Поскольку модель, созданная ABN, понятна для человека, бизнес-аналитику или администратору удобнее использовать для принятия решений именно ее. Алгоритм ABN также более точен, но построение модели требует больше времени.

Кроме этого, Oracle также предлагает свойство Model Seeker. Это свойство позволяет генерировать несколько моделей с помощью алгоритмов ABN и NB, а затем определять, какая из них лучше подходит для конкретной категории. Например, оно может сказать вам, какая модель лучше предсказывает покупателей

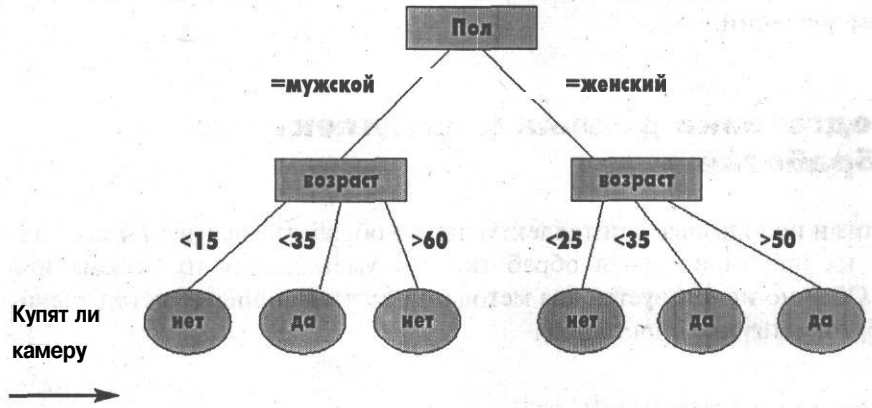


Рис. 9.41. *Дерево решений алгоритма Adaptive Bayes Network*

цифровых камер, а какая лучше предсказывает покупателей кухонных принадлежностей. Model Seeker также генерирует итоговые сведения обо всех созданных им моделях, и пользователь может определить, какая модель лучше.

Кластеризация

Oracle Data Mining поддерживает два алгоритма кластеризации: улучшенный алгоритм k-means и O-Cluster.

Алгоритм k-means — это алгоритм кластеризации, который группирует данные в указанное число k кластеров. Элементы группируются в кластеры на основе их относительной удаленности друг от друга. Таким образом, все точки одного кластера будут ближе друг к другу, чем к точкам других кластеров. Улучшенный алгоритм k-means (Enhanced k-means) представляет собой вариант алгоритма k-means, который формирует кластеры иерархическим образом. Начинает он с того, что все данные помещает в один кластер, а затем расщепляет их на меньшие кластеры до тех пор, пока не будет получено желаемое количество кластеров. Этот алгоритм гораздо эффективнее традиционного алгоритма k-means, поскольку ему требуется только один проход по данным и, следовательно, он может обрабатывать большие объемы данных. Он хорошо работает даже для наборов данных, имеющих менее 10 атрибутов. Расстояние между точками, которое использует k-means, может быть выражено только числовыми атрибутами, поэтому, если у вас имеются дискретные значения (например, цвет красный, черный, голубой), алгоритм k-means использовать нельзя.

Алгоритм O-Cluster определяет кластеры, используя диапазоны значений атрибутов. Пользователю не нужно указывать количество создаваемых кластеров. Этот алгоритм можно использовать для нечисловых атрибутов с дискретным набором значений.

9.2.3. Подготовка данных к интеллектуальной обработке

Чтобы вы могли использовать интеллектуальную обработку данных, может потребоваться их предварительная обработка для уменьшения их объема или сложности. Обычно используется два метода: загрузка (binning) и метод значимости атрибутов (attribute importance).

Загрузка

Алгоритмы интеллектуальной обработки, как правило, требуют, чтобы данные для анализа были «загружены» в ряд отдельных «емкостей». Oracle предоставляет возможность выполнять такую загрузку автоматически или дать пользователю возможность самому загрузить данные, указав граничные значения для «емкостей».

Метод значимости атрибутов

Если ваши данные имеют много предсказательных атрибутов, может оказаться проблематично определить, какие из них влияют на целевой атрибут, а какие не влияют. Например, возраст может определять музыкальные пристрастия клиента, а его рост или цвет глаз может влиять незначительно. Процесс классифицирования, в зависимости от числа предсказательных атрибутов, может занять очень много времени. Метод значимости атрибутов автоматически ранжирует атрибуты по вероятности их влияния на целевой атрибут. Это позволит вам сузить круг значимых предсказательных атрибутов перед построением модели классификации.

Алгоритм метода значимости атрибутов в Oracle Data Mining называется predictor variance (расхождение прогнозов). Алгоритм принимает набор данных вместе с их предсказательными и целевыми атрибутами. Он выясняет, как меняется целевой атрибут при изменении предсказательного атрибута (т. е. высчитывает расхождение целевого атрибута относительно предсказательного атрибута). Если расхождение велико, это означает, что предсказательный атрибут оказывает значительное влияние на целевой атрибут, и его ранг будет высоким.

9.2.4. Стандарт PMML

Язык разметки прогностических моделей (Predictive Modeling Markup Language) — это появляющийся стандарт, основанный на XML, и предназначенный для создания моделей интеллектуальной обработки данных. PMML предлагает не зависящий от производителя метод определения моделей, которыми разные приложения могут обмениваться между собой. Таким образом, одно приложение может создать модель, а другое — применить эту модель к набору данных. Oracle Data Mining поддерживает экспорт и импорт правил ассоциаций и моделей Naive Bayes.

9.2.5. Использование Oracle Data Mining

Oracle Data Mining состоит из двух компонентов: Oracle Mining Server и Oracle Data Mining Java API.

Oracle Mining Server — это компонент базы данных Oracle, который выполняет задачи по интеллектуальной обработке. Mining Server состоит из репозитория, в котором хранятся таблицы с входными и выходными данными анализа.

Oracle data mining API — это интерфейс Java API, который позволяет разработчику приложения писать программы интеллектуальной обработки данных для описанных ранее методов.

Инсталляция и конфигурирование

Oracle Data Mining (ODM) доступен с помощью Universal Installer в виде части инсталляционного пакета Oracle 9i. Если в ходе инсталляции вы решили создать заранее сконфигурированную базу данных (см. главу 2), ODM будет инсталлирован автоматически. Вам придется, однако, разблокировать учетные записи ODM и ODM_MTR, входящие в уже готовую базу данных, и задать новые пароли. Делается это следующим образом:

```
alter user odm identified by <odm_password_here> account unlock;
alter user odm_mtr identified by <odm_password_here> account unlock;
```

Если вы решили создать пользовательскую базу данных, то для инсталляции ODM вы должны запустить Database Configuration Assistant (DBCA) и следовать его инструкциям.

Мы рекомендуем использовать для задач интеллектуальной обработки секционирование (если возможно) и параллельное выполнение, особенно в случае больших наборов данных, поскольку они дают существенный выигрыш производительности.

Чтобы использовать интеллектуальную обработку, нужно запустить ODM Task Monitor. Чтобы запустить ODM Monitor, войдите в базу как пользователь с именем ODM и запустите такую команду на SQL*Plus.

```
execute ODM_START_MONITOR;
```

Чтобы завершить работу программы, запустите такую команду:

```
execute ODM_STOP_MONITOR;
```

Теперь все готово к началу работы с API интеллектуальной обработки.

Задачи интеллектуальной обработки

Типичный процесс работы с интеллектуальным анализом данных включает в себя следующие этапы.

1. ***Предварительная обработка входящих данных.*** Может возникнуть необходимость провести предварительную обработку «сырых» данных перед анализом. Некоторые алгоритмы требуют, чтобы данные были «загружены в емкости». Пользователь может провести предварительную обработку и указать, как следует загружать данные. Этот процесс необязателен, поскольку база Oracle может автоматически выполнить такую загрузку, если пользователь сам этого не сделает.
2. ***Указание анализируемых данных.*** Анализируемые данные должны быть помещены в одну таблицу. Каждый факт анализируемых данных называется случай (case). Например, «покупатель #3, мужчина 35 лет, профессия — учитель, купил цифровую камеру» — это один случай. В таблице случай может представлять собой одну или несколько записей (по одной для каждого атрибута), как это показано на рис. 9.42.

Если все атрибуты одного случая размещаются в одной строке, говорят, что таблица имеет нетранзакционный (nontransactional) формат. Также случай может храниться в нескольких записях в формате (покупатель#, имя_атрибута, значение_атрибута). Такой формат называется транзакционным (transactional).

В каком бы формате ни хранились данные, база Oracle перед анализом переведет их в транзакционный формат.

Спецификация таблицы входных данных определяется с помощью объекта Java под названием Physical Data Specification (PDS).

3. ***Установка параметров анализа.*** Теперь вы должны указать, какую функцию вы хотите использовать (ассоциацию, кластеризацию, классификацию или метод значимости атрибутов). Вы можете указывать параметры на двух уровнях: на уровне функции или метода, который вы хотите использовать, и на уровне конкретного алгоритма, используемого вами для этого метода. Если вы не определите настройки алгоритма, база Oracle будет использовать настройки по умолчанию. Настройки функций анализа хранятся в объекте Java с именем Mining Function Settings (MFS). Интерфейс API, используемый для

Нетранзакционный формат

ID	Возраст	Пол	Занятие	Купил
1	15	М	Инженер	Да
2	27	Ж	Врач	Нет
3	35	М	Учитель	Да
		...		

Транзакционный формат

ID	Атрибут	Значение
1	Возраст	15
1	Пол	М
1	Занятие	Инженер
1	Купил	Да
2	Возраст	27
2	Пол	F
2	Занятие	Врач
2	Купил	Нет
	...	

Рис. 9.42. *Форматы таблиц для интеллектуальной обработки*

создания MFS, зависит от используемого метода интеллектуальной обработки. В следующем разделе мы рассмотрим пример классификации. Важной частью настроек MFS является спецификация Logical Data Specification, в которой вы указываете, каким образом каждый из атрибутов интеллектуальной обработки должен обрабатываться алгоритмом. Сюда входит тип атрибута, т. е. числовой он или категориальный, способ использования, т. е. целевой атрибут или предсказательный, и статус подготовки, т. е. подготовлен атрибут или нет.

После того как объект с настройками функций интеллектуальной обработки создан, хорошо бы проверить его и убедиться в его правильности. Вы также можете сохранить объект с настройками в базе данных, присвоив ему имя. Это позволит использовать эти настройки в разных программах.

4. *Построение модели.* Теперь мы готовы к созданию модели. Чтобы построить модель, вы создаете объект MiningBuildTask, сохраняете его в базе данных и запускаете его она исполнение. Исполнение модели происходит асинхронно. Вы можете в ходе выполнения задачи подавать запросы о ее состоянии. Когда выполнение закончится, модель будет сохранена в базе данных.

5. ***Применение модели к новым данным.*** Когда модель создана, вы можете применять ее к новым данным и делать прогнозы. Вы можете применять модель к новым случаям или одному случаю в таблице. Новые данные должны пройти предварительную обработку (при необходимости) и быть сохранены в таблице, совместимой с той, которая использовалась при построении модели. Чтобы применить модель, вы сначала описываете местоположение новых данных, как в шаге 2. Затем вы указываете местоположение таблицы выходных данных, в которую должны помещаться результаты прогнозирования, и определяете, какие атрибуты следует туда помещать. Наконец, чтобы применить модель, создается объект MiningApplyTask, ему присваивается имя, и он запускается на выполнение. Результаты будут сохранены в базе данных в указанной таблице выходных данных.
6. ***Тестирование модели и вычисление подъема.*** Как уже говорилось ранее, при использовании классификации может понадобиться протестировать модель и вычислить ее подъем (lift). Чтобы протестировать модель и вычислить ее подъем, нужно указать местоположение тестовых данных, которые должны иметь тот же формат, что и формирующие данные. Для тестирования модели создайте объект ClassificationTestTask и запустите его на исполнение. Будет сгенерирована матрица ошибок и сохранена в объекте ClassificationTestResult. Чтобы вычислить подъем, создайте объект MiningComputeLiftTask и выполните его. Значения подъема для каждого дециля записываются в объект MiningLiftResult.

Теперь мы проиллюстрируем некоторые из этих этапов с помощью интерфейсов Oracle Data Mining API.

9.2.6. Пример использования Oracle Data Mining API

Здесь мы представим пример использования API для одного метода, а именно, классификации. Этот пример проиллюстрирует общий ход интеллектуального анализа данных.

Замечание. Подробное описание всех API не входит в задачи данной книги. Вы можете найти полную спецификацию API в установочном комплекте базы Oracle.

Пример 1. Построение модели

Мы построим модель классификации и дадим прогноз, будет ли человек покупать цифровую камеру. Входящие данные хранятся в таблице EASYDW.TEST_BUILD_DATA. Данные хранятся в нетранзакционном формате, что означает, что все атрибуты случая хранятся в одной строке таблицы. Определение таблицы показано в следующем фрагменте кода. Целевой атрибут CAMERABUYER показывает, купил клиент камеру или нет. Он имеет значение 1, если человек приобрел камеру, и 0 — если не приобрел. Пример представляет собой фрагмент кода на Java.

```
SQL> describe test_build_data;
Name                                      Null?    Type
----------------------------------------- -------- ---------------
CUSTOMERID                                         NUMBER(10)
AGE                                                NUMBER
SEX                                                VARCHAR2(8)
OCCUPATION                                         VARCHAR2(10)
CAMERABUYER                                        NUMBER(1)
```

1. ***Предварительная обработка входных данных перед анализом.*** Мы не будем проводить предварительную обработку и дадим базе Oracle выполнить загрузку автоматически.
2. ***Указание используемого сервера интеллектуальной обработки.*** Сначала мы создаем экземпляр объекта DataMiningServer и указываем базу данных, с которой соединяемся с помощью JDBC URL пользователя, от имени которого осуществляем вход. После успешного соединения мы получаем дескриптор соединения (dmsConn), который будет использоваться далее в API.

```
DataMiningServer dms = new
 DataMiningServer
   ("jdbc:oracle:thin:@slawande-pc:1521:orcl",
    "odm", "odm");

oracle.dmt.odm.Connection dmsConn = dms.login();
```

3. ***Определение структуры данных.*** Сначала мы должны создать объект LocationAccessData, который указывает схему и имя таблицы для входных данных. Затем мы создаем объект PhysicalDataSpecification типа

NontransactionalDataSpecification. (Если бы таблица была в транзакционном формате, мы бы использовали тип TransactionalDataSpecification).

```
LocationAccessData lad = new
   LocationAccessData("TEST_BUILD_DATA", "EASYDW");
PhysicalDataSpecification pds = new
   NonTransactionalDataSpecification(lad);
```

4. ***Настройка параметров анализа.*** Теперь мы укажем параметры проведения классификации. Мы дадим базе Oracle возможность определить используемый алгоритм и не будем указывать параметры уровня алгоритма. Для DataPreparationStatus будет использовано значение «unprepared», показывающее, что данные не были предварительно «загружены» (binned) и база Oracle должна выполнить автоматическую загрузку. Как целевой атрибут указывается CAMERABUYER, который является категорийным, т. е. имеющим дискретные значения.

```
ClassificationFunctionSettings cfs
 = ClassificationFunctionSettings.create
   (dmsConn, null, pds, "camerabuyer",
    AttributeType.categorical,
    DataPreparationStatus.getInstance
      ("unprepared"));
```

Если мы хотим использовать алгоритм Naive Bayes, мы также должны указать параметры уровня алгоритма и передать их в параметры уровня функций. Код будет выглядеть примерно так, как приведено ниже. Заметьте, что мы передаем переменную настроек алгоритма Naive Bayes — nbs в класс ClassificationFunctionSettings.

```
NaiveBayesSettings nbs
 = new NaiveBayesSettings(0.01f, 0.01f);
   ClassificationFunctionSettings cfs
    = ClassificationFunctionSettings.create
      (dmsConn, nbs, pds, "targetattr",
       AttributeType.categorical,
       DataPreparationStatus.getInstance
            ("unprepared"));
```

Перед тем как продолжать построение модели, мы проверим настройки и сохраним их в базе данных под именем TEST_DATA_CFS.

```
cfs.validate();
cfs.store(dmsConn, "TEST_DATA_CFS");
```

5. ***Построение модели.*** Чтобы построить модель, мы создаем объект MiningBuildTask и сохраняем его на постоянной основе в базе данных под именем TEST_DATA_CFS_MODEL. Затем запускаем задачу на выполнение и ждем окончания ее работы.

```
MiningBuildTask task = new
   MiningBuildTask(pds, "TEST_DATA_CFS",
         "TEST_DATA_CFS_MODEL");

task.store(dmsConn, "TEST_DATA_CFS_TASK");
task.execute(dmsConn);
MiningBuildStatus = task.waitForCompletion(dmsConn);
```

Полученная модель сохраняется в базе данных. Информацию по различным объектам MiningFunctionSettings, MiningTask и MiningModel можно увидеть, посылая запросы к таблицам метаданных схемы ODM следующим образом.

```
SELECT name, user_name, function_name, algorithm_name
FROM odm_mining_function_settings;

NAME                USER_NAME     FUNCTION_NAME       ALGORITHM_NAME
------------        ----------    ------------ --------------
TEST_DATA_CFS       odm           classification      naiveBayes

SELECT id, name, type, job_id FROM odm_mining_task;

ID     NAME                        TYPE          JOB_ID
---    ------------------  -----  ------
24      TEST_DATA_CFS_TASK  BUILD  24

SELECT name, function_name, algorithm_name, data_location
FROM odm_mining_model;
```

```
NAME                 FUNCTION_NAME       DATA_LOCATION          -
-------------------  ------------------  ---------------------
TEST_DATA_CFS_MODEL  classification      ASYDW.TEST_BUILD_DATA
```

В следующем примере показано, как эта модель применяется к данным.

Пример 2. Применение модели

В этом примере мы применим модель, созданную в предыдущем примере, и сделаем прогноз о том, какие клиенты, скорее всего, купят цифровую камеру.

1. ***Указание используемого сервера интеллектуальной обработки.*** Этот шаг выполняется аналогично предыдущему примеру.
2. ***Определение логической структуры данных.*** Теперь мы указываем местоположение данных, к которым будем применять модель. Эта таблица должна иметь формат, совместимый с формирующими данными. В нашем примере мы будем использовать таблицу EASYDW.TEST_APPLY_DATA.

```
LocationAccessData lad = new
  LocationAccessData("TEST_APPLY_DATA", "EASYDW");
PhysicalDataSpecification pds = new
  NonTransactionalDataSpecification(lad);
```

3. ***Указание местоположения выходных данных.*** При применении модели получающийся прогноз будет помещаться в таблицу. Мы должны указать местоположение этой таблицы, а также определить объект MiningApplyOutput, который описывает, какие столбцы мы хотим в ней иметь. В нашем примере мы будем использовать таблицу EASYDW.TEST_APPLY_OUT.

```
LocationAccessData ladOut = new
  LocationAccessData("TEST_APPLY_OUT", "EASYDW");

MiningApplyOutput mao = new MiningApplyOutput() ;
```

Каждая строка таблицы выходных данных содержит прогноз по целевому атрибуту, в нашем случае — будет ли клиент покупать цифровую камеру или нет. Целевой атрибут представлен в виде прогноза (1 или 0), вероятности прогноза и ранга.

Целевой атрибут мы определяем с помощью объекта ApplyTargetProbabilityItem следующим образом.

```
ApplyTargetProbabilityItem camerabuyerAttr = new
  ApplyTargetProbabilityItem
    (new Attribute("CameraBuyer", DataType.stringType),
     new Attribute("Probability", DataType.stringType),
     new Attribute("Rank", DataType.stringType));
```

Атрибут CAMERABUYER имеет два значения: 1, означающее, что клиент купит камеру, и 0, означающее, что клиент не купит камеру. Мы должны определить эти значения следующим образом.

```
camerabuyerAttr.addTarget
   (new Category("CameraBuyer", "1",
       DataType.getInstance("int")));
camerabuyerAttr.addTarget
   (new Category("NotCameraBuyer", "0",
       DataType.getInstance("int")));
```

Кроме этого, нам нужно сохранить в таблице выходных данных идентификатор клиента, для которого сделан этот прогноз. Он копируется из исходной таблицы EASYDW.TEST_APPLY_DATA, по которой проводится анализ, и, следовательно, мы определяем его с помощью объекта SourceAttributeItem, как это показано ниже. Первый параметр показывает имя и свойства столбца исходной таблицы, а второй параметр определяет имя этого столбца в таблице выходных данных.

```
ApplySourceAttributeItem customerIDAttr = new
  ApplySourceAttributeItem
    (new MiningAttribute("CUSTOMERID",
        DataType.intType,
        AttributeType.notApplicable,
        false, false),
    new Attribute("CUSTOMER_ID", DataType.intType));
```

Наконец, мы добавляем все это в объект MiningApplyAttribute, который мы создали ранее.

```
mao.addItem(camerabuyerAttr);
mao.addItem(customerIDAttr);
```

4. ***Создание и запуск на исполнение задачи по применению модели.*** Теперь мы готовы к применению модели. Этот процесс почти идентичен построению модели. Мы создаем объект MiningApplyTask и передаем ему местоположение входных данных, имя модели (которую мы создали в предыдущем примере), местоположение таблицы выходных данных и параметры вывода результатов применения модели. Затем мы сохраняем этот объект в базе данных под именем TEST_DATA_CFS_APPLY_TASKи запускаем его.

```
MiningApplyTask task = new
   MiningApplyTask  (pds,
      "TEST_DATA_CFS_MODEL",
      mao, ladOut,
      "TEST_DATA_CFS_OUTPUT");

task.store(dmsConn, "TEST_DATA_CFS_APPLYTASK");
task.execute(dmsConn);
```

Когда выполнение задачи завершено, прогнозы сохраняются в таблице EASYDW.TEST_APPLY_OUT. Заметьте, что столбцы соответствуют атрибутам, указанным нами в шаге 3. Для каждого идентификатора клиента имеется две строки: одна — с вероятностью покупки камеры (прогноз = 1), а другая — с вероятностью непокупки камеры (прогноз = 0). Прогнозы ранжируются по наиболее вероятным случаям. Таким образом, клиент с идентификатором 4, скорее всего, не купит камеру (ранг 1 => camera buyer = 0), а клиент с идентификатором 26, скорее всего, купит камеру (ранг = 1 => camera buyer = 1).

```
SELECT * FROM EASYDW.TEST_APPLY_OUT;

CAMERABUYER   PROBABILITY      RANK      CUSTOMERID
-----------   -----------   --------     ----------
0                    .959025771              1            4
1                    .040974229              2            4
0                    .699846267              1            20
1                    .300153733              2            20
1                    .573198618              1            26
0                    .426801382              2            26
...
```

Для анализа результатов вы можете использовать любой аналитический инструмент, предлагаемый компанией Oracle, например, 9i Discoverer или Reports. Большая часть диаграмм, приведенных в этой главе, были получены при анализе результатов работы различных функций интеллектуальной обработки данных с помощью Discoverer. В некоторых случаях для внесения результатов обработки в таблицы может потребоваться немного попрограммировать.

9.3. Заключение

Интеллектуальная обработка — это процесс поиска скрытых закономерностей в данных. Oracle Data Mining встраивает функции интеллектуальной обработки в сервер базы данных, что повышает производительность обработки и позволяет проводить масштабирование до больших объемов. В этой главе мы рассмотрели различные области применения интеллектуальной обработки и то, как можно осуществлять такую обработку с помощью Oracle Data Mining. Мы надеемся, что эта глава позволила вам разобраться в том, как можно использовать интеллектуальную обработку данных для принятия лучших решений в бизнесе.

Приложение

В этой книге описаны способы использования широкого диапазона продуктов Oracle. Чтобы применять программное обеспечение так, как описано в этой книге, вам нужно иметь приводимые ниже продукты. Упомянутые версии являются минимальными, поддерживающими описываемые свойства, на момент написания этой книги. Мы рекомендуем вам использовать самые последние доступные вам версии продуктов.

Oracle Database Enterprise Edition, 9.2.0.1.0

- Oracle Data Mining Option 9.2.0.1.0
- Oracle OLAP Option 9.2.0.1.0, большая часть функций OLAP
- Oracle OLAP Option 9.2.0.2.0, средства автоматического создания рабочих областей.

Oracle 9i Application Server, Enterprise Edition, Release 2, 9.0.2

- JDeveloper 9.0.3 with Business Intelligence Beans
- Oracle 9i Developer Suite (для Discoverer, Report Services и Warehouse Builder)

Перечисленные программы можно скачать с сайта http://otn.oracle.com. Чтобы получить листинги примеров кода SQL, использованных в этой книге, обращайтесь на web-сайт http://www.lilianhobbs.com. Вы также можете связаться с авторами по e-mail ssl@cs.wisc.edu и susan.hillson@oracle.com.

Предметный указатель

A

Adaptive Bayes Network (ABN) алгоритм, 547
AGGMAP, команда, 505, 507
ALTER INDEX REBUILD, выражение, 434
ALTER TABLE EXCHANGE PARTITION, условие, 316-318
Analytic Workspace Manager, 521
  иллюстрация, 522
  что это такое, 521
Apriori, 546
ATTRIBUTE, условие, 197
  в измерениях, 214-215
  что это такое, 214

B

BI Beans, 495, 524-536
  доступ к данным, 524
  задачи, 524
  что это такое, 524
Buffer Cache Advisor, 441
Business Intelligence Beans *СМ. BI Beans*

C

Calculation Builder, 535-536
Capacity Planner, 440
CASE, выражение, 152-153
  использование, 153
  поисковое, 152
  пример использования, 152-153
  что это такое, 152
CONCAT, оператор, 286, 502
CREATE DIMENSION, выражение, 194, 195, 199
CREATE MATERIALIZED VIEW, выражение, 166, 175-176
CREATE OUTLINE, выражение, 442
CREATE TABLE AS SELECT, выражение, 319-320
Cube Viewer, 519
Cube Wizard, 516-518
  выбор измерений, 517
  выбор таблицы фактов, 516
  окно итоговой информации, 518
  соединение таблиц фактов и измерений, 517
  указание мер, 518
CUBE, оператор, 121, 122
  и затраты времени/места, 124
  и поддерживаемые агрегаты, 122
  сравнение выходных данных, 124
  что это такое, 121, 122
  см. также функции SQL
CUME_DIST(), функция, 133, 148
CWM-пакеты, 514

D

Database Configuration Assistant (DBCA), 550
DBMS_AW, пакет, 510
DBMS_MVIEW, процедуры обновления, 178-180
DBMS_OLAP, пакет, 239-245
DBMS_STATS, 78, 184
  использование, 184
  статистика, собираемая с помощью, 438
Decision support systems (DSS), 8
DENSE_RANK(), функция, 130
Dimension Wizard, 334
DISABLE QUERY WRITE, условие, 170
Discoverer Administrator
  Summary Wizard, 365, 366
  автоматические соединения, 358
  выбор схемы, 355
  выбор таблиц/представлений, 357
  доступ к области бизнес-задач, 368
  изменение свойств элемента, 360
  иллюстрация области бизнес-задач, 369
  класс элементов, 364
  обновление сводки, 366
  определение иерархии, 363
  определение соединения, 361
  первый запуск, 355
  присвоение имени области бизнес-задач, 359

создание вычисляемого элемента, 362
создание сводки, 365
список задач, 356
установка области бизнес-задач., 359
что это такое, 349
см. также Oracle Discoverer
Discoverer Desktop, 349
Discoverer Plus, 368-378
Export Wizard, 471
Graph Wizard, 378
SQL Inspector, 376
выбор конкретных данных, 379
выбор отображаемых данных, 371
вычисление итоговых значений, 374
движение вверх и вниз по иерархии, 376
запуск, 368-369
и рабочая книга, 370
иллюстрация отчета, 374
использование, 468-469
настройка отчета, 375
отчет по месяцам, 378
отчеты с использованием условий, 379
помесячный график продаж, 378
раскладка таблицы, 372
сортировка данных, 373
список баз данных для соединения, 370
форматирование заголовков, 372
что это такое, 349
см. также Oracle Discoverer
Discoverer Viewer
выполнение запроса, 351
запуск, 350
использование, 468-469
определение критериев отчета, 470
отчет в браузере, 469
параметры вывода (окно 1), 353
параметры вывода (окно 2), 353
параметры вывода (окно 3), 353
преимущества, 349
продажи по категориям, 352
создание запросов с помощью, 349-351
что это такое, 349
см. также Oracle Discoverer

E

Easy Shopping Inc., 33-34, 45 ,
определение безопасности, 77-78 ,
создание индекса, 72-75
создание ограничений, 72-75
создание таблиц, 72-75
схема, 70-78
табличные пространства, 47, 71-72
файлы данных, 71-72
ENABLE QUERY REWRITE, условие, 170
ENABLE QUERY WRITE, условие, 170, 172
ENABLE/DISABLE, условие, 54, 55
ETL, процесс, 11
загрузка, 265-299
извлечение, 250-251
иллюстрация, 249
инструменты, 325
преобразование, 260-265
трудности разработки, 249
см. также Загрузка данных
Executive information systems (Управленческие информационные системы) (EIS), 18
EXPLAIN PLAN, 113-114, 227, 228, 297
EXPLAIN_MVIEW, утилита, 188-192
в Oracle Enterprise Manager, 193
вызов, 189
выходные данные, 189-190
для существующего материализованного представления, 192
и переписывание запросов, 191
функции, 189
что это такое, 188-189
EXPLAIN_REWRITE, утилита, 231-234
использование, 232
результаты, 232
что это такое, 231
EXTEND_WINDOW, процедура, 258

F

FCEXEC, команда, 506
FCSET, команда, 506
FIRST, функция, 144

G

GRANT, команда, 447

Graph Wizard, 378

GROUP BY, условие, 132, 206
- в запросе к материализованному представлению, 172
- и переписывание запросов, 227-228

GROUPING SETS, функция, 124-127
- возможности, 124
- и поддерживаемые агрегаты, 122
- соединенные, 125
- создание нескольких, 125
- что это такое, 121
- *см. также* SQL-функции

GROUPING, функция, 127-129
- значения, 127
- использование, 128
- поведение, 127
- что это такое, 127
- *см. также* SQL-функции

GROUPING_ID, функция, 128-129

H

HAVING, условие, 132

I

Index Tuning Wizard, 441

INSERT SELECT, выражение, 291

INSERT, выражение, 346

J

Java servlets, 461

JavaServer, страницы, 461

Job Scheduler, 390-395

K

Kimball, Ralph, 9

K-means, алгоритм, 548

L

Lag и lead, (упреждение и запаздывание), функции, 130, 143-144
- использование, 144
- пример, 143
- что это такое, 143
- *см. также* SQL-функции

LAST, функция, 144

LIMIT, команда, 503

LIST BACKUP, команда, 413

LIST, команда, 413

Load Wizard, 267-276
- SQL*Loader, дополнительные опции, 272
- SQL*Loader, режимы работы, 270
- доступ к, 267
- иллюстрация, 268
- мониторинг загрузки, 273
- расписание загрузки, 272
- *см. также* Oracle Enterprise Manager; SQL*Loader

M

Mapping Editor, 339-341
- и соединения, 341
- иллюстрация, 340
- открытие, 339-341

MAX, функция, 141

MERGE, выражение, 298, 300, 346

M

Naive Bayes (NB), алгоритм, 547

New Module Wizard, 329-330

NOVALIDATE, условие, 60, 216

NTITLE(), функция, 136-137

O

Object Management Group (OMG), 21

O-Cluster, алгоритм, 548

OLAP API, 488, 495, 512-513
- многомерная модель, 512-513
- что это такое, 495, 512-513

OLAP DML, 494, 500-508
- возможности, 503-504
- выполнение, 501
- для агрегирования, 504-505
- для прогнозирования, 505-507
- запуск команд, 501
- и SQL, 500
- создание отчетов с помощью, 500-501
- создание пользовательских меню, 507-508
- создание формул, 507
- что это такое, 500

OLAP Summary Advisor, 519

OLAP, каталог, 495, 513-520
требования, 513
что это такое, 513
OLAP, метаданные, 513-520
измерения, 513, 515, 517
куб, 513, 516-518
меры, 513, 518
методы создания, 513
папки мер, 513
создание в Oracle Enterprise Manager, 514-519
сущности, 513
OLAP_TABLE, функция, 511-512
ON PREBUILT TABLE, условие, 172
Operational data store (Склад рабочих данных) (ODS), 14-15
Oracle 9i Application Server, 461-471
HTTP-сервер, 461
Oracle Clickstream, 467-468
Personalization, 20, 466
продукты, 462
разработка, 461
что это такое, 461
Oracle 9i Backup and Recovery Manual, руководство 400
Oracle 9i Database, 18-20
сближение с технологией express server, 18
свойства, 18
функция интеллектуальной обработки данных, 19-20
Oracle 9i OLAP, 488-489
BI Beans, 495, 524-536
OLAP API, 495, 512-513
OLAP DML, 494, 500-508
Release 2, 493
аналитические рабочие области, 493, 496-500
архитектура, 492-495
база данных Oracle 9i, 492-493
каталог OLAP, 495, 513-520
компоненты, 492
система вычислений OLAP, 494
табличные функции, 494
Oracle 9i Option, 488-489
и Discoverer, 489
что это такое, 488
Oracle 9i Reports Server, 483-485
выбор отчетов, 484
определение формата вывода, 484-485
очередь заданий, 486
расписание отчетов, 485
Oracle 9i Reports, 472-485
Queue Manager, 484
Report Wizard, 472-478
в формате Web, 483
интеграция с Oracle 9iAS Portal, 464-465
компоненты, 472
матричный отчет, 480
отчеты с условиями, 481
примеры, 479-481
что это такое, 472
Oracle 9i, 16-19
Partition Change Tracking (PCT), 184-186
Release 2, 488
Summary Advisor, 235-246
Summary Management, 66, 161-247
база данных, 492-493
готовые базы данных, 39-40
индексы, 36
параллельное выполнение DML, 157
продукты, 16
технология RAC, 159
что это такое, 16
Oracle 9iAS Portal, 464-466
интеграция отчетов в, 471
интеграция с Oracle 9i Reports, 464-465
настраиваемые страницы, 465
преимущества, 464-465
что это такое, 464
Oracle Clickstream, 467
информация, 467
что это такое, 467
Oracle Data Guard, 422-423
конфигурация, 422, 423
что это такое, 422
Oracle Data Mining Suite, 539
Oracle data mining, 19-20, 537-556
API, 553-559
алгоритмы, 546-548
анализ подъема, 544-545
задачи, 551-553
заключение, 560

инсталляция/конфигурирование, 550-551
использование, 550-553
категории проблем, 539
классификация, 542-544
кластеризация, 540-542
методы, 539-546
подготовка **данных,** 549-550
правила ассоциаций, 546-547
расхождение прогнозов (predictor variance), 549
форматы **таблиц, 551,** 552
что это такое, 539
этапы анализа, 551-553
см. также Интеллектуальный анализ данных
Oracle Database Configuration Assistant, 39-45
параметры инициализации, 41
переменные-месторасположения файлов, 42
свойства базы данных, 43
хранение базы данных, 44
что это такое, 39
Oracle Discoverer, 349-357
Administrator, 349
Desktop, 349
EUL, 355
Plus, 349
Viewer, 349, 350-352
вопросы безопасности, 366-368
динамические отчеты, 350
и Oracle 9i OLAP Option, 489
иерархии, 362-363
интеграция с Oracle **Portal,** 350
использование, 349
классы элементов, 363-364
область **бинес-задач,** 355-358
ограничения видимости столбцов, 358-360
папки, 360
сводки, 364-366
свойства элементов, изменение, 360-361
типы отчетов, **350-354**
установка среды, 355-368
что это такое, 349
элементы, 349
элементы, 360
элементы, создание, 361-362
Oracle Enterprise Manager
Cube Wizard, 516-518
Job Scheduler, 390-395
Load Wizard, 267-276
OLAP Worksheet в, 501
Table Wizard, **51-52**
безопасность объектов, 67
внешний ключ, 57
и **EXPLAIN_MVIEW,** 193
измерения для OLAP в, 515
измерения, 50
индексы, 63
ограничение null, 57
ограничения, 55
поиск устаревших материализованных представлений, 321
при обновлении сводок, 179, 180
проверка ограничений, 58
просмотр ограничений, 60
просмотр разделов **таблиц, 317**
создание баз данных с помощью, **42-45**
создание индекса в, 86
создание материализованных представлений в, 168
создание метаданных OLAP, **514-519**
создание пользователя, 46
схемы, 48
табличные области, 47
трехуровневая **архитектура,** 382
управление с помощью, **381-396**
учетная запись **администратора,** 386
Oracle Enterprise Manager Console, 382-385
дерево **Проводника, 384**
задания в расписании, 390-395
запуск приложений, 396-397
запуск, 382
иллюстрация, 383
окно **Проводника,** 384-385
первый **запуск,** 382
элементы, 382
Oracle Expert, 440
Oracle managed files (OMF), 452
Oracle Warehouse Builder (OWB), 12, 326-348
Dimension Wizard, 334
Mapping **Editor,** 338-339
New Module Wizard, 329-330
Repository **Assistant,** 328
Warehouse Fact Table Editor, 335-336
генераторы данных, 343
генерация **кода,** 345-346
импорт таблицы, 332
источники **данных,** 331
клиент, **328-329, 331**
конфигурация физически созданной модели, 348

метаданные, 327
определение соответствий источника и цели, 338-343
поиск ключей, 341-342
преимущества, 327
преобразования, 337-338
проверка модели, 344-345
рекомендации по индексам на основе битовых карт, 348
репозиторий, 327, 328
создание измерения, 334-335
создание измерения-времени, 335
создание модуля, 329-331
создание проектной модели/конфигурации, 345-348
создание таблицы вручную, 333
создание таблицы фактов, 335-336
указание источника данных, 331
установка, 327-328
фильтрация данных, 342
формирование таблицы, 331-333
функции, 327
ход проверки, 345
что это такое, 326
этапы установки, 327-328
ORDER BY, условие, 130-131, 147

P

PARALLEL, условие, 68
PARTITION BY, условие, 132-134, 141
иллюстрация, 134
ранжирующие вычисления, 132
что это такое, 132
Partition Change Tracking (PCT), 184-186, 434
и переписывание запросов, 188
и устаревшие материализованные представления, 187
иллюстрация, 185
обновление, 184, 187
поддержка, 187
что это такое, 184
PERCENT_RANK(), функция, 133, 148
PERCENTILE_CONT(), функция, 147-148
PERCENTILE_DISC(), функция, 147-148
Performance Manager, 440
Predictive Modeling Markup Language (PMML), стандарт, 549
PREPARE_SUBSCRIBER_VIEW, процедура, 259
Presentation Wizard, 524-529
выбор мер и измерений, 526
выбор члена измерения, 527
и создание графиков, 532-534
иллюстрация, 525
расположение перекрестной таблицы, 526
форматы отчетов, 524

Q

Query Builder, 476, 519, 525
Query Wizard, 529
QUERY-REWRITE_INTEGRITY, режим, 217-221
ENFORCED, 217, 218, 219, 220
STALE_TOLERATED, 217, 218, 220
TRUSTED, 217, 220, 221

R

RAID, технология, 109
RANK(), функция, 131, 133, 148
RATIO_TO_REPORT, функция, 142-143
Real application clusters (RAC), 23
разделение буферных кэшей, 159
технология, 159
что это такое, 159
Recovery Manager (RMAN), 381, 397
Recovery Wizard, 414-419
запуск, 412
информация о схеме, 414
информация о файлах, нуждающихся в резервном копировании, 415
резервные копии, 410
содержимое резервной копии, 415
Recovery Wizard, 414-416
запуск, 416
конфигурирование, 418
местоположение файлов данных, 418
RELY, условие, 55, 216
REPORT SCHEMA, команда, 413
Report Wizard, 472-478
выбор источника данных, 474
выбор полей отчета, 477
выбор ситуации для отчета, 473
выбор стиля отчета, 474
вычисление итоговых значений, 477

настройки Query Builder, 476
определение SQL-запроса, 475
указание уровня группировки, 476
указание ширины столбцов, 478
см. также Oracle 9i Reports
REPORT, команда, 413, 503
Repository Assistant, 328
REVOKE, команда, 447
ROLLUP, функция, 121, 123-124
и поддерживаемые агрегаты, 123
определение группировок, 20
сравнение выхода, 124
что это такое, 121, 123
см. также SQL-функции

S

SQL Analyze, 440
SQL Inspector, 376
SQL MERGE
загрузка измерений с помощью, 297-300
что это такое 297
SQL*Loader, 267-287
data definition language, 267
дополнительные опции, 272
загрузка в режиме прямого пути (в один раздел), 278-281
загрузка в режиме прямого пути (параллельная), 283-285
загрузка в режиме прямого пути, 271
загрузка в режиме условного пути, 270
использование ограничений без проверки, 289
нарушения ограничений, 288
нестабильные индексы, поиск, 288-290
обработка исключений процесса загрузки, 287
обработка ошибок, 271-272
операции после загрузки, 287-290
оптимизация производительности, 276-278
постановка загрузки в расписание, 272
преобразования с помощью, 286-287
проверка протокола, 273-275, 281-282, 285, 287, 456
режимы работы, 270-271
слежение за выполнением процесса загрузки, 273
управляющий файл, 267-268
файл данных, 269
SQL-текст, совпадение, 204-205
SQL-функции, 121-149
CUBE, 121, 122
CUME_DISTO, 133, 148
DENSE_RANK(), 131
FIRST, 144
GROUP BY, 121, 124-127
GROUPING, 127-129
GROUPING_ID, 128-129
lag и lead, 130, 143-144
LAST, 144
MAX, 141
NTITLE(), 136-137
PERCENT_RANK(), 133, 148
PERCENTILE_CONTO, 147-148
PERCENTILE_DISC(), 147-148
RANK(), 131, 133, 148
RATIO_TO_REPORT, 142-143
ROLLUP, 121, 123-124
SUM, 141, 143
WIDTH_BUCKET(), 136
агрегат сдвигающейся рамки, 130, 137-140
агрегаты, созданные пользователем, 150-152
для агрегирования, 121-128
для аналитических вычислений, 129-145
линейная регрессия, 145
обратный процентиль, 146-147
отчетные агрегаты, 130, 140-142
статистические, 130, 145
SUBPARTITION TEMPLATE, условие, 100, 101
SUM, функция, 141, 143
Summary Advisor Wizard, 235-238
ввод рабочей нагрузки, 235
график рекомендаций, 236
просмотр/изменение рекомендаций, 238
реализация рекомендаций, 238
фильтры, 236
что это такое, 235
Summary Advisor, 235-246, 441
гипотетическая рабочая нагрузка, 243-244
и пакет DBMS_OLAP, 239-245
использование материализованного представления, мониторинг, 245-246
методы запуска, 235
отчеты и скрипты, 241-243
рабочая нагрузка, 236-237, 239-240
размер материализованного представления, оценка, 246
результаты, очистка, 245

рекомендации, 236-238, 241
с помощью Мастера, 235-238
требования для запуска, 235
фильтры, 236-237, 240
что это такое, 165
см. также Summary Management
Summary Management, 66, 161-247
Summary Advisor, 165, 235-246
и существующие сводные таблицы, 172-174
измерения, 165, 194-195
компоненты, 165-166
материализованные представления, 165, 166-175
необходимость, 162-164
переписывание запросов, 165, 202-228
что это такое, 161
Summary Wizard, 365, 366
Super Administrator, 385

T

Table Wizard, 51-52
критерий секционирования, 65
окно для секционирования, 62-63
последнее окно, 60, 61
создание первичного ключа с помощью, 55-56
Tablespace Map, 440
TopSessions, 440

U

USER_EXTERNAL_TABLES, словарное представление, 295

V

VALIDATE_DIMENSION, процедура, 199, 200

W

Warehouse Fact Table Editor, 335-336
Warehouse Module Editor, 334
Web
и хранилища данных, 458-486
публикация данных в, 359
распространение отчетов в, 468-471
создание отчета для, 482-486
функции, 359-360
см. также Интернет, Интранет
WHEN, условие, 313
WHERE, условие, 132, 171, 207, 450
WIDTH_BUCKET(), функция, 136
WITH, условие, 153-155
преимущества, 154
пример использования, 154-155
что это такое, 153

А

Автоматический мониторинг DML, 437
Автоматическое управление памятью, 119-120
Агрегатов вложение, 206-207
Агрегаты, вычисление, 211-212
Агрегирование
возможности, 121
карта, 505
функции для, 121-128
Анализ возможных вариантов, 148-149, 491-492
Анализ набора товаров, 539
Аналитические приложения, 489-492
анализ вариантов, 491-492
аналитические вычисления и агрегирование, 489-491
моделирование, 491
прогнозирование, 490
распределение, 490
финансовые вычисления, 491
что это такое, 489
Аналитические рабочие области, 493, 496-500
временные, 496
Доступ в приложениях, 509-513
концепции, 496-500
постоянные, 496
средства создания, 519-523
что это такое, 488, 496
Аналитических рабочих областей, мастер создания, 422-423
Ассоциаций, правила, 546-547
алгоритм, 546
доверительность, 540
иллюстрация, 541
использование, 540
спрос, 540
что это такое, 539
см. также Интеллектуальный анализ данных; Oracle data mining

Б

Базы данных
Oracle 9i, 492-493
виртуальная частная, 448-450
восстановление, 416-419
выбор, 402
одна или много, 38
положение файлов, 42
присвоение имени, 40
резервная, 421
резервное копирование, 324
свойства, 43
сконфигурированная, 39-40
создание с помощью Oracle Enterprise Manager, 42-45
хранение, 44
Без потерь, соединение, 215
Без префикса, локальные индексы, 107-109
Безопасность, 66-67
Easy Shopping Inc., 77-78
материализованных представлений, 176-177
объектные привилегии, 66-67
поддержание, 447-450
роли, 58
системные привилегии, 66-67
уровень, 447
Бизнес-задач, область, 355-358
в Discoverer Administrator, 369
настройка, 359
определение доступа к, 368
присвоение имени, 359
что это такое, 355
Битовые карты, индексы на основе, 82-87
альтернатива, 90
и соединительный индекс на основе битовых карт, 89
использование, 85
недостатки, 86-87
по двум столбцам, 84
по столбцам внешних ключей, 117
пол, 83
профессия, 84
рекомендации, 348
сжатие, 86, 109
создание в OEM, 86
что это такое, 82
см. также Индексы
Быстрое обновление, 180-184
и материализованные представления, 181
и операции обслуживания разделов, 184-187
пример, 181-182
ситуации, 191
хорошая производительность, 184
что это такое, 180-181
см. также Обновление; Обновление, операция

В

Величинные измерения, 496
Виртуальная частная база данных (VPD), 448-450
для тонкого контроля доступа, 450
что это такое, 450
Вложенных контуров, соединение, 115
Внешние таблицы, 291-300
доступ для чтения, 291
доступ к данным, 295-296
загрузка данных из, 296
иллюстрация, 293
определение метаданных, 292
определение параметров доступа, 292-293
параллельная загрузка данных из, 296-297
положение файлов данных/файлов протокола, 291-292
создание, 292-295
что это такое, 291
см. также Таблицы
Возобновляемые операторы, 452-454
отложенные, 453, 453
статус, 453
Восстановление
конфигурирование, 418
при катастрофах, 419-421
протокол, 419
Восстановление из резервной копии, 414-419
обеспокоенность, 414
с помощью Recovery Wizard, 416-418
тип, выбор, 417
Восстановление после катастроф, 419-421
Enterprise, 422-423
Oracle Data Guard, 421-422
план, 421
что это такое, 419
см. также Восстановление
Восстановление, каталог, 410-411
иллюстрация, 411
размещение в базе данных, 411
регистрация базы данных в, 413
создание, 411
Время загрузки данных, 456

Вычисления, 535-536
аналитические, 489-490
вывод **итогов,** 477
графики, 536
определение параметров, 536
создание, 535
финансовые, 490

Г

Генерация **данных,** 262, 307
Глобальные индексы, 105-106
иллюстрация, 105
код SQL для **создания, 106**
что это такое, 105
см. *также* Секционированные индексы
Графики, 532-534
возможность перемещения по иерархии, 534, 532
иллюстрация, 533
определение внешнего **вида,** 533
отображение данных, 532
пример, 534
Графический пользовательский интерфейс (GUI), инструменты, **38**
Группирующие функции, 133-134
Группы
иллюстрация, 386
создание, 385-386
что это такое, 385-386

Д

Данные
агрегирование, 504-505
генераторы, 343
доступ, внешние таблицы, 295-296
загрузка, 248-325
извлечение, 250-251
изменение, 251-260
интеграция, 261
многомерные, доступ с помощью **SQL, 511-512**
неструктурированные, 22
объем, повышение, 22
определение структуры,
очистка, 261-262, 307
перемещение, **303, 315-320**
покупка, 12
получение подробностей, 352, 376
получение, **262, 307**
преобразование, **11,** 260-265
проверка, 308
публикация, 325
рабочие, **11**
секционирование, **62-63, 90-105**
сжатие, 109-111
фильтрованные, **208-210, 342**
Динамический подбор, 438
Дисковое пространство
автоматизированное управление, 452
возобновляемое выделение **места, 452-453**
использование, 452
мониторинг использования, **451-454**
Доступа параметры, указание, 292-293
Доступность, 23

З

Загрузка в емкости (binning), 549
Загрузка данных, 248-325
в один раздел, 278
загрузка одного раздела в режиме прямого пути, 278-279
и **ETL-процесс,** 248-250
и дублирующиеся даты, 278
из внешних **таблиц,** 291-300
измерения, 297-300
операции после загрузки, 320-324
параллельная в режиме прямого пути, 284
параллельная, 278
первоначальное заполнение, 248
с **помощью SQL*Loader,** 267-287
с помощью транспортируемых табличных областей, 300-306
таблица фактов, 266
таблицы измерений, 265-266
см. также Хранилища данных; **ETL,** процесс
Задания
выбор действия для, 392
выходные данные, 395
информация о, 394
количество, 395
наблюдение, запуск, 394
параметры, 393
расписание, 390-395
резервное копирование, 405
создание, 391
сохранение, 391

Запросы
OLAP, 18, 487
время обработки, определение, 457
вычисление агрегатов в, 211-212
и множественная группировка, 226
производительность, 439
с несколькими материализованными представлениями, 226-228
создание, 475
условия, 210
что это такое, 524
Запросы, оптимизация, 112-119
на основе правил, 112-113
на основе стоимости, 112
соединения, 115-119
Заранее сконфигурированные базы данных, 39-40
Звезда, преобразование по типу, 117-118
доступность, 117
функционирование, 118-119
что это такое, 117
Звезда, схема
иллюстрация, 33
что это такое 36

И

Иерархии, 362-363
Иерархические взаимоотношения, 194
Иерархические индексы, 81-82
виды, 81
идентификаторы строк, 81
ключ индекса, 81-82
сжатие ключей, 109
требования к дисковому пространству, 82
что это такое, 81
см. также Индексы
Извлечение данных, 250-251
Oracle change data capture, 252-260
идентификация измененных данных, 251-252
что это такое, 250
см. также ETL, процесс process
Извлечение/преобразование/загрузка *см. ETL, процесс*
Изменение данных, 251-260
архитектура публикация/подписка, 254
выявление, 251-252
завершение подписки, 260
запросы, 258
изменения, перенос, 260
обработка, 257-260
подписка, 256
публикация, 254
слежение за (CDC), 252-260
создание подписки, 256-257
см. также Извлечение данных
Изменений набор, 254
Изменений таблицы, 252-253
Измерений таблицы
Easy Shopping Inc., 72-75
загрузка, 265-266
и таблицы фактов, 29-30
создание, 51-52
что это такое, 30-31
Измерения, 165, 194-201, 497
величинные, 497
взаимосвязи, объявленные в, 212
время, 196, 212
время, создание (OWB), 335
выбор (Presentation Wizard), 526
выбор членов, 527
выбор, 517
географическое, 194-195
загрузка, 297-300
и переменные, 501-502
метаданные OLAP, 513
нормализованные, 198
определение взаимосвязей, 194
определение с множественными иерархиями, 195-196
определение с нормализованными таблицами, 198-199
определение, с атрибутами, 197-198
оценка, 199-201
переписывание запросов в, 212-215
проверка данных в, 308
проверка, 320
соединение с таблицами фактов, 517
создание (OWB), 334-335
создание, 194-195
уровневые, 497-498
условие ATTRIBUTE в, 214-215
что это такое, 501-502
см. также Summary Management

Имена
базы данных, 40
области бизнес задач, 359
событий, 387
соглашение об, 38-39
Индексирование, 80-88
материализованные представления, 175
необходимость, понимание, 80
решения по, 80
Индексы, 36, 61-62
Easy Shopping Inc., 72-75
Oracle Enterprise Manager, 63
выбор, 61-62
глобальные, 105-106
иерархические, 81-82
изменения, 434-435
локальные, 107-109
на основе битовых карт, 82-87
нестабильность, проверка на, 288-290
нестабильные, перестройка, 290
отсрочка обработки, 277
планирование, 62
разделы, 427-428
секционированные, 104-109
соединительные на основе битовых карт, 87-90
строительные блоки, 80
типы, 61-62, 80
удаление, 284
Инкрементное резервное копирование, 405-406
выбор, 406
уровень, 405-406
что это такое, 405
см. также Резервное копирование
Инструменты для работы с хранилищами, 326-380
Oracle Discoverer, 349-357
Oracle Warehouse Builder (OWB), 326-348
выбор, 326
Интеграция данных, 261
Интеллектуальная обработка данных, 560
Oracle, 19-20, 537-556
алгоритмы, 539
использование, 539
статистические методы и методы машинного обучения, 537
форматы таблиц, 551, 552
что это такое, 19, 537
Интернет, 358-360
Интранет, 358-360
пример содержимого, 359
что это такое, 358

К

Классификация, 542-544
алгоритмы, 547
иллюстрация, 544
матрица ошибок, 544, 545
построение модели, 554-557
при создании профиля клиента, 542-543
процесс, 542-543
тестирование модели, 543-544
что это такое, 542
см. также Интеллектуальная обработка данных; Oracle data mining
Кластеризация, 540-542
алгоритмы, 547-548
в сегментировании рынка, 543
выполнение, 542-543
иллюстрации, 542
что это такое, 540-542
см. также Oracle data mining
Ключей поиск, 341-342
Ключи рабочей системы, 263
Ключи секционирования
для линейного секционирования, 102-103
многостолбцовый, 102-104
что это такое, 90
Ключи хранилища, 31
генерация, 262-264
поиск, 308-309
рабочей системы, 263
суррогатные, 262-264
Комбинированное секционирование, 96-101
иллюстрация, 98
использование, 97
процесс, 98-99
с условием SUBPARTITION TEMPLATE, 100
типы, 97
что это такое, 96
см. также Секционирование
Конечного пользователя, уровень (EUL), 355

Куб
свойства, редактирование, 519
создание, 519
что это такое, 495, 513
см. также метаданные OLAP

Л

Линейная регрессия, анализ, 145
Линейное секционирование, 91-93
в схеме EASYDW, 91-93
иллюстрация, 92
ключи секционирования, 102-104
что это такое, 91
см. также Секционирование
Локальные индексы, 106-109
без префикса, 106-109
разделы, 429
с префиксом, 106-107
типы, 106
что это такое, 106
см. также Секционированные индексы

М

Материализованные представления
ON COMMIT, обновление, 169-170
ON DEMAND, обновление, 170, 178
безопасность, 176-177
включение в переписывании запросов, 170
для репликации данных, 166
заполнение данными, 168-169
и быстрое обновление, 181
индексирование, 175
метод обновления, 169
мониторинг использования, 245-246
несколько, и переписывание запросов, 226-228
обновление, 321-323
определение содержания, 170-171
размеры, оценка, 246
рекомендации, 241
секционирование, 173-174
создание в Oracle Enterprise Manager, 168
создание с условием PREBUILT, 173
создание, 166-175
статистика оптимизатора, 324
устаревшие, поиск, 321
физическое хранение, 167
что это такое, 165, 166
этапы создания, 166
Матрица ошибок, 544, 545
Матричный отчет, 480
Меры
выбор (Presentation Wizard), 526
указание, 518
что это такое, 495, 513
см. также метаданные OLAP
Меры, папки, 513
Метаданные
OLAP, 513-519
OWB, 327
импорт, 305
роль, 21
создание, 292
что это такое, 21
экспорт, 303-304
Многомерность (MOLAP)
и ROLAP, 487-488
продукты, 488
что это такое, 488
см. также Оперативная аналитическая обработка (OLAP)
Многостолбцовые ключи секционирования, 102-104
Многотабличная вставка, 313
Модели
OLAP API, многомерные, 512-513
построение, 554-557
применение, 557-560
Моделирование, 491
E-R, 27-28
пространственное, 28-29
Модули
создание, 329-331
что это такое, 329

Н

Направляемое обучение, 539
Настройка, 439-447
инструменты, 440-441
стабильность плана, 441-447
черная магия, 441
что это такое, 439
Независимые тематические хранилища, 13-14
Неконтролируемое обучение, 539

Ненаправляемое обучение, 539
Нестабильные индексы, 288-290
  перестройка, 290
  поиск, 288-290
нестабильные индексы, **перестройка,** 290
см. также Загрузка данных
Нормализация, **31-32**

О

Обмен разделами, 426
Обновление, **163-164**
  инкрементное, 163
  материализованного представления, 169-170
  опции, 322
  полное, 163
  с помощью Oracle Enterprise Manager, **179-180**
  хранилищ, **434-435**
  что это такое, 163
  см. также Сводные таблицы
Обновление, операция, 177-185
  DBMS_MVIEW, процедуры, 178-180
  ON COMMIT, 169-170
  ON DEMAND, 170, 178
  PCT, **184,** 187
  быстрое, 180-184
  нескольких материализованных представлений, 179
  политика, 178
  что это такое, 177
Обратное соединение, 207-208
  использование, 207-208
  что это **такое,** 207
  см. также Переписывание запросов
Обратный **процентиль,** 146-147
Ограничения, 52-61
  Easy Shopping Inc., 72-75
  null, 57
  аргументы против, 53
  включение без проверки, 289
  внешние, 57
  запись, 59
  запросы к системным таблицам, 59
  и переписывание, 215-217
  нарушения, **обработка,** 288
  определение отношений с помощью, 215
  отключение, 54, 284
  первичные, 215-216
  повторное включение, **283, 286**
  проверка, 58, **74, 77**
  проверка, 59
  просмотр, 60
  создание, 72-75
  ссылочная **целостность,** 279
  что это такое, 52
Онлайновая обработка транзакций **(OLTP),** 7
  автоматизация **процесса,** 7
  в приложениях Oracle, 25
  методы E-R, 9
  оптимальная производительность, 9
Оперативная аналитическая обработка (OLAP), 8-9
  запросы, **18, 487**
  инструменты, 20
  многомерная (MOLAP), 488
  реляционная (ROLAP), **487-488**
  система вычислений, 494
  что это такое, 487
  см. также Oracle 9i OLAP
Оптимизатор на основе правил, **112-113**
Оптимизатор, статистика
  автоматизированный сбор, 437
  для материализованных представлений, 324
  для **таблиц,** 320
  сбор с помощью пакета DBMS_STATS, 438
  сбор, **435-438**
Оптимизация, 79
  запроса, 112-119
  на основе правил, **112-113**
  на основе стоимости, **112**
  соединения, **115-119**
Отношения, 497
Отсечение разделов, 432
Отчетные агрегаты, 130, 140-142
  MAX, 141
  RATIO_TO_REPORT, 142-143
  SUM, **141, 143**
  см. *также* SQL-функции
Отчеты
  Discoverer Plus, 374
  Discoverer Viewer, **469, 470**
  Summary Advisor, 241-243

в рабочей книге, 368
внешний вид, 478
выбор ситуации, 473
выбор стиля, 474
выбор, 484
вычисление итоговых значений, 477
динамические, с графиками, 350
для Web, 482-486
изменение, 376
иллюстрация, 479
интеграция из Discoverer в Oracle 9iAS Portal, 471
источник данных, 474
матричные, 480
настройка, 375
поля, выбор, 477
расписание, 485
распространение, в сети Web, 468-471
с условиями, 379
создание с помощью Presentation Wizard, 524-529
создание с помощью Report Wizard, 472-478
уровни группировки, 476
уровня месяца, 378
условные, 481
формат времени выполнения, 483
формат вывода, 484
форматы, 483
что это такое, 472
ширина столбцов, 478
Очистка данных, 261-262
и объединение/разделение, 262
преобразование для, 307

П

Папки
мер, 513
что это такое, 360
Параллелизм, 155-159
аппаратные требования, 158-159
операции, выигрывающие от применения, 156
уровень, настройка, 157
что это такое, 155
Параллельная загрузка в режиме прямого пути, 283-285
загрузка данных, 284
иллюстрация, 284
отключение ограничений/триггеров, 284
повторное включение ограничений/триггеров, 286
проверка протокола, 285
удаление индексов, 284
что это такое, 283
см. также Режим прямого пути, загрузка
Параллельные запросы, 156
Параллельные управленческие задачи, 438-439
Переменные
и измерения, 501-502
концептуальная схема, 499
что это такое, 498
Перемещение разделов, 432
Перенос данных, 315-320
между таблицами вставкой в режиме прямого пути, 318
с помощью обмена разделами, 316-318
Переписывание запросов, 202-226
включение, 203
вложение агрегатов, 206-207
зависимость от целостности, 221
и grouping sets, 226-231
и PCT, 223-225
и измерения, 212-215
и несколько группировок, 226
и несколько материализованных представлений, 226-228
и ограничения, 215-217
и оптимизатор, 221-222
идентификация запроса, 204
обратное соединение, 207-208
поддержка преобразований, 203
проблемы, выявление, 231-234
режимы целостности, 217-221
с условием GROUP BY, 227-228
совпадение кода SQL, 204-205
фильтрованные данные, 208-210
что это такое, 165
см, также Summary Management
План, стабильность, 441-447
использование сохраненного плана, 443
редактирование сохраненного плана, 444-447
создание сохраненного плана, 442-443
что это такое, 441
Подписка
завершение, 260
обработка, 256
создание, 256-257

Подписчика, представления
подготовка, 259
что это такое, 253
Подъем
вычисление, 544-545
что это такое, 544
см. также Oracle data mining
Полное резервное копирование, 401
Пользовательские агрегаты, 150-152
SalesForecast(), функция, 150-151
в расширяемой основе, 150
интерфейс ODCIAggregate, 150
реализация, 152
что это такое, 150-152
см. также SQL-функции
Пользовательское резервное
копирование, 403-405
Порталы
и хранилища данных, 463
основания для использования, 462-463
пример - Oracle, 463
что это такое, 462
Портлеты, 464
После загрузки, операции, 320-324
обновление материализованного
представления, 321-323
проверка измерения, 320
публикация данных, 325
резервное копирование базы данных, таблицы,
раздела, 324
статистика оптимизатора для материализованных
представлений, 324
статистика оптимизатора для таблиц, 320
Преобразования, 260-265
OWB, 337-338
в Oracle Server, 306-320
в промежуточной области, 264-265
в ходе процесса загрузки, 266
в ходе процесса извлечения, 264
выбор места для осуществления, 264-265
генерация ключей хранилища, 262-264
для образования новых данных, 307
для очистки данных, 307
интеграция данных, 261
очистка данных, 261-262
получение данных, 262
разделение источника данных на несколько
целей, 313-315
с помощью SQL*Loader, 286-287
что это такое, 260
см. также ETL, процесс
Привилегии
ALTER ANY MATERIALIZED VIEW, 177
CREATE MATERIALIZED VIEW, 175-176
Easy Shopping Inc., 77-78
объектные, 66
системные, 79
что это такое, 77-78
Привилегии доступа к объектам, 66
Приложения
аналитические, 489-492
Доступ к аналитическим рабочим
областям в, 509-513
запуск, 396-397
Принятие решений, приложения для, 79
Проверка
данных, 308
компонентов, 344
модели, 344-345
хода выполнения, 345
Прогнозирование, 490
OLAP DML для, 505-507
что это такое, 490
Проекты, 329
Производительность
SQL*Loader, 276-278
быстрое обновление, 184
зависимости, 79
запросов, 439
и секционирование, 91
учет при разработке, 36-37
Промежуточная таблица, 315-320
Пространственное моделирование, 28-29
иллюстрация, 29
что это такое, 28

Р

Рабочая книга, 368, 370
Рабочая нагрузка
ввод, 235, 239
гипотетическая, 243-244

пользовательская, 239-240
фильтрация, 236-237
что это такое, 235
Разделы
добавление/удаление, 424-425
индекс, 427-428
линейное, 431
локальных индексов, 429
облегчают управление, 433
обмен, 426
обслуживание, 424-432
отсечение, 432
перемещение, 432
разрезание, 426, 428-431
слияние, 426-427
списочное, 431
хэш, сращивание, 431
Разрезание разделов, 426, 428-431
иллюстрация, 430
что это такое 426
см. также Секционирование
Рамочные агрегирующие функции, 130, 137-140
Рамочные суммирующие функции, 130, 137-140
Ранжирующие функции, 130-135
CUME_DIST(), 133
NTITLE(), 136-137
PERCENT_RANK(), 133
RANK(), 131, 133
WIDTH_BUCKET(), 136
примеры, 141, 142, 143
что это такое, 130, 130
см. также SQL-функции
Расписание заданий, 390-395
Распределение, 490
Расхождение прогнозов, 549
Режим прямого пути, вставка
перемещение данных между таблицами с помощью, 318
что это такое, 318
Режим прямого пути, загрузка
использование, 276-277
одного раздела, 278-281
параллельная, 283-285
что это такое, 271
см. также SQL*Loader
Резервного копирования, конфигурация, 398-401
выбор каналов, 398
выбор, 401
создание, 398-401
что это такое, 398
Резервное копирование, 35, 397-419
RAM, 409
готовая стратегия, 401-405, 403-405
директория, 398
задание в библиотеке, 405
инкрементное, 405-406
конфигурация, создание, 398-401
обзор процедуры, 403
определение процедуры, 402
перечень содержимого, 415
полное, 401
пользовательская стратегия, 403-405
размеры файлов, 408-410
сохранение на магнитной ленте, RMAN, 400
табличных областей, 407-408
типы, 398
уровень, 405-406
файлы, нуждающиеся в резервном копировании, 415
частота, 400
Рекомендации, 236-238
графики, 236
материализованное представление, 241
просмотр/изменение, 238
реализация, 236-238
см. также Summary Advisor
Реляционная OLAP (ROLAP) и MOLAR, 487-488
что это такое, 487
см. также Оперативная аналитическая обработка (OLAP)
Реорганизация, 423-433
изменения индексов, 434-435
обслуживание разделов, 424-432
причины для, 423-424
Репозиторий, 327
местоположение, 328
табличная область, 328
Роли, 68
EASY_USER, 447
присвоение, 448, 449
создание, 448

C

с иерархией, описанной в объекте-измерении, 207
случаи, 206
см. также Переписывание запросов
С префиксом, локальные индексы, 106-107
С учетом разделов, оптимизация соединения, 116
Сводные таблицы, 65, 162-164
вопросы, 164
иллюстрация, 162
инкрементное обновление, 163
обновленные, 163-164, 177-185
полностью обновленные, 163
существующие и Summary Management, 172-174
устаревшие, 163
что это такое, 161
Связь сущностей, моделирование, 27-28
Сдвигающаяся рамка, операции
добавление/удаление разделов при выполнении, 424-425
что это такое, 424
Секционирование, 62-63, 90-105
и производительность, 91
иллюстрация, 63
комбинированное, 96-101
критерий, 65
линейное, 91-93
материализованные представления, 173-174
методы, 64
по времени, 91
преимущества, 90
процесс, 90-102
списочное, 95-96
хэш, 93-94
что это такое, 62
Секционированные индексы, 104-109
глобальные, 105-106
локальные, 106-109
см. также Индексы
Сжатие, 109-111
атрибут, изменение, 111
иерархического индекса, 109
индекса на основе битовых карт, 109
ключей, 109
сегментов, HO
Симметричные многопроцессорные системы (SMP), 158
Системная статистика, сбор, 438
Системные привилегии, 66-67
Системы с массовым параллелизмом (MPP), 158
Скрипты, Summary Advisor, 241-243
Слияние разделов, 426-427
и разделы индекса, 427-428
иллюстрация, 428
см. также Разделы
События, 386-391
выбор, 389
использование, 386-391
присвоение имени, 387
расписание уведомлений о, 390
создание, 387
уведомление пользователей о, 390
см. также Управление
Соединения
без потерь, 215
вложенных контуров, 115
и Mapping Editor, 341
оптимизация, 115-119
преобразование по типу звезды, 117-118
с учетом разделов, 116
сортировка-слияние, 115
хэш-, 115-116
что это такое (Discoverer), 361
Соединительный индекс на основе битовых карт, 87-90
битовые карты, 87-88
и индекс на основе битовых карт, 89
пример, 87-88, 89
что это такое, 87
см. также Индексы
Соответствия
между источником и целью, 338-339
что это такое, 338-345
см. также Oracle Warehouse Builder
Сортировка-Слияние, соединение, 115
Сохраненные планы, 442-447
использование, 443
исходный, 444
отредактированный, 446

редактирование, 444-447
создание, 442-443
управление, 442
что это такое, 442
см. также План, стабильность
Списочное секционирование, 95-96
иллюстрация, 96
пример, 95-96
что это такое, 95
см. также Секционирование
Среда, настройка (Oracle Discoverer), 355-368
безопасность, 366-368
иерархии, 362-363
классы элементов, 363-364
область бизнес-задач, 355-358
ограничение видимости столбцов, 358-360
папки, 360
сводки, 364-366
свойства элементов, изменение, 360
уровень конечного пользователя, 355
элементы, создание, 361-362
Статистические функции, 145
линейный анализ, 145
что это такое, 130
Стоимостный оптимизатор, 112, 112
и переписывание запросов, 221-222
что это такое, 112
Столбцы
внешние ключи, 117
ограничения видимости, 358-360
число элементов (cardinality), 82
ширина (в отчетах), 478
Суррогатные ключи, 31, 262-264
в таблице фактов, 263
использование, 263
Схемы использования, 21
Схемы, 45-47
Easy Shopping Inc., 70-78
звезда, 32, 35
имя, 46
несколько, 45
объекты, 46
определение, 45

Т

Таблицы
внешние, загрузка с использованием, 291-300
изменение, 252-253
измерение типа родитель-потомок, 497
измерения, 30-31, 51-52
нетранзакционный формат, 551
нормализованные, 198
промежуточные, 315-320
разделы, просмотр, 317
резервное копирование, 324
с хэш-секционированием, 431
сводные, 162-164
соединение при получении данных, 340-341
создание, 72-75
статистика оптимизатора по, 320
транзакционный формат, 551
фактов, 29-30, 29-30, 51-52, 315-320
форматы для интеллектуальной обработки, 551, 552
Табличное пространство, резервное копирование, 407-408
Табличные пространства, 47-50
Easy Shopping Inc., 47, 71-72
изменение в «для чтения и записи», 306
изменение в «только для чтения», 303
имена, 51
репозиторий, 327-328
создание с помощью Oracle Enterprise Manager, 47, 51
создание с помощью SQL, 49
создание, 279, 301, 315
транспортируемые, 300-306
что это такое, 47
Табличные функции, 309-313, 494
OLAP_TABLE, 511-512
TRANSFORM, 311
вызов, 313
объектная технология, 310
что это такое, 309
Тематические хранилища, 13-16
зависимые, 15, 16
и хранилища (warehouse), 32-33
независимые, 13-14
Тестирование
нового программного обеспечения, 455-456
проектной модели, 69
создание системы, 454-455

Транспортируемые табличные пространства, 300-306
изменение табличного пространства на «для чтения/записи», 306
изменение табличного пространства на «только для чтения», 303
иллюстрация, 302
импорт метаданных, 305
использование, 301
перемещение данных, 303
перенос табличного пространства, 304-305
создание табличного пространства, 302
что это такое, 300
экспорт метаданных, 303-304
этапы, 301
см. также Табличные пространства

Триггеры, 252-253
отключение, 279, 284
повторное включение, 283, 286

У

Управление, 381-457
безопасность, 447-450
задачи, 34
задачи, оценка/использование в практике, 456
и OEM, 381-396
и группы, 385-386
и запуск приложений, 396-397
и расписание заданий, 390-395
и события, 386-391
методы, 381
мониторинг использования пространства, 451-454
обзор, 161-247; 457
параллельная обработка, задачи, 438-439
проблемы, 454-457
разделы облегчают, 433
разработка с учетом, 34-36
сохраненный план, 442

Управляющий файл, 267-268
иллюстрация, 270
роль, 267
что это такое, 267
язык определения данных, 267
см. также SQL*Loader

Уровневые измерения, 497-498

Условный отчет, 481

Устаревшие материализованные представления, 187, 321

Устаревшие сводки, 163

Утилита резервного копирования рабочей системы, 397

Ф

Файл данных (SQL*Loader), 269

Файлы данных
Easy Shopping Inc., 71-72
добавление, 47-50
местоположение, 47
указание местоположения, 291-292

Фактов таблицы, 29-30
Easy Shopping Inc., 72-75
выбор, 516
загрузка, 266
и таблицы измерений, 30-31
информация, 29-30
перемещение данных из временной таблицы в, 315-320
секционированные, 315
соединение с измерениями, 517
создание (OWB), 335-336
создание, 51-52
суррогатные ключи в, 263
что это такое, 29-30

Фильтрованные данные, 208-210

Фильтры, 236-237, 240

Финансовые вычисления, 490

Форматирование, управляемое данными, 530-532
иллюстрация, 530
определение правил, 530

Функциональные зависимости, 194, 197

Х

Хранилища данных
автоматическое управление памятью, 119-120
будущее, 24-25
для поиска информации, 8-9
доступность, 23
загрузка данных в, 248-325
и Web, 458-486

и информационные системы электронного бизнеса, 486
и проектирование базы данных, 9-10
и тематические хранилища., 32-33
из рабочих систем, 11-13
индексирование, 80-88
инкрементное построение, 15
история, 6-16
множество тематик, 13
на основе отдельных систем, 11-13
настройка, 439-447
необходимость, 1-2
нормализация, 31-32
обновление, 434-435
объем данных, 22
оптимизация, 79
пользователи/производительность, 23
появление, 6-8
предприятие, 16
проблемы, 20-22
программное обеспечение от Oracle, 461
проектирование, 34
режим реального времени, 24
реорганизация, 423-433
свойства, 79-160
секционирование, 90-105
типы приложений, 24
управление, 20-21, 381-457
что это такое, 1
см. также Средства хранения информации

Хранилища данных в реальном времени, 24

Хранилище данных, разработка, 26-78
безопасность, 66-67
введение, 26-27
генерация/конфигурирование, 345-348
и Oracle Database Configuration Assistant, 39-45
и моделирование взаимосвязи сущностей, 27-28
индексы, 61-62
ключи хранилища, 31
крупный недостаток, 424
материализованные представления, 65
нормализация, 31-32
ограничения, 52-61
повышение производительности, 36-37
пример Easy Shopping, 33-34
проверка, 344-345
проектирование управляемости, 34-36
пространственное моделирование, 28-29
реализация, 37-69, 345
решения по агрегированию, 27
решения по количеству баз данных, 38
секционирование, 62-63
соглашения об именах, 38-39
создание таблицы измерений, 51-52
создание таблицы фактов, 51-52
схемы, 45-47
таблица фактов, 29-30
таблицы измерений, 30-31
тестирование, 69
файлы данных и табличные области, 47-50
что включать в базу, 27

Хэш-секционирование, 93-94
и неравномерность данных, 93
иллюстрация, 94
использование, 94
пример, 94
что это такое, 94
см. также Секционирование

Хэш-соединение, 115-116

Э

Электронная коммерция, информационные системы, 486

Элементы
вычисляемые, создание, 362
классы, 363-364
свойства, изменение, 360-361
создание, 361-362
что это такое, 360

Оглавление

1. Системы хранения информации 5

1.1. Введение 5
1.2. Что такое хранилище информации? 5
1.3. Историческая перспектива 6
1.4. Oracle 9/ 16
1.5. Создание хранилища данных ставит много проблем 20
1.6. Будущее систем хранения информации 24

2. Создание хранилища 26

2.1. Введение 26
2.2. Прочие вопросы дизайна 34
2.3. Реализация схемы 37
2.4. Тестирование проектной модели 69
2.5. Схема для компании Easy Shopping Inc. 70

3. Свойства хранилища данных 79

3.1. Оптимизация хранилища данных 79
3.2. Индексирование 80
3.3. Секционирование данных 90
3.4. Сжатие данных 109
3.5. Техника оптимизации запросов 112
3.6. Автоматическое управление памятью 119
3.7. Функции SQL 121
3.8. Параллелизм 155

4. Управление сводными данными (Summary Management) 161

4.1. Сводные таблицы 161
4.2. Создание материализованного представления 166
4.3. Обновление 177
4.4. Утилита EXPLAIN_MVIEW 188
4.5. Измерения 192
4.6. Переписывание запросов 202
4.7. Summary Advisor 234
4.8. Заключение 247

5. Загрузка данных в хранилище 248

5.1. ETL-процесс 248
5.2. Извлечение данных из рабочих систем 250
5.3. Преобразование данных в общий формат 261
5.4. Загрузка данных в хранилище 265
5.5. Преобразование данных в Oracle Server 306
5.6. Операции, проводимые после загрузки 320
5.7. Использование доступных инструментов для выполнения процесса ETL 325
5.8. Заключение 325

6. Инструменты для работы с хранилищами Oracle 326

6.1. Какие существуют инструменты? 326
6.2. Oracle Warehouse Builder 326
6.3. Oracle Discoverer 349

7. Управление хранилищем 381

7.1. Чем нужно управлять? 381
7.2. Управление с помощью Oracle Enterprise Manager 381
7.3. Резервное копирование и восстановление системы 397

7.4. Восстановление после катастрофы **419**
7.5. Реорганизация хранилища **423**
7.6. Обновление хранилища **434**
7.7. Сбор статистики оптимизатора **435**
7.8. Параллельные управленческие задачи **438**
7.9. Настройка хранилища **439**
7.10. Обеспечение безопасности **447**
7.11. Мониторинг дискового пространства **451**
7.12. Другие проблемы управления **454**
7.13. Заключение **457**

8. Хранилища информации и Web 458

8.1. Общий обзор **458**
8.2. Oracle 9i Application Server **461**
8.3. Oracle 9i Reports **472**
8.4. Хранилища данных и информационные системы для электронной коммерции **486**

9. Oracle 9i OLAP и интеллектуальная обработка информации 487

9.1. Oracle OLAP **487**
9.2. Oracle Data Mining (Интеллектуальная обработка данных) **537**

ИЗДАТЕЛЬСТВО
«КУДИЦ-ОБРАЗ»

Тел.: (095) 333-82-11; ok@kudits.ru, http://books.kudits.ru

ПРИОБРЕТАЙТЕ КНИГИ У НАШИХ ПАРТНЕРОВ

Великий Новгород
ул. Б. Санкт-Петербургская, 44,
(81622) 73-188 доб. 34

Вологда
ООО "Венал" *Оптово-розничная торговля*,
ул. Челюскинцев, д. 9 (8172) 75-21-43

Воронеж
"Книжный мир семьи", пр-т. Революции, 58,
(0732) 51-28-90

Донецк
ЧП Карымов Ратмир Гибадулович,
(10-380-62)381-9232

Екатеринбург
"Книжный мир", ул. 8 Марта, 8г, (3432) 71-18-87
ООО "КДК Дом книги", ул. Блюхера, 51, 359-41-04

Иркутск
"Знание" ул. Ленина, 15, (3952) 24-28-05
"Продалит" ПБОЮЛ Перевозников,
(3952) 23-28-62, 59-13-80, 59-09-90

Калининград
ООО "ПРОМЭКСПОРТИМПОРТ" (0112) 35-37-66

Киев
"Микроника", ул. М. Расковой, 13, (044) 517-73-77
"Технокнига", (044) 268-53-46

Краснодар
"Мир книги", ул. Буденного, 147
"Колос", ул. Красная, 100, (8612) 59-41-32

Минск
Издательство "ТетраСистемс" ул. Железнодорожная,
дом 9, (375-17) 219-73-90, 219-74-01
Книжный магазин books@tut.by, (375-17) 219-73-88
Интернет-магазин http://www.oz.by

Москва
"ОПТИМА+",
(095) 333-65-67, ok@kudits.ru; http://books.kudits.ru

Новосибирск
"Книжный пассаж", ул. Ленина, 10а, (3832) 29-50-30
"Сибирский Дом Книги",
Красный пр-т, 153, (3832) 26-62-39
"Книжный мир", пр-т К. Маркса, 51

Омск
"Книжный Мир", ул. Ленина, 17/19, (3812) 24-32-54

Пермь
"Образование", ул. Солдатова, 37, (3422) 45-96-55
ООО "АКМА" *Оптово-розничная торговля* (3422) 90-93-02,
41-24-60

Ростов-на-Дону
"Мир книги", Ворошиловский пр-т, 33; (8632) 62-54-61
"Деловая литература", (8632) 62-36-55

Самара
"Техническая книга" (8462) 24-29-80

Санкт-Петербург
"Санкт-Петербургский Дом книги"
Невский проспект, 28, (812) 312-01-84
издательство "Наука и Техника", пр. Обуховской Обороны, 107,
(812) 567-70-25, 567-70-26

Саратов
"Книжный Мир"; пр-т Кирова, 32, (8452) 32-98-14

Ставрополь
"Книжный Мир", ул. Мира, 337, (8652) 35-47-90

Таганрог
"Компьютерная книга", ул. Чехова, 31, (8634) 37-13-12

Томск
"Книжный Мир", ул. Ленина, 141, (3822) 51-07-16

Уфа
ООО ПКП "Азия", тел./факс: (3472) 50-39-00
Оптовая торговля Ул. Зенцова, 70
Розничная торговля Магазин "Оазис", ул. Чернышевского, 88
Магазин "Книжник", пр. Октября, 106

Ханты-Мансийск
Магазин "Книги", ул. Ленина, 39

Харьков
Книжный рынок "Райский уголок", ул. Клочковская, 28,
(0572) 549-116

Челябинск
"Книжный Мир", ул. Кирова, 90, (3512) 33-19-58

Ярославль
Магазин "Наука", ул. Володарского, 63, (0852) 25-95-04

ЗАКАЗ КНИГ НАЛОЖЕННЫМ ПЛАТЕЖОМ

Издательство «КУДИЦ-ОБРАЗ» осуществляет рассылку книг по почте.

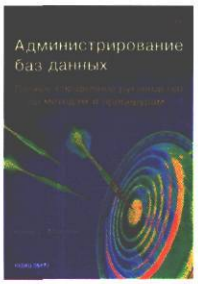
Заказы принимаются по адресу: 121354, Москва, а/я 18; или через Интернет-магазин http://books.kudits.ru.

Заказы из регионов России с авиадоставкой, а также заказы из стран ближнего и дальнего зарубежья обслуживаются только по предварительной оплате.

категория > базы данных

ПРИГЛАШАЕМ
АВТОРОВ КНИГ ПО КОМПЬЮТЕРНОЙ ТЕМАТИКЕ, ПЕРЕВОДЧИКОВ И НАУЧНЫХ РЕДАКТОРОВ

ВОЗМОЖНО, ВАС ЗАИНТЕРЕСУЮТ ДРУГИЕ НАШИ ИЗДАНИЯ



К. С. Маллинс

Администрирование баз данных. Полное справочное руководство по методам и процедурам

ISBN 5-93378-083-9



Д. Шаша, Ф. Бонне

Оптимизация баз данных. Принципы, практика, решение проблем

ISBN 5-93378-096-0



Ш. Шаши, С. Чаула

Основы пространственных баз данных

ISBN 5-93378-091-X



А. Гуруге

Корпоративные порталы на основе XML и Web-служб

ISBN 5-9579-0009-5

По вопросу приобретения книг обращайтесь в издательство **по тел.:** 333-82-11, с **11**[00] до 17[00]. Наш адрес: ул. Профсоюзная, д. 84/32, под. б, эт. 11. ЗВОНИТЕ!

ISBN 5-9579 0036 2



КУДИЦ-ОБРАЗ

Тел./факс: (095) 333-82-11, 333-65-67

E-mail: ok@kudits.ru; http://books.kudits.ru

121354, Москва, а/я 18, "КУДИЦ-ОБРАЗ"

