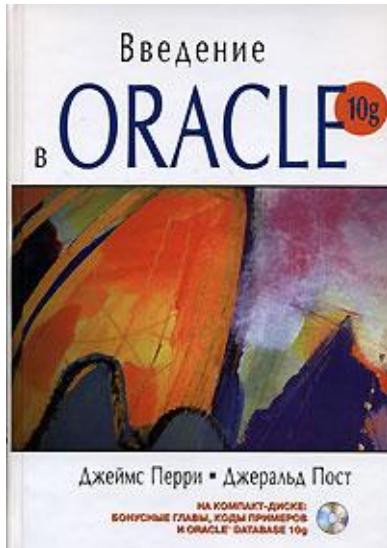


По договору между **издательством "Вильямс"** и Интернет-Магазином "Books.Ru - Книги России" единственный легальный способ получения данного файла  
с книгой **"Введение в Oracle 10g"** (ISBN 5-8459-1113-3) – покупка в Интернет-магазине "Books.Ru - Книги России".

Если вы получили данный файл каким-либо другим образом, вы нарушили законодательство об охране авторского права. Вам необходимо удалить данный файл.



ББК 32.973.26-018.2.75

П27

УДК 681.3.07

Издательский дом “Вильямс”

Зав. редакцией С.Н. Тригуб

Перевод с английского и редакция В.В. Вейтмана и А.В. Назаренко

По общим вопросам обращайтесь в Издательский дом “Вильямс”  
по адресу: [info@williamspublishing.com](mailto:info@williamspublishing.com), <http://www.williamspublishing.com>  
115419, Москва, а/я 783; 03150, Киев, а/я 152

**Перри, Джеймс, Пост, Джеральд.**

X15 П27 Введение в Oracle 10g. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2006. — 704 с. : ил. — Парал. тит. англ.

ISBN 5-8459-1113-3 (рус.)

Данная книга написана в форме самоучителя, который позволит вам овладеть основными навыками работы с Oracle, попрактиковаться в их применении и закрепить полученные знания. Все концепции подробно разбираются и иллюстрируются на примерах. Книга написана доступным языком и хорошо подойдет для людей, не сталкивавшихся ранее с базами данных. В конце книги приводится словарь основных терминов, используемых при работе с любыми базами данных (не только Oracle).

Книга рассчитана на пользователей с любым уровнем подготовки. Может использоваться как учебник при подготовке студентов и аспирантов соответствующих специальностей.

**ББК 32.973.26-018.2.75**

Все названия программных продуктов являются зарегистрированными торговыми марками соответствующих фирм.

Никакая часть настоящего издания ни в каких целях не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами, будь то электронные или механические, включая фотокопирование и запись на магнитный носитель, если на это нет письменного разрешения издательства Prentice Hall, Inc.

Authorized translation from the English language edition published by Prentice Hall, Copyright © 2006 by Pearson Education, Inc., Upper Saddle River, New Jersey, 07458.

Pearson Prentice Hall. All rights reserved. This publication is protected by Copyright and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permission(s), write to: Rights and Permissions Department.

Pearson Prentice Hall™ is a trademark of Pearson Education, Inc.

Pearson ® is a registered trademark of Pearson plc

Prentice Hall ® is a registered trademark of Pearson Education, Inc.

Russian language edition published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2006

ISBN 5-8459-1113-3 (рус.)

ISBN 0-13-195740-6 (англ.)

© Издательский дом “Вильямс”, 2006

© by Pearson Education, Inc, 2006

# Оглавление

<b>Предисловие</b>	<b>21</b>
<b>1. Введение в СУРБД и Oracle 10g</b>	<b>31</b>
<b>2. Обзор SQL и SQL*Plus</b>	<b>95</b>
<b>3. Создание, модификация, переименование и удаление таблиц базы данных</b>	<b>153</b>
<b>4. Изменение данных и наблюдение за таблицей</b>	<b>225</b>
<b>5. Организация запросов к базе данных</b>	<b>305</b>
<b>6. Создание запросов и представлений на основе нескольких таблиц</b>	<b>393</b>
<b>7. Использование PL/SQL</b>	<b>471</b>
<b>8. Использование Forms Builder</b>	<b>545</b>
<b>9. Специальная настройка форм</b>	<b>599</b>
<b>10. Создание и модификация отчетов</b>	<b>CD1</b>
<b>11. Создание интегрированных приложений</b>	<b>CD73</b>
<b>12. Обеспечение защиты базы данных</b>	<b>CD123</b>
<b>13. Администрирование баз данных</b>	<b>CD171</b>
<b>Словарь терминов</b>	<b>675</b>
<b>Предметный указатель</b>	<b>693</b>



# Содержание

Студентам	21
Преподавателям	25
Подход, использованный в книге, и ее основные особенности	25
В помощь студентам и преподавателям ( <a href="http://www.prenhall.com/perrypost">www.prenhall.com/perrypost</a> )	27
Варианты чтения книги	27
Совет по использованию Oracle	28
Благодарности	28
От издательства	29
Благодарности издательства	30
Прочтите перед началом работы	30
<b>1. Введение в СУРБД и Oracle 10g</b>	<b>31</b>
Системы управления базами данных	31
Персональные системы	32
Системы на основе сервера	33
Описание систем в виде объектов и взаимоотношений	35
Модель реляционной базы данных	38
Первичные ключи	38
Нормальные формы	41
Соотношения и внешние ключи	47
Объектно-реляционная модель	48
Установка инструментов Oracle	51
Установка системы управления базой данных Oracle	53
Удаление системы управления базой данных Oracle	56
Установка инструментов Developer Suite	57
Корпоративные службы форм и отчетов	61
Изучение среды Oracle	64
SQL*Plus и iSQL*Plus	64
Инструменты Developer Suite	68
Oracle Enterprise Manager	69
Базы данных, используемые в книге	72
Агентство Redwood Realty	74
Coffee Merchant	77
Rowing Ventures	79
Broadcloth Clothing	83
Повторение пройденного материала	90

<b>2. Обзор SQL и SQL*Plus</b>	<b>95</b>
Введение	95
Типы команд SQL	96
“Анатомия” выражения SQL	97
Интерактивная помощь	99
Взаимодействие с базой данных	100
Вызов SQL*Plus из командной строки DOS	102
Вход в Oracle	102
Ввод и запуск команд SQL	103
Редактирование команд SQL	104
Использование SQL*Plus for Windows	105
Вход в Oracle	105
Ввод и запуск команд SQL	106
Редактирование команд SQL	107
Выход из SQL*Plus	110
Использование iSQL*Plus	110
Вход в Oracle	110
Ввод и запуск команд SQL	111
Редактирование команд SQL	113
Выход из iSQL*Plus	113
Выражения SQL	113
Запуск запросов SQL	115
DLL-выражения	116
DML-выражения	119
TC-выражения	125
DCL-выражения	128
Использование расширений команд SQL*Plus	130
Описание структуры таблицы	131
Захват файлов для печати	132
Команды форматирования	133
Настройка среды SQL*Plus	136
Выполнение файлов сценария	137
Использование переменных	139
Обзор других команд SQL*Plus	140
Создание и запуск сценария	141
Истина или ложь?	145
Заполнить пропущенное	145
Упражнения	147

Redwood Realty	147
Coffee Merchant	149
Rowing Ventures	150
Broadcloth Clothing	152
<b>3. Создание, модификация, переименование и удаление таблиц базы данных</b>	<b>153</b>
Разработка реляционных баз данных	153
Общение с пользователем базы данных	154
Определение требований пользователя	154
Определение бизнес-объектов	155
Нормализация структуры	156
Учетные записи пользователей Oracle	158
Дальнейшие указания для пользователей персональных систем Oracle	159
Создание учетной записи пользователя	159
Вход в базу данных с использованием другого имени пользователя и пароля	160
Изменение системных привилегий пользователя	160
Изменение паролей других пользователей	162
Изменение собственного пароля	163
Удаление учетной записи пользователя	164
Создание таблиц	164
Типы данных Oracle	166
Создание таблицы с использованием SQL*Plus	173
Добавление комментариев к таблицам и столбцам	176
Определение и использование ограничений	178
Условия	178
Именование условий	179
Определение условий	180
Создание условий на столбцы и таблицы	186
Присвоение столбцам значений по умолчанию	188
Изменение таблицы и ее условий	190
Добавление, включение или выключение условий	191
Удаление или переименование условий	193
Изменение заданного значения по умолчанию или типа данных	194
Добавление, удаление и переименование столбцов	195
Отметка столбцов как неиспользуемых и удаление их	197
Отображение имен, структур и комментариев таблиц	198
Просмотр собственных таблиц базы данных	200
Просмотр информации о столбцах	203

Просмотр условий на таблицы и столбцы	204
Просмотр комментариев к таблицам и столбцам	207
<b>Удаление, восстановление и переименование таблиц</b>	<b>209</b>
Удаление таблицы	210
Восстановление удаленной таблицы	211
Очистка корзины	212
Переименование таблиц	212
<b>Создание новых таблиц на основе существующих</b>	<b>213</b>
Резюме	214
Основные термины	215
Повторение пройденного материала	216
Упражнения	218
<b>4. Изменение данных и наблюдение за таблицей</b>	<b>225</b>
Вставка строк в таблицы	225
Задание списка столбцов	230
Условия целостности	230
Пропуск списка столбцов	232
Вставка даты и времени	234
Вставка данных из других таблиц базы данных	238
Создание и использование последовательностей	240
Обновление данных	245
Команда обновления UPDATE	246
Введение в структуру CASE	251
Обновление данных с использованием структуры CASE	252
Подстановочные переменные	254
Удаление строк и усечение таблиц	259
Удаление выбранных строк	260
Удаление всех строк	262
Слияние строк	263
Транзакции базы данных	267
Фиксация	268
Откат	269
Точки сохранения	270
Создание и использование обработчиков событий	273
Введение в обработчики событий	273
Создание и использование обработчика событий BEFORE	276
Создание обработчика событий AFTER, наблюдающего за операциями с таблицей	280
Создание и использование обработчиков событий на уровне выражений	285

Просмотр, изменение и удаление обработчиков событий	288
Резюме	291
Повторение пройденного материала	294
Упражнения	295

---

**5. Организация запросов к базе данных** 305

Отображение информации из одной таблицы базы данных	305
Написание выражений SELECT	306
Выбор всех столбцов	309
Использование оператора DISTINCT	
для отображения уникальных строк	311
Прерывание выполняемого запроса	312
Использование условий поиска для фильтрации результатов	313
Сортировка	321
Включение расчетов в запросы	326
Введение в функции SQL	334
Использование в запросах односторонних функций	335
Использование агрегирующих функций	358
Группировка результатов	366
Фильтрация групп с помощью оператора HAVING	368
Форматирование выхода SQL*Plus и создание простых отчетов	370
Резюме	382
Повторение пройденного материала	385
Упражнения	387

---

**6. Создание запросов и представлений на основе нескольких таблиц** 393

Создание и использование представлений на основе нескольких таблиц	394
Объединение таблиц, имеющих совпадающие значения столбцов	395
Другие типы объединений и условий объединения	404
Операторы действий над множествами	418
Использование подзапросов	424
Создание и использование представлений	441
Зачем нужны представления?	441
Определение и организация запросов	
к однотабличным представлениям	443
Модификация данных с использованием однотабличного представления	447
Создание сложных представлений	449
Создание синонимов для упрощения доступа к таблице	455

Перечисление определений представлений	455
Удаление представлений	457
Резюме	459
Повторение пройденного материала	461
Упражнения	463
4. Broadcloth Clothing	469
<b>7. Использование PL/SQL</b>	<b>471</b>
Общие сведения о PL/SQL	471
Преимущества PL/SQL	472
Типы блоков PL/SQL	473
Неименованные блоки	474
Структура раздела деклараций	475
Раздел исполняемого кода	476
Раздел обработки исключений	478
Создание неименованных блоков	479
Инициализация базы данных Redwood Realty	479
Написание кода для анонимных блоков	480
Настройка среды SQL*Plus	483
Выполнение неименованного блока PL/SQL	484
Модификация неименованного блока	
для отображения нескольких строк	485
Обработка исключений	487
Явные курсоры	490
Итерации в блоках PL/SQL	490
Обработка строк с применением явных курсоров и циклов	493
Использование цикла FOR для работы с явным курсором	498
Выражение IF	501
Именованные блоки	507
Создание, использование и удаление функций	509
Создание функций и сохранение их в базе	509
Вызов функции	513
Модификация функции	515
Получение информации о функции и удаление функции	517
Создание, использование и удаление процедур	520
Преимущества процедур	520
Создание процедур и сохранение их в базе	521
Вызов хранимой процедуры	525
Модификация, повторная компиляция и сохранение процедуры	526
Получение информации о процедуре и удаление процедуры	528
Резюме	531

Основные термины	532
Повторение пройденного материала	533
Упражнения	535

## **8. Использование Forms Builder**

---

Общие сведения о формах	545
Основные типы форм	546
Архитектура среды для поддержки форм	549
Структура формы Oracle	551
Использование инструментальных средств	553
Проверка формы	559
Получение данных с помощью формы	562
Модификация форм	563
Редактор компоновки	564
Объектный навигатор	565
Основные свойства элементов	566
Включение изображений в состав формы	569
Добавление средств поиска	572
Создание списка значений	575
Автоматическое выполнение запроса	577
Переключатели и флажки опций	579
Создание табличных и подчиненных форм	582
Создание табличных форм	582
Создание основной и подчиненной форм	584
Изменение внешнего вида таблицы	588
Установка маски формата	590
Резюме	591
Основные термины	592
Повторение пройденного материала	593
Упражнения	595

## **9. Специальная настройка форм**

---

Создание холста и простого блока данных	602
Создание блока данных для запроса	605
Обеспечение поиска	608
Создание обработчиков событий	611
Определение событий, связанных с формой	612
Создание и редактирование обработчиков	616
Отладка обработчиков событий	619
Обработка ошибок	623
Область видимости и время жизни	627

Создание инструментальных средств	629
Формирование последовательностей	629
Проверка вводимых данных	632
Вычисление итоговых значений для табличных данных	635
Использование нескольких холстов	639
Добавление холста	639
Холсты в виде слоев	641
Холсты в виде вкладок	644
Выбор стиля для формы	647
Создание Web-форм с помощью JDeveloper	649
Модификация таблицы и настройка файлов на сервере	650
Соединение с данными и создание рабочей области	651
Создание модели данных бизнес-компонентов	653
Создание документа JSP для отображения данных	657
Добавление поисковой формы	660
Резюме	665
Основные термины	667
Повторение пройденного материала	668
Упражнения	670

**10. Создание и модификация отчетов****CD1**

Общие сведения об отчетах	CD1
Публикации в Web и бумажные копии отчетов	CD2
Типы отчетов	CD2
Служба отчетов Oracle	CD5
Создание и модификация отчетов	CD7
Структура отчета	CD7
Reports Builder	CD9
Окна Paper Design и Paper Layout	CD13
Окно объектного навигатора	CD17
Доработка отчета	CD20
Выравнивание столбцов	CD22
Установка маски формата и свойств	CD23
Добавление теней и обрамления	CD24
Включение номеров страниц и даты	CD25
Добавление текстовых меток	CD26
Специальная настройка отчетов	CD28
Модель данных	CD29
Добавление полей	CD32
Группы данных и фреймы	CD34
Создание отчета вручную	CD36
Представление изображений из базы данных	CD41

Управление данными в отчете	CD43
Создание пользовательских параметров	CD44
Использование фильтров для ограничения объема отчетов	CD46
Создание обработчиков событий для отчета	CD48
Создание пользовательских шаблонов	CD49
Проектирование пользовательских шаблонов	CD50
Применение шаблонов	CD53
Регистрация шаблонов	CD53
Настройка отчета для публикации в Web	CD55
Среда для динамических отчетов	CD57
Модификация Web-отчета	CD57
Добавление диаграмм к Web-отчетам	CD62
Резюме	CD65
Основные термины	CD66
Повторение пройденного материала	CD67
Упражнения	CD68

## **11. Создание интегрированных приложений**

---

Разработка и интеграция приложений, работающих с базами данных	CD73
Приложение для Redwood Realty	CD74
Обеспечение согласованного внешнего вида	CD76
Создание шаблонов для форм	CD77
Классы свойств и визуальные атрибуты	CD79
Применение шаблонов форм и свойств	CD82
Включение форм и отчетов в приложение	CD85
Исходная форма	CD87
Открытие связанных форм	CD88
Отображение отчетов	CD92
Доставка форм и отчетов на сервер приложений Oracle	CD97
Создание меню	CD100
Назначение меню	CD100
Формирование меню	CD101
Связывание действий с меню	CD103
Доставка и использование меню	CD106
Создание файлов со справочной информацией	CD108
Справочная Web-система Oracle	CD110
Создание справочных HTML-файлов	CD110
Доставка и использование файлов со справочной информацией	CD115
Резюме	CD117
Основные термины	CD118
Повторение пройденного материала	CD118

Упражнения <b>12. Обеспечение защиты базы данных</b> Создание и редактирование учетных записей Аутентификация пользователей Пользовательские роли Системные и объектные привилегии Контроль доступа пользователей к объектам Создание ролей Создание и отмена полномочий Назначение привилегий посредством представлений и процедур Ограничение доступа конкретными строками и столбцами Использование процедур для ограничения возможностей обновления данных Виртуальная частная база данных и гриф секретности Ограничение доступа средствами приложения Защита данных посредством шифрования Защита данных при передаче по глобальной сети Шифрование отдельных данных в базе Защита исходного кода с помощью утилиты wrap Отслеживание действий с базой данных Включение режима мониторинга Просмотр результатов мониторинга Создание обработчиков для мониторинга Детальный мониторинг Резюме Основные термины Повторение пройденного материала Упражнения	CD120 <b>CD123</b> CD123 CD125 CD126 CD129 CD133 CD134 CD136 CD140 CD142 CD143 CD146 CD147 CD150 CD151 CD152 CD155 CD155 CD157 CD159 CD161 CD163 CD163 CD164 CD164 CD167 <b>CD171</b> CD171 Cd173 CD178 CD179 CD180 CD186 CD187 CD187 CD187
<b>13. Администрирование баз данных</b>	
Основные задачи администратора базы данных Использование Enterprise Manager Хранение данных Защита управляющих файлов Создание табличных пространств и файлов данных Настройка операций Undo и Redo Формирование пространства для объектов схемы Установка параметров, определяющих особенности хранения таблиц	CD171 Cd173 CD178 CD179 CD180 CD186 CD187 CD187 CD187

Определение кластеров	CD189
Создание разделов	CD191
Импортирование и экспортование данных	CD193
Использование SQL-сценариев	CD194
Использование Data Pump	CD194
Использование SQL*Loader и внешних таблиц	CD195
Обслуживание СУБД	CD199
Обновления и дополнения	CD199
Запуск и остановка СУБД	CD201
Создание резервных копий баз данных	CD204
Проблемы при создании резервных копий	CD205
Отключение СУБД и создание резервной копии (“холодный” режим)	CD207
Постоянное резервное копирование и архивация (“горячий” режим)	CD209
Повышение производительности базы данных	CD210
Инструменты мониторинга	CD211
Оптимизация запросов	CD217
Получение информации из системных представлений	CD226
Резюме	CD227
Основные термины	CD230
Повторение пройденного материала	CD231
Упражнения	CD233
<b>Словарь терминов</b>	<b>675</b>
<b>Предметный указатель</b>	<b>693</b>



**Кэлли Карлинг и Тобиасу Карлингу —**

дочери и зятю Джима

**Cape Пост —**

жене Джерри

# Об авторах

**Джеймс Т. Перри** читает курс по информационным системам в Школе бизнеса университета Сан-Диего. Степень кандидата наук по информатике Джеймс получил в Пенсильванском университете, а степень бакалавра по математике — в университете Пердью. Он является одним из авторов более восьмидесяти учебников, посвященных работе с системами управления базами данных, Интернету, бухгалтерским системам, Microsoft Office и электронной коммерции. Некоторые из его популярных книг были переведены на китайский, датский, французский, корейский и русский языки. Джеймс имеет тридцать лет преподавательской практики на факультете информационных систем в университете Сан-Диего и факультете информации университета штата Небраска. Он был научным консультантом во многих частных и государственных проектах, включая проект Стратегической оборонной инициативы (“Звездные войны”), к нему также часто обращаются как к специалисту по компьютерной безопасности. Во время, свободное от написания книг и преподавания, Джеймс судит гребные регаты в штате Калифорния.

**Джеральд В. Пост** является профессором Тихоокеанского университета и ведет курс по административным информационным системам. Он имеет степень кандидата наук по экономике и статистике (университет штата Айова), а также степень бакалавра по математике (университет штата Висконсин в О’Клер). Джеральд является автором учебников по управлению базами данных и административным информационным системам, а также напечатал несколько десятков профессиональных работ в таких журналах, как *Communication of the ACM*, *Management Information Systems Quarterly* и *Decision Sciences*. Более двадцати лет он ведет курсы по административным информационным системам и базам данных, является автором нескольких приложений баз данных для коммерческих систем. Во время, свободное от написания книг, любит гонять на мотоцикле или охотиться с собакой.

# Предисловие

## Студентам

В книге рассказывается, как использовать систему управления базами данных Oracle 10g.

Открывает книгу глава 1, в которой приведена основная терминология, объясняется разница между персональными системами управления базами данных (Database Management System — DBMS, СУБД) и системами клиент/сервер, подобными Oracle, рассказывается, что такое сущности и связи, нормализация базы данных и первичные ключи. Затем вы узнаете об установке Oracle в своей системе и сможете получить первое представление о среде Oracle, изучив интерфейс SQL\*Plus, набор инструментов Developer Suite и средство Oracle Enterprise Manager. Наконец, мы очертиим структуру и содержание четырех баз данных, используемых в книге в качестве сквозных примеров: Redwood Realty, Coffee Merchant, Rowing Ventures и Broadcloth Clothing.

В главе 2 описан язык баз данных SQL и интерфейсы базы данных Oracle, включая командную строку операционной системы, SQL\*Plus for Windows и iSQL\*Plus. Руководствуясь пошаговыми инструкциями, вы узнаете, как запустить файл сценария, создающий базу данных Redwood Realty, а затем напишете и выполните выражения SQL в средах SQL\*Plus и iSQL\*Plus. Вы узнаете, как с помощью выражений SQL\*Plus (являющихся расширениями языка SQL\*Plus) описать структуру таблицы, записать результаты для последующей печати и настроить по своему усмотрению среду SQL\*Plus.

В главе 3 речь пойдет о том, как создавать таблицы базы данных, в которых будет содержаться информация. Помимо этого, подробно рассмотрены типы данных Oracle, включая символьные и числовые типы, а также типы, используемые для записи даты и изображений. Вы узнаете, как задавать условия (или ограничения) на столбцы таблицы, в частности, определять первичные и внешние ключи, условия уникальности и проверки (задающие диапазон допустимых значений столбца). Из этой главы вы узнаете, как перечислять имена таблиц и столбцов, а также условия, наложенные на таблицы и столбцы; также будет рассказано, как удалять и переименовывать таблицы. Глава завершается примером, в котором демонстрируется создание новой таблицы на основе уже существующей.

В главе 4 обсуждаются SQL-выражения языка манипулирования данными (Data Manipulation Language — DML), транзакции базы данных и обработчики событий. Руководствуясь пошаговыми инструкциями и уже приобретенным опытом, вы на-

учитесь вставлять новые строки в существующие таблицы, создавать и использовать последовательности Oracle для автоматической генерации значений столбца первичного ключа. Вы узнаете, как модифицировать данные с помощью выражения SQL UPDATE, и попрактикуетесь в удалении строк посредством команды DELETE. В данной главе также подробно описываются транзакции базы данных и связанные с ними выражения COMMIT и ROLLBACK. Вы узнаете, как с помощью транзакций отменять ошибочно выполненные DML-выражения. Наконец, в данной главе описаны обработчики событий (trigger) базы данных. Вы узнаете, как с их помощью перехватывать значения столбцов, которые предполагается изменить, и регистрировать изменения в файле, отслеживая изменения выбранных таблиц и столбцов.

В главе 5 вводится выражение SELECT — команда SQL первостепенной важности, извлекающая информацию из таблиц базы данных и описывающая, как следует извлекать эту информацию. Вы узнаете об операторах выражения SELECT, включая FROM, WHERE и ORDER BY, а также об операторах SQL. В данной главе рассматриваются такие важнейшие понятия, как расчет истекшего времени, использование математических операторов для выполнения численных расчетов и псевдонимы столбцов. Здесь описывается большинство из наиболее часто используемых функций Oracle: символьные и числовые функции, функции даты, функции преобразования, специальные функции, использующиеся для конвертации значений NULL в другие значения, а также обобщенные функции. Глава завершается подробным обсуждением форматирования выхода SQL\*Plus и команд создания отчетов.

В главе 6 продолжается обсуждение выражения SELECT и показывается, как обращаться в одном запросе к нескольким таблицам. Вы узнаете о типах и условиях объединения, включая объединения по эквивалентности, левые, правые и самообъединения, а также декартово произведение. Дополнительными инструментами извлечения информации служат операторы множеств UNION, MINUS и INTERSECT, описание которых также приводится в данной главе. Кроме того, вы изучите представления базы данных и узнаете, как создавать представления и использовать их для собственных потребностей. Далее подробно обсуждаются потенциальные проблемы, связанные с вставкой, обновлением и удалением данных из представлений. Вы узнаете о некоторых важных представлениях словаря данных Oracle, предоставляющих информацию о ваших таблицах и других принадлежащих вам ресурсах.

В главе 7 речь пойдет о программировании. Вы узнаете, как писать анонимные и именованные блоки кода PL/SQL, использовать явные курсоры для обработки результатов, занимающих несколько строк. С помощью хорошо спланированных действий вы пройдете через создание, использование и перечислений функций PL/SQL и процедур PL/SQL. Помимо этого, здесь описаны преимущества функций и процедур по сравнению с использованием анонимных блоков.

В главе 8 вводится концепция форм базы данных. Формы широко используются при написании приложений. Пользователи взаимодействуют с вашей базой данных посредством форм. Новые данные, вводимые в формы, переносятся в таблицы базы данных. Данная глава посвящена средству Forms Builder. Вы научитесь использовать автоматизированные инструменты для создания трех основных типов форм: основных, табличных и подчиненных.

В главе 9 рассказывается, как пойти дальше автоматизированных возможностей Forms Builder. Изучив создание форм с нуля, вы получите более полный контроль над внешним видом формы и узнаете о таких полезных возможностях, как присоединение к форме процедур, реагирующих на события, и расчет промежуточных сумм и выполнения других вычислений в форме. Вы узнаете об обработчиках событий и научитесь создавать формы с несколькими холстами, позволяющие отображать большой объем данных и в то же время не перегружать пользователя информацией.

В главе 10 рассмотрен еще один важный аспект приложений: создание отчетов. Отчеты создаются для того, чтобы отображать на экране сводные данные. Изучив данную главу, вы научитесь создавать отчеты с помощью соответствующего мастера. После этого вы сможете модифицировать и настраивать отчет и добавлять в него все новые и новые возможности. Вы узнаете, как контролировать общую структуру отчета, различные промежуточные результаты и форматирование. Вы увидите, что в отчеты сравнительно просто добавляются графики и внедряются интерактивные фильтры, позволяющие пользователям отбирать данные, которые они желают видеть. Кроме того, вы увидите, как создать и развернуть отчет так, чтобы его можно было просматривать через Интернет с помощью обычного Web-браузера.

В главе 11 разрозненные элементы собираются в единое целое и показывается, как создать интегрированное приложение. Пользователи не должны получать информацию в виде таблиц или SQL-кода. Фактически они могут даже не знать, что за приложением скрывается база данных. Все, что они должны видеть, — это формы и отчеты. В связи с этим ключевым этапом написания приложения является стандартизация форм и отчетов с использованием шаблонов. После этого вы должны использовать обработчики для организации связи между формами и отчетами и скрытия всех деталей от пользователей. Вы можете создать специальные меню и даже написать собственную справочную систему. Закончив с этим, можете доставить приложение на Oracle Application Server (если вы имеете к нему доступ).

В главе 12 рассматривается ряд основных концепций, требуемых для поддержания безопасности. Чтобы приложение с базой данных было безопасным, разработчики и администраторы должны подумать об этом заблаговременно. Из представленного в главе материала вы узнаете, как создавать учетные записи пользователей и с помощью ролей присваивать привилегии, касающиеся безопасности. Кроме того, вы можете ограничить доступ к данным, используя средства управления представлениями и приложением. Вы узнаете, как зашифровать записи базы данных на случай, если

кто-то сможет украсть всю базу данных. Кроме того, в главе объясняется, как настраивать систему наблюдения и пользоваться ею, чтобы узнать, если кто-то попытается атаковать вашу базу данных.

Глава 13 содержит общие сведения об администрировании базы данных. Глава начинается с объяснения различных возможностей средства Enterprise Manager. Данный инструмент служит “мостиком” ко множеству задач АБД. Кроме того, поскольку большинство администраторов баз данных используют команды SQL, в данной главе показано, как с помощью этих команд решаются основные задачи конфигурирования. Если вы желаете запустить собственную копию Oracle, вам в конечном счете придется столкнуться с этими задачами. И даже если вы не имеете персональной базы данных, вам стоит прочесть эту главу, ознакомившись с обязанностями администратора, поскольку это поможет вам решить, подходит ли для вас эта роль. Вы узнаете, как конфигурировать файлы хранилища, поддерживать и обновлять программное обеспечение базы данных. Кроме того, вы узнаете, как экспорттировать и импортировать данные в базу данных и из нее, а также восстанавливать базы после сбоев. В данной главе также описаны инструменты наблюдения и оптимизации производительности базы данных.

Мы надеемся, что при чтении книги вы будете активно участвовать в процессе обучения. Возможно, важнейшими инструментами обучения и запоминания прочитанного являются пошаговые примеры. Придерживаясь подробнейших инструкций, вы будете обучаться методом проб и ошибок. Не имеет значения, что вы сделаете с базой данных в какой-либо главе, — это никак не повлияет на использование этой базы в других главах. Предполагая возможность неумышленного повреждения, база данных обезопасит себя — в начале каждой главы вы заново загружаете либо всю базу данных, либо ту ее часть, изучение которой предусматривается материалом главы. Таким образом, вы всегда начинаете работу с новой (и правильной!) базы данных.

Каждая глава завершается набором вопросов для повторения и закрепления пройденного материала. В вопросы, касающиеся терминологии, включены фразы, истинность/ложность которых вам необходимо определить, фразы, в которые необходимо вставить пропущенные слова, а также вопросы, предполагающие выбор одного или нескольких вариантов ответа. После этих вопросов рассматриваются четыре практических упражнения, предполагающих использование Oracle для решения поставленной задачи. Данные упражнения являются “сквозными”, т.е. они проходят по всем главам. Первое из них касается расширения темы, рассмотренной в главе, и добавления изученных функций в базу данных агентства недвижимости Redwood Realty. Здесь приводятся подробнейшие указания по каждому выполняемому действию. Второе упражнение касается базы данных Coffee Merchant — компании оптовой и розничной торговли кофе. Как и в предыдущем упражнении, вам будут предоставлены подробные указания по решению поставленной задачи. Третье упражнение связано с базой данных компании-строителя гребных регат. Эта база состоит из нескольки-

ких таблиц и других объектов базы данных, необходимых для хранения информации о гонке, ее участниках и выставивших их организациях. Соответствующий раздел упражнений озаглавлен “Rowing Ventures”, в нем приводится краткое описание таблиц базы данных и ее составляющих, необходимых для решения поставленной задачи (тем не менее, решение не расписывается подробно, так что вы можете поступать так, как считаете нужным). Четвертое упражнение связано с базой данных компании Broadcloth Clothing — международной организации, занимающейся пошивом одежды и ее продажей (причем заводы по производству и магазины по продаже одежды разбросаны по всему миру). Данная задача особенно интересна, поскольку связана с иностранными валютами, графиками производства и несколькими другими вопросами, возникающими при организации базы данных глобальной компании. В данном упражнении вы также должны разработать свой вариант решения проблемы, однако часто эта проблема решается проще, чем три остальные.

Внимательно читая текст, старательно следя пошаговым инструкциям и используя материал в конце главы для закрепления пройденного, вы узнаете, как использовать программное обеспечение системы баз данных Oracle для создания и поддержания разумно большого числа таблиц большого размера. Изучение книги — первый шаг на пути к овладению системами баз данных вообще и Oracle в частности.

---

---

## Преподавателям

### Подход, использованный в книге, и ее основные особенности

В данной книге изложение теории баз данных дополняется рассмотрением приложений, иллюстрирующих эту теорию. Читатели узнают, какие типичные действия связаны с работой базы данных и какие команды для этого применяются. После этого приводится подробное пошаговое руководство по выполнению предложенного задания. Например, в главе 4 подробно описываются обработчики событий, рассказывается, почему и когда они должны применяться. После этого читателю предлагается выполнить определенную последовательность действий и создать обработчик событий для указанной таблицы. Затем описывается, в каких случаях может потребоваться наблюдение за модификацией базы данных и регистрация соответствующей информации в контрольной таблице. После этого читателю предлагается выполнить другой набор действий, обновляющий таблицу, для которой определен обработчик событий. Наконец, приводится последовательность действий, отображающая на экране строки контрольной таблицы, содержащие информацию, собранную и записанную созданным ранее обработчиком событий.

Важные термины при первом появлении в тексте выделены *курсивом*. Поскольку терминология вообще является важным компонентом процесса обучения (в частно-

сти, при изучении СУБД), в конце главы приведен список терминов. После резюме главы следует раздел повторения пройденного материала, где необходимо определить истинность или ложность высказывания, заполнить пропущенные фрагменты текста, а также выбрать правильный вариант ответа из нескольких предложенных, что поможет лучше понять и запомнить ключевые концепции, рассмотренные в главе.

Всего в книге рассматриваются четыре базы данных. Одна из них, Redwood Realty, используется в основном тексте всех глав для иллюстрации рассматриваемых концепций. Расширению этой базы данных посвящено первое упражнение в конце главы, где для решения поставленной задачи читателю предлагается выполнить описанную последовательность, закрепляющую понимание понятий, введенных в главе. В данном упражнении читателю предоставляется уже знакомая по главе база данных и необходимый минимум дополнительных сведений, ускоряющий решение задачи.

Остальные три базы данных рассматриваются в упражнениях в конце главы. Компания Coffee Merchant занимается оптовой и розничной торговлей кофе; ее база данных состоит из семи таблиц. Некоторые таблицы Coffee Merchants содержат несколько тысяч строк, что позволяет попрактиковаться работать с нетривиальными таблицами. Таблица *Consumers*, например, содержит 1 587 строк, заполненных контактной информацией. Таблица *Inventory* содержит подробное описание сортов чая и кофе, распространенных по всему миру (от гватемальского кофе Huehuetenango до российского чая Caravan). Столбец *Description* таблицы *Inventory* содержит подробное описание характеристик чая и кофе (иногда это более 100 слов).

Компания Rowing Ventures является типичным устроителем спортивных командных соревнований, с выставлением участников от организаций, различными трассами и конфигурациями лодок. Соответствующая база данных содержит шесть таблиц с небольшим числом строк. На ее примере вы можете последовательно практиковаться в использовании инструментов и методов, типичных для всех баз данных. Упражнения, касающиеся этой базы данных, содержат подробную постановку задачи, но не имеют пошаговых указаний — решение вам предстоит найти самостоятельно.

Суть четвертого упражнения заключается в поддержании базы данных международной компании Broadcloth Clothing, сталкивающейся со множеством задач, типичных для большой международной организации. В частности, вам придется столкнуться с использованием различных языков и валют. Всего база данных Broadcloth Clothing содержит семнадцать таблиц, включая уже упоминавшуюся таблицу *Customer* с контактной информацией о клиентах; таблицу *Factory*, в которой перечисляются фабрики, производящие одежду; и *Compliance*, в которой приводится информация, касающаяся соответствия фабрик нормам безопасности. Подобно примеру с Rowing Ventures, данное упражнение содержит формулировку задания, выполнить которое предлагается читателю (пошаговые инструкции отсутствуют).

## В помощь студентам и преподавателям ([www.prenhall.com/perrypost](http://www.prenhall.com/perrypost))

Предлагаемый учебник одинаково хорошо подходит как для общих курсов по базам данных в высших учебных заведениях, так и для специализированных курсов по Oracle. Независимо от того, какой курс вы будете читать либо изучать, мы предложим вам всестороннюю помощь, облегчающую вашу задачу. Вряд ли вам пригодятся все предлагаемые нами дополнительные материалы, но в зависимости от конкретной задачи вы наверняка найдете среди них что-то полезное для себя. Материалы, предлагаемые в помощь студентам и преподавателям, можно найти на Web-сайте [www.prenhall.com/perrypost](http://www.prenhall.com/perrypost). Дополнительные материалы для преподавателей можно заказать через каталог на сайте [www.prenhall.com](http://www.prenhall.com). Кроме того, с Web-сайта [www.prenhall.com/perrypost](http://www.prenhall.com/perrypost) можно загрузить файлы примеров, рассматриваемых в книге<sup>1</sup>. Обращаем ваше внимание на то, что данный сайт является единственным официальным источником всех файлов данных, касающихся этой книги.

Каждую главу мы рекомендуем начинать “с чистого листа”, запуская сценарий, инициализирующий базу данных Readwood Realty (соответствующее указание приводится в каждой главе). Таким образом вы застрахуетесь от ошибок или побочных эффектов, вызванных работой с предыдущими главами.

## Варианты чтения книги

Существует множество способов изучения систем управлениями базами данных и Oracle (столько же, сколько существует преподавателей). Данная книга разработана достаточно гибко, чтобы удовлетворять нескольким распространенным подходам и стилям преподавания. Всего в ней насчитывается 13 глав, достаточных для формирования полносеместрового курса (из расчета три часа в неделю) по базам данных вообще или Oracle в частности. Тем не менее книгу можно использовать и по-другому. Например, если ваше время ограничено или курс предполагает час или два занятий в неделю, можно изучить только главы 1–7, касающиеся использования SQL. Если предполагается обзорный курс по Oracle, лучше будет использовать материал из всех глав. Хотя мы упорядочили темы определенным образом, вы можете читать их так, как вам будет удобно. В главе 1 вводятся системы управления реляционными базами данных и приводится базовый материал, пропускать который не рекомендуется. В главе 2 вводятся языки SQL и SQL\*Plus; кроме того, вы получите представление о других темах, рассматриваемых далее в книге. Подробное обсуждение запросов SQL приводится в главах 5 и 6, однако при желании эти главы можно прочесть до изучения команд языка определения данных (Data Definition Language – DDL) и языка мани-

---

<sup>1</sup> Для удобства читателей все эти файлы добавлены на прилагаемый компакт-диск в папку `files`. – Примеч. ред.

пулирования данными (Data Manipulation Language – DML). В таком случае глава 5 изучается перед главой 2. Не завершив изучение глав 3 и 4, мы не рекомендуем читать главу 6, где подробно описываются организация запросов и представления, а также несколько таблиц и способы их соединения (левое внешнее объединение, объединение по эквивалентности и т.д.). Кроме того, в главе 6 описываются представления. Глава 7 посвящена программированию с использованием PL/SQL. Это нетривиальная глава, и если вы не желаете изучать программирование и использовать определенные процедуры и функции, вы можете пропустить ее. При желании, данную главу можно прочесть в самом конце, после главы 13.

В главах 8–11 обсуждаются формы и отчеты, а сами главы базируются на том, что изучалось ранее. В главе 12 описывается безопасность базы данных, а в главе 13, посвященной администрированию базы данных, представлен обзор действий АБД. Данная глава хорошо подходит для завершения вводного курса и перехода к более сложной теме администрирования базы данных.

## Совет по использованию Oracle

Вы можете установить программное обеспечение Oracle 10g, доступное на сайте [www.oracle.com](http://www.oracle.com), или использовать любую другую версию этой базы данных. За исключением нескольких необязательных примеров, приведенных в последних главах, работа с книгой не требует никаких специальных привилегий. Все, что от вас требуется, — использовать правильное имя пользователя и пароль, дающие минимальные привилегии (создание сессий и таких объектов, как таблицы) для выполнения упражнений, предлагаемых в книге. Разумеется, если вы установите Oracle на собственный компьютер, вы предоставите себе права администратора базы данных, а следовательно, никаких проблем с привилегиями не будет. Тем не менее знайте, что для работы с книгой это, в общем-то, необязательно.

## Благодарности

Создание учебника — это совместная работа издателя и авторов. Подобно большинству начинаний, создание хорошего учебника требует скоординированной командной работы. Мы хотим поблагодарить рецензентов, комментировавших и уточнявших нашу работу, благодаря чему она стала лучше отвечать требованиям студентов и преподавателей. В частности, мы хотели бы поблагодарить

Владана Йовановича (Vladan Jovanovic), Georgia Southern University

Дженнифер Крейе (Jennifer Kreie), New Mexico State University

Карен Нант (Karen Nantz), Eastern Illinois University

Джереми Смита (Jeremy Smith), Carnegie Mellon University

Томаса П. Стurmа (Thomas P. Sturm), University of St.Thomas, Миннесота

Уильяма Томаса (William Thomas), Juniata College, Пенсильвания

Авторы благодарны за работу профессионалов из издательства Pearson Prentice Hall. Особенно хотелось бы поблагодарить Боба Хорана (Bob Horan), Джейнин Цилиотта (Jeanine Ciliotta), Ану Кордеро (Ana Cordero) и Анну Грейдон (Anne Graydon) за заботу и внимание на каждом шагу этого длинного пути. Все они действительно являются профессионалами!

Наконец, мы хотим поблагодарить наших супруг, Нэнси (Nancy) и Сару (Sarah), за терпение во время нашей работы над книгой. Без вас ничего этого не было бы.

Если вы имеете желание пообщаться с нами по поводу книги, мы будем счастливы услышать ваше мнение и ответить на ваши комментарии, предложения и даже похвалы! Свои сообщения посыпайте на адрес [perrypost@prenhall.com](mailto:perrypost@prenhall.com). Последнюю информацию, файлы данных, обновления и вспомогательные материалы к книге *Введение в Oracle 10g* вы можете найти на Web-сайте [www.prenhall.com/perrypost](http://www.prenhall.com/perrypost).

Джеймс Перри

Джеральд Пост

---

---

## От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо либо просто посетить наш Web-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг. Наши координаты:

E-mail: [info@williamspublishing.com](mailto:info@williamspublishing.com)

WWW: <http://www.williamspublishing.com>

Адреса для писем:

из России:      115419, Москва, а/я 783

из Украины:    03150, Киев, а/я 152

## **Благодарности издательства**

Издательский дом “Вильямс” благодарит Ерофеева Сергея за большой вклад в подготовку издания книги.

---

---

## **Прочтите перед началом работы**

В начале каждой главы приводится последовательность действий, выполняя которые вы инициализируете базу данных Redwood Realty, после чего начинается собственно изучение нового материала. Файлы данных, сценарии и все остальные данные можно найти на прилагаемом к книге компакт-диске или на Web-сайте книги ([www.prenhall.com/perrypost](http://www.prenhall.com/perrypost)). Инициализируя базу данных в начале каждой главы, вы гарантируете, что любые внесенные ранее изменения базы данных не влияют на вашу нынешнюю работу. Другими словами, в начале каждой главы база данных возвращается в исходное состояние. В некоторых первых главах база данных Redwood Realty устанавливается не полностью, поскольку вам нужно лишь несколько ее таблиц и других объектов. Начиная с пятой главы вы будете устанавливать базу полностью. Можете экспериментировать с ней, как вам будет угодно. Изучайте файлы сценариев перед запуском. Все они совместимы с приложением Notepad (Блокнот), так что вы можете найти файл сценария с помощью Windows Explorer, а затем изучить или распечатать его.

# Введение в СУБД и Oracle 10g

**В этой главе...**

- Основная цель и использование систем баз данных
- Основные этапы разработки базы данных
- Установка инструментов Oracle, необходимых для работы с данной книгой
- Дополнительная информация о базах данных, использованных в данной книге

---

## Системы управления базами данных

Система управления базами данных, СУБД (Database Management System — DBMS), является одним из важнейших компьютерных инструментов в современных организациях. В большинстве компаний СУБД используют для записи транзакций и ведения бухгалтерского учета. На коммерческих Web-сайтах СУБД используют в основном “в фоновом режиме” для отчетности и записи данных о транзакциях. Целью использования СУБД является обеспечение надежного хранения и легкого доступа к данным. Вообще, СУБД — это программное обеспечение, взаимодействующее с операционной системой компьютера и использующее его для хранения и извлечения данных. Самы данные хранятся в тщательно разработанной базе данных. Точнее, база данных состоит из данных, а СУБД (или просто *система баз данных*) — это программное обеспечение, контролирующее базу данных.

За годы своего существования системы баз данных неоднократно изменялись. Одним из важнейших изменений было развитие модели *реляционной базы данных*. Большинство существующих систем баз данных, включая Oracle, основаны на этой

модели. Подробнее о реляционном подходе мы будем рассказывать на протяжении всей книги, однако фундаментальная концепция заключается в том, что все данные хранятся в таблицах, состоящих из простых столбцов и строк.

СУБД имеет несколько компонентов, отвечающих за хранение и извлечение данных, определение пользователей и создание приложений. Пользователи взаимодействуют с базой данных с помощью языка запросов, форм ввода данных и отчетов. Стандартным языком запросов, используемым в большинстве основных систем (включая Oracle), является SQL. Некоторые элементы SQL определяются стандартами, из-за чего вам будет легче применять свои знания к различным системам. База данных Oracle очень сильно связана на SQL, поэтому для выполнения практически любого действия в Oracle вначале необходимо изучить язык SQL. Кроме того, вам нужно знать, как создавать формы и отчеты, поскольку именно с помощью этих средств пользователи будут взаимодействовать с вашей базой данных. В связи с этим советуем использовать набор Oracle Developer Suite, содержащий инструменты, которые помогут вам с минимальными усилиями создать формы ввода данных и отчеты различных типов.

Дадим еще несколько определений. *Приложения* состоят из базы данных, а также форм и отчетов, необходимых пользователям для выполнения конкретных задач. *Разработчики* — это проектировщики базы данных, которые пишут запросы и создают формы и отчеты. *Администраторы базы данных*, АБД (Database administrator — DBA), отвечают за управление СУБД, включая установку и обновление программного обеспечения, устраняют различные проблемы, а также занимаются архивированием и восстановлением данных.

## Персональные системы

В небольших организациях СУБД можно запустить на одном персональном компьютере. Как разработчик вы, возможно, установите все на одном компьютере и таким же образом выполните первоначальное тестирование приложения. Однако истинная мощь СУБД проявляется в возможности работы с несколькими пользователями, одновременно изменяющими данные. Персональная система баз данных может быть полезной только в том случае, если она нужна нескольким пользователям. Вы можете выбрать данный подход из-за легкости создания приложений с помощью средств построения форм и отчетов. Однако вы удивитесь, когда узнаете сколько людей захотят использовать систему сразу после ее создания. Другими словами, в момент создания приложение может быть небольшим, однако со временем расширяться до невероятных размеров. В подобных случаях имеет смысл использовать СУБД, поскольку она достаточно гибкая, чтобы поддерживать изменения и расширение. Таблицы, запросы, формы и отчеты, которые вы создадите для небольшого приложения, можно расширить, если несколько пользователей захотят использовать одно и то же приложение.

## Системы на основе сервера

В большинстве производственных систем баз данных используется подход, при котором база данных поддерживается на первичном сервере, а пользователи соединяются с ней через отдельные машины, иногда называемые клиентскими компьютерами. Сервер является центральным хранилищем данных, поэтому все пользователи видят одну и ту же информацию, в том числе и обновленную. Разумеется, данный подход требует, чтобы все эти компьютеры были объединены в какую-то сеть. Из-за своей гибкости и доступности наиболее распространенным подходом сегодня является использованием для соединения компьютеров протоколов Интернета. Данный подход также облегчает доступ к базе данных мобильным клиентам. Фактически клиентские компьютеры сейчас соединяются с сервером через браузер Интернета. Одним из преимуществ данного подхода является то, что все требуемое программное обеспечение для доступа к базе данных и запуска форм и отчетов можно загрузить из Интернета. Соответственно мы можем относительно легко настроить клиентский компьютер, чтобы его можно было использовать с приложением базы данных. Другое преимущество заключается в том, что для взаимодействия с Oracle конечный пользователь может использовать небольшой (недорогой) клиентский компьютер, так как *дорогие клиентские компьютеры не требуются*.

Большинство приложений Oracle запускается на трехсвязной системе (рис. 1.1). Одним компонентом является внутренний сервер базы данных, управляющий собственно базой данных. Этот компьютер (*сервер*) отвечает за хранение и извлечение данных, контролирует операции резервного копирования и транзакции базы данных. Довольно часто данный элемент состоит из кластера компьютеров, что позволяет обеспечить непрерывное надежное предоставление услуг, если один из серверов даст сбой. Средним звеном является сервер приложений Oracle, который управляет учетными записями пользователей и обрабатывает формы и отчеты. Сервер приложений извлекает данные из СУБД, отправляя запросы по сети. Получая в ответ строки данных, он обрабатывает их и интегрирует в формы и отчеты, которые отправляется пользователям. Среднее звено также может быть кластером компьютеров, что позволяет справляться с более интенсивной нагрузкой и компенсировать аппаратные сбои. Верхним (интерфейсным) ярусом является клиентская сторона, которая запускается практически на любом компьютере, поддерживающем браузер Интернета. Одним из требований Oracle к интерфейсному уровню является наличие установленного процессора Java. Как правило, задачи клиента являются довольно простыми и ограничиваются отображением и сбором простых элементов данных. Поскольку пользователи часто требуют, чтобы предоставление информации было мобильным, они часто используют различные компьютеры. Следовательно, аппаратное и программное обеспечение клиентской стороны полезно делать как можно более простым, чтобы снизить стоимость поддержки. Если с клиентским компьютером возникнут проблемы, пользователь просто перейдет на другой компьютер и свяжется с сервером приложений.

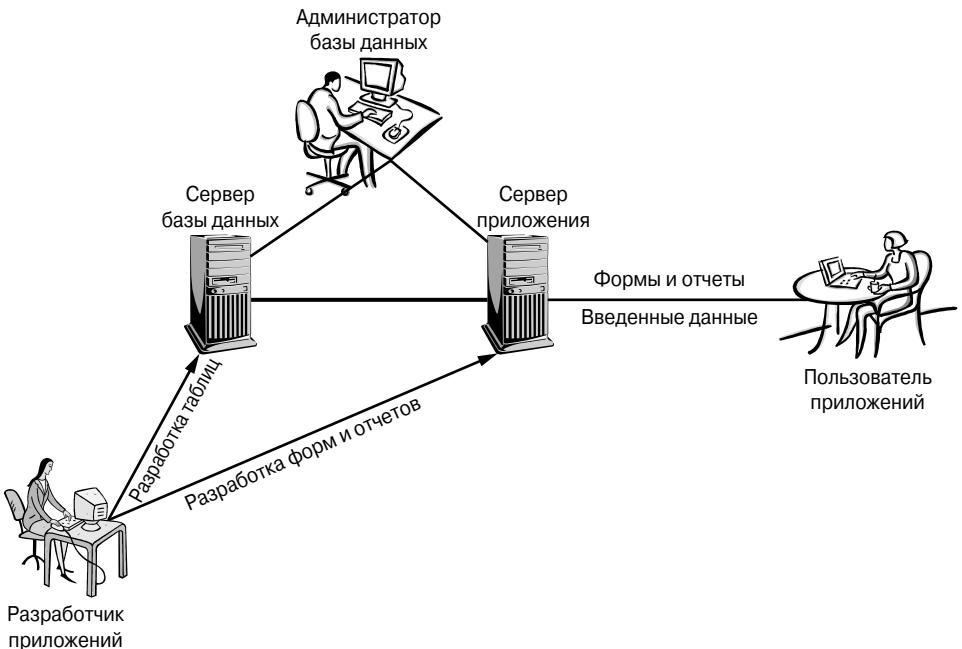


Рис. 1.1. Трехсвязная система СУБД

Как вы могли догадаться, основная цель многопользовательской многоуровневой системы — добиться одновременной работы базы данных и приложений. В критических приложениях система должны продолжать работать даже при сбое некоторых компонентов. Единственная надежная возможность решить эту проблему — реализовать резервирование на каждом уровне. Разумеется, это дороже (в основном из-за того, что вашей организации потребуется дополнительно заплатить компании Oracle за лицензию). Однако если ваша система и информация очень важны, то добавочная стоимость — ничто по сравнению с бизнес-потерями и ценой за восстановление после аппаратного сбоя.

Запуск базы данных на нескольких уровнях имеет несколько достоинств, однако есть и недостаток, проявляющийся, когда клиентская сторона использует Web-браузер. Из-за того что сетевые соединения являются потенциально медленными, данные загружаются на компьютер клиента. Изменения, которые вносят в данные, не записываются в серверной базе данных, пока пользователь не щелкнет на кнопке записи данных. Подумайте, что может произойти, если несколько сотрудников одновременно работают над одним проектом? Задерживая обновление, вы увеличиваете вероятность того, что кто-то другой изменит данные до того, как вы запишете свои изменения. СУБД может распознать такую ситуацию, однако из-за задержки будет немножко сложнее определить, какое из измененных значений правильное.

Поскольку Oracle зависит от Java, проверьте, чтобы в вашем Web-браузере был установлен процессор Java (или соответствующий модуль). Как правило, стандартный процессор устанавливается по умолчанию, но в некоторых организациях из соображений безопасности его отключают. В таком случае все, что вам нужно, вы найдете на Web-сайте Java (<http://www.java.com>), загрузив и установив наиболее “свежую” версию процессора.

---

## Описание систем в виде объектов и взаимоотношений

Главная цель использования базы данных — это сбор и эффективное хранение необходимой информации, чтобы ее можно было быстро обновлять и извлекать. Одна из проблем заключается в том, чтобы точно определить, какие данные требуются организации. Для ее решения разработчики вначале обязывают пользователей и менеджеров определить проблемы, которые требуется решить. Но дело в том, что пользователи обычно немного знают о базах данных. Поэтому вы должны спросить их о том, какие бизнес-операции и данные они собирают и используют при принятии решений. Затем вам нужно так организовать эту информацию, чтобы ее можно было эффективно хранить в базе данных. Если сможете, нарисуйте диаграмму и покажите ее пользователям, чтобы они подтвердили, что вы правильно поняли, как следует организовать информацию.

Для моделирования системы баз данных очень удобна модель “сущность–связь” (или классовый подход). Обращаем ваше внимание на то, что некоторые пользователи по-прежнему используют старую терминологию и обозначения *диаграммы “сущность–связь”* (Entity-Relationship Diagram — ERD). Позже были принята терминология и рисунки *UML* (Unified Modeling Language — унифицированный язык моделирования). Общие принципы обоих подходов очень похожи и отличаются лишь построением диаграмм. Диаграммы UML легче читать, и этот подход стандартизован, следовательно, более универсален. Однако вы можете использовать термины и диаграммы по своему выбору.

Фундаментальным этапом моделирования является идентификация сущностей (или классов сущностей) для которых вы должны собирать данные. Каждая *сущность* (или *класс*) имеет *атрибут*, определяющий собираемые данные. Например, во многих базах данных распространена сущность *Customer*. Объект *Customer* имеет различные атрибуты: *CustomerID*, *Last Name*, *First Name*, *Phone*, *Address* и *City*. Атрибуты представляют собой данные, собираемые для каждого класса. Например, определенный экземпляр данных *Customer* может иметь следующий вид:

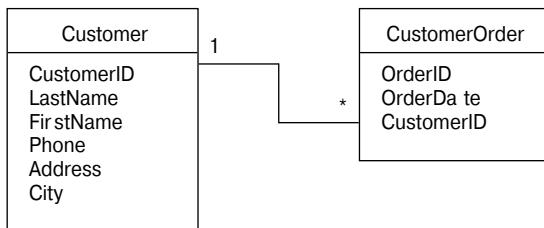
151, 'Jones', 'Mary', '111-2222', '123 Main', 'Eureka'

**ТАБЛИЦА 1.1.** Общие типы данных

<i>Тип данных</i>	<i>Описание</i>	<i>Примеры</i>
Text	Любой тип символов, чисел или знаков пунктуации	123 Main Street
Number	Числа хранятся так, чтобы их можно было объединять или совершать над ними арифметические действия. Вы можете задавать число десятичных знаков (задав 0, можно получить целые числа). В Oracle данный тип данных используется также для денежных единиц	1.2345 15.32
Date	Используется для хранения дат и времени. Везде, где это возможно, стоит использовать данный формат, а не текстовое представление, поскольку в таком случае система сможет вычислить разницу между датами	13-JUL-2006
Binary	Используется для хранения данных о нетрадиционных объектах (например, изображениях, файлах электронных таблиц или других больших элементах)	Picture.jpg

В конечном счете вам еще нужно будет записать тип данных, содержащихся в каждом атрибуте. В предыдущем примере большинство атрибутов содержит текстовые данные, т.е. данные, которые содержат символы, числа и знаки препинания. Различные общие типы данных, которые вам наверняка встретятся, представлены в табл. 1.1. После того как вы определите, какие атрибуты будете хранить, вам потребуется записать тип данных, вмешаемых каждым атрибутом. Ситуация с некоторыми атрибутами сравнительно понятна, — например, для записи адреса клиента используется тип данных *Text*. Относительно других атрибутов вам придется изучать существующие данные и консультироваться с пользователями. Типы данных *Number* и *Date* доступны для данных, задействованных в расчетах. Например, вы можете хранить в поле базы данных не возраст сотрудников, а даты их рождения. Позже, используя столбец “День рождения”, вы сможете определить возраст каждого сотрудника, рассчитав разницу между текущей датой (по часам компьютера) и датой рождения сотрудника. Подобным образом номер счета можно считать строкой символов, поскольку вам вряд ли придется выполнять математические действия с атрибутом *InvoiceID*. При этом поле цены единицы продукции в счете обязательно должно быть числовым, чтобы при расчетах эти поля можно было суммировать, рассчитывать произведение цены единицы товара на их количество и определить общую сумму за товар. Если сделать все позиции текстовыми, то вы сможете хранить и извлекать данные, но не сможете рассчитать суммы или вычислить разницу между двумя датами.

База данных практически всегда имеет больше одного объекта, а иногда — сотни объектов. Как правило, объекты каким-то образом связаны между собой. Данные



**Рис. 1.2.** Отношение “один ко многим” между классами *Customer* и *CustomerOrder*

ассоциации, соотношения или связи реализуются путем хранения связанных данных в похожих столбцах. Например, еще одним распространенным бизнес-объектом является *Customer Order* (“заказ клиента”), в котором записываются заказы, помещаемые клиентами (еще один объект). *Customer Order* содержит различные атрибуты, например *OrderID* и *OrderDate*. Кроме того, он содержит атрибут *CustomerID*, который предлагает связь с таблицей *Customer*. Например, определенный объект *Customer Order* может содержать такие данные: 1201, '06-JUN-2006', 151. Здесь 151 — это *CustomerID* (“идентификатор клиента”); зная это, вы можете просмотреть все остальные данные, найдя атрибут *CustomerID* со значением 151 в объекте *Customer*.

На рис. 1.2 показано, как можно изобразить связь между объектами *Customer* и *CustomerOrder*. Каждый объект представлен прямоугольником, атрибуты расположены в прямоугольнике, в верхней части которого указано имя объекта. Соотношения между двумя объектами определяются с помощью соединительной линии. Как правило, ее рисуют для того, чтобы показать, что значения *CustomerID* в классе *CustomerOrder* будут соединяться со значениями *CustomerID* в таблице *Customer*. Примечание возле каждого конца линии указывает тип соотношения. В данном случае установлена зависимость типа “один ко многим”, поскольку возле левого конца линии указана 1, а возле правого — \* (звездочка). Звездочка указывает на многосторонность отношения “один ко многим”. Читая данную диаграмму слева направо, можно сказать, что клиент может помещать несколько заказов (это обозначено звездочкой) на правой стороне связи, обозначенной *CustomerOrder*. С другой стороны, любой заказ может поступать только от одного клиента, на что указывает цифра 1 на стороне связи, обозначенной *Customer*. Таким образом, если вы ищете форму заказа, то находите только одного клиента. Два клиента не могут подавать один заказ. Откуда вы это знаете? Таким образом представлено правило организации, поэтому либо вы должны знать, что это стандартный подход в бизнесе, либо вам должны сообщить об этом пользователи.

В реальном проекте ваша основная работа на этом этапе заключается в общении с пользователями и определении всех классов объектов и соответствующих атрибутов. Вы должны начать составлять диаграмму классов и отношений между ними. Возможно, некоторые связи вы установили изначально; тем не менее, в следующем

разделе мы представим некоторые правила, которыми вы должны руководствоваться при разработке.

---

## Модель реляционной базы данных

Исследования баз данных показали, что самым эффективным способом хранения и извлечения большей части информации являются реляционные базы данных. Чтобы добиться такой эффективности, к разработке базы данных необходимо подходить очень ответственно. Фундаментальным компонентом реляционной базы данных является таблица. Е. Ф. Кодд (E. F. Codd), признанный отец реляционных баз данных, разработавший этот подход, предпочитал термин *связь* (relation), а не *таблица*. Именно отсюда и пошел термин *реляционные* (relational) *системы баз данных*.

Таблица состоит из строк и столбцов и содержит данные, представляющие один объект в бизнес-модели. Ее столбцы называются *атрибутами*, а строки представляют отдельные экземпляры данных. Приведенную терминологию легче понять, обратившись к примеру, поэтому на рис. 1.3 показаны две таблицы гипотетической реляционной базы данных. Точнее, данные таблицы представляют диаграмму отношений, представленную на рис. 1.2. Обратите внимание на то, что каждая строка является простой, т.е. она представляет одну сущность. Каждый фрагмент данных является простым, поскольку содержит значения, описывающие одну концепцию. Например, вы не можете записать два номера телефона для одного человека. Если вам нужно будет записать несколько номеров телефона какого-то клиента, вы либо добавляете новые столбцы к таблице (CellPhone, HomePhone и т.д.), либо помещаете эту информацию в отдельную таблицу. Обратите также внимание на то, что таблицы связаны между собой посредством данных в столбце CustomerID. Посмотрев на первую строку таблицы CustomerOrders, видим, что заказ был помещен клиентом с идентификатором 151. Это значение атрибута CustomerID можно использовать для поиска информации, касающейся данного клиента, — соответствующей строки таблицы Customers. Такое соединение таблиц не случайно, так как вы должны встроить его в структуру базы данных. Именно значения согласующего столбца в двух связанных таблицах позволяет системе баз данных соединять связанные строки двух таблиц. Другими словами, значения согласующего столбца в двух отдельных таблицах являются “克莱ем”, позволяющим Oracle или другой системе баз данных синтезировать информацию по отдельным таблицам или фактам.

### Первичные ключи

Существует несколько основных правил разработки таблиц, которые будут эффективно и точно работать в реляционной системе баз данных. Однако для понимания этих правил вначале необходимо ознакомиться с несколькими определениями. Реляцион-

CustomerOrders

<b>OrderID</b>	<b>OrderDate</b>	<b>CustomerID</b>
1201	06-JUN-2006	151
1202	06-JUN-2006	155
1203	07-JUN-2006	151

Customer s

<b>CustomerID</b>	<b>LastName</b>	<b>FirstName</b>	<b>Phone</b>	<b>Address</b>	<b>City</b>
151	Jones	Mary	111-2222	123 Main	Eureka
152	Smith	Susan	222-5555	738 Elm	Eureka
153	Brown	David	111-2355	235 East	Eureka
154	Sanchez	Maria	999-3332	351 Ocean	Arcata
155	Steuben	Saul	555-2351	111 Main	Orick
156	Hayworth	Michele	231-3252	761 West	Loleta

Рис. 1.3. Примеры таблиц в реляционной базе данных

ная база данных — это набор связанных таблиц. Таблица — это набор строк и столбцов (атрибутов), описывающих объект. Атрибут — это характеристика объекта. Стока содержит данные для одного экземпляра объекта.

Каждая таблица должна иметь *первичный ключ* — значение, единственным образом определяющее каждую строку и располагающееся в определенном столбце или наборе столбцов. Для нескольких объектов вы создадите собственный столбец первичных ключей и позволите системе управления реляционной базой данных автоматически заполнить столбцы уникальными значениями. В нашем простом примере генерируемым ключом, скорее всего, будет CustomerID. Если система генерирует первичный ключ автоматически, то он будет уникальным. Но если первичный ключ создается вручную, то гарантировать его уникальность затруднительно. Пользователи могут делать ошибки и присваивать одно и то же значение двум клиентам — особенно в большой системе или системе, в которой несколько пользователей взаимодействуют с базой данных и могут одновременно и независимо вводить первичные ключи. К счастью, Oracle содержит инструменты, позволяющие генерировать значения первичных ключей, которые гарантированно будут уникальными. Более того, вы можете указать Oracle автоматизировать процесс генерации ключей. В примере, показанном на рис. 1.3, ключ OrderID может генерировать база данных, однако его может генерировать и отдел продаж — возможно, извлекая из напечатанной формы.

Для каждой таблицы, хранимой и используемой базой данных, вам нужно согласовать с пользователями, как вы будете выбирать первичные ключи. (Поскольку некоторые ключи лучше других.) Например, вернемся к нашей таблице Customer:

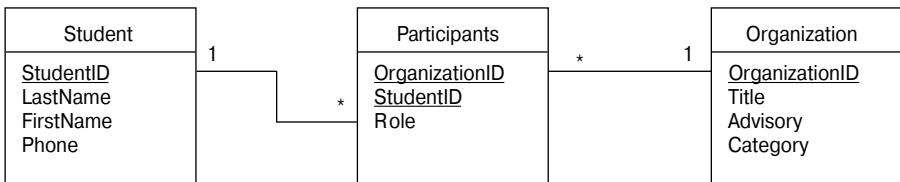


Рис. 1.4. Отношения “один ко многим” и “многие ко многим”

вы вряд ли захотите использовать в качестве первичного ключа номер телефона из строки клиента. Что произойдет, если заказать товар пожелают два члена одной семьи? Нужно ли вам их различать? Рассмотрим худшую ситуацию: что произойдет, если клиент поменяет номер телефона, а старый номер достанется кому-то другому?

При работе с первичными ключами следует быть осторожными. Вы не будете генерировать новые ключи для каждой таблицы. Фактически вам часто придется работать с таблицами, в которых первичные ключи состоят из нескольких столбцов. Такие ситуации имеют особое значение, важное для понимания структуры реляционной базы данных. Если два (или более) столбца являются частью первичного ключа, это означает, что между объектами существует отношение “многие ко многим”, которое представляют эти ключи.

Пример отношения “многие ко многим”, которое вам, возможно, уже встречалось, приведен на рис. 1.4. Структура объекта *Student* понятна — в нем содержатся общие данные о студенте (имя, фамилия, номер телефона). Объект *Organization* представляет собой клубы, спортивные команды и другие организации студенческого городка. Объект *Participants* занимает промежуточное положение — в нем отслеживается, какие студенты являются членами каких организаций. В объекте *Participants* первичный ключ образуют атрибуты *OrganizationID* и *StudentID*. Для того чтобы единственным образом определить строку таблицы, ключ должен содержать оба указанных столбца. Пример данных, составляющих таблицы, приведен на рис. 1.5.

Атрибут *OrganizationID* должен быть представлен в ключе потому, что каждый студент может быть членом нескольких организаций. Атрибут *StudentID* должен быть представлен в ключе потому, что в каждой организации может состоять множество студентов. Если частью ключа будет только атрибут *OrganizationID*, то мы не сможем единственным образом определить строку, поскольку большинство идентификаторов введено несколько раз, — фактически они вводятся для каждого студента. Если частью ключа будет только атрибут *StudentID*, то мы снова получим дублирующиеся значения, поскольку студент может принадлежать к нескольким организациям. Формируя *составной первичный ключ* из атрибутов *OrganizationID* и *StudentID*, мы можем единственным образом идентифицировать строку, поскольку студент, состоящий в организации, в списке членов данной организации указан только раз.

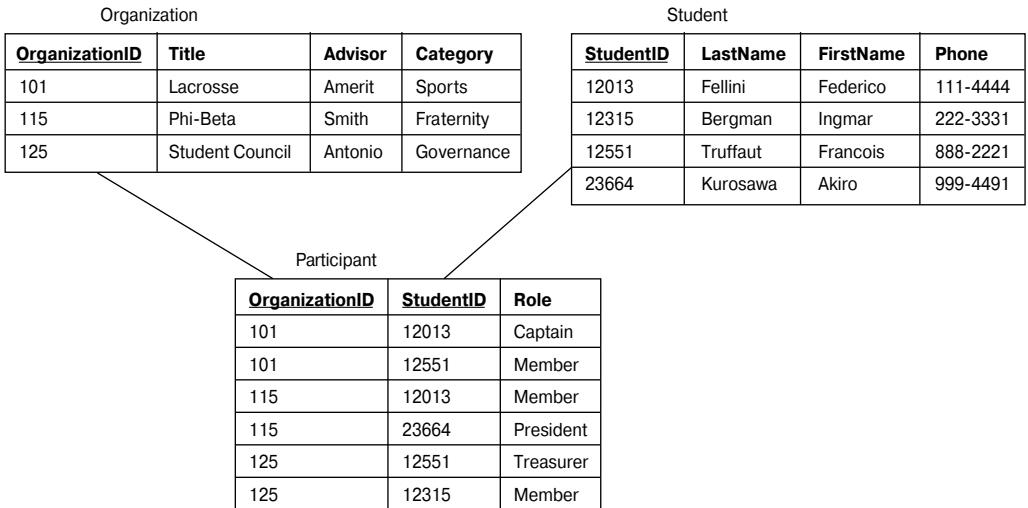


Рис. 1.5. Пример информации, требующей ключа, сформированного двумя столбцами

## Нормальные формы

Определение столбцов, используемых в качестве первичных ключей, является важным этапом разработки таблиц для реляционной базы данных. Тем не менее это не все. Каждая таблица в реляционной базе данных должна удовлетворять определенным условиям. Если эти условия не удовлетворяются, использование базы данных будет сопряжено с множеством проблем, включая потерю важных данных. Итак, как правило, вы начинаете с формы или описания данных. По ним вы создаете объекты. Затем благодаря процессу *нормализации* данные объекты превращаются в таблицы. Ниже мы подробно распишем процедуру создания таблиц с очень хорошими характеристиками. Проще всего понять данный процесс на примере. Рассмотрим его.

На рис. 1.6 показана стандартная бизнес-форма для заказа клиента. Посмотрите на форму и подумайте, можете ли вы определить основные объекты? Объект *Customer* сравнительно очевиден — особенно из-за того, что он заключен в отдельную рамку. Если вы немного об этом подумаете, то можете прийти к выводу, что объектом может быть весь элемент *CustomerOrder*. Рассмотрим оставшиеся данные. Поскольку они также отображаются в отдельном прямоугольнике, заманчиво предположить, что упорядочиваемые элементы представляют третий и последний объект. Для начала ответ хороший, однако его нужно улучшить.

Для того чтобы понять, зачем нужна нормализация базы данных, попробуйте создать таблицу базы данных без планирования какой-либо структуры. Что произойдет, если все данные из формы *Customer* поместить в одну таблицу? На рис. 1.7 показано несколько элементов данных. Во-первых, обратите внимание на то, что вам не обязательно сохранять столбец *Value*, поскольку его можно вычислить по существу-

Customer Order				
OrderID				Date
CustomerID First Name      Last Name Phone Address City, State Zipcode				
ItemID	Price	Description	Quantity	Value
1526	32.95	Basketball	1	32.95
3921	79.92	Running shoes	1	79.92
4797	1.59	Racquetballs	3	4.77

Рис. 1.6. Данные формы перед нормализацией

ющим данным. Как правило, все, что можно вычислить, хранить не обязательно. База данных располагает инструментами, которые выполняют для вас все расчеты, поэтому вам совсем не обязательно занимать место этой информацией.

Если вы попытаетесь ввести данные в предложенную таблицу, то быстро обнаружите ряд проблем — в основном это чрезмерное количество повторов. Для каждой из трех заказанных вещей вам придется не только повторно вводить все данные заказа, но и все данные о клиенте. А постоянный повтор одних и тех же данных — это бездумное расходование места. Однако есть и другие проблемы. Что произойдет, если клиент (Customer) 155 отметит свой заказ и вы удалите две строки таблицы? Вы потеряете всю информацию о клиенте! Другой вопрос: что вы можете сказать о продукте 1577? Он еще не заказан, поэтому для хранения указанных данных нет места; следовательно, вы не знаете ничего. Изучая предложенную таблицу, вы можете найти и другие проблемы подобного рода. Как исправить дизайн таблицы и избежать этих проблем? Гарантировать, что все таблицы соответствуют трем правилам нормализации (или трем нормальным формам)! Данные правила нормализации применяются последовательно, начиная с первой нормальной формы.

## Первая нормальная форма

Для того чтобы таблица соответствовала *первой нормальной форме*, она должна иметь только атомные элементы. Атомный означает, что значения нельзя разделить на более мелкие составляющие. Другими словами, каждая ячейка таблицы содержит простые элементы данных, которые не могут повторяться или иметь вложенные значения. Технически все ячейки на рис. 1.7 являются атомными. Однако здесь мы сделали

не все возможное. В реальной жизни номер `ItemID` повторяется для каждой записи `OrderID`, что явно видно по форме заказа, приведенной на рис. 1.6.

Если вы желаете уменьшить число очевидных повторов в форме, посмотрите, как организована таблица, приведенная на рис. 1.8. По многим показателям данный подход даже хуже рассмотренного выше, поскольку теперь реляционная база данных не знает, сколько элементов может быть заказано в каждой форме, поэтому планировать распределение места под базу данных довольно сложно, кроме того, затруднен и поиск в ней. Таблица, предложенная на рис. 1.8, не находится в первой нормальной форме. Это можно исправить с помощью повтора данных и сведения таблицы к виду, представленному на рис. 1.7, однако такое улучшение не кажется целесообразным. Вместо этого разумно разбить полученную таблицу на две новые. На те же две таблицы можно разбить и версию, представленную на рис. 1.7, руководствуясь, однако, другими причинами. По сути, вы извлекаете данные, находящиеся в повторяющемся блоке исходной таблицы, и помещаете их в новую таблицу. Тем не менее в новую таблицу вам нужно включить столбец `OrderID`, поскольку с его помощью полученные таблицы будут связаны между собой.

Результат изменений показан на рис. 1.9. Обратите внимание на то, что в первой предложенной таблице в качестве первичного ключа используется `OrderID`, поскольку теперь вам уже не нужно повторять данные о заказе и клиенте для каждой заказываемой вещи. Во второй предложенной таблице в качестве первичного ключа используются `OrderID` и `ItemID` из-за отношения “множество–множество”. Каждый заказ может содержать несколько вещей, а каждая вещь может быть заказана множество раз. Обе предложенные таблицы имеют атомные данные и теперь находятся в первой нормальной форме.

## Вторая нормальная форма

Если вы внимательно посмотрите на вторую таблицу на рис. 1.9, то увидите, что проблемы по-прежнему существуют. Всякий раз, когда вы вводите `ItemID` 1526, вам нужно ввести описание и цену. Из-за траты места и времени на ввод данных внедрение информации о вещах в данную таблицу заказа означает, что вы по-прежнему никак не можете найти информацию о продуктах, которые еще не были заказаны. Проблема возникает из-за того, что описание вещи и цена зависят только от номера `ItemID`. Если рассмотреть элемент, значение `ItemID` которого равно 1526, то эта вещь будет иметь то же описание и цену, что и вчера. Данные значения не зависят от значения `OrderID`.

OrderID	OrderDate	CustID	FName	LName	Phone	Address	City	Sta	ZIP	ItemID	Price	Description	Qty
1201	06-JUN-06	151	Mary	Jones	111-22222	123 Main	Eureka	CA	95001	1526	32.95	Basketball	1
1201	06-JUN-06	151	Mary	Jones	111-22222	123 Main	Eureka	CA	95001	3921	79.92	Running Shoes	1
1201	06-JUN-06	151	Mary	Jones	111-22222	123 Main	Eureka	CA	95001	4797	1.59	Racquetballs	3
1202	06-JUN-06	155	Saul	Steuben	555-2351	111 Main	Orick	CA	95022	1526	32.95	Basketball	1
1202	06-JUN-06	155	Saul	Steuben	555-2351	111 Main	Orick	CA	95022	3144	15.72	Baseball	1

**Рис. 1.7.** Все данные размещены в одной таблице – пример плохого дизайна

OrderID	OrderDate	CustID	FName	LName	Phone	Address	City	State	ZIP	ItemID	Price	Description	Qty
1201	06-JUN-06	151	Mary	Jones	111-22222	123 Main	Eureka	CA	95001	1526	32.95	Basketball	1
1202	06-JUN-06	155	Saul	Steuben	555-2351	111 Main	Orick	CA	95022	1526	32.95	Basketball	1

**Рис. 1.8.** Уменьшение числа повторов за счет использования неатомной структуры – по-прежнему плохо

OrderID	OrderDate	CustID	FName	LName	Phone	Address	City	State	ZIP
1201	06-JUN-06	151	Mary	Jones	111-2222	123 Main	Eureka	CA	95001
1202	06-JUN-06	155	Saul	Steuben	555-2351	111 Main	Orick	CA	95022

OrderID	ItemID	Price	Description	Quantity
1201	1526	32.95	Basketball	1
1201	3921	79.92	Running Shoes	1
1201	4797	1.59	Racquetballs	3
1202	1526	32.95	Basketball	1
1202	3144	15.72	Baseball	1

Рис. 1.9. Выделение блока повторяющихся данных

Таблица находится во *второй нормальной форме*, если все столбцы, не относящиеся к ключу, зависят от всего ключа. Кроме того, она должна быть в первой нормальной форме. Таблица, приведенная на рисунке, не находится во второй нормальной форме, поскольку описание и цена зависят только от ItemID, но не от OrderID. С другой стороны, заказанное количество вещей зависит и от OrderID, и от ItemID, поскольку оно сообщает нам, сколько единиц каждого товара необходимо предоставить конкретному покупателю.

Подумайте еще раз о столбце цены. Фиксирована ли цена на вещь или зависит от конкретного заказа? Однозначного ответа на этот вопрос не существует, поскольку бизнес может вестись по различным правилам. Вопросы подобных зависимостей в различных организациях могут решать по-разному. Некоторые имеют традиционные ответы, пригодные для большинства случаев, однако вам необходимо поговорить с пользователями, чтобы понять, как они подходят к этой проблеме. Если вы создаете форму заказа для реальной компании, то, скорее всего, будете использовать две цены: прейскурантную цену, которая зависит только от ItemID, и отпускную цену, которая зависит от OrderID и ItemID. Таким образом, компания может предлагать скидку, основываясь на любой выбранной характеристике. Для упрощения обсуждения далее мы будем предполагать, что каждая вещь имеет одну фиксированную цену.

Чтобы исправить таблицу, которая сейчас находится не во второй нормальной форме, вам нужно расщепить ее еще раз. Поместите столбцы, зависящие только от частичного ключа, в отдельную таблицу (рис. 1.10). Теперь таблицы OrderItems и Items находятся во второй нормальной форме. В действительности предложенная таблица order-customer также находится во второй нормальной форме, поскольку ее первичный ключ состоит из одного столбца.

order-customer

OrderID	OrderDate	CustID	FName	LName	Phone	Address	City	State	ZIP
1201	06-Jun-06	151	Mary	Jones	111-2222	123 Main	Eureka	CA	95001
1202	06-Jun-06	155	Saul	Steuben	555-2351	111 Main	Orick	CA	95022

OrderItems

OrderID	ItemID	Quantity
1201	1526	1
1201	3921	1
1201	4797	3
1202	1526	1
1202	3144	1

Items

ItemID	Price	Description
1526	32.95	Basketball
3144	15.72	Baseball
3921	79.92	Running Shoes
4797	1.59	Racquetballs

Рис. 1.10. Выделение столбцов, зависящих только от значения ItemID

### Третья нормальная форма

В таблице *order-customer* (см. рис. 1.10) все еще имеются проблемы. Если мы вернемся к исходной таблице, приведенной на рис. 1.7, то увидим эти проблемы еще отчетливее. Для каждого заказа в таблице по-прежнему повторяется одна и та же информация о клиенте. В первоначальном варианте информация о клиенте повторялась для каждой заказанной вещи. Проблема заключается в том, что данные о клиенте (имя, адрес и пр.) зависят только от атрибута *CustomerID*, но абсолютно не зависят от *OrderID*. Если вы знаете значение атрибута *CustomerID*, то знаете все о данном клиенте и вам совсем не нужен атрибут *OrderID*.

Таблица находится в *третьей нормальной форме*, когда любой столбец, не относящийся к ключу, зависит только от ключа. Кроме того, она должна находиться во второй нормальной форме. В настоящем примере данные о клиенте зависят от атрибута *CustomerID*, который не является частью первичного ключа, поэтому таблица не находится в третьей нормальной форме. Как и ранее, для решения этой проблемы таблицу необходимо расщепить на две. Вам следует вынести столбцы, зависящие только от атрибута *CustomerID*, и поместить их в собственную таблицу.

Полученные в результате таблицы показаны на рис. 1.11. Вы можете изучить их и убедиться, что они находятся в третьей нормальной форме. Главное, помните, что таблица находится в третьей нормальной форме тогда и только тогда, когда каждая ячейка содержит атомные данные, а любой столбец, не входящий в ключ, зависит от всего ключа и только от ключа.

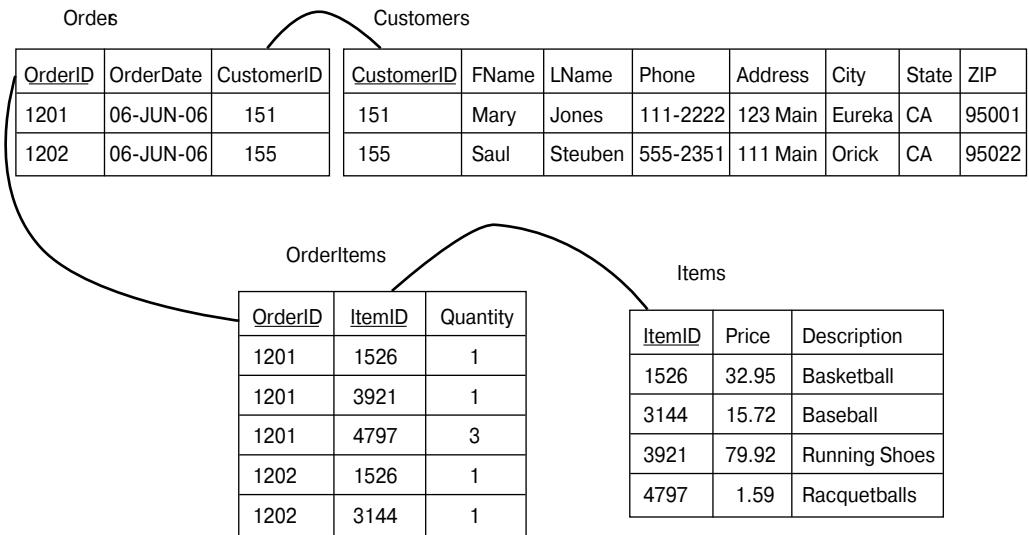


Рис. 1.11. Выделение столбцов, зависящих только от атрибута CustomerID

В таблице **Orders** первичным ключом является **OrderID**, который определяет дату заказа (**OrderDate**) и клиента, сделавшего заказ (**CustomerID**). Ключом таблицы **Customers** является **CustomerID**, а все остальные столбцы представляют атрибуты клиентов. Подобным образом ключом таблицы **Items** является **ItemID**, определяющий цену и описание вещи. Таблица **OrderItems** появилась из-за повторяющегося фрагмента в форме заказа. Атрибуты **OrderID** и **ItemID** должны быть ключами из-за отношения “многие ко многим”. Столбцы количества зависят от заказа и вещи, поэтому данная таблица также находится в третьей нормальной форме.

Таблицы в третьей нормальной форме имеют сравнительно мало проблем. Каждая таблица представляет отдельную концепцию, а дублирование данных минимально. Вы можете добавлять в таблицу строки и удалять их, не теряя важные данные в других таблицах. Если, например, вы удалите заказ, у вас по-прежнему останутся контактные данные клиента.

Третья нормальная форма — это не последний этап нормализации, однако ситуации, требующие форм более высокого ранга, сравнительно редки и в данной книге не рассматриваются. Подробнее о них вы можете прочесть в хорошем учебнике по разработке баз данных. В любом случае вам нужно представить таблицы в третьей нормальной форме, а лишь затем приступать к созданию их в реляционной базе данных.

## Соотношения и внешние ключи

На рис. 1.11 также показано, по какому принципу соединяются таблицы, чтобы с помощью данных можно было восстановить исходную форму. Данные соотношения не

случайны. Вам следует быть очень аккуратными при расщеплении таблиц и следить за тем, чтобы их можно было объединить на основе соотношений. Как правило, данные связи настраиваются через ключевые столбцы, поскольку именно ключи являются эффективной ссылкой на исходные данные. Например, таблица *Orders* содержит значение *CustomerID*, которое можно использовать для поиска связанных данных в таблице *Customers*.

Рассмотрим таблицу *Orders* более внимательно. Столбец *OrderID* является первичным ключом, поскольку именно он требуется, чтобы гарантировать уникальность каждой строки. В частности, обратите внимание на то, что столбец *CustomerID* не является частью первичного ключа в таблице *Orders*. Если бы мы не сделали его ключом, то можно было бы сказать, что между заказами и клиентами существует соотношение “множество–множество”, т.е. с данным заказом может быть связано несколько клиентов. В большинстве бизнес-транзакций такое утверждение ложно. Следовательно, в таблице *Orders* вы не должны вносить в ключ столбец *CustomerID*.

Тем не менее, поскольку столбец *CustomerID* является первичным ключом в таблице *Customers*, в таблице *Orders* он называется *внешним ключом* (foreign key). Позже вы увидите, что данная терминология очень важна в Oracle, поскольку отношения задаются через внешние ключи. Обратите внимание на то, что таблица *OrderItems* содержит два внешних ключа: *OrderID* и *ItemID*, так как оба они являются первичными ключами в “родных” таблицах.

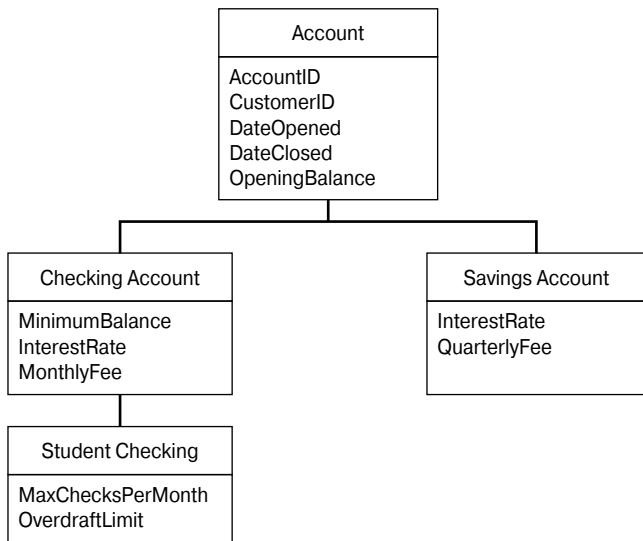
Обратите внимание на то, что таблицы *Customers* и *Items* не содержат внешних ключей. Иногда подобные таблицы называются *базисными*, поскольку содержащаяся в них информация не зависит от других таблиц. Достоинством базисных таблиц является то, что мы можем создать их и загрузить в них данные, не затрагивая связанные таблицы. И наоборот, если базе данных известна связь между внешним и первичным ключом, то вы не можете загрузить данные в таблицу *Orders* после создания нескольких клиентов. Например, если вы попытаетесь загрузить данные в таблицу *OrderItems* (см. рис. 1.11) до загрузки данных в таблицу *Items*, от которой зависят строки таблицы *OrderItems*, Oracle выдаст сообщение об ошибке. Точнее, если вы попытаетесь создать заказ и ввести значение *CustomerID*, равное 190, вы должны уже иметь соответствующую строку в таблице *Customers* с таким значением ключа. Данное свойство называется *целостностью на уровне ссылок* (referential integrity) и означает, что вы не можете вставлять данные с одной стороны связи, если на другой стороне не существует согласующегося значения.

## Объектно-реляционная модель

В 1990-х годах программисты начали использовать новый подход, именуемый объектно-ориентированным программированием, ООП (Object-Oriented Programming – OOP). Особенno эффективен он для программирования в графической среде, так как многие распространенные задачи можно выполнить с помощью предопре-

деленных объектов. Используя существующие объекты, программист существенно экономит время, поскольку ему не нужно многократно переписывать один и тот же код. В конечном счете программистам требуется сохранить на диске некоторую информацию об объектах. Разработчики также пытаются привнести в бизнес-мир идеи классов и объектов. В те времена предполагалось, что классы можно будет определять и использовать повторно во всей организации. Например, компания могла иметь класс `Customers`, который был бы доступен всем разработчикам приложений. В терминологии ООП класс — это описание объекта. Он состоит из заголовка, набора атрибутов и методов или функций, которые может выполнять класс. Обратите внимание на то, что данное определение похоже на определение сущности (entity), используемое для создания таблиц для базы данных, только базы данных, возможно, не обязаны хранить внутренние процедуры функций для классов. Объектно-ориентированные базы данных пытались создать несколько компаний. Некоторые из них попытались объединить преимущества объектной модели с реляционной моделью. До настоящего момента ни чисто объектно-ориентированные, ни объектно-реляционные возможности баз данных не получили заметного коммерческого успеха. Тем не менее Oracle поддерживает несколько возможностей объектного типа, которые вы можете использовать для определения таблиц. Насколько они вам пригодятся — зависит от задач, которые вы будете решать.

Объектно-ориентированный подход вводит две основные возможности в реляционную модель базы данных: во-первых, вы можете сохранять более сложные типы данных; во-вторых, можете создавать подтаблицы при наличии иерархии классов. Первую возможность понять проще. Напомним, что реляционная база данных требует, чтобы данные были атомными. Однако вы ограничены возможностью хранения данных общих типов, определенных СУБД (текст, числа, дата и бинарные значения). Что произойдет, если вам потребуется новый тип данных? Например, вы захотите записывать географические координаты складов. Положение определяется широтой и долготой, однако исходя из пожеланий пользователей вы можете указывать еще и высоту над уровнем моря. Мы привыкли, что положение точки в пространстве задается тремя координатами. Вместе с тем любая функция, рассчитывающая расстояние, должна получить в качестве аргументов все три значения. Однако реляционная база данных требует, чтобы вы записали координаты в виде трех отдельных столбцов. Хотя данное неудобство не очень значительное, оно означает, что база данных не совсем так представляет данные, как видим их мы. Эту проблему можно решить, используя объектно-ориентированный подход и определив новый тип данных `Location`, который будет состоять из трех значений: `longitude` (долгота), `latitude` (широта) и `altitude` (высота над уровнем моря). Далее три указанных значения будут трактоваться как один элемент при копировании данных или при передаче их функции. Для определения нового типа сложных данных в Oracle используется команда `CREATE TYPE`, позволяющая тонко настраивать базу данных. Однако будьте осторожны.



**Рис. 1.12.** Пример наследования классов объекта — счет в банке

Создав сложный тип данных в Oracle, вы можете затруднить в дальнейшем перенос данных на другие системы.

Ключевой особенностью объектно-ориентированного программирования является наследование. *Наследование* означает возможность определения и создания объектов, являющихся частными случаями существующих, при этом оба объекта имеют общее поведение, которое не нужно реализовывать заново для нового объекта. Как это выглядит? Программисты вначале определяют родовой класс, а затем выводят более подробные классы объектов из данного общего *суперкласса*. На рис. 1.12 показан простой пример банковской компьютерной системы. Родовой класс *Account* содержит данные, требуемые для всех типов счетов. На уровне программирования он также содержит методы для открытия и закрытия счетов, однако методы не хранятся в базе данных, поэтому здесь они не показаны. Затем разработчики могут определить подклассы — *Checking Account* и *Savings Account*. Как видно на диаграмме, оба новых класса выведены из родового класса *Account*. Это означает, что они наследуют все свойства и методы суперкласса (который называется так из-за того, что находится выше производных классов). Наследование позволяет идти дальше и определять новые типы счетов, например *Student Checking Account*. Программистам нужно определить только новые свойства — все остальные свойства и методы обрабатываются суперклассом.

Для работы с наследованием в Oracle используется объединение типов данных с подтипами и вложенными таблицами. По сути, вы можете определить различные типы данных для всех типов счетов и записать их как вложенные столбцы в таблице *Account*. Подробности этого процесса в настоящей книге не рассматриваются.

При рассмотрении модели наследования стоит отметить то, что оно является иерархическим. Частично развитие реляционной модели было вызвано необходимостью борьбы с недостатками иерархического подхода. В некотором смысле объектно-ориентированный подход представляет попытку усилить структуру реляционной базы данных. Фактически вы не обязаны использовать объектно-ориентированные возможности. Вы всегда можете найти способ реализовать то, что нужно, с помощью реляционной структуры. В нашем примере банковских счетов вы можете просто определить все четыре класса как отдельные таблицы, а затем записать данные для отдельных счетов в соответствующих таблицах. Единственное, что нужно сделать, — это определить AccountID в качестве первичного ключа во всех таблицах. Если же использовать чистый объектно-ориентированный подход, то СУБД будет автоматически обрабатывать связи таблиц. В общем, из всех объектно-ориентированных возможностей вы, скорее всего, будете использовать только команду CREATE TYPE, которая создает новые типы данных.

---

## Установка инструментов Oracle

Система управления базой данных Oracle представляет собой сложный программный пакет и состоит из нескольких десятков приложений, каждый из которых включает десятки и даже сотни компонентов. Хотя данная система тесно интегрируется с операционной системой, Oracle старается поддерживать независимость от платформы, поэтому приложения, созданные в Oracle, будут запускаться на любой платформе с установленной системой Oracle. Основные элементы системы показаны на рис. 1.13. Администратор базы данных, АБД (Database Administrator — DBA), обычно отвечает за установку и управление системой баз данных, включая резервное копирование и восстановление. Разработчик приложений проектирует таблицы формы, отчеты и запросы базы данных, а также пишет код, отвечающий за процессы, затребованные пользователем. Сама по себе СУБД интегрируется с операционной системой, причем основной акцент делается на хранении и извлечении данных с дисков. СУБД имеет *процессор данных*, обрабатывающий большую часть реальных операций с компьютерной системой. В базе данных вся информация хранится в таблицах, а СУБД поддерживает *словарь данных*, с помощью которого определяются все таблицы и столбцы, хранимые в базе данных. Кроме того, СУБД имеет *подсистему безопасности*, определяющую пользователей и вводящую права доступа для контроля над действиями пользователей. Все компоненты процессора, словаря и системы безопасности устанавливаются автоматически. СУБД имеет *процессор запросов*, который с помощью языка PL/SQL определяет, как хранить и извлекать данные в ответ на запросы пользователей и приложений. Данная система является довольно мощной и оптимизирует извлечение данных. Кроме того, СУБД включает *средства администрирования*, помогающие АБД настраивать СУБД и следить за ее состоянием.

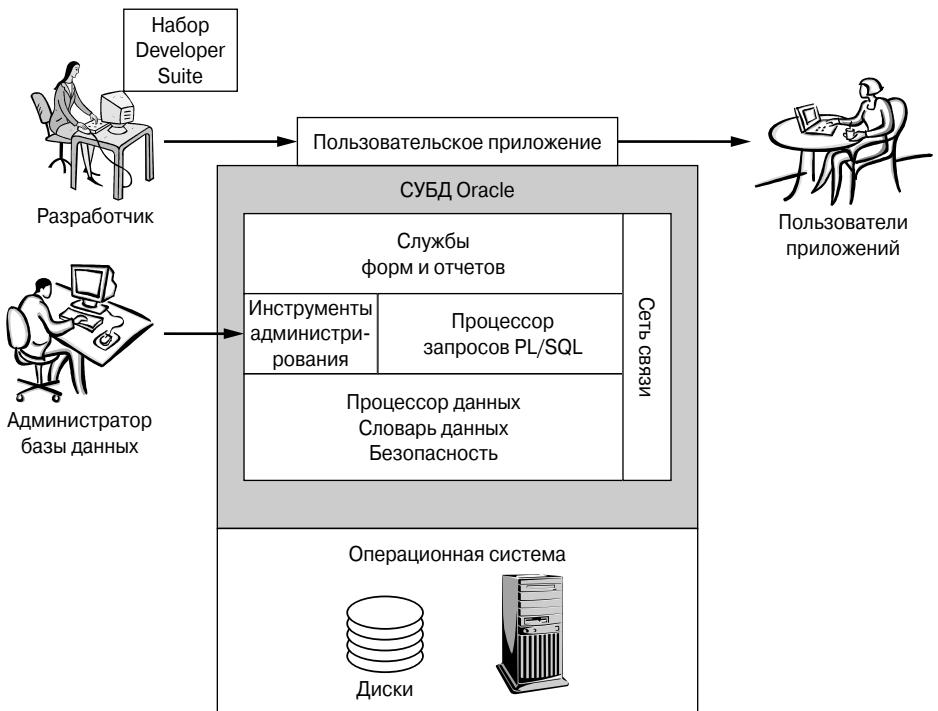


Рис. 1.13. Элементы СУБД

и производительностью. В Oracle эти средства состоят из нескольких запросов и пакетов, а также инструмента *Enterprise Manager* — графического средства, сводящего воедино несколько важных средств администрирования Oracle и предоставляющего АБД очень легкий доступ к ним. Вся система зависит от *сети связи*, соединяющей базу данных с пользователями и другими компьютерами. При установке базы данных следует аккуратно настроить систему, чтобы в сети Oracle она определялась единственным образом.

Отдельным компонентом СУБД являются *службы форм и отчетов*. Если вы устанавливаете СУБД на сервер для производственной среды, то необходимо отдельно установить службы форм и отчетов, являющиеся частью системы Oracle Enterprise. Фактически в большинстве организаций устанавливают службы форм и отчетов на отдельный компьютер. Впрочем, Oracle содержит и отдельный компонент служб форм и отчетов, который можно установить на тот же сервер, что и СУБД.

После установки компонентов разработчик приложения устанавливает набор инструментов *Developer Suite*. В большинстве случаев этот набор устанавливают на отдельном компьютере. Тем не менее допускается и установка СУБД Oracle и *Developer Suite* на одной машине. Запомните, что вы не должны пытаться установить *Developer Suite* на том же компьютере, на котором запущены службы форм и отчетов.

## Установка системы управления базой данных Oracle

Важно понимать, что инструменты Oracle устанавливаются несколькими блоками, а СУБД тесно связана с вашей операционной системой. Поскольку производители постоянно ужесточают безопасность операционных систем (Linux, Windows XP, Unix и др.), вы можете столкнуться с проблемами при установке программного обеспечения Oracle. В некоторых случаях вам придется отключать различные элементы систем безопасности. Например, многим инструментам Oracle мешают брандмауэры.

База данных Oracle 10g очень сильно зависит от протоколов Интернета, а это означает, что большинство инструментов запускается через Web-браузер. Кроме того, это означает, что большая часть “общения” основана на стандартных протоколах Интернета. Данный подход хорош тем, что стандартные протоколы Интернета поддерживают большинство систем. Единственным недостатком является то, что базовая безопасность Интернета — минимальна, поэтому производители, компании и образовательные учреждения добавляют свои средства безопасности. При типичных настройках по умолчанию многие из этих инструментов блокируют соединения Oracle. Например, формы и отчеты Oracle связываются с Enterprise Manager через определенные порты TCP/IP, которые обычно блокируются. Если вы устанавливаете программное обеспечение СУБД на несколько машин, кому-то придется так настроить брандмауэры, чтобы разрешить ряд соединений. Если вы устанавливаете СУБД и Developer Suite на собственный компьютер, проблем будет меньше. Тем не менее вам все равно придется открыть несколько портов или отключить внутренний брандмауэр.

Итак, вначале вам следует попробовать стандартную установку, переписать все соединения и порты, используемые системой, и проверить результаты установки. Если компонент не работает, попробуйте открыть нужный порт в брандмауэре. Если это не поможет — попробуйте отключить программное обеспечение брандмауэра.

Далее вам нужно установить на свой компьютер систему баз данных Oracle. Если вы используете существующую систему Oracle, то данные этапы проходить не нужно. Однако целесообразно ознакомиться с ними, чтобы лучше разбираться в установке системы баз данных.

Итак, для того чтобы установить систему управления базой данных Oracle, выполните следующие действия.

1. Найдите основной установочный диск или папку. При наличии высокоскоростного соединения Интернета текущие копии можно загрузить из сети Oracle Technology Network (<http://otn.oracle.com>). В любом случае вам придется зарегистрировать свою копию и принять условия Oracle.
2. Убедитесь, что на целевом компьютере имеется по крайней мере несколько гигабайтов свободного места, и начните установку основной базы данных, открыв нужную папку. Если потребуется, запустите файл `setup.exe`. Запустив программу установки, щелкните на кнопке `Install/Deinstall Products`.

3. Далее нужно заполнить поля основного экрана установки, но вы можете принять значения, установленные по умолчанию, если они подходят для вашего компьютера. Прежде всего введите *пароль* и повторите его в окне подтверждения. Запишите пароль и храните в безопасном месте — это системный пароль, который понадобится позже.
4. Проверьте папку, в которую устанавливается Oracle, и убедитесь, что на диске доступно по крайней мере 2 Гбайт свободного места (рис. 1.14). Выберите тип установки — возможно, единственным доступным выбором будет Personal Oracle. Если пожелаете, можете изменить глобальное имя базы данных (Global Database Name) на что-то более приятное, чем просто ORCL.
5. *Данный этап очень важен.* Перепишите всю информацию, которая отображается на экране, — она понадобится вам позже.
6. Щелкните на кнопке **Next** и просмотрите список устанавливаемых компонентов. Щелкните на кнопке **Install** и подождите несколько минут. После установки программного обеспечения СУБД и создания исходной базы данных вам предложат изменить пароль. На данном этапе это можно не делать, поскольку в текущий момент все учетные записи отключены. Щелкните на кнопке **OK**.
7. Откроется окно (рис. 1.15), в котором сообщается, что установка завершилась успешно. Кроме того, здесь указываются адреса и порты TCP/IP, требуемые некоторыми инструментами. Скопируйте данный список и запишите содержащуюся в нем информацию. Если вы устанавливаете СУБД в производственной сети, вам нужно будет передать данные специалистам по безопасности и работе сети, чтобы они могли открыть порты для данных утилит. Когда вы будете готовы, щелкните на кнопке **Exit**.
8. Далее система запустит Enterprise Manager. Если запуск пройдет некорректно, перезагрузите компьютер. Затем запустите браузер и используйте адреса, скопированные на предыдущем этапе. (*Подсказка:* `http://<server name>:5500/em`) После этого вы должны поместить на рабочем столе ярлык-закладку на этот сайт, поскольку она понадобится вам позже.
9. Теперь вам нужно войти в Enterprise Manager, используя имя пользователя `system` и пароль, введенный на этапе 3.
10. Создайте новую учетную запись для себя, поскольку использовать системную учетную запись неудобно. Щелкните на ссылке **Administration**, затем выберите **Users** в разделе **Security**. Щелкните на кнопке **Create**. Чтобы создать учетную запись нового пользователя, введите его имя и пароль, а затем щелкните на кнопке **OK**. Добавьте введенную информацию к своим записям.

В текущий момент в системе запущены основной процессор базы данных и система запросов. Кроме того, имеется учетная запись, с помощью которой вы можете



Рис. 1.14. Выбор установки Oracle



Рис. 1.15. Установка прошла успешно, но не забудьте скопировать все URL

создавать собственные таблицы и запросы. Помимо этого, вам необходимо установить набор Developer Suite, который предоставит инструменты для создания форм и отчетов.

## Удаление системы управления базой данных Oracle

Если в процессе установки произойдут существенные ошибки или если вы обнаружите, что нужно было задать другие опции, программное обеспечение придется переустанавливать. Процесс удаления программного обеспечения требует терпения и времени.

Итак, чтобы удалить систему баз данных Oracle с вашего компьютера, выполните нижеприведенные действия.

1. Вначале, используя диспетчер задач Windows, остановите все службы Oracle. В основном меню Windows выберите Start/All Programs/Administrative Tools/ Computer Management (Пуск/Все программы/Администрирование/Управление компьютером), затем раскройте узел Services (Службы) в узле Services and Applications (Службы и приложения), как показано на рис. 1.16, прокручивая список до тех пор, пока не найдете все запущенные службы Oracle. Для каждой найденной службы щелкните на кнопке Stop (Остановить). Затем щелкните правой кнопкой на элементе, выберите опцию Properties (Свойства) и измените тип запуска с Automatic (Авто) на Manual (Вручную). Перегрузите компьютер.
2. После запуска компьютера база данных Oracle работать не будет. В основном меню Windows выберите Start/All Programs/Oracle ... home/Oracle Installation Products/Universal Installer (Пуск/Все программы/Oracle ... home/Oracle Installation Products/Universal Installer). Щелкните на кнопке Deinstall Products и отметьте все отображенные элементы. На рис. 1.17 показана система с установленной базой данных и службами форм. Щелкните на кнопке Remove, чтобы начать процесс удаления. Если потребуется, подтвердите свое решение с помощью кнопки Yes и продолжайте процесс.
3. После удаления указанных продуктов закройте программу установки. Теперь необходимо вручную удалить файлы с компьютера. Используя программу Windows Explorer, дойдите до папки OracleHome и удалите ее полностью. Кроме того, вам нужно будет удалить всю папку C:\Program Files\Oracle.
4. Чтобы удалить все, нужно убрать несколько записей Oracle из системного реестра компьютера. Будьте очень осторожны и перепроверяйте все свои действия. Если вы удалите нужную запись, восстановить ее не удастся. Итак, в основном меню Windows выберите Start/Run (Пуск/Запуск программы), затем введите regedit и нажмите клавишу <Enter>. Далее, как показано на рис. 1.18, раскройте узел HKEY\_LOCAL\_MACHINE, дойдите до папки SOFTWARE, выделите запись ORACLE и нажмите клавишу <Delete>.

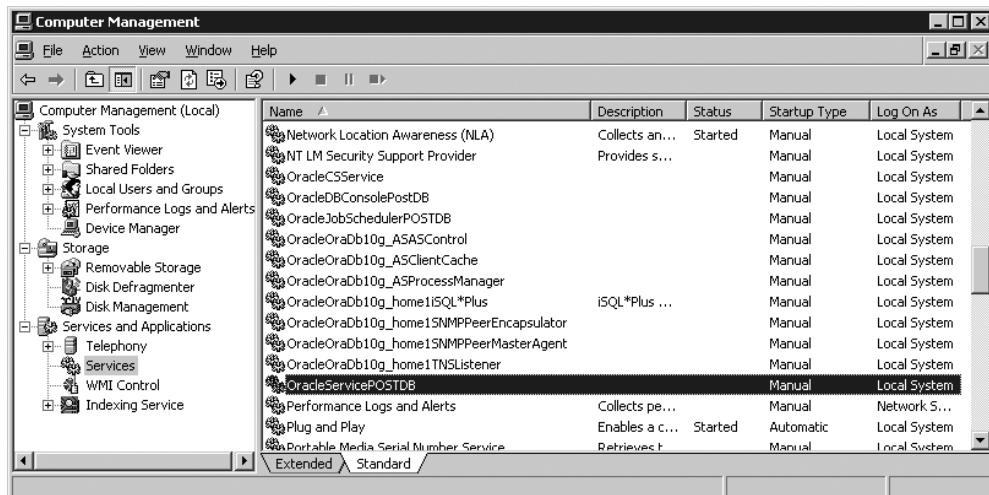


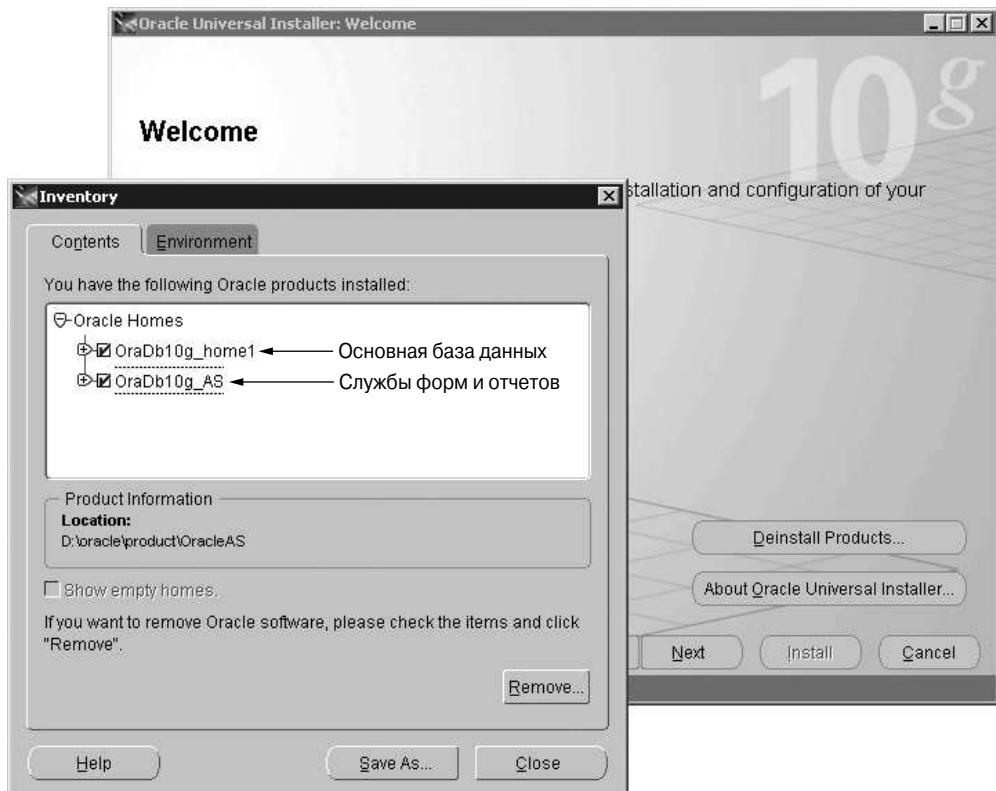
Рис. 1.16. Остановка службы Oracle

5. Нужно найти записи для служб и удалить их (они находятся в ветке HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Services и HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet002\Services). Помимо этого, следует удалить все службы, имеющие в названии слово “Oracle”: OracleOraDb10g\_home1TNSListener, Oracle.<dbname>, OracleDBConsole<dbname>, OracleOraDB10g\_home1iSQL\*Plus, OracleCSService, OracleDBConsole<dbname> и OracleService <dbname>. Для этого используйте средство Edit/Find (Правка/Найти) редактора реестра и удалите все записи, имеющие в названии слово “Oracle”. Закончив с этим, закройте редактор.
6. Желательно также удалить существующие записи из меню Start (Пуск). Откройте меню Start (Пуск), записи Oracle, щелкните на них правой кнопкой, а затем выберите опцию Delete (Удалить). Можно также удалить значение переменной среды Oracle. Для этого необходимо найти запись HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet002\Control\SessionManager\Environment в реестре или щелкнуть правой кнопкой мыши по значку компьютера и выбрать закладку Advanced (Дополнительно).

Теперь удалите большинство фрагментов данных, внесенных Oracle на ваш компьютер. Для надежности можно перегрузить компьютер. После этого можно вставить в привод диск базы данных Oracle и повторно установить все, что вам требуется.

## Установка инструментов Developer Suite

Инструменты Developer Suite необходимо устанавливать на компьютерах разработчиков. Их ни в коем случае нельзя устанавливать на компьютер, на котором запущены

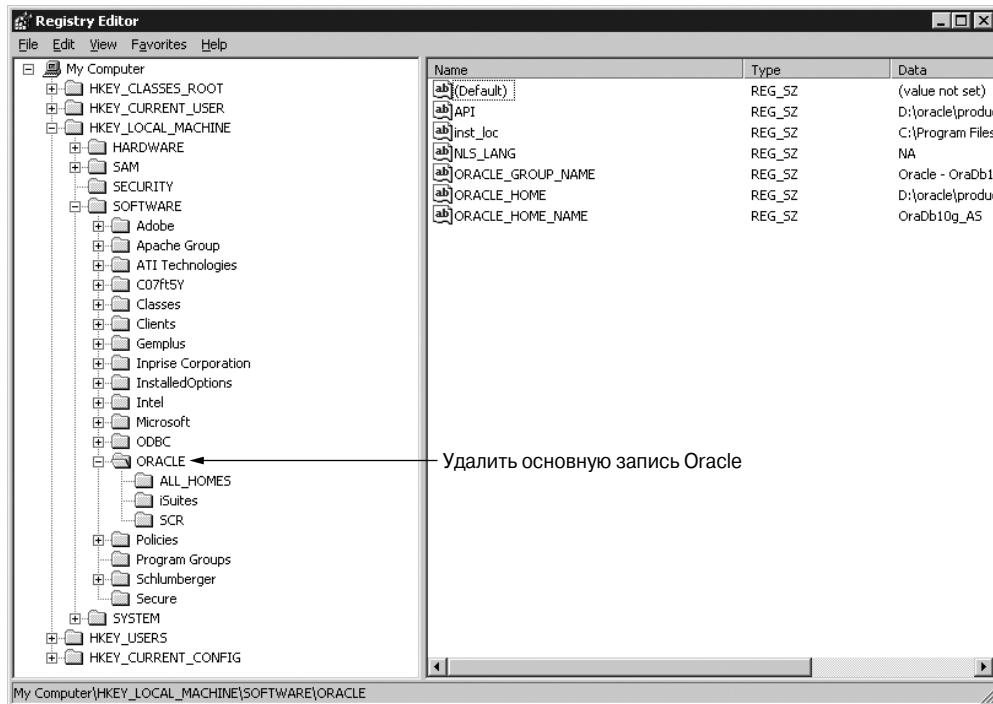


**Рис. 1.17.** Удаление продуктов Oracle

службы форм и отчетов Oracle Application Server. В идеальном случае указанные инструменты устанавливаются на машину с уже запущенной персональной системой Oracle; все, что вам нужно, — это добавить Developer Suite в другую папку.

Для того чтобы установить данный набор инструментов разработчика, выполните следующие действия.

1. Найдите и запустите первый диск Developer Suite. В программе установки щелкните на кнопке **Install**. После проверки компьютера отобразится экран приветствия. Щелкните на кнопке **Next**.
2. Установите положение папки Oracle Home (рис. 1.19). Если вы уже установили на компьютер базу данных, вам следует указать путь к новой папке (path) и новое название (Name) папки Developer Suite. Убедитесь, что на выбранном диске свободно по крайней мере 1 Гбайт места. После этого щелкните на кнопке **Next**.
3. Если объем свободного места ограничен, вы можете выбрать не все инструменты разработки. В любом случае вам потребуется набор Rapid Application Development,



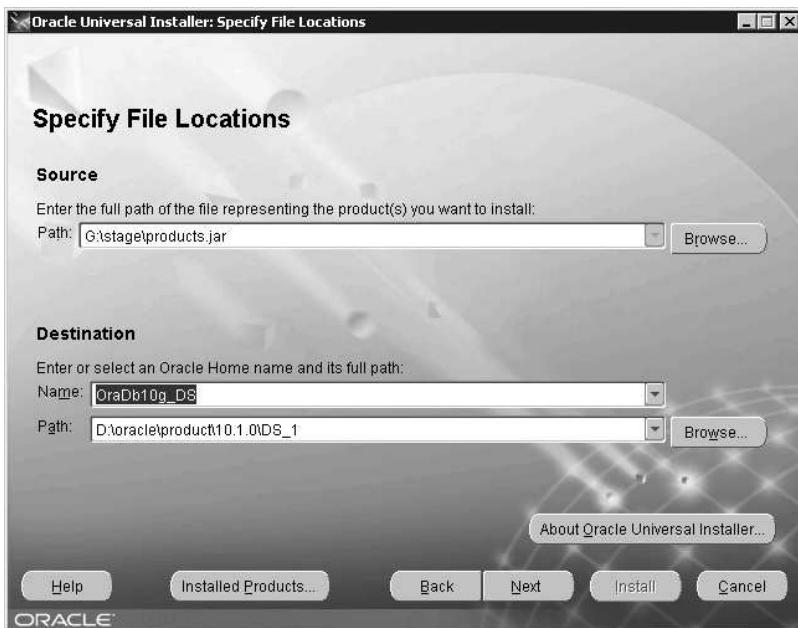
**Рис. 1.18.** Удаление записей Oracle из реестра

поэтому можете смело выбирать вариант *Complete*, поскольку для него требуется ровно столько же свободного места. Щелкните на кнопке *Next*.

4. Если вы желаете проверить возможность отправки отчетов по электронной почте, введите имя исходящего SMTP-сервера. Возможно, на этом этапе вам придется согласовать свои действия с администраторами, отвечающими за работу сети или безопасность. Пока же можете не заполнять это поле и щелкнуть на кнопке *Next*.
5. Проверьте список устанавливаемых компонентов и щелкните на кнопке *Install*. Подождите некоторое время и вставьте второй диск примерно после выполнения 70% установки. После завершения работы программы установки закройте ее.

Итак, теперь вы можете создавать формы и отчеты. В основном меню Windows появится несколько новых записей для Oracle, включая список элементов, связанных с Oracle Developer Suite.

Компоненты Oracle могут устанавливаться во множество различных конфигураций, поэтому рассмотреть все возможности довольно проблематично. Но один из вариантов заслуживает отдельного рассмотрения. Довольно часто первичную базу данных устанавливают на отдельный сервер. После этого на отдельные компьютеры устанавливают Developer Suite. Данный подход полезен из-за того, что если по какой-



**Рис. 1.19.** Задание уникального имени и пути

либо причине инструментальный компьютер выйдет из строя, то это не повлияет ни на что другое, кроме того, все могут совместно использовать одну базу данных. Однако если вы устанавливаете Developer Suite на каждом компьютере, вам придется дополнитель но выполнить ряд действий, а именно: настроить сеть Oracle на компьютере разработчика, чтобы инструменты “знали”, как найти сервер базы данных. Если речь идет о большой организации, установившей Oracle Internet Directory или Oracle Names Server, то эти действия можно не выполнять. Кроме того, описанную процедуру можно пропустить, если вы устанавливаете основное программное обеспечение базы данных на том же компьютере, что и программы разработчика.

Для того чтобы настроить сеть Oracle на компьютере разработчика, выполните такие действия.

1. Из меню Start (Пуск) Windows выберите Oracle Developer Suite/Configuration and Migration Tools/Net Manager. Раскройте узел ServiceNaming и посмотрите, введены ли какие-либо имена (рис. 1.20). Обычно в данном месте имеется только запись extproc. Ничего не делайте с ней, вам нужно будет добавить еще одну запись, указывающую местоположение сервера.
2. Выделив узел ServiceNaming, щелкните на кнопке Create (+) и введите имя службы для сервера. Оно может быть любым, но желательно, чтобы оно ассоциировалось у вас с сервером (в нашем примере используется имя LabDB). Далее щелкните на кнопке Next.

3. В окне Protocol оставьте TCP/IP-соединение, заданное по умолчанию, и щелкните на кнопке Next.
4. В третьем окне введите имя сервера, содержащего базу данных. Возможно, данное имя вы получите от системного администратора, но чаще всего это сетевое имя компьютера, присвоенное в Windows. Примите номер порта по умолчанию (1521) и щелкните на кнопке Next.
5. Теперь необходимо ввести глобальное имя базы данных, которое было присвоено на этапе 4 установки базы данных. Значение этого параметра по умолчанию равно ORCL, однако, как показано на рис. 1.21, вы можете заменить его каким-то более значимым, а затем щелкнуть на кнопке Next.
6. *Данный этап очень важен.* Вам предлагается протестировать введенные значения. Щелкните на кнопке Test. Вначале программа попытается использовать регистрационное имя по умолчанию, однако данная учетная запись (scott) сейчас заблокирована, поэтому первый тест не будет пройден. Щелкните на кнопке Change Login и введите собственное имя пользователя и пароль. Затем снова щелкните на кнопке Test.
7. После того как тестовый процесс успешно соединится с базой данных, закройте тестовую форму и щелкните на кнопке Finish. Вернитесь к основному окну Net Manager и воспользуйтесь File/Save Network Configuration, чтобы сохранить новое соединение. После этого закройте окно Net Manager.
8. Если в ходе теста не удастся соединиться с базой данных, вы можете повторно ввести свою учетную запись или системную учетную запись. Если все остается по-прежнему, то это, скорее всего, связано с тем, что имя сервера базы данных было введено неверно. Возможно, это связано с тем, что брандмауэр или сеть блокирует доступ к серверу, и вам потребуется помочь сетевого администратора.

Итак, теперь ваш компьютер способен запускать инструменты разработки Oracle и соединяться с базой данных.

## Корпоративные службы форм и отчетов

Поскольку Application Server и Forms and Reports Services для работы с данной книгой вам не нужны, они не включены на компакт-диск. Тем не менее Application Server и Forms and Reports Services могут оказаться полезными, если нужно попрактиковаться в реальной разработке. Если вы желаете поэкспериментировать с данными продуктами, можете загрузить их из OTN. Однако для этого вам потребуются минимум два компьютера (а скорее всего — три). Если вы желаете запустить полную версию Application Server, вам потребуется установить корпоративную базу данных на один сервер, Application Server — на другой, а Developer Suite — на отдельную рабочую станцию. Если же вы располагаете только одним сервером и рабочей стан-



**Рис. 1.20.** Конфигурирование сети Oracle

цией, но хотите поэкспериментировать с сервером приложений, установите Developer Suite на свою рабочую станцию. После этого установите базу данных на сервер, затем установите службы форм и отчетов. Основные этапы установки служб форм и отчетов для данного случая описаны ниже.

Для того чтобы установить службы форм и отчетов, выполните нижеприведенные действия.

1. Загрузите из OTN отдельный пакет Forms and Reports Services. Будьте внимательны на сайте, Application Server состоит не только из Forms and Reports Services, но сейчас вам нужен только этот продукт. Разархивируйте файлы во временную папку и запустите `setup.exe`. В окне Welcome щелкните на кнопке `Next`.
2. Службы форм и отчетов вам нужно поместить в отдельную папку, поэтому в нужном поле введите соответствующий путь. Проверьте, чтобы на выбранном диске было доступно не менее гигабайта свободного места, и введите имя, например `OraDB10g_AS`. Чтобы начать установку, щелкните на кнопке `Next`.
3. Далее вы можете добавить языки к серверу форм и отчетов. Если вы желаете поэкспериментировать с приложением, которое поддерживает несколько языков (или



**Рис. 1.21.** Ввод глобального имени базы данных

вам это необходимо), добавьте требуемые языки. В противном случае оставьте настройки, заданные по умолчанию, и щелкните на кнопке **Next**.

4. Теперь необходимо дать название экземпляру базы данных — введите **Ora10gAS**. Кроме того, требуется ввести еще один пароль администратора. Запишите имя пользователя (**ias\_admin**) и выбранный вами пароль. Щелкните на кнопке **Next**.
5. Если вам требуется, чтобы сервер отправлял отчеты пользователям по электронной почте, следует ввести имя исходящего SMTP-сервера, которое вы можете узнать у сетевого администратора. Кроме того, вам, возможно, потребуется обратиться к сотруднику, отвечающему за безопасность, который настроит сеть так, чтобы вы могли использовать сервер. Пока что оставьте это поле пустым — вы сможете его заполнить позже в любой момент. Щелкните на кнопке **Next**, а затем — на кнопке **Install** и подождите несколько минут.
6. После завершения установки на экране отобразятся два URL (рис. 1.22). Запишите или запомните их. Порт 80 — это стандартный Web-порт. Для того чтобы проверить его, свяжитесь из браузера с сервером. На экране должна появиться стандартная информационная страница. Другой URL задает использование порта 1810 протокола TCP/IP для доступа к Enterprise Manager на сервере приложений. Данный адрес применяется для настройки опций для служб форм и отчетов (используйте имя **ias\_admin** и пароль, введенный на этапе 4).

Службы Application Services запускаются в каком-то смысле независимо и имеют собственные системы управления и наблюдения. На первый взгляд, настройка сложная, однако в конце концов, если вы желаете использовать формы и отчеты в производственной среде, данные службы вам понадобятся.



**Рис. 1.22.** Скопируйте и сохраните конфигурацию Forms and Reports Services

## Изучение среды Oracle

Установив системы Oracle, вы, скорее всего, пожелаете с ними поэкспериментировать, чтобы понять, что они могут. Тем не менее, как будет показано ниже, системы баз данных требуют, чтобы вы аккуратно определили и создали таблицы до того, как начнете делать все остальное. Однако немногого “пощупать” Oracle все же необходимо, чтобы разобраться с инструментами и возможностями этой базы данных. Поэтому в данном разделе мы рассмотрим относительно простые задачи, используя одну небольшую таблицу. Из данных примеров вы не почувствуете всю мощь системы баз данных, однако они помогут вам понять общую организацию некоторых инструментов.

### SQL\*Plus и iSQL\*Plus

Лучший способ знакомства с двумя интерфейсами Oracle, SQL\*Plus и iSQL\*Plus, — это их использование. Прежде всего вам необходимо создать таблицу, а затем загрузить в нее какие-то данные. Поскольку мы не собираемся превращать изучение Oracle в практикум по набору текста, данный этап сделает за нас SQL-сценарий.

(Самостоятельно писать SQL-выражения вы научитесь, прочитав последующие главы, в которых речь пойдет о создании таблиц и их заполнении информацией.) Для запуска сценария используется популярный интерфейс баз данных Oracle, именуемый SQL\*Plus. Для начала найдите в папке Ch01Data на компакт-диске файл данных о студентах (Ch01Explore.sql). Запишите полный путь к файлам данных.

1. Запустите SQL\*Plus из основного меню Windows. Путь к нему показан на рис. 1.23. Найдите домашнюю папку Oracle, затем Application Development и выберите SQL Plus.
2. Откройте базу данных. На рис. 1.24 показан стандартный экран регистрации. Если вы запустили базу данных на своем компьютере, вам потребуются имя пользователя и пароль, созданные после установки СУБД. Кроме того, вам понадобится имя вашего компьютера, которое вы вводите в поле Host String. Если Oracle устанавливали не вы, спросите у администратора, какими должны быть три указанные строки. Введите их и щелкните на кнопке OK.
3. После запуска SQL\*Plus отображается пустое окно. Далее вам нужно запустить сценарий SQL, написанный для данной главы. Для того чтобы узнать, каково точное положение диска, используйте Windows Explorer. В SQL\*Plus введите

```
@<путь>\Ch01Explore
```

Вместо <путь> введите требуемый путь. Пример команды и результат ее выполнения показаны на рис. 1.25. Данный сценарий создает таблицу Clients и загружает в нее шестнадцать имён. Таблица состоит из пяти столбцов: ClientID, FirstName, LastName, Category и Balance. Столбец Category предоставляет данные о роде занятий клиента (Builder, Banker и т.д.); Balance — это сумма, которую должен клиент за услуги, предоставленные ранее.

4. Чтобы просмотреть данные, введите команду `SELECT * FROM Clients;` и нажмите клавишу <Enter>. Не забудьте ввести точку с запятой в конце выражения. После выполнения команды вы увидите строки данных, которые, скорее всего, будут запутанными, поскольку SQL\*Plus переносит длинные строки на новую строку.
5. Выйдите из Oracle и SQL\*Plus, набрав `Exit` и нажав клавишу <Enter>.

Кроме того, Oracle содержит более графически-ориентированный инструмент запросов iSQL\*Plus, обладающий дополнительной возможностью запуска через Интернет с помощью обычного браузера. Данный инструмент недоступен из стандартного меню, для его запуска нужно соединиться с сервером через Web-браузер.

Для того чтобы запустить iSQL\*Plus и получить доступ к базе данных Oracle, выполните следующие действия.

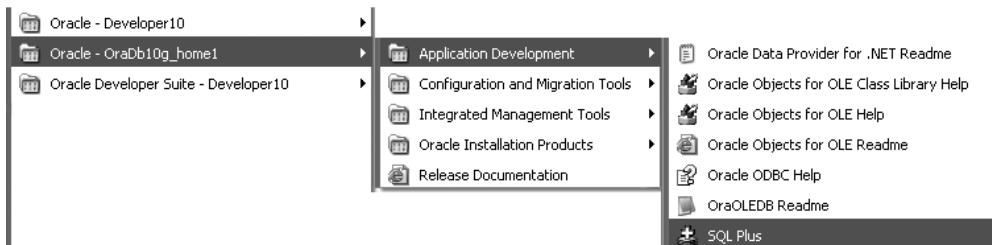


Рис. 1.23. Запуск SQL\*Plus

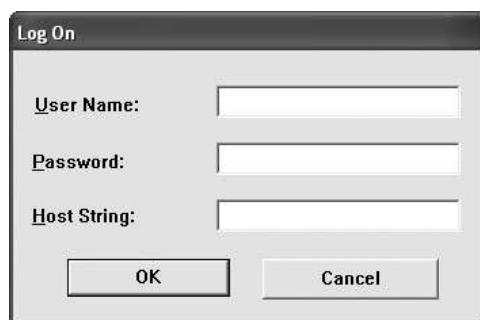


Рис. 1.24. Стандартный экран регистрации

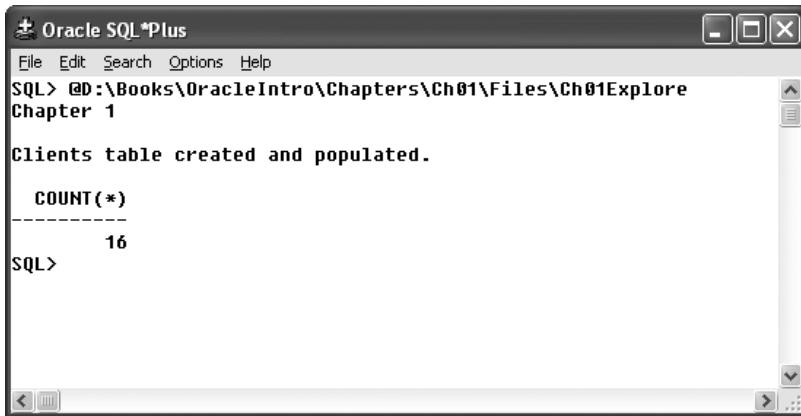


Рис. 1.25. Запуск сценария настройки

1. Запустите Web-браузер. В адресной строке введите URL сервера iSQL, который вы вводили при установке базы данных. Если вы не изменяли значения, установленные по умолчанию, то адрес будет следующим (вместо <server name> подставьте имя своего компьютера):

`http://<server name>:5560/sqlplus`

**Совет.** Если вы не можете определить имя своего компьютера, используйте URL, приведенный ниже. Если вы устанавливали базу данных на свой компьютер, это почти наверняка сработает.

`http://127.0.0.1:5560/sqlplus`

(Данный вариант *не сработает*, если вы обращаетесь к удаленному серверу. В таком случае спросите у администратора URL для доступа к iSQL\*Plus.)

2. Введите имя пользователя Oracle в текстовом поле `Username` и пароль — в поле `Password`. Поле `ConnectIdentifier` заполнять или изменять не нужно. Щелкните на кнопке `Login`, чтобы открыть окно `Workspace`.
3. Щелкните в текстовом окне `Workspace` и введите следующую информацию. В конце строки клавишу `<Enter>` нажимать не обязательно (в отличие от SQL\*Plus):

```
SELECT * FROM Clients;
```

4. Щелкните на кнопке `Execute`, чтобы выполнить команду. Далее Oracle извлечет 16 строк из таблицы `Clients` и отобразит их (рис. 1.26).

На рис. 1.26 показана команда и результаты ее выполнения. Обратите внимание на то, что браузер так форматирует таблицу, чтобы были лучше видны ее строки и столбцы. Практически для всех примеров, рассмотренных в книге, вы можете использовать SQL\*Plus либо iSQL\*Plus. Мы предпочтаем использовать для иллюстрации SQL\*Plus, поскольку данный инструмент более распространен и используется на различных платформах (а не только в Windows). В любом случае оба инструмента поддерживают стандартные запросы. Основные недостатки SQL\*Plus состоят в том, что вам нужно установить его на компьютере, а для получения читаемых результатов часто требуется дополнительное форматирование. С другой стороны, хотя iSQL\*Plus очень легко запустить, этот инструмент не поддерживает все конфигурационные опции, доступные в SQL\*Plus. Большинство пользователей по-прежнему используют SQL\*Plus, поскольку данный инструмент давно представлен на рынке и люди просто научились его использовать.

Для того чтобы удалить таблицу, созданную ранее с помощью SQL-выражения, выполните нижеприведенные действия.

1. Очистите окно `Workspace`, щелкнув на кнопке `Clear`, которая находится с правой стороны окна, сразу под разделительной линией. Текстовое окно `Workspace` очищается, и iSQL\*Plus удаляет результаты запроса.
2. Щелкните в текстовом окне `Workspace` и введите приведенную ниже строку. Затем щелкните на кнопке `Execute`, чтобы выполнить запрос.

```
DROP TABLE Clients PURGE;
```

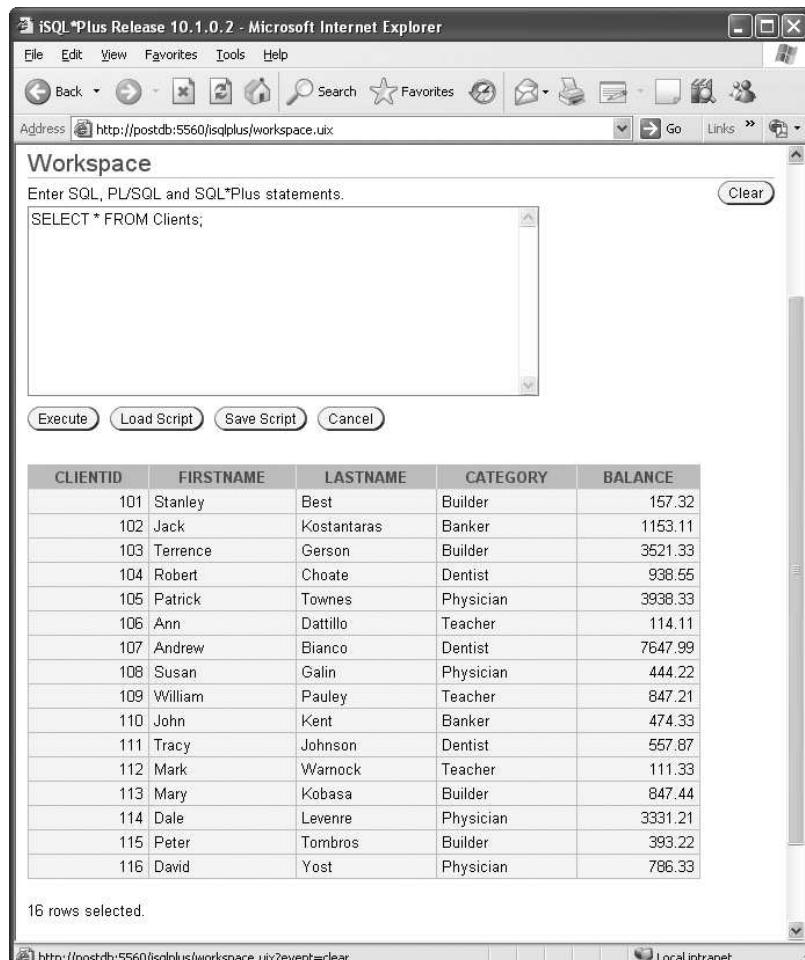


Рис. 1.26. Запуск простого запроса с iSQL\*Plus

- Поскольку в текущее время вы работаете с iSQL\*Plus, щелкните на ссылке, чтобы выйти из Oracle.
- Закройте браузер.

## Инструменты Developer Suite

Двумя основными инструментами Developer Suite являются Forms Builder и Reports Builder. Оба они предлагают множество возможностей, изучение которых требует некоторого времени. Тем не менее рассмотреть простую форму отчета в любом случае полезно, так как это поможет вам лучше понять их. Единственная проблема

связана с тем, что запуск обоих приложений требует выполнения ряда определенных действий, в ходе которых могут возникнуть сложности, решить которые на данном этапе вам будет сложновато. Поэтому пока довольствуйтесь предлагаемым рисунком.

На рис. 1.27 показана простая форма, которую можно использовать для просмотра и ввода данных в таблицу *Clients*. Поскольку мы работаем с единственной таблицей, форма выглядит относительно простой. Вы можете прокручивать строки данных, щелкнув на кнопках *NextRecord* (▶) и *PreviousRecord* (◀) расположенных на панели инструментов. Если вы изменили данные, щелкните на кнопке *Save* (💾), чтобы записать изменения в базу данных. Вы можете даже использовать форму как инструмент запроса. Щелкните на кнопке *Enter Query* (🔍), введите значения в форму (например, *Banker* в окне *Category*), а затем щелкните на кнопке *Execute Query* (\_EXECUTE). После этого в форме будут отображены только записи, согласующиеся с введенными значениями. Более сложные формы умеют рассчитывать суммы и использовать списки подстановок, облегчающие пользователям ввод данных.

На рис. 1.28 показан простой отчет, основанный на данных из таблицы *Clients*. Данный отчет является сгруппированным, поскольку он рассчитывает сумму для всех позиций столбца *Category* (*Banker*, *Builder* и т.д.). В разделе подробной информации указывается баланс для каждого клиента из заданной категории. Большую часть работы по созданию данного отчета выполнил инструмент *Report Wizard*. Полученную форму можно распечатать или просмотреть в Web-браузере. Для печатных отчетов Oracle позволяет генерировать PDF-файл, который можно отправить по электронной почте, чтобы пользователи просмотрели его в электронном виде или распечатали.

Ниже мы расскажем, как с помощью различных мастеров создавать полезные формы и отчеты. Вы также узнаете, как редактировать формы и отчеты, настраивая их и добавляя различные полезные функции. Если вам интересно, можете изучить пример формы (*Clients.fmb*) и отчета (*ClientBalances.jsp*), записанные на компакт-диск (откройте папку, соответствующую данной главе). Щелкнув на любом из указанных файлов, вы откроете его в соответствующем редакторе Oracle. Вы можете легко запустить отчет, чтобы посмотреть, как это выглядит, однако запуск форм требует несколько больше усилий, и вам сейчас нет смысла тратить время на изучение деталей этого процесса. Подробно все шаги рассмотрены в главе 8, где вы сможете поработать с отчетами в свое удовольствие.

## Oracle Enterprise Manager

Средства Forms Builder и Reports Builder используют преимущественно разработчики приложений. Результаты их работы используются конечными пользователями. Кроме того, разработчики применяют инструменты запросов SQL\*Plus или iSQL\*Plus. Пользователям редко предлагается прямой доступ к инструментам запроса. Если даже вы ограничите деятельность пользователей выполнением простых запросов со

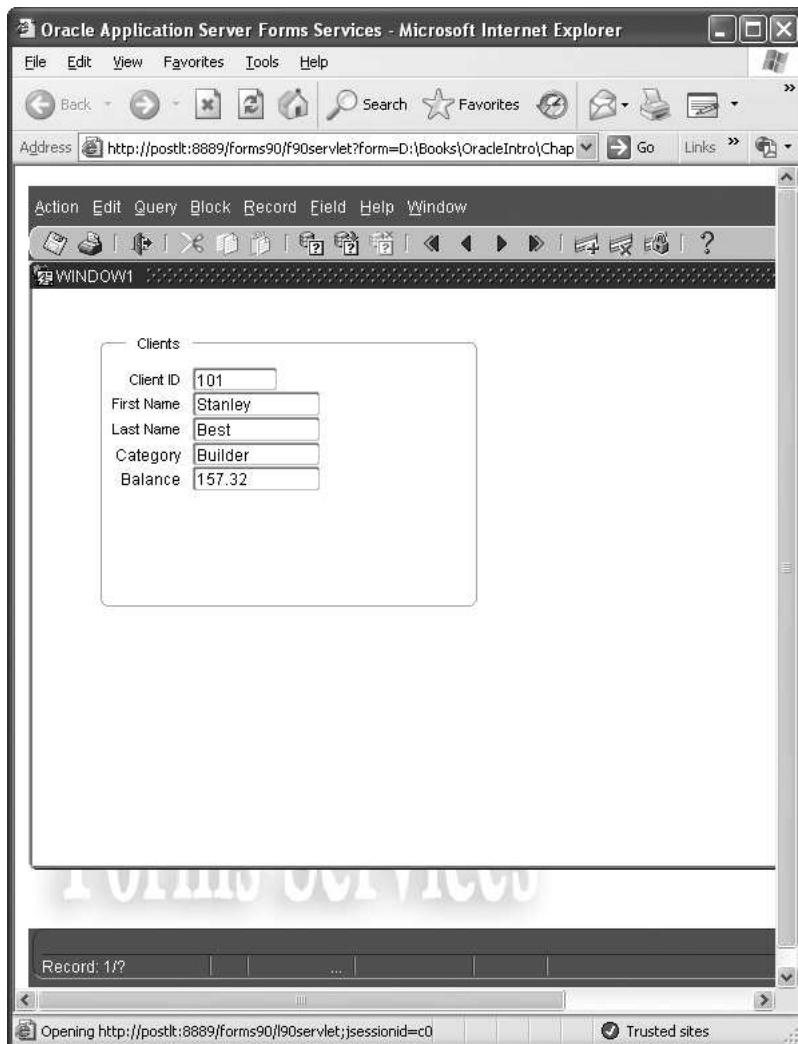


Рис. 1.27. Простая форма для таблицы Clients

статусом “только чтение”, вам придется долго объяснять им синтаксис SQL, и всегда будет существовать вероятность того, что они неверно интерпретируют результаты запроса. Несмотря на это, некоторые продвинутые пользователи могут потребовать предоставить им доступ к данным инструментам. Инструменты запроса интенсивно используют и администраторы базы данных. Практически любую задачу, с которой сталкивается АБД, можно решить с помощью специального запроса или пакета. Тем не менее изучение функций всех этих инструментов требует времени, к тому же администраторы обычно должны запоминать команды, поскольку система запросов не предлагает никаких подсказок.

## Client Balances

Category	Client ID	First Name	Last Name	Balance
Banker	110	John	Kent	\$474.33
	102	Jack	Kostantaras	\$1,153.11
	<b>Total:</b>			<b>\$1,627.44</b>
Builder	101	Stanley	Best	\$157.32
	103	Terrence	Gerson	\$3,521.33
	113	Mary	Kobasa	\$847.44
	115	Peter	Tombros	\$393.22
	<b>Total:</b>			<b>\$4,919.31</b>
Dentist	107	Andrew	Bianco	\$7,647.99
	104	Robert	Choate	\$938.55
	111	Tracy	Johnson	\$557.87
	<b>Total:</b>			<b>\$9,144.41</b>
Physician	108	Susan	Galin	\$444.22
	114	Dale	Levenre	\$3,331.21
	105	Patrick	Townes	\$3,938.33
	116	David	Yost	\$786.33
Teacher	<b>Total:</b>			<b>\$8,500.09</b>
	106	Ann	Dattillo	\$114.11
	109	William	Pauley	\$847.21
	112	Mark	Warnock	\$111.33
<b>Total:</b>				<b>\$1,072.65</b>
<b>Total:</b>				<b>\$25,263.90</b>

**Рис. 1.28.** Простой отчет для таблицы Clients

Чтобы помочь администратору базы данных отслеживать ее активность и выполнять стандартные задачи, было создано средство Enterprise Manager, которое представляет собой графический интерфейс, запускаемый в браузере через Интернет. Следовательно, его относительно просто использовать (даже начинающему администратору), и он позволяет контролировать базу данных из удаленных мест. Основным недостатком является то, что Enterprise Manager может использовать только администратор базы данных. Если вы установили копию СУБД Oracle на собственный компьютер, то сможете использовать учетную запись System, которая выполняет роль администратора базы данных. Если вы ограничены доступом к общему компьютеру в лаборатории, администратор вряд ли предоставит всем желающим такие же права, какие имеет сам, поэтому в таком случае вы не сможете запустить примеры, предполагающие использование Enterprise Manager.

Если же вы располагаете правами администратора, то запустить Enterprise Manager будет достаточно просто. Будьте аккуратны и не пытайтесь ничего менять, если не уверены в том, что делаете. Некоторые изменения можно будет исправить только путем переустановки системы баз данных.

Итак, чтобы запустить Enterprise Manager, выполните следующие действия.

1. Определите порт TCP/IP, настроенный при установке системы. Запустите Web-браузер и введите URL в адресную строку. Ниже приводится значение по умолчанию (введите имя сервера без угловых скобок).

`http://<server name>:5500/em`

2. После запроса введите имя пользователя и пароль. Отобразится домашняя страница Enterprise Manager, содержащая основную информацию о состоянии базы данных.
3. Обратите внимание на основные закладки вверху окна браузера (Home, Performance, Administration и Maintenance) — они предоставляют доступ к большинству функций данного средства. На рис. 1.29 показана открытая страница Administration. Опции группируются по основной задаче (например, Storage или Security).
4. Щелкнув на ссылке *Users* под опцией *Security*, вы можете добавлять пользователей базы данных или модифицировать права доступа существующих пользователей. Если вы установили Oracle на собственный компьютер, можете создать новую учетную запись для себя, чтобы вам не приходилось постоянно использовать запись *System*. Другие закладки предлагают еще больше средств управления и ссылок, демонстрирующих огромное количество задач, которые может выполнять администратор базы данных. Повторим еще раз: не вносите изменения, о которых потом придется сожалеть. Учетная запись администратора обладает значительными привилегиями, поскольку администратору нужен полный доступ, позволяющий настраивать базу данных и помогать пользователям. Но в то же время данные привилегии могут нанести серьезный урон базе данных.
5. Щелкните на ссылке *Logout* вверху страницы браузера и закройте его, пока вы всех не напугали.

## Базы данных, используемые в книге

Лучший способ научиться использовать системы баз данных — это создать несколько простых баз и поэкспериментировать с ними. Для иллюстрации различных концепций мы будем использовать вымышленное агентство недвижимости Redwood Realty. Небольшой офис данной компании, в котором работают двадцать девять профессиональных агентов и вспомогательных сотрудников, расположен в г. Арката (Калифорния) и обслуживает округ Хамбольд (на сервере штата Калифорния). В данной главе мы с помощью подробных примеров покажем, как создавать элементы приложения для данного агентства недвижимости. В конце главы представлены четыре практических упражнения. Первое упражнение в каждой главе связано с примером, рассмотренным в главе (агентство Redwood Realty); от вас будет требоваться развить результат, полученный в тексте, т.е. добавить новые функции, закрепить изученный

**Administration**

Instance	Storage	Security	Enterprise Manager Administration
<a href="#">Memory Parameters</a>	<a href="#">Controlfiles</a>	<a href="#">Users</a>	<a href="#">Administrators</a>
<a href="#">Undo Management</a>	<a href="#">Tablespaces</a>	<a href="#">Roles</a>	<a href="#">Notification Schedule</a>
<a href="#">All Initialization Parameters</a>	<a href="#">Datafiles</a>	<a href="#">Profiles</a>	<a href="#">Blackouts</a>
	<a href="#">Rollback Segments</a>		
	<a href="#">Redo Log Groups</a>		
	<a href="#">Archive Logs</a>		
	<a href="#">Temporary Tablespace Groups</a>		
Schema	Warehouse		
<a href="#">Tables</a>	<a href="#">Packages</a>	<a href="#">Array Types</a>	<a href="#">Cubes</a>
<a href="#">Indexes</a>	<a href="#">Package Bodies</a>	<a href="#">Object Types</a>	<a href="#">OLAP Dimensions</a>
<a href="#">Views</a>	<a href="#">Procedures</a>	<a href="#">Table Types</a>	<a href="#">Measure Folders</a>
<a href="#">Synonyms</a>	<a href="#">Functions</a>		<a href="#">Dimensions</a>
<a href="#">Sequences</a>	<a href="#">Triggers</a>		<a href="#">Materialized Views</a>
<a href="#">Database Links</a>	<a href="#">Java Sources</a>		<a href="#">Materialized View Logs</a>
	<a href="#">Java Classes</a>		<a href="#">Refresh Groups</a>
Configuration Management	Workload	Resource Manager	Scheduler
<a href="#">Last Collected Configuration</a>	<a href="#">Automatic Workload Repository</a>	<a href="#">Resource Monitors</a>	<a href="#">Jobs</a>
<a href="#">Database Usage Statistics</a>	<a href="#">SQL Tuning Sets</a>	<a href="#">Resource Consumer Group Mappings</a>	<a href="#">Schedules</a>
		<a href="#">Resource Consumer Groups</a>	<a href="#">Programs</a>
		<a href="#">Resource Plans</a>	<a href="#">Job Classes</a>
			<a href="#">Windows</a>
			<a href="#">Window Groups</a>
			<a href="#">Global Attributes</a>

**TIP** Use the Enterprise Manager 10g Java Console to manage Streams, Advanced Replication, Advanced Queues, XML Database, Spatial and Workspace.

**Рис. 1.29.** Закладка Administration в Enterprise Manager

материал и т.д. Остальные три упражнения требуют применения полученных знаний для создания или расширения объектов базы данных для трех различных примеров.

Основная цель настоящей книги — показать, как создаются конкретные элементы, с которыми вы, скорее всего, столкнетесь при работе с реальными базами данных. В дальнейшем, когда вы будете реализовывать реальные проекты, приведенные примеры помогут создать основу практически для любой СУБД. В данном разделе кратко разбираются примеры, приведенные в конце главы. Для всех из них написаны сценарии SQL, с помощью которых вы можете создавать и загружать данные в таблицы и инициализировать все элементы баз данных. Данные сценарии разбиты по главам и представлены на прилагаемом к книге компакт-диске. Для некоторых глав на диске представлены дополнительные файлы — примеры форм и отчетов. Мы предполагаем, что вы скопировали данные файлы на жесткий диск, создав общую папку примеров с подпапками для каждой главы.

## Агентство Redwood Realty

Агентство Redwood Realty будет рассматриваться во всех главах книги и являться связующей нитью, используемой для изучения общих концепций баз данных, а также понятий, характерных только для Oracle. В каждой главе мы подробно опишем использование базы данных Redwood Realty. Как вы знаете<sup>1</sup>, агентства недвижимости помогают людям продавать и покупать квартиры, взимая за свои услуги комиссионные. По договоренности агенты по недвижимости взимают комиссионные с продавца, а не с покупателя недвижимости. Задействовать агентство недвижимости выгодно по нескольким причинам. Основным из них является полное юридическое сопровождение сделки: агенты по недвижимости знают все нормы и законы, регулирующие операции с недвижимостью, и могут подсказать своим клиентам, как выбраться из моря бумажной работы, связанной с продажей/покупкой жилья. Агентами являются те сотрудники, которые сдали соответствующий экзамен и получили лицензию на право работы. Данная лицензия позволяет продавать недвижимость в штате, в котором она была получена (в нашем случае — в Калифорнии). Агенты связываются с потенциальными продавцами и отслеживают историю похожих продаж, чтобы установить стартовую цену дома или другого имущества. Дом помечается как выставленный на продажу, и его вводят в региональную базу данных MLS (Multiple Listing Service). Агент, работающий с продавцом, является *агентом по продаже*. Потенциальные покупатели могут связываться с данным агентом по продаже или нанять своего агента (возможно, из другой компании). Когда любой из агентов, работающих на агентство Redwood Realty, выставляет дом на продажу, он должен ввести описание в базу данных компании. Кроме того, агенты должны отслеживать информацию о клиенте.

На рис. 1.30 показана схема взаимосвязи восьми таблиц, составляющих базу данных Redwood Realty. Каждый прямоугольник обозначает таблицу, имя которой указано в верху в затененной области. Первичный ключ (или ключи) таблицы указан ниже имени и подчеркнут. Столбцы первичного ключа помечены слева буквами “PK” (сокращение от Primary Key). Под ними с пометкой “FK” (сокращение от Foreign Key) указаны внешние ключи. Некоторые первичные ключи также являются внешними (см., например, таблицу CustAgentList на рис. 1.30). В таком случае столбец помечается и буквами “PK”, и буквами “FK”. (Никакого особого смысла в цифрах после букв “FK” нет — это просто способ отслеживания удаленных внешних ключей.) К рис. 1.30 вам придется обращаться довольно часто, поэтому рекомендуем вложить в книгу закладку.

Кратко говоря, значение таблиц заключается в следующем. Таблица ContactReason содержит три столбца, в которых клиенты Redwood Realty рассортированы по

---

<sup>1</sup>Ниже изложены принципы операций с недвижимостью, принятые в США. Поскольку настоящая книга посвящена базам данных Oracle, мы не будем акцентировать внимание на отличиях законодательств разных стран в вопросах, касающихся покупки/продажи квартир. — Примеч. ред.

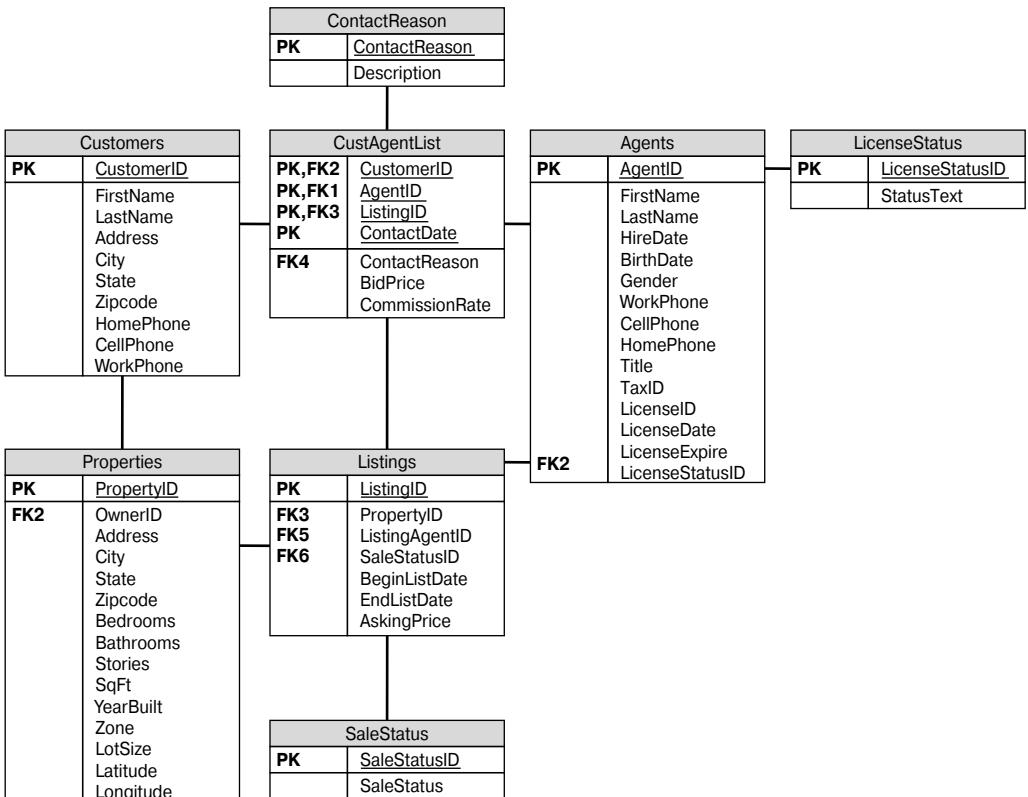


Рис. 1.30. Диаграмма базы данных Redwood Realty

причинам контакта с фирмой: Buy (“покупка”), Sell (“продажа”) или Casual (“случайно”). В таблице *Customers* перечислено 2 500 покупателей и продавцов, имевших дела с Redwood Realty за последнее время. Данные включают всю необходимую контактную информацию. В таблице *Agents*, состоящей из 29 строк, перечислена информация об агентах, работающих на Redwood Realty. В таблице *Listings* каждая из 502 строк представляет объект, выставленный на продажу. Здесь указывается агент по продаже и описание дома, а также дата начала и конца продажи. В данной таблице указывается цена, запрошеннная владельцем (*AskingPrice*). Таблица *Properties* содержит всю информацию о 2 000 объектах, перечисленных в таблице *Listings* и таблицах *Customers*. Не все дома, указанные в таблице *Properties*, выставлены на продажу — только 502 из них. Остальные 1 492 являются недвижимостью в округе Хамбольд. Помимо типичной информации об объектах, каждая запись содержит данные, имеющиеся в архивах округа, — полный метраж, жилая площадь, год постройки, район, серия, число комнат, санузлов, а также широта и долгота! Крошечная таблица *SaleStatus* содержит всего три строки: For Sale (“на продажу”).

жу”), Pending (“в процессе”) и Sold (“продано”). Более полезная в форме таблицы, на которую ссылаются раскрывающиеся списки формы, SaleStatus предлагает постоянный контроль за состоянием объектов в таблице Listings (ссылающейся на SaleStatus). Таблица LicenseStatus содержит 16 строк, описывающих различные условия на лицензии для работы с недвижимостью в Калифорнии. Состояние может изменяться от “Licensed” (“есть лицензия”) до “Surrendered” (“отказано в лицензии”). Такие условия действительно существуют и определяют, может ли агент работать в данном штате. Наконец, таблица CustAgentList – это “клей”, связывающий три таблицы, Customers, Agents и Listings. Таблица CustAgentList содержит внешние ключи, являющиеся первичными в таблицах Customers, Listings, Agents и ContactReason. Всего она состоит из 1 018 строк. Только в этой таблице можно найти цену, которую потенциальные покупатели предлагают за недвижимость. Следовательно, ненулевые значения BidPrice могут содержать только те объекты, для которых в таблице ContactReason указано “Buу”.

Когда рынок недвижимости лихорадит, работы у агентов прибавляется, и они могут работать с несколькими покупателями и продавцами одновременно. Особен-но сложны продажи, поскольку покупатели и продавцы часто проводят несколько раундов переговоров, пока не сойдутся на окончательной цене. Агенты должны от-слеживать все предложения. Фактически одной из важнейших видов информации для агента является список контактов. Люди, который посмотрели один дом, могут не предложить за него приемлемую цену, однако они, скорее всего, заинтересуются другими домами. Хороший агент всегда имеет в своем арсенале несколько домов на продажу и может направить потенциальных покупателей в дом, отвечающий их тре-бованиям. Агенты зарабатывают свои деньги на комиссионных с продажи или покуп-ки дома. Как правило, агент по продаже и покупке часть своих комиссионных отдает агентству недвижимости, в котором он работает. Традиционно агентства берут за под-писание контракта 6% комиссионных, но иногда данная сумма может быть предметом торга. На рис. 1.31 показана контактная форма, которую используют агенты для от-слеживания предложений и продаж других агентов. Разумеется, компания собирает дополнительные данные о сотрудниках и клиентах, однако из соображений экономии места мы не будем приводить эту форму полностью. Ядром данной формы является таблица CustomerAgentList. Другие формы используются для ввода данных.

На рис. 1.32 показана часть отчета о продажах для агентов. В отчете указаны некоторые важные данные о недвижимости, представляемой каждым агентом. Как и ранее, об объекте собрано гораздо больше деталей – число санузлов, этажей, размер дома в квадратных футах, а также год его постройки. Кроме того, агенты хранят контактную информацию о владельцах недвижимости.

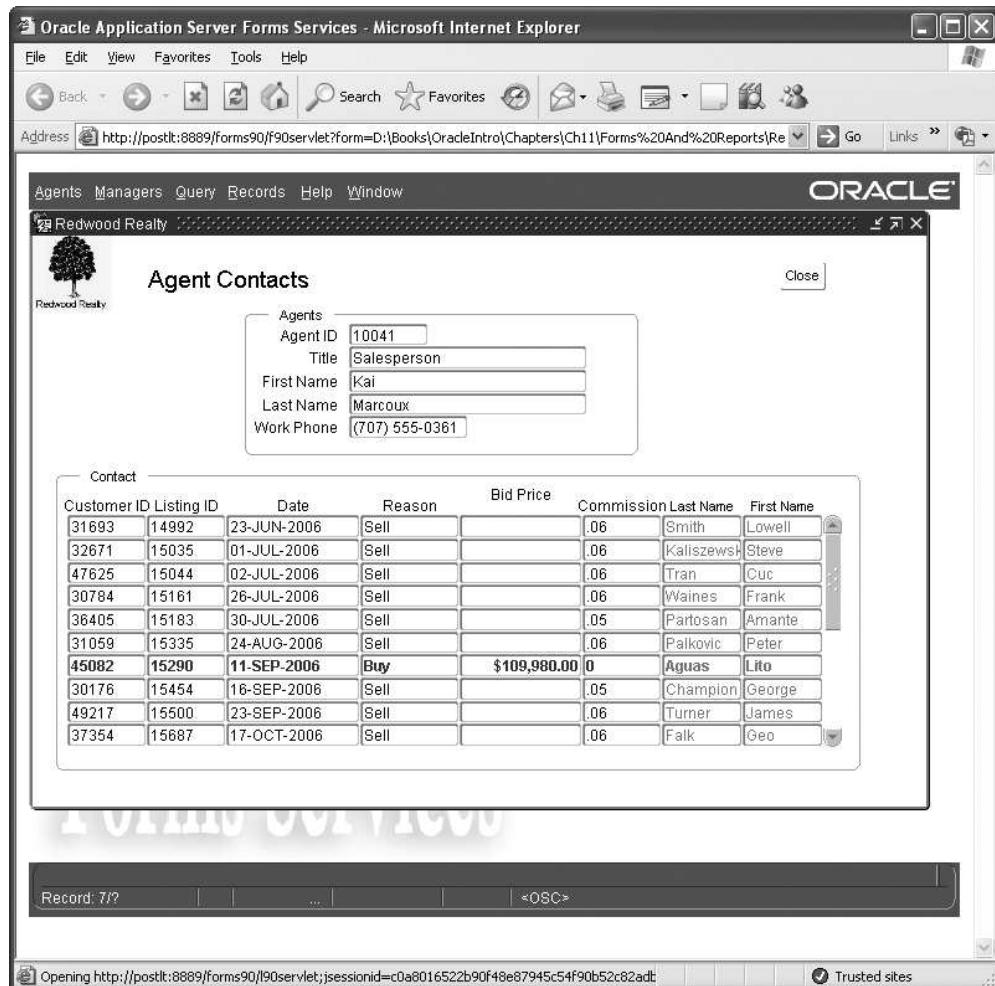


Рис. 1.31. Контактная форма для агентов и клиентов

## Coffee Merchant

Coffee Merchant — это небольшой магазин, отправляющий кофе и чай множеству своих клиентов по всему миру. По своей сути, это Web-компания, компьютерные нужды которой относительно просты: записывать заказы и отслеживать клиентов. Каталог товаров Coffee Merchant достаточно велик: какао-бобы от Kona Extra Fancy (США), а также различные сорта из Суматры, Эфиопии, Йемена, Колумбии, Коста-Рики и Папуа-Новой Гвинеи (и это лишь часть из них).

Бизнес-модель магазина Coffee Merchant представлена в семи таблицах: States, Consumers, Employees, OrderLines, Orders, Inventory и Countries. По мере приобретения опыта менеджеры планируют включить в базу данных перевозку то-



## Agent Listings

Last Name	Allee	First Name	Lora	Agent ID	15293
Work Phone	(707) 555-9990				
Listing ID		List Date		Asking Price	
15099		12-JUL-06		\$230,000	Eureka
15863		11-NOV-06		\$208,000	Arcata
15461		17-SEP-06		\$200,000	Eureka
15537		26-SEP-06		\$178,035	Arcata
16120		18-DEC-06		\$163,360	Eureka
15982		29-NOV-06		\$155,010	Eureka
15480		18-SEP-06		\$149,228	Arcata
15232		06-AUG-06		\$145,000	Loleta
15623		04-OCT-06		\$145,000	Arcata
15699		20-OCT-06		\$130,950	Eureka
16135		19-DEC-06		\$129,800	Arcata
15705		20-OCT-06		\$129,740	Eureka
15045		03-JUL-06		\$125,000	Arcata
15384		05-SEP-06		\$119,000	Eureka
15715		22-OCT-06		\$105,000	Arcata
15663		10-OCT-06		\$101,000	Arcata
15268		10-AUG-06		\$100,000	McKinleyville
<b>Total: 17</b>				<b>\$2,514,123</b>	

**Рис. 1.32.** Пример отчета о выставлении объекта на продажу

вара. Таблица *States* содержит информацию о 50 штатах США и округе Колумбия. В столбцах представлены сокращенное обозначение штата, полное название штата, налог с оборота, численность населения, размер штата в квадратных милях, а также URL официального Web-сайта штата (ну мало ли, вдруг вам захочется узнать, какой цветок является символом штата Миссисипи). Таблица *Consumers* содержит имя и адрес бывших и нынешних клиентов (более чем 1 500 строк-клиентов). В таблице *Employees* хранится информация о 22 сотрудниках: дата найма, ставка комиссионного вознаграждения, дата рождения, пол и имя. Заказы, направленные Coffee Merchant, помещаются в таблицы *Orders* и *OrderLines*. 500 строк таблицы *Orders* содержат указатели на клиентов и сотрудников, имеющих отношение к конкретным заказам, даты заказов, а также заказы клиентов (если они есть). Таблица *OrderLines* состоит из 2 192 строк отдельных позиций заказов, причем все записи ссылаются на родительские заказы с помощью соответствующей пары внешних первичных ключей. В таблице *Inventory* перечислены все сорта чая и кофе, которые Coffee Merchant может предложить своим клиентам. В 128 строках таблицы *Inventory* представлено все: от обычного колумбийского кофе до экзотических сортов китайского чая. Поле *Description* данной таблицы не часто используется в книге, однако оно содержит подробное описание чая или кофе, включая место произрастания, вкусовые

характеристики, факты, касающиеся обработки продукта, и т.п. В таблице `Countries`, состоящей из 256 строк, описываются страны в определенный момент времени. Таблица `Inventory` ссылается на таблицу `Countries` как на источник сведений о стране-производителе продукта. Ниже мы изложим некоторые факты относительно компаний и опишем, как сотрудники обрабатывают заказы.

Чтобы заполнить заказы, полученные через Web-сайт или по телефону, в компании Coffee Merchant задействованы 22 человека. Для того чтобы определить, кто лучше всех продает товар, руководство отслеживает, какой сотрудник отвечает за конкретный заказ. Заказом, полученным по телефону, занимается сотрудник, получивший его; Web-заказы распределяются между всеми сотрудниками.

Подробности отгрузки/доставки не выделяются в отдельную таблицу. По сложившейся традиции для работы с интерактивными покупателями компания предпочитает использовать формы заказа Oracle, а не Web-формы. Кроме того, формы Oracle разработаны для сотрудников, а не потенциальных клиентов.

На рис. 1.33 показана стандартная форма заказа, используемая в компании Coffee Merchant. Информация о клиенте вводится в отдельную форму, и для выбора подходящего клиента в данной форме используется список подстановки. Подобным образом компания поддерживает отдельный список предлагаемых продуктов (каталог). Данные элементы определяются номером в форме заказа, а для получения подробной информации о каждой позиции используется список подстановки. Обратите внимание на то, что столбец `Value` рассчитывается в самой форме — в базе данных он не хранится. Кроме того, компания вводит две позиции, которых нет в простой форме заказа, рассмотренной в главе. Во-первых, существует цена по каталогу и отпускная цена продукта. Отпускная цена определяется при продаже на основе информации, хранимой в базе данных. Во-вторых, сотрудники могут предлагать клиентам дополнительные скидки, т.е. уменьшать отпускную цену на несколько процентов.

На рис. 1.34 представлен типичный административный отчет компании Coffee Merchant. В нем рассчитываются общие продажи за квартал для каждого штата и каждой категории. Пока что магазин отпускает две категории товара: чай и кофе, однако в будущем ассортимент может увеличиться. В расчетах компания использует календарный год, поэтому значение квартала легко рассчитать по дате заказа. Для того чтобы уменьшить ошибки ввода данных, компания использует отдельные таблицы для разных штатов и стран.

## Rowing Ventures

Rowing Ventures отвечает за координацию гребных регат в более десяти местах США и других стран. Помимо наблюдения за регатой и предоставлением места под рекламу на различных соревнованиях по гребле, Rowing Ventures хранит в своей базе данных множество информации о заездах каждой регаты, участниках, их спонсорах, лодках каждого заезда, номерах и экипажах лодок.

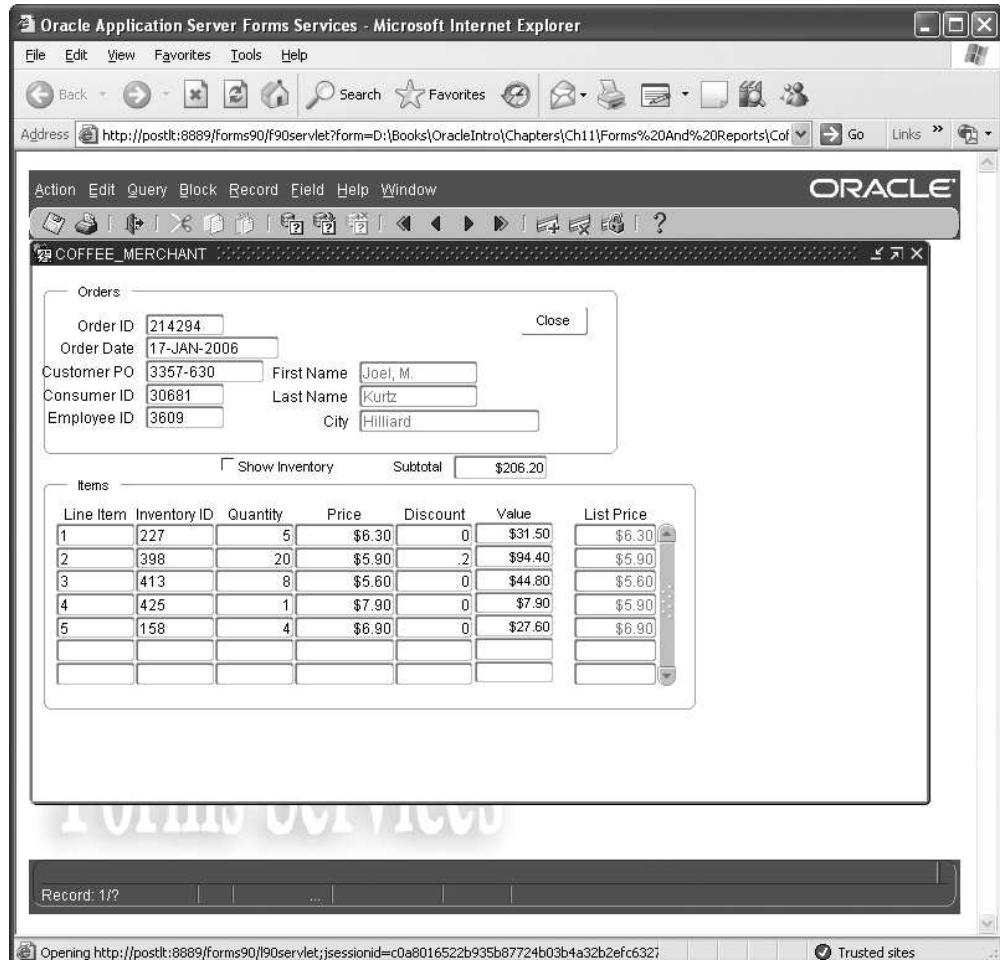


Рис. 1.33. Стандартная форма заказа компании Coffee Merchant

Сравнительно небольшая база данных образована шестью таблицами. В таблице Boat указана категория лодки, присвоенный номер, а также клуб или организация, от которых выступает команда. В таблице Organization представлена информация, касающаяся клуба или компании, участвовавшей в одной или нескольких гонках. В данной таблице указаны телефон, адрес, страна, название организации, а также основные новости, касающиеся группы или области, в которой располагается группа. В таблице RaceTimes указано время финиширования, порядок на финише, а также указатели на гонки и лодку. Таблица Race содержит имена всех групп гонок. На текущий момент представлены только два типа гонок: Club Eights Men и Club Fours Men. В таблицах Joining Race, RaceTimes и Boat содержатся перечень всех гонок, проведенных до текущего момента, название команд, время заездов и порядок



### Quarterly Sales By State

Item Type C		2005-4	2006-1	2006-2	
Year/Quarter	State	Value	Value	Value	Total
	AL	\$487	\$288	\$158	\$933
	AZ	\$216	\$1,049		\$1,264
	CA	\$3,676	\$2,934	\$354	\$6,964
	CO	\$492	\$321		\$813
	CT	\$1,504	\$1,366	\$311	\$3,180
	DC	\$315			\$315
	DE	\$138	\$636		\$775
	FL	\$1,281	\$1,144	\$105	\$2,530
	GA	\$931	\$561		\$1,492
	IA	\$561	\$67		\$628
	IL	\$1,675	\$1,590	\$602	\$3,868
	IN	\$298	\$361		\$658
	KS		\$392	\$226	\$618
	KY	\$198	\$434	\$202	\$834
	LA	\$314	\$615		\$929
	MA	\$1,556	\$933	\$707	\$3,196
	MD	\$81	\$424		\$504
	ME	\$89			\$89
	MI	\$749	\$836	\$154	\$1,740
	MN	\$1,120	\$1,159	\$494	\$2,773
	MO	\$1,304	\$1,077	\$455	\$2,835
	MS	\$809	\$644		\$1,453
	NC	\$450	\$866	\$86	\$1,401
	NE	\$44			\$44
	NH	\$251	\$226		\$478
	NJ	\$2,413	\$1,080	\$220	\$3,714
	NM		\$345	\$200	\$546
	NV	\$529	\$424		\$952
	NY	\$4,733	\$4,245	\$85	\$9,063
	OH	\$2,406	\$1,688	\$385	\$4,479
	OK	\$195	\$368		\$563
	OR	\$299	\$210		\$508
	PA	\$1,797	\$2,966	\$414	\$5,177
	RI		\$600		\$600
	SC	\$9	\$256		\$265
	TN	\$556	\$1,011	\$144	\$1,711
	TX	\$4,657	\$2,752	\$333	\$7,742
	UT	\$643	\$269		\$912
	VA	\$889	\$458	\$444	\$1,791
	VT	\$76			\$76
	WA	\$342	\$796		\$1,138
	WI	\$1,738	\$1,151	\$38	\$2,927
	WV		\$73		\$73
		\$39,820	\$36,612	\$6,118	\$82,551

Рис. 1.34. Квартальный отчет о продажах по типу товара и штатам

финиширования. В таблице BoatCrew перечислены имена спортсменов команды лодок. В таблице Person указаны имя, пол, вес и дата рождения каждого участника. По информации из таблицы Person определяют, в какой гонке может официально участвовать данный человек, здесь также установлено ограничение по возрасту, если оно есть (например, если речь идет о профессиональных гонках). Из таблиц Join-

ing Race, RaceTimes, Boat, BoatCrew и Person можно получить всю информацию о том, кто, когда и с каким результатом участвовал в какой гонке.

Ключевыми элементами базы данных является возможность регистрации участников. За большинством лодок закреплена команда, т.е. на них находятся несколько человек. Поскольку лодки дорогие, их владельцами являются организации, которые и выставляют их на соревнование. Многие из организаций — это университетские клубы гребли, однако существуют и частные команды. При регистрации лодки на гонке компания Rowing Ventures должна отследить данные обо всех членах команды и самой организации. Существует множество категорий соревнований, которые основываются на типе судна, количестве гребцов, а часто еще на поле и возрасте участников команды. Все лодки на данном соревновании получают номера, соответствующие дорожкам, по которым они идут, поэтому и на воде, и на финишной черте их легко идентифицировать.

В спорте принята своя терминология. Если вы посещали соответствующие Web-сайты или присутствовали на гонках, то должны это знать. Например, на весельной лодке каждый спортсмен имеет одно весло. Существует гребля парными веслами. Гоночная лодка на восемь человек (шэлл) всегда имеет рулевого, положение которого обозначается цифрой “0” в столбце Position таблицы BoatCrew. На данной лодке, именуемой “восьмеркой”, находится восемь гребцов, имеющих по одному веслу. Положения данных гребцов нумеруются с носа до кормы в убывающем порядке, начиная с положения 8. При нумерации от носа до кормы гребцы левого и правого бортов чередуются (начиная с левого борта). Гребцы, учащиеся средней школы или колледжа, обычно располагаются либо на носу, либо по правому борту. Профессиональным гребцам (27 лет и старше) часто доверяют левый или правый борт. На “четверке” (четырехвесельной лодке) можно обойтись и без рулевого. В лодке такого типа может быть четыре гребца с четырьмя или восемью веслами. (*Это не шутка!*)

На рис. 1.35 показан вариант формы, используемой для записи результатов гонок. Места присваиваются согласно порядку появления команд на финише, параллельно записывается время каждой лодки. Все дополнительные данные записываются перед гонками (преимущественно при регистрации лодок и команд). После гонок служащий просто выбирает соответствующие гонки, вводит занятое место, время, показанное лодкой, и выбирает лодку из раскрывающегося списка.

Результаты гонок печатают и размещают так, чтобы их могли видеть команды и болельщики (рис. 1.36). В конечном итоге результаты отправляют в различные профильные издания. Гребцы будут рады увидеть свои имена в газете — даже если это простая студенческая газета. PDF-версию некоторых результатов также помещают на Web-сайте, чтобы болельщики и участники могли проверить и сравнить результаты.

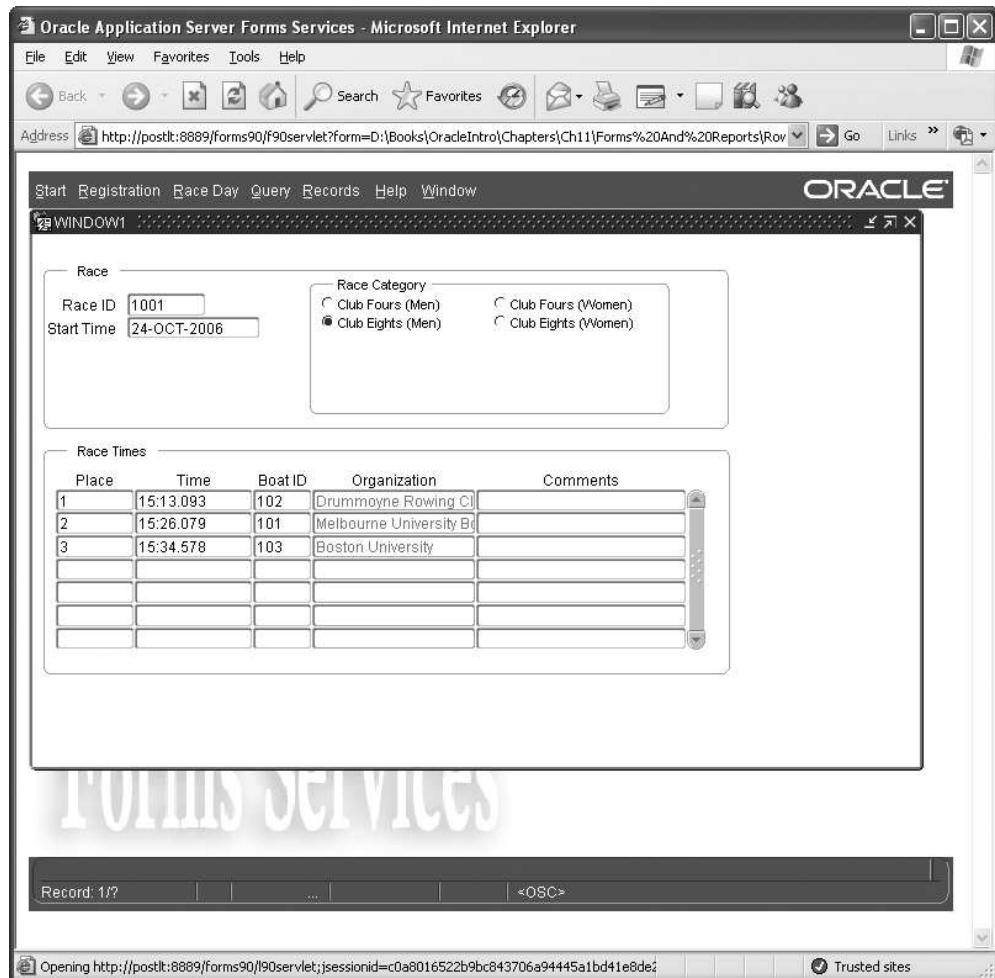


Рис. 1.35. Форма результатов гонок

## Broadcloth Clothing

Broadcloth Clothing — это компания, производящая одежду, которая продается преимущественно через универмаги и сети магазинов. Broadcloth Clothing представляет собой сравнительно сложный случай, и с точки зрения собранных данных он гораздо больше предыдущих. Сложность состоит в том, что компания специализируется на дизайне и координации. Большую часть одежды производят на различных фабриках по всему миру, имеющих разных владельцев. Broadcloth Clothing разрабатывает новые линии одежды, получает заказы из магазинов (*клиенты*), а затем заключает контракты с различными фабrikами для производства определенного количества товара. В данном случае под дизайном понимается новая модель или стиль одежды,



## Race Results

### Rowing Ventures

Race ID	Start Time	Category	Place	Time	Boat ID	Position	Last Name	First Name
1001	24-OCT-06	Club Eights (Men)	1	15:13.093	102	0	Rogers	June
		Drummoyne Rowing Club				1	Pangburn	Richard
						2	Berry	Glenn
						3	Farrar	Donald
						4	Wood	Larry
						5	Wheeler	Michael
						6	Miller	Ralph
						7	Moore	Paul
						8	Sutter	Robert
			2	15:26.079	101	0	Wolfe	Carole
		Melbourne University Boat Club				1	Rappaport	Scott
						2	Eklund	Gene
						3	Cory	Lou
						4	Simpson	Ted
						5	Briggs	Dan
						6	Lewis	Donald
						7	Witte	Charles
						8	Johnson	Douglas
			3	15:34.578	103	0	Rose	Barbara
		Boston University				1	White	Dwight
						2	Kern	Ray
						3	Morrow	Ray
						4	Brown	Mark
						5	King	Michael
						6	Wisefield	Joey
						7	Wulfod	Harry
						8	Jones	Bruce

Рис. 1.36. Результаты гонок с именами гребцов

например, новая рубашка, блузка или брюки. Каждая модель может иметь несколько размеров или цветов. Например, блузка определенного стиля, пошитая на определенного типа фабрике, является одной моделью одежды. Модель синего цвета и шестого размера является отдельным предметом одежды.

Как показано на рис. 1.37, клиенты заказывают по нескольку штук отдельных предметов. Как правило, это несколько размеров и цветов одной модели. Тем не менее компания должна отслеживать отдельные предметы. Позже, если потребуется, можно создать отчет с указанием общего числа проданных элементов различных стилей моделей.

Международная природа производства несколько затрудняет вопросы доставки. Во многих странах введены таможенные пошлины на одежду. В любом случае требуется, чтобы партия товара прошла через таможню для оценки. Фабрика пытается наладить массовое производство предметов одежды, чтобы снизить их стоимость, однако при этом ей нужно группировать поставки, чтобы уменьшить стоимость их доставки. Например, в понедельник компания может сшить тысячу блузок, а во втор-

**BROADCLOTH\_CLOTHING**

**Customer Order**

Order ID	1001	Order Date	30-APR-2006	Purchase Order No	3661201
Customer ID	110	Company Name	Gap		
Bill Address	1 Jefferson St.	Delivery Location	Corporate		
Bill City	San Francisco	Delivery Address	1 Jefferson St.		
Bill Postal Code	94133	Delivery City	San Francisco		
Bill State	CA	Delivery Postal	94133		
Bill Nation	USA	Delivery State	CA		
Delivery Nation	USA				
Price Adjustment	0	Order Exchange Date	30-MAY-2006	Order Currency	USD

**Order Items**

Item ID	Quantity	Sale Price	Item Size	List Price	Description
1005	385	63.8	2	75	Sweater, cable knit
1035	347	127.69	12	150	Skirt, medium, silk
1048	66	212.52	6	250	Jacket, long, light

Record: 1/?

Рис. 1.37. Форма заказа Broadcloth Clothing

ник — партию брюк. Данные предметы одежды сортируют в среду и упаковывают так, чтобы часть блузок и часть брюк отправить одному клиенту, а остальное распределить по другим клиентам. После изготовления достаточно большой партии одежду отправляют экспортеру вместе со счетом-фактурой (рис. 1.38). Каждой партии товара присваивается идентификатор, позволяющий отслеживать доставку груза различным клиентам. Иногда груз дополняют объяснениями для таможенной службы.

Одной из сложнейших обязанностей менеджеров Broadcloth Clothing является планирование производства. Они должны собрать заказы от клиентов и определить, сколько единиц каждой продукции необходимо произвести. После этого им требуется определить, по каким фабрикам следует распределить заказы. Поскольку фабрики

Action Edit Query Block Record Field Help Window

ORACLE

WINDOW1

Shipment Item Customer

Shipment ID	1001	Ship Method	Container	Ship Date	10-AUG-2006
Factory ID	100	Customer ID	120	Ship Address	1501 5th Ave
Ship Cost	105	Ship City	Seattle	Ship State	WA
Ship Currency	USD	Ship Nation	USA	Ship Postal	98101
Exchange Date	10-AUG-2006	Customs Comments			
Customs ID	114-583				

Total Quantity 266

Item ID	Quantity Shipped	Order ID
1031	266	1007

Record: 1/?

Opening http://postit:8889/forms90/f90servlet;jsessionid=c0a8016522b95fc483a2e751493f87328e344

Trusted sites

Рис. 1.38. Доставка груза с фабрики клиенту

также производят товары для других компаний, менеджеры должны выбрать оптимальный вариант заказа с учетом нескольких фабрик. Кроме того, компания Broadcloth Clothing отслеживает качество каждой партии продукции, чтобы определить лучших поставщиков. В последнее время компания стала дополнитель но проводить выборочные проверки в рабочих условиях. Она отправляет на фабрики инспекторов, оценивающих различные условия и гарантирующих, что все рабочие имеют допустимый возраст, здоровы и получают надлежащую заработную плату. Один из вариантов отчета о фабрике показан на рис. 1.39. В данном случае просто рассчитывается суммарное производство различных элементов одежды на различных фабриках. Печатая отчет для различных периодов времени, компания может определить, какие фабрики использовать в будущем для производства подобных товаров.



Broadcloth Clothing

## Production Analysis

Model ID	Description	Factory ID 100		101 Quezon City Total Produced 283	102 Zhongshan City Total Produced
		City	Bangkok		
			Total Produced		
1001	Sweater, scoop neck				
1002	Sweater, cable knit				
1004	Blouse, long collar				
1005	Blouse, frills			387	
1006	Sweater, button				
1007	Pants, plain front, narrow leg				
1009	Skirt, pleated, single color				
1010	Skirt, pleated, plaid		266		73
1011	Skirt, long, wool				
1012	Skirt, medium, silk				347
1013	Jacket, fitted, short				
1014	Jacket, loose, long				
1016	Jacket, long, light			42	
1017	Scarf, wool				
1018	Stockings, stripes				
1019	Hat, floppy				
1020	Tie, silk		266	341 1053	420

**Рис. 1.39.** Отчет о производстве на фабриках

После надлежащей “отладки” системы производства работают впечатляюще. Несмотря на это, ежедневные операции всегда сложнее, чем ожидалось заранее. Когда все идет нормально, нужные вещи производятся вовремя и поставляются соответствующим клиентам. Проблема в том, что в реальной жизни очень многое может пойти не так, как планировалось. Менеджерам необходимо постоянно проверять заказы, производство и данные о поставках, чтобы они могли выявить проблему сразу после ее возникновения. Кроме того, требуется информация для создания обходных путей. Например, если заказчику поступил не тот товар, менеджеру нужно найти требуемый груз или запланировать изготовление новой партии на доступной фабрике. Имея в базе данных всю необходимую информацию, менеджеры могут быстро найти альтернативные варианты. Разумеется, им все равно придется созваниваться с клиентами и фабриками для уточнения деталей, однако необходимую начальную информацию удобно извлекать из базы данных.

## Резюме

Системы управления базами данных (СУБД) – это мощные программные инструменты, позволяющие легко и безопасно делиться информацией и создавать приложения. Базы данных следует разрабатывать особенно тщательно, чтобы использовать все преимущества реляционного подхода. Процесс разработки удобно начинать с опре-

деления основных объектов. После этого следует применять три правила нормализации, чтобы создать удобные таблицы. Основное правило, гарантирующее, что таблица находится в третьей нормальной форме, заключается в том, что каждый столбец должен содержать однозначные данные, а все столбцы, не относящиеся к ключевым, — зависеть от всего ключа и только от него. Помимо этого, вам необходимо гарантировать, что каждая таблица идентифицируется первичным ключом и что связи между таблицами отражают принципы работы компании, использующей базу данных.

Установкой программного обеспечения Oracle обычно занимается администратор базы данных, однако вам придется устанавливать копии базы и на свой компьютер. Для работы с упражнениями, рассматриваемыми в настоящей книге, вы должны установить основную базу данных, а также набор инструментов Developer Suite. Вы можете установить их на одном компьютере (только обязательно в различных папках). Удаление программного обеспечения Oracle несколько сложнее, поскольку оно требует редактирования файла реестра Windows.

Прочитав данную книгу, вы научитесь использовать инструменты Oracle, позволяющие создать основы для разработки приложений и администрирования базы данных. Поскольку практически все задачи в Oracle можно решить с помощью SQL, в книге будет интенсивно использоваться SQL\*Plus. Если вам понадобиться связаться с сервером через Интернет-соединение, полезным может оказаться основанное на Web средство iSQL\*Plus. Набор инструментов Developer Suite позволяет быстро создавать формы и отчеты и использовать при этом минимум программирования. Данный набор является настраиваемым, так что вы можете использовать программируемые элементы для создания мощных приложений. В качестве графического инструмента Oracle, используемого для решения стандартных задач администрирования базы данных, применяется Enterprise Manager. Новичкам обычно легче использовать Enterprise Manager, чем запоминать все административные запросы. Несмотря на это, для использования данного инструмента требуется обладать полномочиями администратора базы данных, и в любом случае основные запросы вам все равно придется запомнить.

## Основные термины

- Инструменты администрирования
- Приложение
- Связь
- Атомный
- Атрибут
- Базисная таблица
- Класс

- Сеть связи
- Сложный первичный ключ
- Словарь базы
- Процессор базы
- Типы данных
- База данных
- Администратор базы данных (АБД)
- Система управления базой данных (СУБД)
- Developer Suite
- Разработчик
- Enterprise Manager
- Объект
- Диаграмма “сущность–связь” (Entity-Relationship Diagram – ERD)
- Первая нормальная форма
- Внешний ключ
- Службы форм и отчетов
- Наследование
- Отношение типа “многие ко многим”
- Нормализация
- Отношение типа “один ко многим”
- Первичный ключ
- Процессор запросов
- Целостность на уровне ссылок
- Реляционная база данных
- Зависимость
- Стока
- Вторая нормальная форма
- Подсистема безопасности
- Суперкласс
- Таблица
- Третья нормальная форма
- Универсальный язык моделирования (Unified Modeling Language – UML)

## Повторение пройденного материала

### Истина или ложь?

1. В производственной ситуации сервер приложений не должен устанавливаться на тот же сервер, что и база данных.
2. Таблица может находиться в третьей нормальной форме, не находясь при этом в первой.
3. Столбец, являющийся внешним ключом, всегда должен быть частью первичного ключа в той же таблице.
4. Если набор Developer Suite устанавливается на тот же компьютер, что и основная база данных, его следует устанавливать в другую папку.
5. Для использования Enterprise Manager необходимы привилегии администратора базы данных.

### Заполнить пропущенное

1. Если вы создаете таблицу и определяете два столбца как часть первичного ключа, вы указываете, что между данным двумя столбцами существует отношение \_\_\_\_\_.
2. Чтобы таблица находилась в первой нормальной форме, все ее столбцы должны содержать \_\_\_\_\_ данные.
3. Одной из основных объектно-ориентированных возможностей, предлагаемых Oracle, является команда \_\_\_\_\_, позволяющая пользователям определять собственные типы данных.
4. Первичный ключ, состоящий из нескольких столбцов, называется \_\_\_\_\_ ключом.
5. Для того чтобы гарантировать точность данных, используется принцип \_\_\_\_\_, исходя из которого пользователь может вводить значения CustomerID в таблицу Order только в том случае, если информация об этом пользователе уже имеется в таблице Customer.

### Варианты ответов

1. Какой инструмент вы будете использовать, если требуется найти телефонный номер клиента?
  - a) SQL\*Plus
  - b) Enterprise Manager
  - c) Forms Builder
  - d) Reports Builder
  - e) Application Server

2. Данна следующая таблица:

`ListingProperty(ListingID, StartDate, AgentID,  
PropertyID, Bedrooms, Bathrooms, Size)`

В ней число спален (`Bedrooms`), санузлов (`Bathrooms`) и размер (`Size`) зависят от значения `PropertyID`. Данная таблица удовлетворяет следующим условиям:

- а) Первая нормальная форма.
  - б) Вторая нормальная форма.
  - в) Третья нормальная форма.
  - г) Вторая и третья нормальная форма.
  - д) Имеет структуру “сущность–связь”
3. В прикладной системе за создание форм и отчетов отвечает \_\_\_\_\_.  
  - а) Пользователь.
  - б) Администратор базы данных.
  - в) Создатель запросов.
  - г) Разработчик приложений.
  - д) Генеральный директор.
4. Первый этап удаления базы данных Oracle – это  
  - а) Запуск средства Add/Remove programs (Установка и удаление программ) Windows.
  - б) Запуск программы установки Oracle и выбор варианта удаления.
  - в) Удаление записей Oracle из реестра Windows.
  - г) Останов служб Oracle.
  - д) Прекращение платежей Oracle.
5. В таблице реляционной базы данных информация, описывающая один экземпляр объекта, хранится  
  - а) В столбце таблице.
  - б) В строке таблицы.
  - в) Как целая таблица.
  - г) Как первичный ключ в таблице.
  - д) В словаре данных.

## Упражнения

### 1. Readwood Realty

Для того чтобы создать базу данных для организации, вам в конечном итоге нужно создать список нормализованных таблиц. На этом этапе будет полезно начать с диаграммы основных объектов и их взаимоотношений.

1. Вернитесь к описанию примера главы и выпишите основные объекты. По крайней мере, ваш список должен включать *Properties*, *Agents*, *Listings*, *Customers* и *CustomerAgentList*, так как все они упоминались в описании примера.
2. Подумайте, какими могут быть дополнительные подстановочные таблицы подстановок (например, *ContactReason*) с тремя-четырьмя столбцами данных? Задача данных таблиц — предоставлять информацию для раскрывающихся списков, чтобы пользователи могли выбирать причину из списка, а не вводить ее вручную. Помимо того, что это облегчает ввод данных, подстановка уменьшает число ошибок, вызванных использованием сокращений и опечатками.
3. Изучите форму и отчет, показанные в тексте, и определите объекты, необходимые для их создания. Нарисуйте диаграмму объектов для данного случая. Определите столбцы, принадлежащие к каждой нормализованной таблице. С помощью линий обозначьте взаимоотношения и добавьте метки, указывающие на связь “один ко многим” *Подсказка*. Если вы не создавали дополнительные таблицы подстановки, решение должно состоять примерно из восьми таблиц.

### 2. Coffee Merchant

Чтобы создать базу данных для организации, вам нужно создать список нормализованных таблиц. На этом этапе будет полезно начать с диаграммы основных объектов и их взаимоотношений.

1. Вернитесь к описанию примера главы и выпишите основные объекты. Ваш список должен включать по крайней мере четыре таблицы: *Orders*, *OrderLines*, *Inventory* и *Consumers*.
2. Компания так использует таблицу *States*, чтобы сотрудники могли не набирать название штата, а выбрать его из списка. При таком подходе получаются согласованные значения для отчета. Подобным образом компания перечисляет страны — производители каждого продукта из каталога, поэтому имеется отдельная таблица стран.

3. Нарисуйте диаграмму взаимоотношений объектов для данной задачи. Определите столбцы, принадлежащие к каждой нормализованной таблице. С помощью линий обозначьте взаимоотношения и добавьте метки, указывающие на связь “один ко многим”. *Подсказка.* Если вы не создавали дополнительные таблиц подстановки, решение должно состоять примерно из семи таблиц.

### 3. Rowing Ventures

Используя форму и отчет, приведенные в главе, определите основные объекты и взаимоотношения, необходимые для создания этой базы данных. Найдите хотя бы один Web-сайт, посвященный гонкам, и создайте список гоночных классов. Нарисуйте диаграмму взаимоотношений. *Подсказка.* На диаграмме должно присутствовать по крайней мере шесть таблиц, однако если вы добавили подстановочные таблицы, их может быть больше.

### 4. Broadcloth Clothing

Используя форму и отчет, приведенные в главе, определите основные объекты и взаимоотношения, необходимые для создания этой базы данных. Данное задание сравнительно объемно, поэтому делайте его по частям. Начните со стандартной формы заказа, а затем добавьте информацию по поставкам. Фабричное производство может быть довольно сложным на момент, когда вы будете собирать для смен данные о производстве и качестве. Еще несколько таблиц добавит элемент условий работы. В целом базовая диаграмма без таблиц подстановки должна состоять примерно из тринадцати таблиц.



# ГЛАВА 2

## Обзор SQL и SQL\*Plus

**В этой главе...**

- Классификация и распознавание типов SQL-выражений
- Редактирование и запись SQL-выражений
- Использование трех различных интерфейсов SQL, доступных в Oracle
- Написание SQL-выражений для создания таблиц и ввода данных в таблицы
- Применение SQL-выражений для удаления строк, добавления столбцов и удаления таблиц
- Использование расширений SQL\*Plus для захвата выхода и форматирования результатов

---

---

### Введение

Язык структурированных запросов (Structured Query Language – SQL) представляет собой промышленный стандартный язык, используемый для взаимодействия с реляционными базами данных. Впервые SQL был описан в работе доктора Е.Ф. Кодда (E.F. Codd), разработавшего в 1970-х System R и SQL для IBM. На данном стандарте основаны все инструменты доступа к Oracle. Промышленные стандарты для SQL устанавливаются двумя сообществами – ANSI (American National Standards Institute – Национальный институт стандартизации США) и ISO (International Standards Organization – Международная организация по стандартизации). ANSI стандартизовал SQL в 1986 году. В 1989 году ANSI и ISO опубликовали стандарт SQL89 (или SQL1). Позже ANSI пересмотрел первый стандарт и продублировал его как SQL92

(SQL2). Последний стандарт ANSI, SQL99 (SQL3), содержит указания по концепциям объектно-ориентированных баз данных. Следует отметить высокую переносимость SQL — данный язык можно использовать для доступа к большому числу систем баз данных, включая Oracle, DB2 (IBM), SQL Server (Microsoft) и MySQL.

SQL — это не язык программирования. Он предоставляет выражения, используемые для ввода, извлечения, модификации, удаления и отображения данных, хранимых в таблицах базы данных. Используя SQL, вы можете формулировать запросы для немедленного выполнения, создавать различные объекты и выполнять операции по ведению базы данных. За прошедшие годы развилось несколько версий стандарта. Последняя из них, SQL99, полностью распознается системой управления базой данных Oracle. Преимущество стандарта заключается в том, что любые выученные вами команды SQL и навыки, которыми вы овладели при использовании Oracle, пригодятся вам и при работе с другими системами управления реляционными базами данных. Небольшие отличия в реализациях SQL от различных производителей баз данных возможны, но, как правило, они являются крайне незначительными (например, использование для выделения имен не двойных кавычек, а квадратных скобок). Другими словами, изучив сейчас SQL, вы сможете работать с интерфейсами практически всех других систем управления базами данных.

Несмотря на то что в Oracle реализован ANSI SQL, здесь также предлагается богатый набор расширений к SQL, называемых командами SQL\*Plus. Хотя расширения SQL\*Plus нельзя использовать со всеми системами баз данных на основе SQL, они достаточно полезны при работе с Oracle. Расширения SQL\*Plus предлагают множество элементов, облегчающих разработку, в частности, форматирование данных, возвращаемых в ответ на запрос, облеченный в стандартную форму SQL. Еще раз повторим, что в данной книге и в документации, предлагаемой Oracle, эти полезные расширения называются командами SQL\*Plus.

## Типы команд SQL

Существует пять типов выражений SQL: выражения запроса, DDL-выражения (Data Definition Language — язык определения данных), DML-выражения (Data Manipulation Language — язык манипулирования данными), ТС-выражения (Transaction Control — управление транзакциями) и DCL-выражения (Data Control Language — язык управления данными).

### Выражения запроса

Выражения запроса позволяют извлекать строки из одной или нескольких таблиц. В качестве примера выражения запроса можно привести SELECT (мы придерживаемся принятого в SQL принципа зарезервированных слов — слова, определяющие элементы языка SQL и не используемые в качестве идентификаторов, пишутся прописными буквами).

## DDL-выражения

Выражения SQL, принадлежащие к данной группе, позволяют определять структуры (таблицы и представления), образующие базу данных. В данной главе мы будем использовать пять таких выражений: CREATE, ALTER, DROP, RENAME и TRUNCATE. Данные команды позволяют соответственно создавать и определять таблицы или пользователей; модифицировать структуру базы данных; удалять структуру (например, таблицу); изменять ее имя; а также удалять все позиции таблицы.

## DML-выражения

Как следует из названия, выражения манипулирования данными позволяют модифицировать содержимое таблиц путем ввода новых данных, изменения существующих данных или удаления из таблицы целых строк данных. DML-выражения INSERT, UPDATE и DELETE мы разберем в следующих главах.

## TC-выражения

Выражения управления транзакциями предлагают инструменты, позволяющие сделать постоянными любые изменения строк таблицы или отменить эти изменения. Команда COMMIT делает постоянными любые изменения, внесенные в строки таблицы. Команда ROLLBACK является обратной — она аннулирует все изменения строк таблицы, сделанные после последнего сохранения. Еще в данную группу входит выражение SAVEPOINT, позволяющее отмечать точку, к которой вы можете вернуться, отменив все внесенные изменения.

## DCL-выражения

Выражения языка управления данными определяют, кому разрешен доступ к различным таблицам, кто может входить в систему Oracle, а также какие права доступа и привилегии имеют пользователи различных таблиц базы данных. Для определения прав доступа и привилегий пользователей используется выражение GRANT, а для отмены выбранных прав и привилегий — команда REVOKE. Например, вы можете задать, какой пользователь может извлекать информацию из данных другого пользователя или запретить пользователю создавать таблицы.

## “Анатомия” выражения SQL

Выражения SQL применяются для выполнения задач, связанных с базой данных. После командного слова или фразы (например, CREATE) вы включаете фразу, уточняющую действие, которое следует выполнить. Выражения SQL, введенные посредством интерфейса SQL\*Plus, могут занимать одну или несколько строк. Для продолжения выражений SQL на следующей строке необходимо нажать клавишу <Enter>. В ре-

The screenshot shows the Oracle SQL\*Plus interface on a Windows system. The title bar reads "Oracle SQL\*Plus". The menu bar includes File, Edit, Search, Options, Help. The main area displays three separate SQL queries and their results.

```

SQL> SELECT FirstName, LastName
  2  FROM Agents
  3 ORDER BY LastName, FirstName;

FIRSTNAME          LASTNAME
-----            -----
Lora               Allee
Tobias             Carling
Belinda            Chong
Elizabeth          Dahl
Cornelis            Dann
Crystal             Fernandez
Jackson            Flamenbaum
David               Gagnon
Barbara             Herring
James                Kellogg
Patricia            Lewis

FIRSTNAME          LASTNAME
-----            -----
Kai                Marcoux
Essi               Okindo
Nancy              Piperova
Lee                Reed
Clair               Robinson
Cecilia             Romero
Ramanathan          Rose
Tim                Schutz
Ricki               Selby
Heather             Sheibani
Danial              Silverburg

FIRSTNAME          LASTNAME
-----            -----
Stanislaw           Soltwedel
Tim                 St-Onge
Jessica             Taylor
Edwin               Townsend
Bruce               Voss
Sindisiwe           Weber
Christine           Williams

29 rows selected.

```

**Рис. 2.1.** Пример команды SQL, введенной с помощью SQL\*Plus for Windows

зультате точка ввода переносится на начало новой строки, и при этом не требуется использовать метку продолжения строки. В любом случае разрыв слова на две строки не допускается. Если вы пользуетесь построчным интерфейсом, подобным тому, что предлагает SQL\*Plus, то последовательные строки будут пронумерованы. Выражение SQL не обязательно полностью писать в одну строку, довольно часто выражения SQL в данной книге представляются в несколько строк, каждая из которых содержит отдельную фразу SQL. На рис. 2.1 показан пример выражения SQL SELECT, введенного посредством интерфейса SQL\*Plus.

Сокращенная запись слов, входящих в выражение SQL, недопустима (хотя некоторые слова SQL\*Plus имеют краткие формы; подробнее об этом — ниже). Слова должны разделяться по крайней мере одним пустым символом (<пробел>, <Tab> или <Enter>). Слова SQL можно набирать прописными или строчными буквами. Ключевые слова SQL (слова, по определению имеющие специальное значение в языке SQL) в данной книге для удобства представлены прописными буквами. Самыми распространенными знаками пунктуации в командах SQL являются запятая (разделение списка имен, например, названий столбцов) и круглые скобки (обособление элементов). Точки применяются в именах объектов для разделения отдельных составляющих, например `Inventory.Price`. Одинарные кавычки (апострофы) замыкают последовательности литературных символов (например, даты). В двойные кавычки

берутся строки, используемые, например, как альтернативные имена столбцов. Точка с запятой указывает конец выражения SQL. Ввод точки с запятой и нажатие клавиши <Enter> в SQL\*Plus приводит к выполнению команды SQL. Существуют и другие знаки пунктуации, которые мы подробно рассмотрим ниже.

## Интерактивная помощь

Один из самых полезных навыков при изучении Oracle связан с получением помощи. Например, вы ввели команду SQL с ошибкой и интерпретатор SQL\*Plus отобразил сообщение об ошибке, ее код, номер строки, содержащей ошибку, и отметил положение ошибки в пределах этой строки. На рис. 2.2 показан пример ошибки, порожденной командой CREATE TABLE. Пытаясь создать новую таблицу ContactReason, мы неправильно ввели команду, описывающую структуру таблицы. Некоторые ошибки могут быть очевидными, а другие обнаружить очень трудно. Интерпретатор SQL\*Plus указывает, что ошибка находится в строке 2, отображает ее код (ORA-00922) и сообщение “missing or invalid option” (“пропущенная или неверная опция”). Большинство сообщений об ошибках Oracle обычно лаконичны. Если вы не можете сразу определить проблему, помните, что в Интернете вы можете легко найти дополнительную информацию, которая поможет понять и исправить ошибку. Приставка “ORA” в начале кода ошибки Oracle указывает, что ошибку (иногда именуемую исключительной ситуацией) сгенерировала система баз данных. Пятизначное число после ORA, присваиваемое Oracle Corporation, указывает точную природу ошибки — в данном случае, что мы пропустили запятую во второй строке. Звездочка в сообщении об ошибке часто указывает точное положение ошибки в операторе, вызвавшем сбой. Однако в примере на рис. 2.2 звездочка просто отмечает начало выражения или фразу, в которой была обнаружена ошибка. В частности, звездочка указывает, что ошибка произошла при обработке открывающей круглой скобки, за которой следует определение первого столбца таблицы ContactReason. В данном случае сообщение “missing or invalid option” недостаточно конкретно, чтобы быть особенно полезным.

Помощь по сообщениям об ошибках Oracle доступна на сайте OTN (Oracle Technology Network) и других Web-сайтах. На сайте <http://ora-code.com> представлен полный список кодов ошибок ORA по порядку. Возможно, вам придется несколько раз щелкнуть мышью, чтобы найти требуемый код на нескольких страницах перечня, однако это того стоит.

Чтобы получить больше информации о сообщениях об ошибках в стиле ORA, выполните действия, приведенные ниже.

1. Запустите Web-браузер, введите в адресной строке <http://ora-code.com> и нажмите клавишу <Enter>. Появится страница Oracle Database Error Code.
2. Щелкните на ссылке с цифрой 3, чтобы перейти на третью страницу перечня ошибок.

The screenshot shows the Oracle SQL\*Plus interface. The title bar reads "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following SQL code:

```
SQL> CREATE TABLE ContactReason
  2      (ContactReasonID NVARCHAR2(10)
  3      Description NVARCHAR2(30)
  4      );
  5      (ContactReasonID NVARCHAR2(10)
  *
ERROR at line 2:
ORA-00922: missing or invalid option
```

Below the code, the prompt "SQL>" is visible. The status bar at the bottom shows navigation icons.

**Рис. 2.2.** Сообщение об ошибке в команде SQL и соответствующий код ошибки

3. Щелкните на ссылке ORA-00922 (значения увеличиваются слева направо и сверху вниз).
4. Появится полное описание ошибки. По-видимому, Oracle считает, что мы добавили неверный оператор. Внимательно изучая строку SQL, видим, что в конце строки 2 отсутствует запятая. Пока что данную ошибку мы исправлять не будем.
5. Закройте Web-браузер.

## Взаимодействие с базой данных

Помимо определения языка SQL, в стандарте SQL99 приводятся указания по выполнению выражений SQL. В частности, данные указания (*связующие стили*) определяют минимальный набор выражений SQL, который должен поддерживаться для определенного связующего стиля. Всего существует четыре метода выполнения: прямое (немедленное) выполнение, связывание модулей, внедренные SQL-выражения и интерфейс уровня вызовов (Call-Level Interface — CLI).

Прямое выполнение позволяет общаться с системой управления базой данных непосредственно через клиентское приложение, подобное SQL командной строки, SQL\*Plus или iSQL\*Plus. Интерпретатор SQL\*Plus проверяет командную строку, чтобы гарантировать, что она не содержит синтаксических ошибок. Если все правильно, то интерпретатор отправляет строку в базу данных для выполнения. Отметим, что команды SQL *нечувствительны* к регистру. Даже если мы будем писать зарезервированные слова SQL прописными буквами, интерпретатор SQL\*Plus и база данных Oracle, выполняющая команды, регистр не различают — команды и любые слова могут содержать только прописные, только строчные буквы или и те и другие в произвольной комбинации. Более того, SQL\*Plus игнорирует пробелы и разрывы строки в выражении. Пробелы и разрывы строк мы используем исключительно для удобочитаемости.

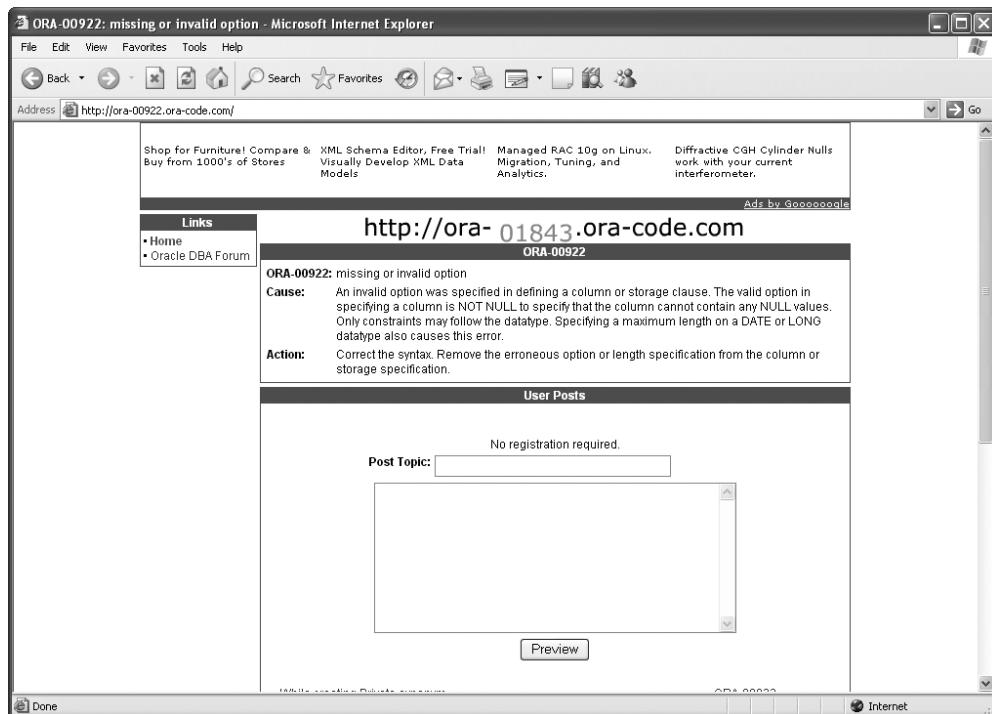


Рис. 2.3. Дополнительная информация о коде ошибки Oracle

Интерпретатор SQL взаимодействует с сервером базы данных, расположенным на том же компьютере или на другом компьютере сети. При прямом выполнении вы вводите выражение SQL непосредственно на клиентском компьютере и, нажимая соответствующую клавишу, отправляете его серверу базы данных. Вскоре сервер возвращает результаты интерфейсу клиента. Если, например, выражение SQL предполагало извлечение данных, интерфейсу клиента будут отправлены строки результата.

Метод связывания модулей позволяет создавать блоки выражений SQL. Готовый блок можно вводить в приложение с помощью программы, именуемой компоновщиком (linker). В результате получается завершенная процедура, содержащая все, что требуется для выполнения ряда операций с базой данных.

Внедренные выражения SQL вводятся непосредственно в базовый язык программирования (например, C++ или COBOL). Препроцессор Oracle изучает выражения SQL на предмет правильности и отделяет их от другого скомпилированного кода на упомянутом языке программирования. Код SQL преобразовывается в форму, которую понимает система баз данных. Внедрив выражения SQL непосредственно в базовый язык программирования, мы позволим программе обращаться к таблицам базы данных посредством этих выражений и извлекать реальные строки данных.

Интерфейс уровня вызовов позволяет вызывать выражения SQL через “дверцу в коде”, передавая выражения SQL непосредственно обрабатывающим подпрограмм-

мам. Выражения не компилируются и не подвергаются другой обработке до получения их сервером базы данных (после чего они выполняются непосредственно системой управления базой данных).

Ниже описаны три различных способа организации запросов и модификации баз данных Oracle: через интерфейс командной строки из операционной системы; через клиентский продукт командной строки SQL\*Plus от Oracle; и через Web-среду Oracle, названную *iSQL\*Plus*.

Иногда разница между SQL и SQL\*Plus на первый взгляд незаметна, однако они отличаются, и вам следует знать об этом. SQL — это непроцедурный язык, используемый для связи с системой управления реляционной базой данных. SQL\*Plus — это разработанная Oracle среда, в которой вы можете выполнять выражения SQL. Позже в данной главе мы расскажем о втором значении термина SQL\*Plus — командах SQL\*Plus. Команды SQL\*Plus — это расширение SQL от Oracle, позволяющее форматировать результаты, отображать определения таблиц, редактировать и сохранять файлы. Подробнее об этом рассказывается в разделе “Использование расширений SQL\*Plus”.

Ниже мы рассмотрим использование трех различных методов для входа в систему баз данных Oracle с использованием SQL\*Plus на компьютере с ОС Windows. Во-первых, вы научитесь вызывать SQL\*Plus из командной строки DOS (Disk Operating System — дисковая операционная система). Во-вторых, узнаете, как использовать более доброжелательное к пользователю средство SQL\*Plus for Windows. И в-третьих, вы разберетесь с вызовом *iSQL\*Plus*, интерфейсом базы данных Oracle, основанном на Web. Все названные методы имеют достоинства и недостатки. Прочитав данную книгу, вы сможете выбрать интерфейс, оптимальный для ваших потребностей.

## Вызов SQL\*Plus из командной строки DOS

Если вы запустили Oracle на собственном ПК с операционной системой Windows или используете ПК Windows для взаимодействия с Oracle на сервере, то для взаимодействия с базой данных можете использовать окно DOS или среду SQL\*Plus. Поскольку мы рекомендуем использовать средства SQL\*Plus, предлагающие множество знакомых по Windows инструментов (например, копирование/вставка и т.д.), возможности командной строки мы рассмотрим кратко.

### Вход в Oracle

SQL\*Plus можно вызвать из командной строки DOS, после этого войти в Oracle и далее вводить команды SQL. Итак, чтобы открыть окно DOS и вызвать SQL\*Plus в операционной системе Windows, выполните следующие действия.

```

C:\>sqlplus
SQL*Plus: Release 10.1.0.2.0 - Production on Sat Sep 23 12:40:26 2006
Copyright <c> 1982, 2004, Oracle. All rights reserved.

Enter user-name: redwood
Enter password:

Connected to:
Personal Oracle Database 10g Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> help describe
DESCRIBE
Lists the column definitions for a table, view, or synonym,
or the specifications for a function or procedure.
DESC[RIBE] <[schema.]object[@connect_identifier]>

SQL>

```

**Рис. 2.4.** Использование SQL\*Plus в окне DOS

1. Щелкните на кнопке Start (Пуск) панели задач, выберите Run (Выполнить), введите command в текстовом окне Open (Открыть) и щелкните на кнопке OK. Откроется окно DOS.
2. Введите sqlplus и нажмите клавишу <Enter>. Появится запрос имени пользователя.
3. Введите свое имя пользователя и нажмите клавишу <Enter>. Появится запрос пароля.
4. Введите пароль (на экране он не отображается) и нажмите клавишу <Enter>. Если вы все правильно ввели, появится запрос SQL>.
5. Введите help describe и нажмите клавишу <Enter>, чтобы отобразилась справочная информация о заданной команде SQL\*Plus (рис. 2.4).
6. Чтобы выйти из Oracle, введите exit и нажмите клавишу <Enter>. Вы вернетесь к запросу DOS C:\>.
7. Чтобы закрыть окно DOS, еще раз введите exit и нажмите клавишу <Enter>.

## Ввод и запуск команд SQL

В предыдущем примере вы запустили команду SQL\*Plus — одну из команд, определенных Oracle, для расширения стандартного набора команд SQL. Эта команда описывает структуру таблицы Dual, устанавливаемой Oracle автоматически. Перед выполнением указанных ниже действий найдите на компакт-диске файл Ch02SaleStatus.sql и скопируйте его на компьютер. Запомните полный путь к папке, в которую вы его скопировали. Данный файл не стоит записывать в папке My Documents (Мои документы),

поскольку это слишком длинный путь. Сохраните его в корне диска С (C:\) или в папке Temp (C:\Temp\).

Чтобы запустить команду SQL в SQL\*Plus, откройте окно DOS.

- Щелкните на кнопке Start (Пуск) панели задач, выберите Run (Выполнить), введите command в текстовом окне Open (Открыть) и щелкните на кнопке OK. Откроется окно DOS.
- Запустите SQL\*Plus: введите `sqlplus <имя_пользователя>/<пароль>` и нажмите клавишу <Enter>, подставив вместо <имя\_пользователя> свое имя пользователя, а вместо <пароль> — пароль, разделив их косой чертой. (Помните, что в данном случае пароль *отображается* на экране!)
- Введите команду SQL\*Plus CLEAR SCREEN (прописными или строчными буквами) и нажмите клавишу <Enter>, чтобы очистить экран.
- После запроса `SQL>` введите `START`, нажмите клавишу <пробел>, введите полный путь к файлу (`Ch02SaleStatus.sql`), обратную косую черту `Ch02SaleStatus.sql` (использование прописные буквы необязательно). После этого нажмите клавишу <Enter>. Если, например, вы записали файл (в данном случае называемый сценарием) в корне диска С, то вам нужно будет набрать следующую строку и нажать клавишу <Enter>:

```
SQL> START C:\Ch02SaleStatus.sql
```

- В ответ Oracle отобразит подтверждение “Table created” (“таблица создана”).
- Окно DOS не закрывайте — мы к нему еще вернемся.

## Редактирование команд SQL

Если вам требуется изменить только что введенную команду SQL (возможно, нужно исправить ошибку), вызовите редактор, введя в запросе `SQL>` команду `edit`. Windows откроет программу Notepad (Блокнот), позволяя изменить команду, а затем записать результат.

Итак, чтобы запустить команду SQL в SQL\*Plus, активизируйте открытое ранее окно DOS, а затем выполните нижеприведенные действия.

- Ведите `edit` и нажмите клавишу <Enter>. Windows откроет программу Блокнот — предопределенный редактор для окна DOS. В окне данной программы откроется сценарий `CREATE TABLE`, записанный в файле `Ch02SaleStatus.sql`. Содержимое этого сценария записывается во временный файл `afiedt.buf` (имя файла предопределено Oracle).
- Если потребуется, щелкните мышью слева от первой строки перед словом `CREATE`.
- Ведите `--Create a new table` и нажмите клавишу <Enter>. Любая строка, начинающаяся с двух минусов, является комментарием, который Oracle игнорирует.

4. Щелкните на слове File (Файл) в строке меню, выберите Exit (Выход) и щелкните на кнопке Yes, чтобы сохранить изменения.
5. Измененное выражение SQL (включая добавленный комментарий) появится ниже сообщения Wrote file afiedt.buf.
6. Чтобы выполнить модифицированное выражение SQL, введите косую черту и нажмите клавишу <Enter>. В ответ Oracle генерирует сообщение об ошибке (ORA-00955), поскольку таблица уже существует, а выражение SQL пытается создать ее повторно.
7. Введите exit и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus. Еще раз наберите exit и нажмите клавишу <Enter>, чтобы закрыть окно DOS.

---

---

## Использование SQL\*Plus for Windows

Гораздо легче использовать среду SQL\*Plus с помощью SQL\*Plus for Windows, поскольку в этом случае мы имеем возможность работать с диалоговым окном Windows и более понятным интерфейсом. Кроме того, в SQL\*Plus for Windows легче задавать положения файлов. Если вы можете выбирать, какой интерфейс операционной системы использовать, — DOS- или Windows-версию, — советуем выбрать последнюю.

При входе в Oracle с помощью SQL\*Plus for Windows отображается запрос командной строки SQL> и предлагается среда, обладающая описанными ниже характеристиками.

- Принимает вводимые вами выражения SQL и передает их серверу Oracle.
- Интерпретирует команды редактора, форматирования и работы с файлами.
- Форматирует результаты, возвращаемые в ответ на запросы SQL, и отображает отчеты на экране.
- Помнит установленные вами настройки среды (например, число строк между заголовками и ширина строки экрана).
- Предлагает простой построчный редактор, поддерживающий ввод и модификацию набираемых выражений.
- Принимает выражения SQL, записанные в файлах.

### Вход в Oracle

Для того чтобы войти в Oracle с помощью SQL\*Plus for Windows, выполните следующие действия.

1. Щелкните на кнопке Start (Пуск), выберите All Programs (Все программы) (Windows XP), затем выберите Oracle-OraDb10g\_home1, выберите ApplicationDevelopment и щелкните на SQLPlus. Откроется диалоговое окно LogOn.

**Совет.** Для удобства можно создать ярлык для SQL\*Plus, щелкнув правой кнопкой мыши на SQLPlus, выбрав Send To (Отправить), а затем Desktop (create shortcut) (Рабочий стол (создать ярлык)).

2. Введите имя пользователя в текстовом окне User Name и нажмите клавишу <Tab>. Введите пароль в поле Password. Если потребуется, нажмите клавишу <Tab> и, если необходимо, введите строку хоста в текстовом окне Host String.

**Совет.** Если вы неосторожно нажали клавишу <Enter> после ввода имени пользователя, Oracle закроет диалоговое окно Log On и запросит у вас пароль. Введите его и нажмите клавишу <Enter>.

3. Щелкните на кнопке OK, чтобы войти в базу данных Oracle. Для удобства можете развернуть окно SQL\*Plus на весь экран.

Если вы все правильно ввели, SQL\*Plus предоставит вам доступ к базе данных Oracle, покажет номер версии SQL\*Plus, дату и время, а также уведомление об авторских правах. Кроме того, будет указана база данных Oracle (например, Personal Oracle), с которой вы соединились. В последней строке отображается приглашение командной строки SQL>, свидетельствующее о том, что приложение SQL\*Plus готово получать команды. (Окно SQL\*Plus не закрывайте — оно нам еще понадобится.)

Если вы получите сообщение об ошибке, например ORA-01017: invalid user-name/password; logon denied, это означает, что вы ввели неверное имя пользователя или пароль. Спросите у своего администратора базы данных правильное имя и пароль. Если вы увидите сообщение об ошибке с кодом ORA-12154, например ORA-12154: TNS: could not resolve the connect identifier specified, — обратитесь к администратору.

После того как вы успешно вошли в систему и увидели приглашение SQL>, вы начали новый сеанс Oracle. После приглашения вы можете ввести только одну команду SQL\*Plus. Для очень больших команд просто нажимайте клавишу <Enter> между вводом слов, чтобы перевести строку. Никакого ограничения на число строк в команде SQL\*Plus не существует.

## Ввод и запуск команд SQL

После входа в Oracle через интерфейс Windows SQL\*Plus вы можете выполнять команды SQL и SQL\*Plus.

Чтобы отобразить структуру таблицы SaleStatus и добавить в нее строки данных, выполните следующие действия.

1. После отображенного приглашения SQL введите команду SQL\*Plus describe salesstatus, которая перечислит названия всех столбцов таблицы и их типы дан-

ных, а затем нажмите клавишу <Enter>. Oracle извлечет и отобразит информацию о таблице *salestatus*. Обратите внимание на то, что названия столбцов написаны прописными буквами (они хранятся именно в таком виде).

2. Введите в таблицу какие-нибудь данные, используя команду *INSERT*. После *SQL>* введите указанные ниже строки, нажав клавишу <Enter> после слова *SaleStatus*. Введите точку с запятой, но *не нажимайте* клавишу <Enter> в конце второй строки.

```
INSERT INTO SaleStatus  
VALUES (101, 'For Sale');
```

3. Нажмите клавишу <Enter>, чтобы отправить завершенное выражение *SQL* к *SQL\*Plus*, а затем к *Oracle* для выполнения. *Oracle* выполняет команду, если последним символом, набранным перед нажатием клавиши <Enter>, была точка с запятой (это нужно для работы с одно- или многострочными командами). Появится сообщение “*1 row created*”, которое означает, что вы успешно добавили данные в таблицу.
4. Введите следующие два выражения *SQL*, нажимая клавишу <Enter> в конце каждой строки, завершающейся точкой с запятой. (Использование прописных букв существенно *только* в данных в одинарных кавычках. Наберите текст точно так, как показано ниже.)

```
INSERT INTO SaleStatus VALUES (102, 'Pending');  
INSERT INTO SaleStatus VALUES (103, 'Sold');
```

Если выражения были введены правильно, после каждого выражения *SQL\*Plus* отобразит сгенерированное *Oracle* сообщение – “*1 row created*” (рис. 2.5).

5. Введите *CLEAR SCREEN* и нажмите клавишу <Enter>, чтобы очистить экран *SQL\*Plus* (окно *SQL\*Plus* не закрывайте).

## Редактирование команд SQL

Команды, набранные в *SQL\*Plus*, можно изменить, набрав *edit* в командной строке *SQL>* (процедура аналогична рассмотренной выше для редактирования команд *SQL* в окне DOS). Тем не менее при работе с книгой мы рекомендуем использовать другой метод, чтобы вы могли записывать все, что набираете, и спокойно его редактировать. Довольно часто команды *SQL* длинные, что затрудняет их повторный набор для устранения ошибок. Мы считаем, что лучшей альтернативой является создание команд *SQL* и *SQL\*Plus* с помощью текстового редактора, подобного Блокноту (использование текстового процессора, подобного Word, не рекомендуется).

Набрав команду *SQL*, вы копируете ее в буфер Windows <*Ctrl+A*>, чтобы выделить весь текст, затем выбираете из меню *Copy* («*Скопировать*»), активизируете окно *SQL\*Plus*, вставляете в него команду и запускаете ее. Если возникнет ошибка, исполь-

SQL\*Plus: Release 10.1.0.2.0 - Production on Mon Sep 25 14:39:42 2006  
Copyright (c) 1982, 2004, Oracle. All rights reserved.

Connected to:  
Personal Oracle Database 10g Release 10.1.0.2.0 - Production  
With the Partitioning, OLAP and Data Mining options

```
SQL> describe salestatus
Name          Null?    Type
-----        -----   -----
SALESTATUSID  NOT NULL NUMBER(38)
SALESTATUS      NVARCHAR2(10)

SQL> INSERT INTO SaleStatus
2  VALUES (101,'For Sale');

1 row created.

SQL> INSERT INTO SaleStatus VALUES (102, 'Pending');

1 row created.

SQL> INSERT INTO SaleStatus VALUES (103, 'Sold');

1 row created.

SQL> |
```

**Рис. 2.5.** Ввод команды SQL через окно SQL\*Plus

зуете Блокнот и исправляете ошибку, а затем копируете текст в SQL\*Plus. Преимущество такого подхода заключается в том, что файл Блокнота можно сохранить, а затем по необходимости загрузить в SQL\*Plus повторно. Использование команды `Edit` в командной строке SQL\*Plus можно назвать более неуклюжим способом. Любые записываемые команды SQL (или группы отдельных команд в одном файле, разделенные точкой с запятой) называются *сценарием*. Вообще, удобно иметь копии многих (если не всех) написанных вами строк SQL как запись структур таблиц и данных, содержащихся в этих таблицах. Когда сценарии безопасно хранятся на диске, вы можете запускать их на другом компьютере с Oracle, открыв SQL\*Plus и набрав `start` и полный путь к файлу сценария (включая его название). Ниже мы попрактикуемся это делать.

Для того чтобы отредактировать команду SQL в Блокноте, а затем выполнить ее в SQL\*Plus, выполните следующие действия.

1. В окне SQL\*Plus введите приведенные ниже команды. Нажмите клавишу `<Enter>` после ввода названия таблицы (`SaleStatus`) и еще раз после точки с запятой во второй строке.

```
INSERT INTO SaleStatus
VALUES (101, 'Unknown');
```

Oracle возвратит сообщение об ошибке, указывающее, что система обнаружила *нарушение условия уникальности*.



**Рис. 2.6.** Выбор редактируемого текста

2. Щелкните мышью в начале первой строки и растяните прямоугольник выделения до точки с запятой во второй строке. Приглашение SQL, номера строк, повторенную команду и сообщение об ошибке выделять не нужно (рис. 2.6).
3. Выберите Edit из строки меню и щелкните на позиции Copy.

**Совет.** Для экономии времени нажмите комбинацию клавиш <Ctrl+C>, чтобы скопировать выбранный текст в буфер.

4. Запустите Блокнот (щелкните на кнопке Start (Пуск), затем выберите All Programs (Всепрограммы), Accessories (Стандартные) и Notepad (Блокнот)) или запустите любой другой текстовый редактор.
5. Находясь в программе Блокноте, щелкните на кнопке Edit (Правка), затем выберите Paste (Вставить) (или нажмите комбинацию клавиш <Ctrl+V>). Команда SQL появится в Блокноте. Теперь вы можете исправить ее и записать результат в виде файла.
6. Исправьте команду: измените значение 101 на 104 — это идентификатор новой строки таблицы.
7. Скопируйте исправленную команду в SQL\*Plus: нажмите комбинацию клавиш <Ctrl+A>, чтобы выбрать весь текст, нажмите комбинацию клавиш <Ctrl+C>, чтобы скопировать его в буфер, переключитесь на окно SQL\*Plus, нажмите <Ctrl+V>, чтобы вставить новый код в окно SQL\*Plus, и нажмите клавишу <Enter>, чтобы запустить команду. На этот раз она выполняется правильно и Oracle возвращает подтверждающее сообщение — “1 row created”.
8. Откройте окно Блокнота, откройте меню File (Файл), а затем выберите Save (Сохранить). Дойдите до папки, в которой хотите сохранить файлы сценариев для главы 2, введите Ch2<имя>SaleStatus.sql в окне File name (Имя файла) (вместо “<имя>” введите свое имя), откройте список Save as type (Тип файла), выберите All Files (Всефайлы), а затем щелкните на кнопке Save.
9. Закройте программу Блокнот. Если вы собираетесь активно вводить команды SQL\*Plus, окно Блокнота удобно держать открытым.

## Выход из SQL\*Plus

Выйти из SQL\*Plus можно тремя способами.

- Выберите File в строке меню, а затем Exit.
- Введите Exit в командной строке SQL>.
- Щелкните на значке X в строке заголовка.

При выходе из SQL\*Plus Oracle отключает вас и автоматически закрывает соединение с базой данных, т.е. описанная выше последовательность действий — это процедура выхода из Oracle.

---

## Использование iSQL\*Plus

Другим способом доступа к Oracle является использование iSQL\*Plus — основанной на Web среды SQL\*Plus. Для обращения к iSQL\*Plus необходимо ввести URL вида `http://<имя_компьютера>.<имя_домена>:<порт>/sqlplus/`.

В предыдущем примере `<имя_компьютера>` — это имя компьютера, а значение `<порт>` равно 5560. Если вы устанавливали Oracle на собственный компьютер, вы сможете обращаться к iSQL\*Plus с помощью URL `http://127.0.0.1:5560/sqlplus/`.

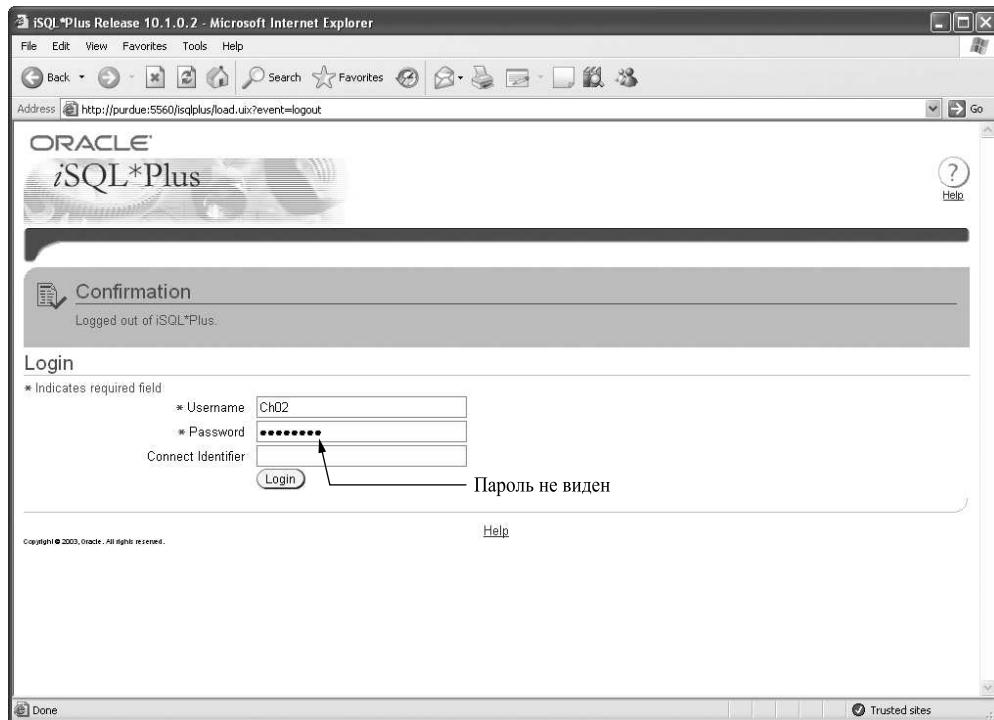
Если ни один из указанных URL не подошел, обратитесь к администратору, чтобы он выдал необходимые сведения. На рис. 2.7 показан экран регистрации в iSQL\*Plus с использованием имени компьютера PURDUE. Имя домена не указывается, поскольку среда iSQL\*Plus располагается на той же машине, что и база данных Oracle (компьютере PURDUE). Точно так же, как при использовании SQL\*Plus, вы вводите имя пользователя, пароль и (возможно) идентификатор соединения, после чего щелкаете на кнопке Login и входите в базу данных Oracle. Если вы не уверены насчет значения идентификатора соединения — обратитесь к преподавателю.

## Вход в Oracle

Ну что ж, попробуем войти в Oracle через iSQL\*Plus — версию SQL\*Plus на основе Web.

Для того чтобы войти в Oracle с помощью iSQL\*Plus и ввести команду SQL, выполните следующие действия.

1. Запустите Web-браузер и введите URL базы данных Oracle. Если вы не уверены насчет значения URL — обратитесь к преподавателю. Отобразится Web-страница iSQL\*Plus.
2. Введите в текстовом окне Username имя пользователя, нажмите клавишу <Tab> и введите пароль.



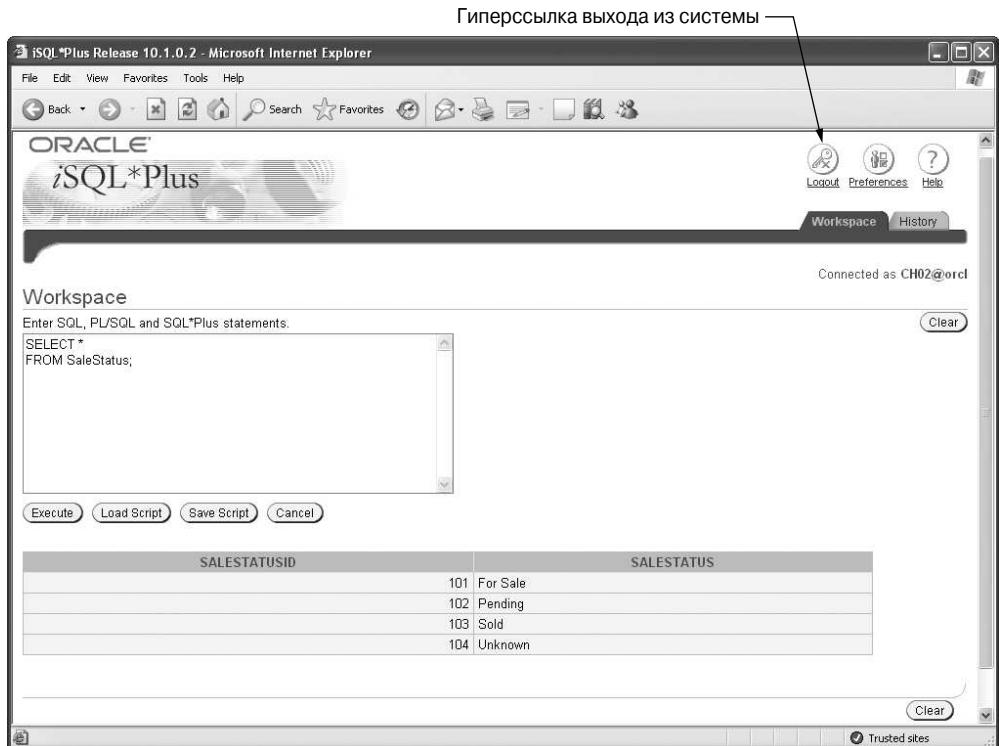
**Рис. 2.7.** Вход в Oracle с использованием iSQL\*Plus

3. Если нужно заполнить поле Connect Identifier, нажмите клавишу <Tab> и введите идентификатор соединения. На рис. 2.7 показана Web-страница iSQL\*Plus с выбранным нами именем пользователя (Ch02) и паролем.
4. Щелкните на кнопке Login, чтобы войти в базу данных Oracle. Если вы ввели правильное имя пользователя и пароль, iSQL\*Plus отобразит Web-страницу Workspace.
5. Щелкните на окне с пометкой “<Enter> SQL, PL/SQL and SQL\*Plus statements”, чтобы поместить в окно точку ввода.

## Ввод и запуск команд SQL

Ввод команд SQL и SQL\*Plus происходит достаточно понятно. Предлагаемый метод может вам понравиться: он позволяет перемещать точку ввода в пределах команды, вносить изменения, а затем отправлять команду на обработку с помощью кнопки Execute.

Для того чтобы ввести команду SQL и выполнить ее, выполните следующие действия.



**Рис. 2.8.** Команда SQL и извлеченные строки в браузере

1. Введите следующее выражение SQL:

```
SELECT *
FROM SaleStatus;
```

2. Щелкните на кнопке Execute, расположенной под окном, в котором вы ввели команду. Результат ее выполнения будет показан внизу окна: появятся строки, записанные в созданной вами таблице SaleStatus (рис. 2.8).

Итак, вы вводите команду в рабочей области, а результат ее выполнения появляется внизу экрана. Если результат занимает больше места, чем может вместить экран, iSQL\*Plus отображает кнопку Next Page внизу экрана, щелкнув на которой можно открыть на следующие окна. Среда iSQL\*Plus также позволяет выполнять набранные команды, загружать файлы сценариев SQL, записывать сценарии, появляющиеся на экране, а также проверять историю введенных команд (гиперссылка History); кроме того, имеется множество других возможностей. Щелкнув на кнопке Help вверху экрана, вы откроете в окно интерактивной помощи.

## Редактирование команд SQL

Команды SQL или SQL\*Plus редактируются точно так же, как текст в текстовом редакторе: вы устанавливаете курсор там, где требуется внести изменения, вносите правку, а затем щелкаете на кнопке **Execute**, чтобы выполнить модифицированную команду. Чтобы записать измененный вариант, щелкните на кнопке **Save Script**, а затем выберите, в какой папке сохранить файл сценария.

Чтобы отредактировать команду SQL в окне браузера *iSQL\*Plus*, выполните следующие действия.

1. Щелкните мышью между словом `SaleStatus` и заключительной точкой с запятой.
2. Нажмите клавишу `<Enter>`, чтобы перейти на новую строку, а затем введите  
`ORDER BY SaleStatus DESC`
3. Щелкните на кнопке **Execute**, чтобы отобразить новые результаты. Возвращаемые строки Oracle сортирует в поле `SaleStatus` в порядке убывания.
4. Запишите новый сценарий как файл: щелкните на кнопке **Save Script**, если появится диалоговое окно, предупреждающее об опасности загрузки файлов из Интернета, щелкните на кнопке **Save**.
5. Дойдите до папки, в которой записаны сценарии главы 2, введите имя `Ch2<имя>ReverseSaleStatus.sql`, в окне **Save as type** выберите **All Files** и щелкните на кнопке **Save**. После этого *iSQL\*Plus* запишет ваш файл сценария. Чтобы загрузить его впоследствии, щелкните на кнопке **Load Script**, затем на кнопке **Browse** и введите путь к файлу, который требуется загрузить (позже мы попрактикуемся в этом).
6. Щелкните на кнопке **Clear**, чтобы очистить окно **Workspace**.

## Выход из *iSQL\*Plus*

Для того чтобы выйти из *iSQL\*Plus*, выполните следующие действия.

1. Щелкните на ссылке **Logout** (вверху и внизу окна), чтобы выйти из Oracle. Oracle отобразит подтверждение, сообщающее, что выход прошел правильно.
2. Закройте Web-браузер.

## Выражения SQL

Фундаментальными операциями, выполняемыми через интерфейс *SQL\*Plus*, являются организация запросов к базе данных, создание и удаление различных объектов

```

*+ Oracle SQL*Plus
File Edit Search Options Help
DROP TABLE Agents      CASCADE CONSTRAINTS PURGE
*
ERROR at line 1:
ORA-00942: table or view does not exist

DROP TABLE LicenseStatus CASCADE CONSTRAINTS PURGE
*
ERROR at line 1:
ORA-00942: table or view does not exist

SQL> |

```

**Рис. 2.9.** Сообщения об ошибках, генерируемые при выполнении команд `DROP TABLE`

базы данных (включая таблицы), а также добавление, обновление и удаление строк таблиц с помощью команд SQL. Ниже мы рассмотрим структуру всех пяти типов выражений SQL, разберем их использование на примере базы данных Redwood Realty и будем периодически создавать команды SQL для исследования их влияния на базу данных. Чтобы ознакомиться с различными выражениями SQL, вначале нужно запустить сценарий SQL, определяющий две новые таблицы и заполняющий их данными. Создав хотя бы несколько таблиц для базы данных Redwood Realty, можно будет уже будет поэкспериментировать с описанными в книге командами SQL.

Итак, чтобы использовать файл сценария для создания двух таблиц и заполнения их данными, выполните следующие действия.

1. Скопируйте с компакт-диска (папка, соответствующая главе 2) файл сценария `Ch02Agents.sql` и запишите или запомните полный путь к нему (например, `C:\Temp\` или `C:\My Documents\SQL\`).
2. Если требуется, запустите SQL\*Plus и введите свое имя пользователя и пароль. Войдите в систему.
3. В командной строке `SQL>` наберите `START <путь>\Ch02Agents.sql`, где `<путь>` — полный путь к файлу сценария.
4. Нажмите клавишу `<Enter>`, чтобы запустить серию команд SQL из файла сценария. В ответ Oracle выведет два сообщения об ошибках (рис. 2.9) под каждой из команд `DROP TABLE`. Не обращайте на них внимания, они просто указывают, что вы начинаете работу с двух пустых таблиц.
5. Введите выражение SQL\*Plus `CLEAR SCREEN` (можно и строчными буквами) и нажмите клавишу `<Enter>`, чтобы очистить экран SQL\*Plus от всех сообщений.
6. Введите `exit` и нажмите клавишу `<Enter>`, чтобы закрыть SQL\*Plus и выйти из Oracle.

При выполнении файла сценария создаются две таблицы, `Agents` и `LicensesStatus`, которые затем заполняются данными. Таблица `Agents` содержит 29 строк,

в которых перечисляются агенты по недвижимости, работающие в компании Redwood Realty. Таблица LicenseStatus содержит 16 строк с информацией о состоянии лицензий агентов (*Licensed* (“действительна”), *Expired* (“истек срок действия”), *Revoked* (“аннулирована”) и т.д.).

## Запуск запросов SQL

Пожалуй, основной задачей языка SQL является *организация запросов* к базе данных и получение ответов на них. Для извлечения информации в SQL применяется команда **SELECT**. Данная команда извлекает информацию из одной или нескольких таблиц. Она задает, какие поля таблицы (включая расчеты в этих полях) извлечь, к каким таблицам обратиться, чтобы найти затребованные поля, какие строки таблицы включить в возвращаемые данные и в каком порядке должны идти строки.

Рассмотрим синтаксис команды **SELECT**. Вообще, *синтаксис* — это правила, касающиеся правописания и грамматики языка; в нашем случае — SQL. Синтаксические правила не допускают вольностей и позволяют получить ответ, только если вы правильно набрали выражение (“правильно” — с точки зрения правил синтаксиса). В данном и последующих примерах, приведенных в данной книге, в квадратных скобках ([ и ]) указываются необязательные части выражения, а в фигурных ({ и }) приводится список элементов, разделенных вертикальной чертой, один из которых вы должны выбрать.

```
SELECT [DISTINCT|ALL] [*] [column_expression [AS new_name]] [,...]
FROM table_name
[JOIN table_name [alias] ON (table_column <operator>table_column)]...
WHERE search_condition
[GROUP BY column_list] [HAVING search_condition]
[ORDER BY column_list]
```

Возможно, что вначале синтаксис команды **SELECT** покажется вам непонятным. Не беспокойтесь. Вы настолько часто будете использовать команду **SELECT** во множестве различных форм, что быстро привыкнете к ней. Например, с помощью этой команды можно перечислить содержимое таблицы *Agents* (все строки и столбцы):

```
SELECT * FROM Agents;
```

Звездочка указывает, что перечислить нужно все существующие столбцы таблицы, а оператор **FROM** — что Oracle должна вернуть строки из таблицы *Agents*. В результате мы получаем 15 столбцов и 29 строк. Чтобы получить часть данной информации, можно использовать следующий запрос **SELECT**:

```
SELECT firstname, lastname, gender
FROM agents
WHERE gender = 'F'
ORDER BY lastname, firstname;
```

Имена столбцов после оператора **SELECT** указывают на то, что из таблицы *Agents* нас интересуют только имя, фамилия и пол агента. Более того, условие **WHERE** огра-

ничивает показываемые результаты строками с информацией по агентам-женщинам (`gender = 'F'`). Перед отображением результатов оператор `ORDER BY` сортирует строки по фамилии, а затем по имени, если встречаются люди с одинаковой фамилией. Попрактикуемся использовать выражение SQL `SELECT` с интерпретатором *iSQL\*Plus*. Прежде всего закройте клиентскую программу `SQL*Plus`, если она еще запущена, введя `exit` в командной строке `SQL>`.

Чтобы с помощью `SELECT` отобразить на экране избранные строки таблицы `Agents`, выполните следующие действия.

1. Запустите Web-браузер, затем *iSQL\*Plus*. (Введите имя пользователя и пароль и щелкните на кнопке `Login`.)
2. Установите курсор в окне `Workspace` и введите приведенные ниже четыре строки. Использование прописных букв некритично за исключением буквы “M” в одинарных кавычках (третья строка). Обратите внимание на отсутствие завершающей точки с запятой, поскольку *iSQL\*Plus* это не требуется.

```
SELECT firstname, lastname, gender, licensestatusid
FROM agents
WHERE gender = 'M'
ORDER BY licensestatusid, lastname
```

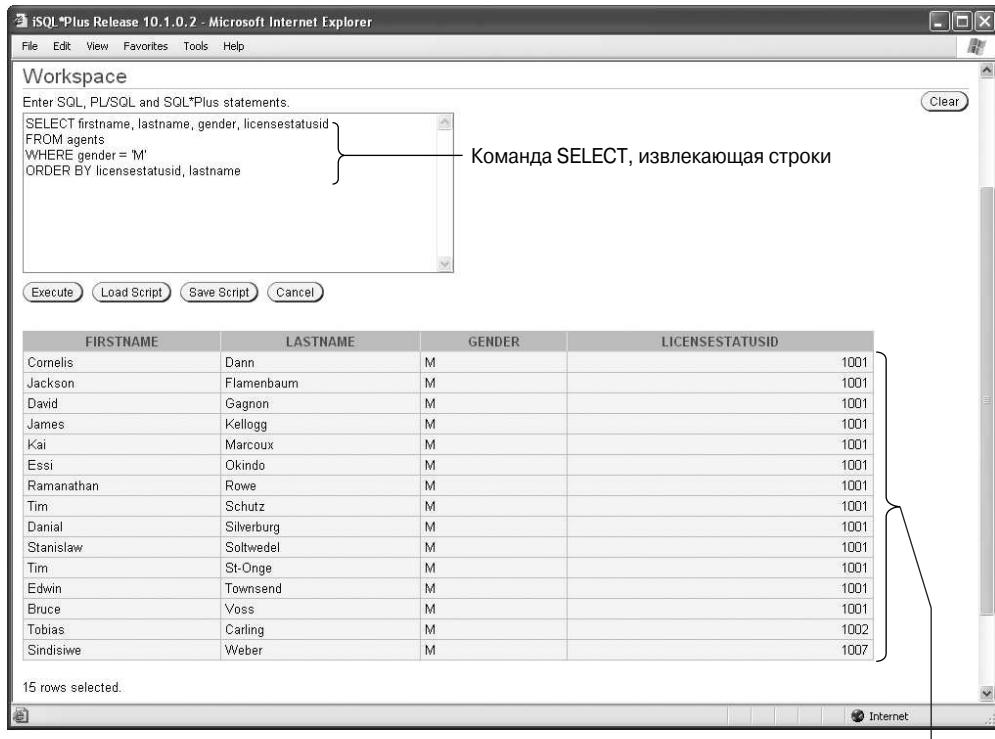
3. Щелкните на кнопке `Execute`, чтобы передать команду Oracle. В ответ возвращается 15 строк (рис. 2.10).
4. Щелкните на кнопке `Logout`. (Данная кнопка есть вверху и внизу экрана.)
5. Закройте браузер. Существуют и другие способы записи команды `SELECT`; их мы рассмотрим в главе 3.

## DLL-выражения

В число самых используемых выражений SQL, написанных на языке DDL (Data Definition Language — язык определения данных), входят `CREATE TABLE`, `ALTER TABLE` и `DROP TABLE`. Существуют и другие DDL-выражения, которые будут рассмотрены в следующих главах, однако приведенные операторы являются наиболее характерными представителями этой группы.

## Создание таблицы

Одним из основных выражений языка определения данных является `CREATE TABLE`. SQL поддерживает три типа таблиц: базовые, производные и наблюдаемые таблицы. Вы преимущественно будете работать с *базовыми таблицами* —объектами схемы, содержащими данные (как таблица `Agents`, созданная при запуске файла сценария `Ch02Agents.sql`). *Производные таблицы* представляют собой результаты, возвращае-



Строки, извлеченные из таблицы Agents

Рис. 2.10. Строки, возвращаемые в ответ на команду SELECT

мые системой баз данных в ответ на запрос (SELECT). *Наблюдаемые таблицы*, подробно рассмотренные в главе 6, являются именованными производными таблицами. Наблюдаемые таблицы обычно представляют собой сложные выражения SELECT, которым присвоено имя, позволяющее обращаться к ним так же, как к обычным базовым таблицам. Ниже мы изучим структуру и использование команды CREATE TABLE для создания базовой таблицы.

Ниже приведен сокращенный синтаксис команды CREATE TABLE, применявшийся при создании базовой таблицы. (Полное описание синтаксиса CREATE TABLE можно найти в руководстве SQL Reference Oracle Corporation.)

```
CREATE TABLE table_name
(column_name type [NOT NULL]
[CONSTRAINT constraint_definition DEFAULT expression]
[,column_name type [NOT NULL]
[CONSTRAINT constraint_definition DEFAULT expression]...]
)
[PRIMARY KEY (column_name [,column_name ...])
[FOREIGN KEY (column_name [,column_name ...])
REFERENCES foreign_table (column_name [,column_name ...])]
```

Приведенная ниже команда CREATE TABLE создает для базы данных Redwood Realty таблицу ContactReason, содержащую два столбца, с помощью которых указывается причина контакта клиента с агентством. На данный момент предусмотрено три причины: Buy (Покупка), Casual (Случайно), Sell (Продажа). Причина Buy означает, что клиент интересуется покупкой дома; Casual — что человек просто “зашел посмотреть” и в текущий момент не планирует операций с недвижимостью; Sell указывает, что соответствующий клиент связался с Redwood Realty для продажи дома. Итак, указанная команда CREATE TABLE создает таблицу ContactReason и определяет ContactReason как первичный ключ (напомним, что первичный ключ единственным образом определяет все строки таблицы).

```
CREATE TABLE ContactReason
  (ContactReason NVARCHAR2(15) NOT NULL,
   Description NVARCHAR2(50),
   CONSTRAINT pk_ContactReason PRIMARY KEY (ContactReason)
);
```

В первой строке таблице присваивается имя. После этого в круглых скобках идет список названий столбцов. Третья строка содержит определение связи для одного столбца. Условие (constraint) — это ограничение на значение, которое может иметь столбец. Условие указывает, что первый столбец, ContactReason, является первичным ключом. Следовательно, каждое значение в этом столбце должно быть уникальным для данной таблицы (значения Buy, Casual и Sell уникальны — каждое из них появляется только в одной строке таблицы).

После каждого имени столбца следует тип данных и размер. Тип данных столбца ContactReason равен NVARCHAR2 — строка символов национального языка размером до 15 символов. (*Набор символов национального языка* позволяет записывать информацию из языков, отдельные символы которых нельзя представить в восьмибитовой схеме кодирования.) Необязательный оператор NOT NULL указывает, что значение в этом столбце нельзя пропускать. Второй столбец, Description, позволяет использовать строки символов национального языка размером до 50 символов.

## **Добавление и отбрасывание столбцов**

Одной из удобных особенностей реляционных баз данных является возможность добавлять, изменять и удалять столбцы. Предположим, что таблица сотрудников содержит несколько сотен строк с типичной информацией о персонале: имя, адрес, телефон, дата найма и дата рождения. За последний год (или около того) каждый сотрудник получил для использования в разъездах служебный мобильный телефон. Изначально таблица не содержала столбца, предназначенного для номера мобильных телефонов. Можете ли вы добавить еще один столбец для новой информации? Да! Используйте для этой цели оператор SQL ALTER TABLE. Ниже показана команда, позволяющая добавить в таблицу Employee столбец MobilePhone.

```
ALTER TABLE Employee
    ADD MobilePhone VARCHAR2(14);
```

Oracle добавляет столбец `MobilePhone` в конец таблицы и присваивает всем его позициям значение `NULL`. Позже в нужные строки таблицы можно ввести номера мобильных телефонов. Подобным образом можно модифицировать или удалять существующие столбцы таблицы. Для этого используются соответственно команда `ALTER TABLE` с уточнением `MODIFY` или `DROP COLUMN`. Например, приведенная ниже команда SQL удаляет из таблицы столбец `YearStarted`.

```
ALTER TABLE Employee
    DROP COLUMN YearStarted;
```

Подобным образом указанная ниже команда заменяет текущий тип данных столбца типом `DATE`.

```
ALTER TABLE Employee
    Modify YearStarted DATE DEFAULT SYSDATE;
```

Кроме того, оператор `DEFAULT` гарантирует, что в любое незаполненное поле `YearStarted` будет автоматически внесена сегодняшняя дата — значение, возвращаемое функцией `SYSDATE`.

Время от времени требуется удалять нежелательные позиции таблицы. Предположим, например, что вам больше не требуется таблица `CheapWebHost`, содержащая имена и URL дешевых Web-хостов. Вы уже выбрали хост, установили на него систему, поэтому указанная таблица стала бесполезной. Разумеется, можно удалить все строки, однако при этом останется структура таблицы. Чтобы вместе со всеми строками убрать определение таблицы, используйте команду `DROP TABLE`, имеющую следующий синтаксис:

```
DROP TABLE table_name [CASCADE CONSTRAINTS] [PURGE]
```

При удалении таблицы с опцией `CASCADE CONSTRAINTS` таблица вместе со всей содержащейся в ней информацией, другими объектами, ссылающимися на таблицу или зависящими от нее (включая представления, процедуры и присоединенные процедуры; см. ниже), переносится в корзину. При использовании необязательного оператора `PURGE` таблица удаляется, но не помещается в корзину. Вместо этого она безвозвратно удаляется из системы. Например, приведенная ниже команда удаляет таблицу `CheapWebHost` и очищает корзину.

```
DROP TABLE CheapWebHost CASCADE CONSTRAINTS PURGE;
```

## DML-выражения

Одной из фундаментальных функций любой базы данных является возможность манипулирования содержащейся в ней информацией. Именно для этого предназначены SQL-выражения языка манипулирования данными (Data Manipulation Language — DML), позволяющие добавлять, модифицировать и удалять данные из таблиц базы

данных. Возможность вставки новых строк данных, модификации ошибочной или изменившейся информации, а также удаления устаревших записей необходима, чтобы информация не устаревала и была точной. Для реализации названных возможностей применяются выражения SQL `INSERT`, `UPDATE` и `DELETE`.

## Вставка данных

Команда `INSERT` применяется для вставки значения в некоторые или во все столбцы (или поля) таблицы. Синтаксис стандартной команды `INSERT` достаточно прост:

```
INSERT INTO table_name [(column_name [, column_name ...])]  
VALUES (value [, value...])
```

При вставке значений во все столбцы таблицы можно использовать следующую форму команды:

```
INSERT INTO table_name  
VALUES (value, value, value, ...)
```

Обратите внимание на то, что в предыдущей форме опущен список столбцов таблицы — указываются только значения, которые они получают. Если вам требуется вставить значения во все столбцы, в команде `INSERT` оператор `VALUES` должен содержать значения для каждого столбца — ни одно значение пропустить нельзя. Если вы не знаете, какое значение должен иметь столбец, используйте вместо него слово `NULL`. (`NULL` — это зарезервированное слово, означающее, что значение неизвестно.) Ниже показано, как с помощью команды `INSERT` добавить новую строку в таблицу `LicenseStatus`, зная все значения и порядок, в котором идут поля таблицы.

```
INSERT INTO LicenseStatus  
VALUES (1017, 'Reactivated');
```

При использовании данной формы список значений должен удовлетворять следующим условиям.

- Если в команде `INSERT` не заданы названия столбцов, для каждого столбца должно указываться значение, причем данные значения должны идти в том же порядке, в каком они определены в таблице.
- Неизвестные значения должны представляться значением `NULL`.

Если требуется ввести лишь несколько значений, то можно в любом порядке перечислить названия столбцов, получающих значения, а затем указать список `VALUES`, содержащий получаемые значения. Предположим, например, что в таблицу `Agents` требуется добавить нового агента по недвижимости, а все, что у вас есть, — это присвоенное брокером данному агенту значение `AgentID`, а также имя и фамилия агента. В таком случае проще всего написать команду `INSERT` со списком из трех столбцов, за которым в операторе `VALUE` следуют три значения. Попробуем это сделать с помощью SQL\*Plus.

The screenshot shows the Oracle SQL\*Plus interface. The title bar reads "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following text:

```

SQL*Plus: Release 10.1.0.2.0 - Production on Sat Sep 30 09:42:54 2006
Copyright (c) 1982, 2004, Oracle. All rights reserved.

Connected to:
Personal Oracle Database 10g Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> INSERT INTO Agents (AgentID, FirstName, LastName)
  2  VALUES (10001, 'Melinda', 'Whitmore');

1 row created.

SQL>

```

**Рис. 2.11.** Вставка новой строки и проверка ее значений

Для того чтобы добавить нового агента в таблицу Agents, зная только значения трех столбцов, выполните следующие действия.

1. Откройте SQL\*Plus и войдите в Oracle со своим именем пользователя и паролем.
2. В командной строке SQL> наберите следующий текст:

```

INSERT INTO Agents (AgentID, FirstName, LastName)
VALUES (10001, 'Melinda', 'Whitmore');

```

3. Нажмите клавишу <Enter>, чтобы выполнить команду. SQL\*Plus проверит команду и передаст ее базе данных Oracle. Oracle ответит сообщением “1 row created”, указывающим, что база данных добавила одну строку (рис. 2.11).
4. Проверьте, что строка действительно была вставлена, затребовав информацию об агентах. Для этого в командной строке SQL> введите следующее:

```

SELECT AgentID, FirstName, LastName
FROM Agents
WHERE AgentID < 10200;

```

5. Нажмите клавишу <Enter>, чтобы выполнить команду. Oracle покажет, что заданному критерию отвечают две строки, одна из которых была введена вами только что. Не закрывайте окно SQL\*Plus — мы продолжим работать с ним позже.

Рассмотрим третью форму команды INSERT, использующую оператор SELECT для копирования строк из другой таблицы. В данной форме вместо VALUES применяется оператор SELECT:

```

INSERT INTO table_name (column_name1, column_name2, ...)
  SELECT column1, column2, ...
    FROM table_name [WHERE search_condition]

```

Здесь вы перечисляете столбцы, в которые требуется вставить значения, а затем с помощью команды SELECT извлекаете аналогичное число столбцов для вставки (тип новых и старых столбцов также должен совпадать). Необязательный оператор

WHERE позволяет фильтровать строки. Например, с его помощью вы можете добавить строки в таблицу Agents, копируя значения из другой таблицы, столбцы которой имеют тот же тип данных, что и столбцы таблицы Agents:

```
INSERT INTO Agents (AgentID, LastName, FirstName)
    SELECT NewAgentNumber, AgentLastName, AgentFirstName
        FROM NewlyLicensedAgents;
```

Помните, что тип данных столбцов таблицы Agent должен быть таким же, как у таблицы NewAgentNumber, кроме того, каждая строка, добавляемая в таблицу, должна иметь уникальный идентификатор нового агента, не совпадающий ни с одним номером в таблице Agents. Как видите, с помощью данного метода удобно копировать большие объемы данных из одной таблицы в другую.

## Обновление строк

Вас не должно удивлять, что иногда информацию в таблицах приходится исправлять из-за ошибок ввода данных или обновлять из-за изменения информации. Адреса клиентов могут регулярно изменяться, сотрудников повышают по службе, а наличный товар обновляется постоянно. В связи с этим приходится использовать команду UPDATE, позволяющую модифицировать одно или несколько полей в одной или нескольких строках таблиц Oracle. Синтаксис команды UPDATE выглядит так:

```
UPDATE table_name
    SET column_name = expression [, column_name = expression, ...]
        [WHERE search_condition]
```

Видно, что оператор SET обязательный, а оператор WHERE — нет. В целом вы задаете имя обновляемой таблицы плюс пары “имя столбца—новое значение”. Если вы зададите условие WHERE, обновляться будут только строки, удовлетворяющие условиям поиска.

Предположим, что вам требуется добавить номера мобильного телефона и лицензии агента Melinda Whitmore (информацию, которой вы не располагали при вставке в таблицу соответствующей записи). Для решения этой задачи можно использовать команду UPDATE, найдя нужную запись и обновив требуемые значения.

Для того чтобы обновить три значения для определенного агента по недвижимости, выполните следующие действия.

1. В окне SQL\*Plus введите CLEAR SCREEN, чтобы очистить экран от предыдущей информации.
2. Откройте программу Блокнот и введите приведенный ниже код (нажмите клавишу <Enter>, чтобы перейти на следующую строку для трех первых строк, однако *не нажимайте* клавишу <Enter> в конце четвертой строки). Будьте очень внимательны — номер мобильного телефона, имя и фамилия заключаются в одинарные кавычки. Прописные и строчные буквы используйте точно так, как показано ниже.

Новые значения для указанных столбцов записи Мелинды Уитмо

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> UPDATE Agents
  2 SET CellPhone = '(707) 555-8099', LicenseStatusID = 1001
  3 WHERE FirstName = 'Melinda' AND LastName = 'Whitmore';

1 row updated.

SQL> COMMIT;

Commit complete.

SQL>

```

**Рис. 2.12.** Обновление строки в таблице Agents

(Оператор COMMIT рассмотрен ниже в этой главе.)

```

UPDATE Agents
SET CellPhone = '(707) 555-8099', LicenseStatusID = 1001
WHERE FirstName = 'Melinda' AND LastName = 'Whitmore';
COMMIT;

```

3. В программе Блокнот нажмите комбинацию клавиш **<Ctrl+A>** чтобы выбрать весь текст, затем нажмите комбинацию клавиш **<Ctrl+C>**, чтобы скопировать его в буфер.
4. Переключитесь на SQL\*Plus, нажмите комбинацию клавиш **<Ctrl+V>**, чтобы вставить текст в SQL\*Plus, и нажмите клавишу **<Enter>** чтобы выполнить команду COMMIT. Если SQL\*Plus укажет на наличие синтаксических ошибок, переключитесь на Блокнот, исправьте нужное место и повторите шаг 3 и 4. Oracle отобразит сообщения: “1 row updated” и “Commit complete” (рис. 2.12). Теперь поля CellPhone и LicenseStatusID записи MelindaWhitmore будут обновлены новыми значениями. (Чтобы проверить внесенные изменения, можете использовать команду SELECT.)
5. Переключитесь на Блокнот, запишите команду SQL, если она вам нужна, и закройте Блокнот. Окно SQL\*Plus не закрывайте.

## Удаление строк из таблицы

Для удаления строк из таблицы базы данных применяется команда SQL **DELETE**. С ее помощью можно удалить одну, несколько или все строки таблицы. Впрочем, для удаления всех строк лучше использовать команду **TRUNCATE**. Разница между командами **DELETE** и **TRUNCATE** заключается в том, что в определенных обстоятельствах строки, удаленные с помощью команды **DELETE**, можно восстановить с помощью команды **ROLLBACK**, которая описана в следующем разделе. Если же строки были удалены с помощью команды **TRUNCATE**, то восстановить их невозможно. Кроме

того, команда TRUNCATE выполняет определенные дополнительные действия, например, высвобождает место, которое занимала таблица, и т.п. Команда DELETE имеет следующий синтаксис:

```
DELETE FROM table_name
[WHERE search_condition]
```

Команда DELETE FROM требует, чтобы вы указали таблицу, из которой необходимо удалить строки. Необязательная команда WHERE используется для того же, что и команда SELECT или UPDATE: для указания условий поиска, которым должна удовлетворять удаляемая строка. Если вы пропустите команду WHERE (распространенная ошибка новичков), то Oracle удалит все строки таблицы. Отметим, что удалять строки из таблицы можно как с помощью DELETE, так и с помощью TRUNCATE, хотя ни одна из названных команд не удаляет определение таблицы. Чтобы удалить таблицу и ее строки, применяйте команду DROP TABLE (она описана ранее в этой главе).

Для того чтобы удалить строку из таблицы Agents, выполните следующие действия.

- Если вы закрыли SQL\*Plus, откройте его и войдите в базу данных, используя ваше имя пользователя и пароль. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы начать с командной строки SQL и чистого экрана.
- После SQL> введите приведенную ниже команду, чтобы удалить из таблицы Agents нового агента (Melinda Whitmore), и нажмите клавишу <Enter> в конце обеих строк:

```
DELETE FROM Agents
WHERE AgentID = 10001;
```

Oracle отобразит подтверждающее сообщение “1 row deleted”.

- Проверьте, что запись MelindaWhitmore удалена из таблицы, отобразив на экране имена и фамилии всех агентов. Для этого в командной строке SQL> введите следующую команду (рис. 2.13):

```
SELECT FirstName, LastName
FROM Agents
ORDER BY LastName DESC;
```

- Как видите, запись MelindaWhitmore исчезла из списка агентов. (Изображение, которое вы увидите на экране, может отличаться от приведенного в книге и иметь больше заголовков столбцов. Это зависит от настройки среды SQL\*Plus, имеющей LINELENGTH. Подробно LINELENGTH рассматривается в следующем разделе.) Не закрывайте SQL\*Plus. Если вы закроете SQL\*Plus, а затем откроете его, когда будете читать следующий раздел, некоторые действия будут выполняться неверно.

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> DELETE FROM Agents
  2 WHERE AgentID = 10001;
1 row deleted.

SQL> SELECT FirstName, LastName
  2 FROM Agents
  3 ORDER BY LastName DESC;

FIRSTNAME          LASTNAME
-----            -----
Christine          Williams
Sindisiwe          Weber
Bruce              Voss
Edwin              Townsend
Jessica             Taylor
Tim                St-Onge
Stanislaw           Soltwedel
Danial              Silverburg
Heather             Sheibani
Ricki               Selby
Tim                Schutz
Ramanathan         Rose
Cecilia             Romero
Clair               Robinson
Lee                Reed
Nancy              Piperova
Essi                Okindo
Kai                Marcoux
Patricia            Lewis
James               Kellogg
Barbara             Herring
David               Gagnon
Jackson             Flamenbaum
Crystal             Fernandez
Cornelis            Dann
Elizabeth           Dahlen
Belinda             Chong
Tobias              Carling
Lora                Allee

29 rows selected.

SQL>

```

Подтверждение удаления указанной строки

**Рис. 2.13.** Удаление строки и проверка результата

## ТС-выражения

Предназначение ТС-команд SQL понятно из названия: ТС — это аббревиатура от Transaction Control (“управление транзакциями”). *Транзакцией* базы данных называется группа выражений SQL, представляющих собой логическую единицу работы. Выражения в транзакциях представляют собой нераздельно связанную группу, отдельные выражения которой должны завершаться успешно. В противном случае необходимо “откатить” весь набор выражений SQL, чтобы база данных вернулась в состояние, предшествовавшее выполнению транзакции. Распространенным примером транзакции является перевод денег с депозита на текущий счет. Одно выражение UPDATE уменьшит количество денег на депозите на величину перевода, а последующая команда UPDATE увеличит на ту же величину количество денег на текущем счете. Если по какой-либо причине одна из команд UPDATE даст неверный результат, фонды затеряются в киберпространстве и в бухгалтерских книгах банка не удастся свести баланс. Выражения управления транзакциями предотвращают такую возможность, связывая две операции в одну транзакцию и требуя, чтобы обе команды UPDATE записывались в базу данных одновременно или же чтобы их эффект обнулялся для возврата базы данных в состояние, предшествовавшее выполнению транзакции.



**Рис. 2.14.** Пример структуры транзакции

## Принятие или отклонение изменений

При выполнении DDL-выражения, например, использовании CREATE TABLE для добавления новой таблицы в базу данных, Oracle сразу изменяет базу данных, что видно любому пользователю, обращающемуся к таблице. Для DML-выражений (INSERT, UPDATE и DELETE) это не так, если вы не “включите” специальную переменную SQL\*Plus AUTOCOMMIT. При “включении” AUTOCOMMIT (SET AUTOCOMMIT ON) любое изменение, вызываемое DML-выражением, записывается в базу данных сразу же после выполнения.

Чтобы записать результаты одного или нескольких выражений SQL, образующих ряд выражений транзакции, вы фиксируете транзакцию, выполняя выражении COMMIT. Чтобы обратить результат одного или нескольких выражений SQL, образующих транзакцию, вы выполняете откат с помощью команды ROLLBACK. Любые изменения базы данных, выполненные до команды ROLLBACK (от последнего выражения COMMIT), отбрасываются. Когда вы отключаетесь от базы данных, Oracle автоматические фиксирует все изменения базы данных, внесенные пользователем. На рис. 2.14 показана одна транзакция, размещенная между командами COMMIT.

Напомним, что на предыдущем этапе вы ввели нового агента (Мелинду Уитмо), обновили ее номер мобильного телефона и ввели команду COMMIT, чтобы сделать постоянной добавленную и обновленную запись об агенте. После этого вы удалили записи Мелинды из таблицы Agents. Если вы не выходили из системы Oracle после выполнения команды DELETE, то можете восстановить запись Мелинды. Если же вы вышли из системы после удаления записи, восстановление невозможно, поскольку Oracle выполняет “тихую” команду фиксации транзакций. Ниже описаны действия, необходимые для отмены операции DELETE и восстановления записи Мелинды Уитмо в таблице Agents.

Итак, чтобы отменить все операции, выполненные после последней команды COMMIT, выполните следующие действия.

1. Введите ROLLBACK, затем ; (точка с запятой) и нажмите клавишу <Enter>. Oracle отобразит сообщение “Rollback complete”.
2. В командной строке SQL> введите следующий текст:

```
SELECT LastName, CellPhone  
FROM Agents  
ORDER BY LastName DESC;
```

3. После этого нажмите клавишу <Enter>, чтобы проверить, что запись Мелинды Уитмо восстановлена. Обратите внимание на то, что ее имя снова появилось в списке агентов (второе сверху) вместе и обновленным номером мобильного телефона. Следовательно, вы отменили команду DELETE — последнюю транзакцию после предыдущего выражения COMMIT.

Поскольку запись Мелинды в таблице Agents нам не нужна, введите представленные ниже команды, чтобы навсегда удалить ее запись из базы данных.

Для того чтобы безвозвратно удалить запись, выполните следующие действия.

1. В командной строке SQL> введите команду.

```
DELETE FROM Agents  
WHERE AgentID = 10001;
```

2. Чтобы удаление было безвозвратным, введите команду  
COMMIT;

Oracle ответит сообщением “Commit complete”.

3. Введите Exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus.

## Установка точек отката

В любом месте транзакции можно установить точку сохранения, которая позволит отменить любые изменения, внесенные после нее. Точки сохранения широко используются в очень длинных транзакциях для минимизации числа команд, которые нужно отменять при сбое транзакции. Для создания точек сохранения применяется команда SAVEPOINT, которая имеет следующий синтаксис:

```
SAVEPOINT savepoint_name
```

В данном выражении savepoint\_name — любое значимое имя. Например, можно объявить точку сохранения, если ввести команду

```
SAVEPOINT AfterFinancialUpdates;
```

Если вы выполняете DML-выражения до и после команды SAVEPOINT, состояние базы данных, существовавшее в точке AfterFinancialUpdates, можно восстановить с помощью выражения

```
ROLLBACK TO SAVEPOINT AfterFinancialUpdates;
```

## DCL-выражения

В многопользовательской базе данных Oracle доступ пользователей к системе необходимо контролировать. Любое обращение к базе данных можно ограничить — пользователю можно разрешать и запрещать доступ к различным объектам базы данных (таблицы, представления и отчеты). Для обеспечения безопасности базы данных применяются DCL-выражения (Data Control Language — язык управления данными), в которые входят команды SQL, разрешающие и запрещающие доступ к системе и содержащимся в ней объектам. Когда вы создаете таблицы и другие объекты в своей схеме (определенной именем пользователя), другие пользователи не могут обращаться к этим объектам, если вы явно не разрешите им этого. Кроме того, новые пользователи вообще могут разве что войти в базу данных, если администратор базы данных — самый надежный пользователь системы — не предоставит им привилегии на использование систем и объектов. Для предоставления привилегий применяется команда GRANT, для удаления — команда REVOKE. Хотя вы слабо можете повлиять на привилегии, присвоенные вам, вы можете предоставить другим пользователям привилегии на использование созданных объектов. Соответственно, вы можете и лишить привилегий на собственные объекты.

Существуют два типа привилегий: касающиеся системы и объектов. *Привилегии системы* позволяют пользователю выполнять в Oracle определенные действия (создавать таблицы, представления и т.д.). *Привилегии объектов* разрешают пользователю совершать определенные действия по отношению к объектам базы данных (например, вставлять данные в таблицу другого пользователя).

### Предоставление привилегий

Для предоставления привилегий на пользование системой и объектами применяется команда GRANT, которая имеет следующий синтаксис.

```
GRANT privilege [, privilege [, privilege ] ...] | ALL PRIVILEGES
ON object
TO user [IDENTIFIED BY password] [, user2 [, user3]...] | PUBLIC
[WITH GRANT OPTION]
```

В приведенном выражении вместо всех слов, выделенных курсивом, необходимо подставить соответствующие значения. Предположим, например, что ваше имя пользователя Pine и вы владеете таблицей Agents. Вы желаете, чтобы пользователь Elm мог просматривать объекты вашей таблицы Agents, но при этом вы *не хотите*, чтобы этот пользователь мог изменять или удалять записи из таблицы Agents. Для этого необходимо войти в базу данных под именем Pine и ввести следующую команду SQL:

```
GRANT SELECT ON Pine.Agents TO Elm;
```

## Лишение привилегий

Любой пользователь, владеющий объектом, может запретить доступ других пользователей к своим (созданным этим пользователем) объектам базы данных. Используемая для этого команда REVOKE имеет следующий синтаксис:

```
REVOKE privilege[, privilege[, privilege] ...] | ALL PRIVILEGES
ON object
FROM user [, user2 [, user3 ]...] | PUBLIC
```

Чтобы лишить пользователя Elm возможности отображать на экране строки таблицы Agents, владельцем которой является Pine, пользователь Pine должен ввести следующую команду SQL:

```
REVOKE SELECT ON Pine.Agents FROM Elm
```

Существует несколько привилегий объектов, которые вы можете предоставить другим пользователям: SELECT, INSERT, UPDATE, DELETE и EXECUTE.

Администраторы базы данных обладают достаточными привилегиями для предоставления другим пользователям разнообразных привилегий системы: CREATE SESSION, CREATE TABLE, CREATE USER, CREATE VIEW, CREATE SYNONYM и т.д. Проверить, какими привилегиями обладаете вы, можно с помощью простой команды SELECT. Войдите в Oracle, используя SQL\*Plus, и выполните описанные ниже действия.

Чтобы проверить, какими привилегиями системы вы обладаете, выполните следующие действия.

- Если необходимо, запустите SQL\*Plus и войдите в базу данных со своим именем пользователя и паролем.
- Ведите следующую команду (будьте внимательны, не пропустите два символа подчеркивания):  

```
SELECT *
FROM user_sys_privs;
```
- Нажмите клавишу <Enter>, чтобы передать команду базе данных. В ответ Oracle отобразит ваше имя пользователя и перечислит все привилегии, которыми вы обладаете, например, право создавать таблицы.
- Ведите exit и нажмите клавишу <Enter>, чтобы выйти из базы данных Oracle и закрыть SQL\*Plus.

В первом столбце возвращаемой таблицы указано ваше имя пользователя, затем идет столбец привилегий (каждая привилегия приведена в отдельной строке). Кроме того, указывается, можете ли вы предоставлять привилегии другим пользователям. На рис. 2.15 показан список системных привилегий, предоставленных пользователю Woody. Разумеется, в вашем случае список может быть другим.

Подробно (с примерами) предоставление и лишение привилегий будут рассмотрены в главе 12 “Поддержание безопасности базы данных”.

USERNAME	PRIVILEGE	ADM
WOODY	CREATE VIEW	NO
WOODY	CREATE TABLE	NO
WOODY	CREATE SESSION	NO
WOODY	CREATE SEQUENCE	NO

Рис. 2.15. Список системных привилегий

## Использование расширений команд SQL\*Plus

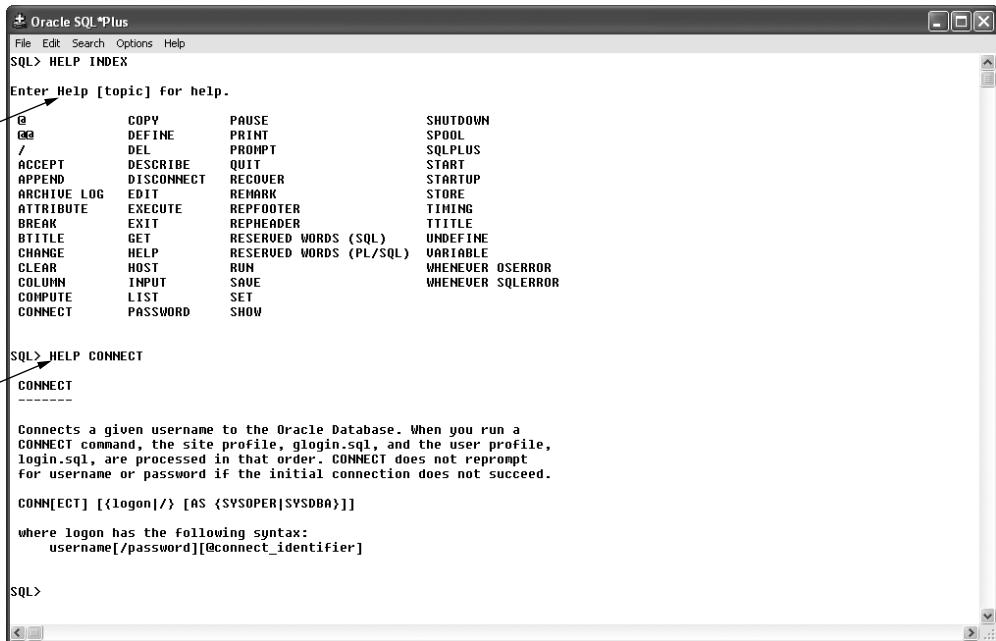
Клиент SQL\*Plus от Oracle предлагает несколько *команд SQL\*Plus*, которые отличаются от стандартных команд SQL и позволяющих форматировать получаемые столбцы, копировать возвращаемые результаты в буфер, запускать внешние файлы сценариев, отображать структуру таблицы и определения типов данных, менять пароль и соединяться с системой баз данных — всего предлагается около 80 команд. Некоторые из этих команд вы уже использовали при изучении материала данной главы (например, CLEAR SCREEN). Все они определены только в среде SQL\*Plus (но не обязательно в среде iSQL\*Plus) и только для Oracle. Одной из команд SQL\*Plus, позволяющих экономить время, является команда START, которую вы уже использовали ранее для запуска файла сценария, содержащего команды SQL.

Мы не будем описывать все отдельные команды SQL\*Plus, а рассмотрим их по категориям или выполняемым функциям. Как вы могли заметить, команды SQL\*Plus не заканчиваются точкой с запятой, кроме того, многие команды SQL\*Plus допускают сокращенную форму записи (DESCRIBE или DESC), а для команд SQL такое невозможно. Наконец, некоторые команды SQL\*Plus не поддерживаются в iSQL\*Plus. Следовательно, все примеры, приведенные в конце главы, предполагают использование клиентской программы SQL\*Plus.

*Примечание.* Вы уже знаете, что в SQL\*Plus необходимо нажимать клавишу <Enter> в конце каждой строки, чтобы перейти на следующую строку или выполнить команду SQL, завершающуюся точкой с запятой. Начиная с этого момента мы часто будем опускать такие подробности, надеясь, что вы их не забудете, т.е. мы будем говорить “введите команду” или “выполните команду”, не уточняя “нажмите клавишу <Enter>”.

Для того чтобы вывести на экран список всех команд SQL\*Plus, выполните следующие действия.

- Если необходимо, запустите SQL\*Plus и войдите в базу данных, используя свое имя пользователя и пароль.



Отображает помощь по указанной команде SQL\*Plus

Перечисляет команды SQL\*Plus

**Рис. 2.16.** Перечень команд SQL\*Plus и справка о команде CONNECT

2. Ведите CLEAR SCREEN, чтобы очистить экран.
3. Ведите команду HELP INDEX (набрать ее можно и строчными буквами) без точки с запятой, чтобы отобразить список из более чем 50 команд SQL\*Plus.
4. Ведите HELP CONNECT (рис. 2.16).

После ввода выражения HELP CONNECT SQL\*Plus отобразит информацию об использовании и синтаксисе команды SQL\*Plus. Если вы затребуете помощь о команде SQL, например CREATE TABLE, SQL\*Plus вернет сообщение об ошибке, указывающее, что найти помощь по данной теме не удается. Помните, что ответы на все вопросы, связанные с SQL, можно найти в сети OTN (Oracle Technical Network). Для решения вопросов, связанных с SQL\*Plus, используйте команду SQL\*Plus HELP.

## Описание структуры таблицы

Команда SQL\*Plus DESCRIBE отображает определения столбцов таблицы, представления или синонима. Кроме того, она приводит спецификацию функции или процедуры (подробнее об этом — в следующих главах). Если вы не можете вспомнить названия определенных столбцов таблицы — выведите на экран список всех их.

```

SQL> DESCRIBE Agents
Name          Null?    Type
-----        -----
AGENTID      NOT NULL NUMBER(38)
FIRSTNAME    NVARCHAR2(30)
LASTNAME     NVARCHAR2(30)
HIREDATE     DATE
BIRTHDATE    DATE
GENDER       NVARCHAR2(10)
WORKPHONE   NVARCHAR2(20)
CELLPHONE   NVARCHAR2(20)
HOMEPHONE   NVARCHAR2(20)
TITLE        NVARCHAR2(20)
TAXID        NVARCHAR2(20)
LICENSEID    NVARCHAR2(20)
LICENSEDATE  DATE
LICENSEEXPIRE DATE
LICENSESTATUSID NUMBER(38)

SQL> DESCRIBE SaleStatus
Name          Null?    Type
-----        -----
SALESTATUSID NOT NULL NUMBER(38)
SALESTATUS   NVARCHAR2(10)

SQL> DESCRIBE LicenseStatus
Name          Null?    Type
-----        -----
LICENSESTATUSID NOT NULL NUMBER(38)
STATUSTEXT   NVARCHAR2(25)

SQL>

```

Рис. 2.17. Отображение определений столбцов таблицы

Для того чтобы отобразить названия и типы данных столбцов, выполните следующие действия.

1. В командной строке SQL\*Plus выполните команду  
`DESCRIBE Agents`  
Oracle отобразит имена и типы данных 15 столбцов таблицы Agents.
2. Введите `DESCRIBE SaleStatus`.
3. Введите `DESCRIBE LicenseStatus` (рис. 2.17). не закрывайте SQL\*Plus.

## Захват файлов для печати

Иногда требуется записать результаты сеансов SQL\*Plus (команды SQL и SQL\*Plus, а также результаты их выполнения). Записав команды, выполняемые Oracle, подтверждающие сообщения и результаты запросов в файл, вы можете впоследствии его напечатать. Хотя для решения подобной задачи можно использовать средства Windows (например, нажать клавишу `<Print Screen>`, чтобы занести в буфер находящееся в текущий момент на экране содержимое окна SQL\*Plus), они неудобны, если выход Oracle превышает одну страницу. В связи с этим рекомендуется применять команду SQL\*Plus `SPOOL`, позволяющую захватывать в файл весь текст, появляющийся в окне SQL\*Plus (как вводимые команды, так и результаты, возвращаемые базой данных).

В *iSQL\*Plus* команда SPOOL отсутствует. (Если вы попытаетесь выполнить SPOOL в *iSQL\*Plus*, то получите соответствующее сообщение об ошибке.) SPOOL — это “бистабильная система”, т.е. вы включаете ее для записи, а затем отключаете. Синтаксис команды вызывается с помощью HELP SPOOL и выглядит следующим образом:

```
SPOOL [file_name[.ext] [CREATE | REPLACE | APPEND | OFF | OUT]]
```

Любое слово, представленное прописными буквами (за исключением OFF и OUT), можно сократить до первых трех букв (например, SPO вместо SPOOL). Чтобы реализовать буферизацию данных в файл, который впоследствии можно напечатать, после команды SPOOL указывается полное имя файла (включая путь). После того как вы указали файл, все, что будет вводиться или отображаться на экране, будет записываться в файл. Так будет продолжаться до тех пор, пока вы либо не выйдете из SQL\*Plus, либо не введете SPOOL OFF. Попрактиковаться в использовании данной программы вы сможете в следующем разделе, где рассматриваются команды форматирования.

## Команды форматирования

Для форматирования заголовков столбцов и данных, содержащихся в этих столбцах, применяется команда SQL\*Plus COLUMN. Полный ее синтаксис достаточно длинный, поэтому мы приведем только сокращенную версию:

```
COLUMN [{column_name | expression} [option...]]
```

Одними из наиболее важных опций являются CLEAR, FORMAT, HEADING, JUSTIFY и WRAP. Команда COLUMN задает атрибуты отображения столбца, включая текст заголовка, выравнивание столбца, формат числовых данных, а также возможность перехода данных на новую строку, если это требуется. Например, приведенная ниже команда SQL\*Plus устанавливает характеристики отображения столбца AgentID до конца сеанса SQL\*Plus.

```
COLUMN AgentID FORMAT 99999 HEADING 'Agent ID'
```

В частности, на экране отображается только пять цифр идентификатора агента, а заголовок изменен с принятого по умолчанию (название столбца таблицы) на “Agent ID”. С помощью нескольких строк SQL\*Plus вы можете установить любое количество названий столбцов и задать множество настроек. Чтобы удалить форматирование, установленное в ходе текущего сеанса SQL\*Plus, применяется команда CLEAR COLUMNS. При ее выполнении удаляется все форматирование столбцов. Отметим, что команда COLUMN особенно полезна в часто используемом файле сценария. Разберем подробнее команды SQL\*Plus SPOOL и COLUMN.

Для того чтобы скопировать выход окна SQL\*Plus в файл и отформатировать столбцы, выполните следующие действия.

1. При открытом окне SQL\*Plus запустите программу Блокнот.
2. В Блокноте введите следующие команды SQL\*Plus (не нажимайте клавишу <Enter> в конце последней строки):

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> CLEAR COLUMNS
columns cleared
SQL> COLUMN AgentID FORMAT 99999 HEADING 'Agent ID'
SQL> COLUMN FirstName FORMAT A15 HEADING 'First Name'
SQL> COLUMN LastName FORMAT A15 HEADING 'Last Name'
SQL> SELECT AgentID, FirstName, LastName
  2  FROM Agents;
Agent ID First Name      Last Name
-----  -----
 10041 Kai            Marcoux
 10235 Tobias          Carling
 10429 Elizabeth        Dahlien
 10497 Ramanathan       Rowe
 10849 Heather          Sheibani
 10913 Bruce            Uoss
 11775 Cecilia          Romero
 12211 Cornelis         Dann
 12301 Daniel           Silverburg
 12499 Clair             Robinson
 12715 Nancy            Piperova

Agent ID First Name      Last Name
-----  -----
 12765 Edwin            Townsend
 12875 Essi              Okindio
 12963 Lee               Reed
 13353 Stanislaw         Soltwedel
 13555 Belinda           Chong
 13649 Ricki             Selby
 13771 Jessica            Taylor
 14117 Tim                St-Onge
 14447 Jackson           Flamenbaum
 14599 Barbara            Herring
 14601 Sindisiwe         Weber

Agent ID First Name      Last Name
-----  -----
 14677 James             Kellogg
 14883 Crystal            Fernandez
 15061 Christine          Williams
 15233 David              Gagnon
 15293 Lora                Allée
 15349 Tim                 Schutz
 15521 Patricia            Lewis

```

Рис. 2.18. Изменение заголовков столбцов

```

CLEAR SCREEN
CLEAR COLUMNS
COLUMN AgentID FORMAT 99999 HEADING 'Agent ID'
COLUMN FirstName FORMAT A15 HEADING 'First Name'
COLUMN LastName FORMAT A15 HEADING 'Last Name'
SELECT AgentID, FirstName, LastName
FROM Agents;

```

- Запишите данный фрагмент в файле Блокнота Ch2<имя>FormatColumns.sql в удобной для вас папке.
- Скопируйте весь текст из Блокнота в SQL\*Plus. SQL\*Plus немедленно интерпретирует команду COLUMN, однако не будет выполнять выражение SQL SELECT (если вы не нажали клавишу <Enter> после точки с запятой в файле программы Блокнот).
- Нажмите клавишу <Enter>, чтобы выполнить команду SELECT и отформатировать столбцы заданным образом (рис. 2.18).

*Модель форматирования A15*, указанная после модификатора FORMAT, означает, что столбец является буквенно-цифровым, и ограничивает отображаемый в нем текст

15 позициями. Более длинные записи усекаются справа. Маска формата 99999 означает, что столбец является числовым, а SQL\*Plus будет отображать до пяти цифр. Если изменить маску формата, например на 999, в столбце появится запись ####, указывающая, что значения в столбце шире, чем позволяет маска формата. В таком случае увеличьте количество цифр-заполнителей. Далее предположим, что мы задали следующее условие:

```
COLUMN AgentID FORMAT 99,999 HEADING 'Agent ID'
```

Тогда идентификационный номер агента будет идти с запятой между третьей и четвертой значащей цифрой (например, 10,041).

Обратите внимание на то, что заголовки столбцов отличаются от своего обычного формата. Каждый заголовок содержит два слова, написанных не прописными буквами, как принято по умолчанию. Скорее всего, вы увидите, что под заголовками идет всего 11 строк с информацией об агентах. Добавьте к этому две строки заголовка и пустую строку, предшествующую заголовку, и вы получите 14 строк — значение по умолчанию переменной среды SQL\*Plus PAGESIZE. Переменная PAGESIZE определяет число строк, отображаемых между верхом заглавия и низом страницы. Вы можете изменить ее значение, добавив в файл сценария выражение SQL\*Plus SET PAGESIZE <value>, где <value> — число строк на странице. Максимальное значение PAGESIZE равно 50 000.

Для того чтобы модифицировать файл сценария SQL, выполните следующие действия.

1. Откройте файл сценария FormatColumns в программе Блокнот, щелкните слева от команды CLEAR COLUMNS и нажмите клавишу <Enter>, чтобы создать пустую строку. В этой строке между CLEAR SCREEN и CLEAR COLUMNS введите команду

```
SET PAGESIZE 45
```
2. Выберите пункт File (Файл) и Save (Сохранить), чтобы сохранить модифицированный файл SQL\*Plus.
3. Скопируйте все строки из программы Блокнот в окно SQL\*Plus.
4. В окне SQL\*Plus нажмите клавишу <Enter>, чтобы выполнить команду SELECT (рис. 2.19).
5. В SQL\*Plus введите следующую команду, чтобы восстановить исходное значение переменной среды PAGESIZE и очистить экран:

```
SET PAGESIZE 14
CLEAR SCREEN
```

6. Закройте окно Блокнота, но не закрывайте SQL\*Plus.

Существуют и другие команды форматирования SQL\*Plus, с которыми вы еще встретитесь в книге. Подробнее они будут рассмотрены по мере появления.

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", "Help". The command line shows the following SQL commands:

```

SQL> SET PAGESIZE 45
SQL> CLEAR COLUMNS
SQL> columns cleared
SQL> COLUMN AgentID FORMAT 99999 HEADING 'Agent ID'
SQL> COLUMN FirstName FORMAT A15 HEADING 'First Name'
SQL> COLUMN LastName FORMAT A15 HEADING 'Last Name'
SQL> SELECT AgentID, FirstName, LastName
  2  FROM Agents;

```

The output displays the results of the query:

Agent ID	First Name	Last Name
10041	Kai	Marcoux
10235	Tobias	Carling
10429	Elizabeth	Dahlen
10497	Ramanathan	Rowe
10849	Heather	Sheibani
10913	Bruce	Voss
11775	Cecilia	Romero
12211	Corneilis	Dann
12301	Danial	Silverburg
12499	Clair	Robinson
12715	Nancy	Piperova
12765	Edwin	Townsend
12875	Essi	Okinido
12963	Lee	Reed
13353	Stanislaw	Soltwedel
13555	Belinda	Chong
13649	Ricki	Selby
13771	Jessica	Taylor
14117	Tim	St-Onge
14447	Jackson	Flamenbaum
14599	Barbara	Herring
14601	Sindisiwe	Weber
14677	James	Kellogg
14883	Crystal	Fernandez
15061	Christine	Williams
15233	David	Gagnon
15293	Lora	Allee
15349	Tim	Schutz
15521	Patricia	Lewis

29 rows selected.

SQL>

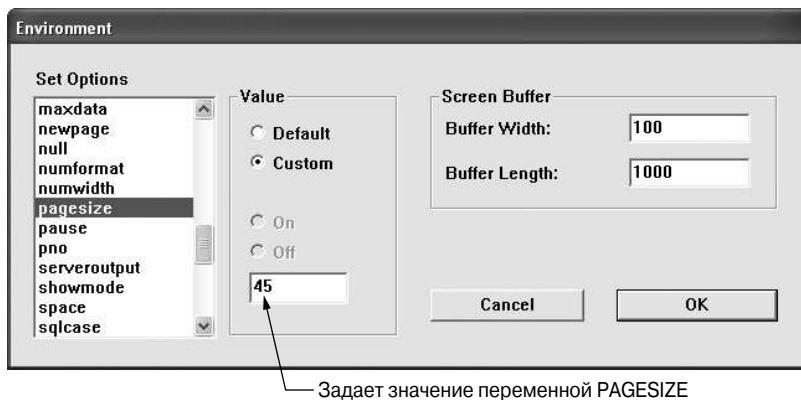
Рис. 2.19. Изменения значения переменной среды PAGESIZE

## Настройка среды SQL\*Plus

Некоторые переменные среды SQL\*Plus (например, PAGESIZE) можно установить на всегда, чтобы их не приходилось задавать при каждом входе в Oracle через SQL\*Plus. Например, вам удобно использовать значение PAGESIZE, равное 50, а значение LINESIZE (число символов в строке), равное 75. Для настройки среды SQL\*Plus применяются команды SQL\*Plus из строки меню.

Итак, чтобы модифицировать любую переменную среды SQL\*Plus, выполните следующие действия.

1. В окне SQL\*Plus (не iSQL\*Plus) выберите Options в строке меню и позицию Environment. SQL\*Plus откроет диалоговое окно Environment.
2. Прокрутите список Set Options вниз, пока не увидите надпись pagesize. Активизируйте ее (щелкните на ней) и установите переключатель Custom на панели Value. Обратите внимание на то, что в текстовом окне появилось значение 14, соответствующее текущему значению размера страницы (вы восстановили это значение, выполняя предыдущие шаги).



**Рис. 2.20.** Настройка среды SQL\*Plus

3. Дважды щелкните на значении 14, чтобы выбрать всю строку, и введите число 45 (рис. 2.20).
4. Прокрутите окно просмотра вниз и выберите в нем надпись `sqlprompt`. Щелкните на кнопке `Custom`. Обратите внимание на то, что в окне в группе `Value` появилось принятное по умолчанию приглашение командной строки `SQL>`. Пока что изменять его не будем.
5. Щелкните на кнопке `OK`, чтобы подтвердить внесенные изменения в размер строки. Диалоговое окно `Environment` закроется.
6. В командной строке SQL\*Plus введите `SHOW PAGESIZE` и нажмите клавишу `<Enter>`, чтобы отобразить на экране текущее значение переменной `PAGESIZE`.
7. Чему теперь равно значение переменной `LINESIZE`? Чтобы узнать это, введите `SHOW LINESIZE` и нажмите клавишу `<Enter>`.

Вообще, можно настроить значение любой переменной SQL\*Plus, имя которой отображается в списке `Set Options` диалогового окна `Environment`. Кстати, если вам нужен длинный список значений всех переменных среды, введите команду `SHOW ALL`.

## Выполнение файлов сценария

Скоро вы обнаружите, что определенные наборы команды SQL и SQL\*Plus приходится запускать довольно часто. Вместо того чтобы постоянно набирать десятки строк кода, можно записать команды SQL\*Plus в файле, чтобы их можно было запустить все сразу (почти так же, как вы поступали, когда копировали строки из программы Блокнот в SQL\*Plus, а затем запускали весь код). Набор выражений SQL и SQL\*Plus в файле называется *сценарием*. Записав сценарий, вы можете открыть весь файл, не открывая программу Блокнот и не занимаясь копированием. Все нужные действия представляются двумя командами SQL\*Plus, которые отвечают за загрузку и запуск

Agent ID	First Name	Last Name
10041	Kai	Marcoux
10235	Tobias	Carling
10429	Elizabeth	Dahlen
10497	Ramanathan	Rowe
10849	Heather	Sheibani
10913	Bruce	Voss
11775	Cecilia	Romero
12211	Cornelis	Dann
12301	Daniel	Silverburg
12499	Clair	Robinson
12715	Hancy	Piperova
12765	Edwin	Townsend
12875	Essi	Okindo
12963	Lee	Reed
13353	Stanislaw	Soltwedel
13555	Belinda	Chong
13649	Ricki	Selby
13771	Jessica	Taylor
14117	Tim	St-Onge
14447	Jackson	Flamenbaum
14599	Barbara	Herring
14601	Sindisiwe	Weber
14677	James	Kellogg
14883	Crystal	Fernandez
15061	Christine	Williams
15233	David	Gagnon
15298	Lora	Ailee
15349	Tim	Schutz
15521	Patricia	Lewis

29 rows selected.

SQL>

Команда START (в вашем случае не видна)

**Рис. 2.21.** Запуск файла сценария в SQL\*Plus

файлов сценария: START и @. Их выполнение дает идентичный результат: загрузку файла сценария и выполнение всех его строк.

Для того чтобы выполнить записанный на диске файл сценария, выполните следующие действия.

- Найдите папку, содержащую записанный ранее файл сценария Ch2<имя> FormatColumns.sql, и запомните путь к нему.
- В командной строке SQL\*Plus введите следующий код:

```
START <путь>\Ch2<имя>FormatColumns
```

Здесь <путь> — полный путь с указанием диска и папки, в которой записан файл сценария; <имя> — имя, подставленное вами ранее. Например:

```
START C:\My Documents\Ch2ElmFormatColumns
```

- Нажмите клавишу <Enter>, чтобы выполнить команду START. SQL\*Plus загрузит файл сценария и выполнит содержащиеся в нем команды. В результате на экране SQL\*Plus отобразится 3 столбца и 29 идентификаторов и имен сотрудников агентства недвижимости — все под одним заголовком (рис. 2.21).

*Примечание.* Мы удалили первую строку, CLEAR SCREEN, чтобы вы могли увидеть команду START, запускающую файл сценария. В вашем случае ее видно не будет.

Если хотите, вместо слова START можете подставить знак @ — результат от этого не изменится.

## Использование переменных

SQL\*Plus позволяет определять переменные, которые можно использовать вместо реальных значений в выражениях SQL. Такие переменные, именуемые *подстановочными*, применяются для подстановки значений в выражения SQL. Особенно они полезны в файлах сценариев SQL. Понять, что является переменной, довольно легко — это обычные текстовые строки, которые выглядят так, как названия столбцов, но начинаются с амперсанта (&). Таким образом, предположим, что SQL\*Plus выполняет следующее выражение SELECT:

```
SELECT AgentID, FirstName, LastName  
FROM Agents  
WHERE LastName = '&lastnameplease';
```

Здесь SQL\*Plus требует, чтобы пользователь предоставил значение “lastnameplease” (ввел фамилию) перед отправкой запроса в Oracle. Используя переменные, очень удобно создавать обобщенные файлы сценариев. Рассмотрим предыдущий пример: кто угодно может затребовать информацию об агенте, указав “в процессе” его фамилию. Рассмотрим на реальном примере, как это действует.

Для того чтобы использовать подстановочную переменную в файле сценария, выполните следующие действия.

1. Переключитесь на SQL\*Plus и введите CLEAR SCREEN (нажмите клавишу <Enter>), чтобы очистить рабочую поверхность.
2. Откройте программу Блокнот, наберите следующий сценарий SQL, а затем запишите его как текстовый файл с именем Ch2<имя>SubVar.sql:

```
SELECT AgentID, FirstName, LastName, BirthDate  
FROM Agents  
WHERE LastName = '&lastnameplease';
```

3. Вернитесь в SQL\*Plus и наберите следующую строку.

```
@<путь>\Ch2<имя>SubVar
```

Здесь <путь> — полный путь с указанием диска и папки, в которой записан файл сценария; <имя> — имя, подставленное вами ранее.

4. Нажмите клавишу <Enter>, чтобы запустить файл сценария. После этого SQL\*Plus запросит фамилию.
5. Наберите Carling (точно так, как написано) и нажмите клавишу <Enter>. Oracle отобразит выбранные столбцы таблицы Agent для агента Carling.
6. Повторите шаги 3 и 4.
7. После запроса “<Enter> value for lastnameplease” введите Fernandez

The screenshot shows the Oracle SQL\*Plus interface. In the top menu bar, 'File', 'Edit', 'Search', 'Options', and 'Help' are visible. The main window displays a script file named 'Ch2SubVar'. The script contains several lines of SQL code. The first two lines are:

```
SQL>@C:\Docs\Ch2SubVar
Enter value for lastnameplease: Carling
```

These lines are part of a substitution variable assignment. The 'old' value is '3: WHERE LastName = '%lastnameplease%''. The 'new' value is '3: WHERE LastName = 'Carling''. Below this, the script lists agent information in a table format:

Agent ID	First Name	Last Name	BIRTHDATE
10235	Tobias	Carling	19-OCT-75

Following this, another substitution variable assignment is shown:

```
SQL>@C:\Docs\Ch2SubVar
Enter value for lastnameplease: Fernandez
```

The 'old' value is '3: WHERE LastName = '%lastnameplease%''. The 'new' value is '3: WHERE LastName = 'Fernandez''. Below this, the script lists agent information in a table format:

Agent ID	First Name	Last Name	BIRTHDATE
14883	Crystal	Fernandez	29-NOV-64

The prompt 'SQL>' is visible at the bottom left.

Рис. 2.22. Использование подстановочной переменной в файле сценария

и нажмите клавишу <Enter>. SQL\*Plus подставит набранную вами строку вместо переменной и передаст весь запрос Oracle. В ответ Oracle вернет другую строку (рис. 2.22).

8. Выходите из SQL\*Plus: введите `Exit` и нажмите клавишу <Enter>.
9. Закройте программу Блокнот.

## Обзор других команд SQL\*Plus

В настоящей книге вам встретятся еще несколько команд SQL\*Plus, по которым иногда требуется интерактивная помощь (справка). Помните, что описание и синтаксис команд SQL\*Plus можно получить в SQL\*Plus, если ввести `HELP`, а затем имя команды. Вы можете вывести на экран список всех команд SQL\*Plus, набрав `HELP INDEX`. Ниже мы кратко опишем эти команды, не вдаваясь в особые подробности (помимо того, что данные команды полезны, их очень легко понять и использовать).

Команда `HOST`, недоступная в *i*SQL\*Plus, выполняет команды операционной системы, не покидая SQL\*Plus. Вместо `HOST` в командной строке `SQL>` вы также можете использовать `$`. Обе команды вызывают операционную систему. Если речь идет о Windows, то откроется окно DOS и появится запрос командной строки операционной системы с указанием текущего пути. Чтобы вернуться в SQL\*Plus, введите `Exit` и нажмите клавишу <Enter>. Например, если вам требуется перечислить файлы SQL в текущей папке, вы можете ввести `HOST`, а затем в командной строке DOS ввести `DIR *.SQL`. Более того, если вы хотите автоматизировать команду DOS, используйте конструкцию “`HOST+команда`”. Рассмотрим, например, выполнение следующей команды в SQL\*Plus:

```
HOST DEL C:\Docs\myfile.sql
```

SQL\*Plus вызывает операционную систему компьютера и указывает ей удалить (`DEL`) файл `myfile.sql` из папки `C:\Docs\`, после чего закрыть окно DOS, практи-

чески сразу вернувшись в SQL\*Plus. Ниже мы используем эту возможность в файле сценария. Разумеется, в командной строке операционной системы вы можете выполнить различные команды, включая изменение текущей папки.

Переменные среды TERMOUT и PAGESIZE также весьма полезны в файле сценария, когда требуется выполнить ряд команд SQL и SQL\*Plus. TERMOUT применяется, когда на экране не нужны информационные сообщения, подобные “1 row created”. Если ввести команду SET TERMOUT OFF (или SET TERM OFF), это подавит информацию, выводимую на экран из данного файла сценария (и только из него). Если ввести команду SET TERM ON, окно вернется в обычный режим отображения. Как только выполнение сценария, содержащего команду TERMOUT, завершается, TERMOUT включается автоматически. Другой переменной среды SQL\*Plus является FEEDBACK, определяющая, будут ли отображаться на экране *информационные сообщения* — “1 row created” или “table dropped”. После ввода команды SET FEEDBACK OFF информативные сообщения подавляются, после ввода SET FEEDBACK ON — снова активизируются. Если отображение сообщений не включить явно, состояние FEEDBACK остается равным OFF до конца сеанса SQL\*Plus, независимо от того, где оно было отключено (в файле сценария или в командной строке). Ввод SET PAGESIZE 0 подавляет отображение заголовков столбцов, что бывает полезно, когда результаты запроса выводятся в файл без заглавий столбцов.

В файлы сценариев вы можете включать комментарии. Если комментарий занимает одну строку, он начинается с двух дефисов (--). Комментарии в несколько строк начинаются с косой черты и звездочки /\*), а заканчиваются звездочкой и косой чертой (\*/). Кроме того, в начале однострочного комментария можно использовать команду REM. По возможности, старайтесь избегать варианта /\* текст \*/, поскольку вы можете забыть закрыть комментарий, и тогда остаток сценария будет считаться комментарием.

Команда PROMPT выводит на экран сообщение или пустую строку. Особенно она полезна в файле сценария, когда требуется отобразить сообщение о ходе работы, уведомляющее о состоянии файла сценария. Например, после выполнения приведенной ниже команды на экране появится текст, который следует после команды PROMT.

```
PROMPT Successfully ran script to delete files.
```

## Создание и запуск сценария

Как говорилось ранее, файл сценария — это набор команд SQL и SQL\*Plus, применяющихся для выполнения определенной задачи. Напомним, что для запуска сценария необходимо ввести команду START, а после нее — путь и имя файла сценария. После этого SQL\*Plus загрузит указанный файл и последовательно выполнит все его строки. Для логического завершения данной главы полезно собрать некоторые изученные в ней команды в файл сценария. Кроме того, вы сможете вызывать несколько системных

таблиц, поддерживаемых базой данных, которая содержит имена всех объектов, принадлежащих схеме (пользователю). Вообще, Oracle поддерживает список объектов, принадлежащих текущему пользователю в представлении `user_tables`. Указанная таблица является “таблицей таблиц”, поскольку содержит более 45 столбцов, описывающих характеристики принадлежащих вам таблиц, включая число строк в каждой таблице, процент свободного места и имена таблиц. (Чтобы отобразить на экране список столбцов данной системной таблицы, введите `DESCRIBE user_tables`.) Чуть позже мы напишем сценарий, отображающий имена всех принадлежащих вам таблиц, число строк в этих таблицах и описание столбцов в каждой таблице, — все это из одного файла сценария, который вы сможете периодически запускать по мере изменения базы данных.

Далее вы напишете сценарий, описывающий имена принадлежащих вам (вашей схеме) файлов. Файл сценария будет обращаться к системной таблице, чтобы перечислить имена таблиц вашей учетной записи. На основе данного списка имен вы динамически создадите другой файл сценария, содержащий набор команд `DESCRIBE` (по одной на каждую таблицу, найденную в вашей учетной записи). Наконец, ваш файл сценария выполнит динамически созданный файл сценария, а затем удалит его после выполнения. В конце вы получите отображенный в окне SQL\*Plus список имен таблиц и список столбцов, содержащихся в каждой таблице, с указанием типа данных, соотнесенных с каждым столбцом. Не волнуйтесь, если не все строки кода будут вам понятны. В данном случае наша цель — показать, как создавать файлы сценариев, которые затем передаются SQL\*Plus для выполнения.

Итак, для того чтобы создать файл сценария и сохранить его, выполните следующие действия.

1. Прежде всего определите, куда вы будете записывать файл сценария. Например, вы можете выделить для этого папку `C:\Temp` или использовать корневую папку `C:\`.
2. Откройте программу Блокнот и введите показанный ниже код. Не забудьте подставить вместо `C:\Docs\` точный путь к папке, получающей файл сценария (это нужно сделать в трех местах; все они указаны на рис. 2.23):

```

REM Describe my tables
CLEAR SCREEN
--Создать и выполнить файл
SET FEEDBACK OFF
SET PAGESIZE 0
SPOOL C:\Docs\Describe.sql
SELECT 'DESCRIBE' || table_name || ';' 
FROM user_tables
WHERE table_name NOT LIKE 'BIN%'
ORDER BY 1;
SPOOL OFF
PROMPT
@@C:\Docs\Describe.sql

```

```

REM Describe my tables
CLEAR SCREEN
--Create a file that is, in turn, executed
SET FEEDBACK OFF
SET PAGESIZE 0
SPOOL C:\Docs\Describe.sql
SELECT 'DESCRIBE ' || table_name || ';' 
FROM user_tables
WHERE table_name NOT LIKE 'BIN%'
ORDER BY 1;
SPOOL OFF
PROMPT
@C:\Docs\Describe.sql
HOST del C:\Docs\Describe.sql
SET FEEDBACK ON

```

**Рис. 2.23.** Файл сценария

```

HOST del C:\Docs\Describe.sql
SET PAGESIZE 14
SET FEEDBACK ON

```

3. Выберите пункт **File** (Файл), а затем **Save** (Сохранить). Щелкните в текстовом окне **Save as type** (Тип файла), выберите **All Files** (Все файлы), перейдите в папку, выбранную для сохранения файла сценария. Введите **Ch2<имя>GenScript.sql** (вместо <имя> введите свое имя) в окне **File name** (Имя файла) и щелкните на кнопке **Save** (Сохранить). Не закрывайте Блокнот на случай, если SQL\*Plus выдаст ошибку и сценарий придется править.
4. Откройте SQL\*Plus и войдите в систему, используя свое имя пользователя и пароль. В командной строке SQL введите следующий текст:
 

```
START <путь>Ch2<имя>GenScript
```

 Вместо <путь> подставьте путь к файлу сценария, а вместо <имя> — свое имя.
5. Нажмите клавишу <Enter>, чтобы запустить файл сценария. На экране вы должны увидеть то же, что показано на рис. 2.24.
6. Если появятся сообщения об ошибках, переключитесь на программу Блокнот, найдите опечатки, еще раз сохраните файл и повторите п. 4.
7. Закройте Блокнот и выйдите из SQL\*Plus.

## Резюме

Существует пять категорий выражений языка структурированных запросов (Structured Query Language — SQL): запросы, определения данных (DDL), команды манипулирования данными (DML), выражения управления транзакциями (TC) и команды языка управления данными (DCL). Запросы SQL извлекают информацию из базы данных. Выражения DML **INSERT**, **UPDATE** и **DELETE** модифицируют данные. Ко-

```

+ Oracle SQL*Plus
File Edit Search Options Help
DESCRIBE AGENTS;
DESCRIBE LICENSESTATUS;
DESCRIBE SALESTATUS;

Name          Null?    Type
-----        -----    -----
AGENTID      NOT NULL NUMBER(38)
FIRSTNAME    NVARCHAR2(30)
LASTNAME     NVARCHAR2(30)
HIREDATE     DATE
BIRTHDATE    DATE
GENDER       NVARCHAR2(10)
WORKPHONE   NVARCHAR2(20)
CELLPHONE   NVARCHAR2(20)
HOMEPHONE   NVARCHAR2(20)
TITLE        NVARCHAR2(20)
TAXID        NVARCHAR2(20)
LICENSEID    NVARCHAR2(20)
LICENSEDATE  DATE
LICENSEEXPIRE DATE
LICENSESTATUSID NUMBER(38)

Name          Null?    Type
-----        -----    -----
LICENSESTATUSID NOT NULL NUMBER(38)
STATUSTEXT    NVARCHAR2(25)

Name          Null?    Type
-----        -----    -----
SALESTATUSID NOT NULL NUMBER(38)
SALESTATUS    NVARCHAR2(10)

SQL>

```

Рис. 2.24. Результат выполнения файла сценария

манды языка определения данных определяют в базе данных структуры (например, таблицы). Выражения управления транзакциями отменяют или фиксируют любые изменения, внесенные последними незафиксированными командами DML. Команды языка управления данными обеспечивают безопасность, разрешая (или запрещая) пользователям обращаться к выбранными объектам базы данных. К базе данных “клиент/сервер” Oracle можно обращаться из операционной системы компьютера, используя клиент Windows SQL\*Plus, и из Web-интерфейса iSQL\*Plus. Оба метода имеют свои достоинства, однако самым распространенным вариантом является доступ с использованием клиентского приложения SQL\*Plus.

Расширения стандартного языка SQL компании Oracle называются выражениями SQL\*Plus. Данные команды позволяют форматировать выход, захватывать извлекаемые результаты, запускать файлы сценариев, отображать структуру таблицы и настраивать среду интерфейса клиента SQL\*Plus. Чтобы отобразить список данных полезных расширений SQL, введите `HELP INDEX` в командной строке SQL. Чтобы получить дополнительную справку по заинтересовавшей вас команде SQL\*Plus, введите `HELP` и имя команды. Последовательности команд SQL и SQL\*Plus образуют сценарий SQL. Записывая блоки команд в файл, вы можете впоследствии вызывать эти блоки с помощью единственной команды. Файлы сценариев могут содержать переменные, значения которых задает пользователь во время выполнения сценария. Переменные позволяют отложить определение имен объектов и действий до времени выполнения и создать динамический и настраиваемый сценарий, удовлетворяющий требованиям множества пользователей базы данных.

## Основные термины

- Базовая таблица
- Фиксировать
- Условие
- Производная таблица
- Модель формата
- Набор символов национального языка
- NULL
- Запрос
- Откат
- Сценарий
- Команды SQL\*Plus
- Подстановочная переменная
- Привилегии системы
- Транзакция
- Наблюдаемая таблица

## Повторение пройденного материала

### Истина или ложь?

1. Язык структурированных запросов (Structured Query Language – SQL) – это язык программирования.
2. Выражения SQL языка манипулирования данными (Data Manipulation Language – DML) позволяют создавать такие объекты, как таблицы и представления.
3. Взаимодействовать с Oracle непосредственно из операционной системы компьютера гораздо легче, чем использовать SQL\*Plus.
4. Для удаления записей из таблицы базы данных применяется команда DELETE FROM.
5. Запись, удаленную из таблицы, можно восстановить, если до выхода из системы или фиксации удаления использовать команду ROLLBACK.

### Заполнить пропущенное

1. Выполните выражение SQL \_\_\_\_\_, чтобы добавить записи в таблицу.
2. Выражение DESCRIBE относится к командам \_\_\_\_\_, а не к командам SQL.
3. Чтобы удалить таблицу из схемы, используйте команду \_\_\_\_\_.

4. Чтобы сделать постоянными изменения базы данных, вызванные командами INSERT, DELETE и UPDATE, введите команду SQL \_\_\_\_.
5. Команда SQL\*Plus \_\_\_\_, введенная с правильным модификатором, позволяет контролировать формат возвращаемых данных в SQL\*Plus.

## Варианты ответов

1. Какая из приведенных ниже команд позволяет захватывать в файл все, что появляется на экране SQL\*Plus?
  - a) PRINT ON.
  - б) SPOOL <filename>.
  - в) SPOOL OFF.
  - г) CAPTURE <filename>.
2. Какая команда позволяет удалить все форматирование столбцов, установленное в течение сеанса SQL\*Plus?
  - a) FORMAT COLUMN OFF.
  - б) COLUMN FORMAT CLEAR.
  - в) CLEAR COLUMNS.
  - г) Ни одна из перечисленных.
3. Какая из приведенных команд позволяет добавить строку в таблицу LicenseStatus (предполагается, что для всех столбцов тип данных предлагаемых значений правильный)?
  - a) INSERT INTO LicenseStatus VALUES (876,'Out of commission');
  - б) INSERT LicenseStatus VALUES (876,'Out of commission');
  - в) INSERT FROM LicenseStatus VALUES (876,'Out of commission');
  - г) CREATE FROM LicenseStatus VALUES (876,'Out of commission');
4. Какая из приведенных команд удаляет запись Danial Silverburg из таблицы Agents Redwood Realty?
  - a) DELETE FROM Agents LastName = 'Silverburg' AND FirstName ='Danial'
  - б) DROPFROMAgentsWHERELastName = 'Silverburg' AND FirstName ='Danial'
  - в) DELETE FROM Agents WHERE LastName ='Silverburg' AND FirstName ='Danial';
  - г) DELETE Agents WHERE LastName ='Silverburg' AND FirstName ='Danial'
  - д) Правильные варианты в) и г).
5. С помощью какой команды можно подавить дальнейшее отображение информации на экране информативных сообщений, подобных “3 rows deleted”?
  - a) SET TERMOUT ON.

**ТАБЛИЦА 2.1.** Данные, которые требуется ввести в таблицу ContactReason

ContactReason	Описание
Buy	Offer to buy a property (“Желает купить недвижимость”)
Casual	General customer probably looking for properties (“Обычный клиент, возможно, интересующийся недвижимостью”)
Sell	Listing to sell a property (“Желает продать недвижимость”)

- б) SET PAGESIZE 0.
- в) SET SPOOL OFF.
- г) SET INFORM OFF.
- д) SET FEEDBACK OFF.

## Упражнения

### Redwood Realty

Администратор базы данных Redwood Realty требует создать в базе таблицу ContactReason. В этой таблице должны быть представлены три причины контакта людей с агентством: чтобы купить дом; чтобы продать дом или чтобы связаться с брокером для каких-то других целей. Три данные категории должны храниться в таблице вместе с более длинными описаниями. Три строки, которые вы должны вставить в таблицу ContactReason, приведены в табл. 2.1.

1. Войдите в Oracle, используя SQL\*Plus. Откройте программу Блокнот и сразу сохраните файл как Ch2<имя>RedwoodEOC.sql. Не забудьте щелкнуть на Save as type (Тип файла) и выбрать пункт All Files (Все файлы).
2. В программе Блокнот введите следующий текст, нажимая клавишу <Enter> в конце первых двух строк (После завершающей точки с запятой клавишу <Enter> не нажимайте):
 

```
CREATE TABLE ContactReason
(ContactReason NVARCHAR2(15) PRIMARY KEY,
 Description NVARCHAR2(50));
```
3. Скопируйте все четыре строки из программы Блокнот в SQL\*Plus. Нажмите клавишу <Enter>, чтобы выполнить команду CREATE TABLE. База данных Oracle должна вывести на экран сообщение “Table created”.
4. Если на предыдущем этапе появились сообщения об ошибках, найдите их и исправьте в программе Блокнот, после чего повторите п. 3.
5. Переключитесь на Блокнот, нажмите клавишу <Enter>, чтобы перейти на следующую строку, и введите следующие команды INSERT (после последней строки также нажмите <Enter>):

```

INSERT INTO ContactReason VALUES ('Buy',
'Offer to buy a property');
INSERT INTO ContactReason VALUES ('Casual',
'General customer probably looking for
properties');
INSERT INTO ContactReason VALUES ('Sell',
'Listing to sell a property');

```

6. Выделите в Блокноте три команды `Insert`, включая третью точку с запятой и пустую строку после нее. Вставьте данный текст в SQL\*Plus. Три команды будут выполнены немедленно.
7. Переключитесь на программу Блокнот, запишите файл, выйдите из программы Блокнот и вернитесь в SQL\*Plus.
8. Введите `COMMIT;` (не пропустите в конце точку с запятой) и нажмите клавишу `<Enter>`, чтобы сделать постоянной вставку трех указанных строк.
9. В SQL\*Plus введите и выполните следующую команду `INSERT` (между буквами `n` и `t` в слове `Don't` вводятся две одинарные кавычки):  

```
INSERT INTO ContactReason VALUES
('Don''t know', 'Unknown reason');
```
10. В SQL\*Plus введите следующую команду и нажмите клавишу `<Enter>`, чтобы выполнить ее:  

```
SELECT *
FROM ContactReason;
```
11. Введите `ROLLBACK;` (не забудьте точку с запятой) и нажмите клавишу `<Enter>`, чтобы отменить последнее действие.
12. Чтобы создать текстовый файл, в который можно записывать информацию и который можно сохранить на диске, выполните в SQL\*Plus следующие команды (вместо `<путь>` подставьте полный путь (например, `C:\Main\`)):  

```
SPOOL <путь>\Ch2ContactReasonList.txt
DESCRIBE ContactReason
SELECT * FROM ContactReason;
SPOOL OFF
```
13. Откройте созданный выше файл `Ch2ContactReasonList.txt` в программе Блокнот, вставьте сверху новую строку, введите свое имя, запишите файл, напечатайте его, а затем закройте программу Блокнот.
14. Переключитесь на SQL\*Plus и удалите буферный файл с помощью следующей команды (вместо `<путь>` введите реальный путь к файлу):  

```
HOST del <путь>\Ch2ContactReasonList.txt
```
15. Чтобы удалить таблицу и ее содержимое, введите следующую команду.  

```
DROP TABLE ContactReason;
```
16. Выйдите из SQL\*Plus.

## Coffee Merchant

Администратор базы данных Coffee Merchant требует, чтобы вы создали несколько таблиц и немного изменили представленные в базе данные. Предполагается, что в данном примере вы будете использовать SQL\*Plus и iSQL\*Plus. Если по какой-либо причине вы не можете задействовать iSQL\*Plus (интерфейс базы данных на основе браузера), просто используйте SQL\*Plus. Итак, администратор базы данных желает, чтобы вы создали и заполнили таблицу States, которая после выполнения работы будет содержать данные о 50 штатах США и округе Колумбия. Страна этой таблицы включает двухбуквенное сокращение названия штата, полное название штата, ставку налога, примерную численность населения, размер штата в квадратных милях и URL официального Web-сайта штата (если он есть).

1. Запустите SQL\*Plus и войдите в Oracle. Откройте программу Блокнот и сразу сохраните файл под именем Ch2<имя>CoffeeEOC.sql. Не забудьте щелкнуть на кнопке Save as type (Тип файла), а затем выбрать AllFiles (Все файлы).
2. Введите в программе Блокнот следующий текст, нажимая клавишу <Enter> в конце каждой строки (не забудьте набрать заключительную закрывающую скобку и точку с запятой):

```
CREATE TABLE States
  (StateID NCHAR(2) PRIMARY KEY,
  StateName NVARCHAR2(20) NOT NULL,
  TaxRate NUMBER(7, 4),
  Population INTEGER,
  LandArea INTEGER,
  WebURL NVARCHAR2(50)
);
```

3. Запишите файл в Блокноте, скопируйте в буфер весь текст до завершающей точки с запятой, но без следующей за ним пустой строки. Закройте программу Блокнот.
4. В SQL\*Plus нажмите комбинацию клавиш <Ctrl+V>, чтобы вставить текст, и клавишу <Enter> — чтобы выполнить команды сценария. Oracle ответит сообщением “Table created”.
5. В SQL\*Plus введите и выполните следующие выражения SQL:

```
INSERT INTO STATES
VALUES ('AK', 'Alaska', 0, 643786,
      571951, 'www.state.ak.us');
CLEAR SCREEN
COLUMN StateID FORMAT A7
COLUMN StateName FORMAT A15
COLUMN TaxRate FORMAT 0.9999
COLUMN Population FORMAT 99,999,999
COLUMN LandArea FORMAT 99,999,999
COLUMN WebURL FORMAT A15
SELECT * FROM STATES;
COMMIT;
```

```
DELETE FROM STATES;
SELECT * FROM STATES;
ROLLBACK;
SELECT * FROM STATES;
```

6. Найдите на компакт-диске файл Ch02PopulateStates.sql и запишите полный путь к нему. После этого введите в SQL\*Plus указанный ниже код, нажав клавишу <Enter>, чтобы запустить файл сценария. (Вместо <путь> введите путь к файлу сценария.)

```
START <путь>\Ch02PopulateStates.sql
```

Последним сообщением, которая отобразится после ряда строк 1 row created, должно быть “Commit complete”.

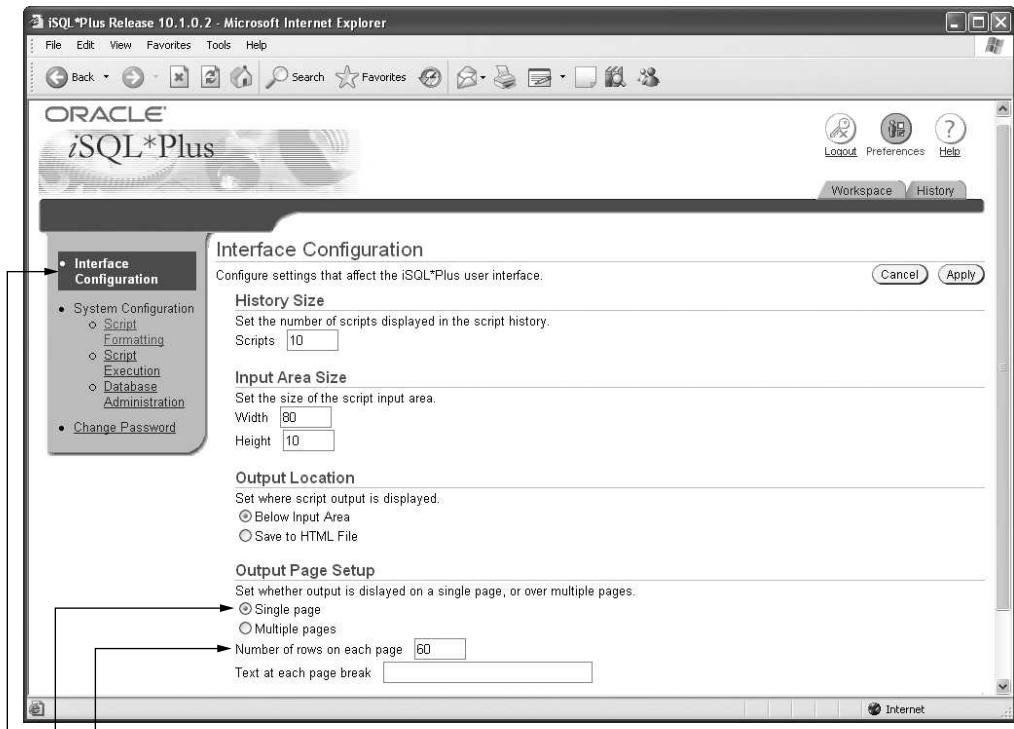
7. Введите Exit, чтобы выйти из Oracle и закрыть SQL\*Plus.
8. Откройте Web-браузер, откройте iSQL\*Plus и войдите в Oracle, используя свое имя пользователя и пароль.
9. Щелкните на ссылке Preferences возле правого верхнего края экрана, а затем (если потребуется) на ссылке Interface Configuration слева. Далее щелкните на кнопке Single page под заголовком OutputPageSetup, дважды щелкните в текстовом окне Number of rows on each page и введите 60 (рис. 2.25). Щелкните на кнопке Apply, а затем щелкните на закладке Workspace в правом верхней углу страницы.
10. Щелкните в текстовом окне Workspace и введите следующий текст, нажимая клавишу <Enter> в конце каждой строки:

```
COLUMN Density FORMAT 9,999
SELECT StateID, StateName,
Population/LandArea AS Density
FROM States
ORDER BY Density DESC;
```

11. Щелкните на кнопке Execute, чтобы просмотреть результаты: появится список штатов и округ Колумбия. Наиболее плотно населенные штаты будут идти вверху списка, а менее плотно населенные — внизу.
12. Выберите File в строке меню вашего браузера. Выберите Print из меню и щелкните на кнопке Print диалогового окна Print, чтобы напечатать запрос iSQL\*Plus и результат его выполнения.
13. Щелкните на гиперссылке Logout, чтобы выйти из базы данных. Когда появится Web-страница подтверждения выхода, закройте браузер.

## Rowing Ventures

Компания Rowing Ventures отвечает за координацию гребных регат в более чем десятке мест всего мира. Помимо надзора за трассой и предоставления места для торговли



Число строк на каждой странице задано равным 60

Кнопка *Single page*

Гиперссылка *Interface Configuration*

**Рис. 2.25.** Изменение переменной среды iSQL\*Plus

и рекламы на гонках, Rowing Ventures хранит массу информации о заездах регаты, участниках, их спонсорах, лодках, выставленных на каждый заезд, номерах и экипажах этих лодок.

Запустите SQL\*Plus и войдите в Oracle. После этого на компакт-диске, прилагаемом к этой книге, найдите и запустите файл сценария Ch02RowingVentures.sql. Данный файл создает таблицы Boat и Organization. Откройте файл, предназначенный для буферизации данных, чтобы записать следующие этапы на диск, и выполните команды DESCRIBE Organization и DESCRIBE Boat, чтобы отобразить на экране названия столбцов указанных таблиц. Если хотите, можете распечатать данный файл, но после этого не закрывайте его. Далее с помощью команды SELECT для всех записей выведите на экран следующие поля таблицы Organization: OrganizationID, Phone и поле адреса. Закройте буферный файл и распечатайте его. Очистите экран, затем выведите на него столбцы BoatID и BoatCategory для всех записей таблицы Boat. Удалите первые две записи, задав их значения BoatID в команде DELETE.

Создайте еще один буферный файл (например, SPOOL C:\Docs\Myfile.txt) и выведите на экран столбцы BoatID и BoatCategory для всех записей в таблице Boat. Выполните команду, отменяющую последнее удаление записи. Распечатайте буферный файл. Закройте SQL\*Plus.

## Broadcloth Clothing

Руководство компании-производителя женской одежды Broadcloth Clothing требует, чтобы вы перечислили содержимое нескольких таблиц корпоративной базы данных, направив эту информацию в файл, который можно напечатать. Итак, прежде всего найдите на компакт-диске файл сценария Ch02Broadcloth.sql и запишите путь к нему. Далее запустите SQL\*Plus, войдите в Oracle, используя свое имя пользователя и пароль. Запустите буферизацию данных в файл, выполнив следующую команду:

```
SPOOL
<путь>\<имя>Ch02BroadclothSpool.txt
```

Здесь вместо <путь> необходимо ввести путь к файлу, а вместо <имя> — ваше имя (оно должно быть уникальным для выбранной папки). После этого, используя записанный ранее путь, запустите файл сценария Ch02Broadcloth.sql, создающий и заполняющий две таблицы, с которыми вы будете работать в этом упражнении (файл сценария создает таблицы Factory и WorkingConditions). Используя команду DESCRIBE, перечислите для обеих таблиц имена полей и их типы данных. После этого перечислите значения в столбцах ComplianceID и ConditionCategory для всех строк в таблице WorkingConditions. Удалите из таблицы WorkingConditions все строки, значение ComplianceID которых равно 2003. Снова отобразите те же два столбца для всех строк, используя команду SELECT. Отмените удаление и остановите буферизацию в файл. Отредактируйте этот файл в программе Блокнот и добавьте свое имя в первой строке (вставьте в начале файла новую строку). Запишите буферный файл и напечатайте его. Закройте программу Блокнот и введите Exit в SQL\*Plus, чтобы выйти из SQL\*Plus и Oracle.

Запустите iSQL\*Plus, войдите в базу данных, отобразите структуры таблицы Factory (имена и типы данных), после чего напечатайте Web-страницу с этими результатами. Щелкните на кнопке Clear, чтобы очистить окно Workspace и удалить результаты. Затем введите команду SELECT, чтобы отобразить все столбцы таблицы Factory. Напечатайте запрос и результаты, показанные на Web-странице. Щелкните на кнопке Clear, чтобы удалить запрос и результаты. После этого введите и выполните команду DELETE, чтобы удалить строки таблицы Factory, в которых разница времени по Гринвичу (GMTDifference) равна +8 часов. Снова отобразите все строки таблицы Factory и напечатайте эту Web-страницу. Щелкните на кнопке Logout, чтобы выйти из Oracle, и закройте браузер.

# ГЛАВА 3

## Создание, модификация, переименование и удаление таблиц базы данных

**В этой главе...**

- Создание нормализованных таблиц
- Создание учетной записи пользователя и присвоение ей пароля
- Предоставление учетной записи пользователя привилегий объектов и системы
- Команды Oracle создания таблиц
- Типы данных, доступные в Oracle
- Добавление к таблицам условий целостности и условий домена
- Добавление и удаление столбцов из таблиц
- Хранение описаний таблиц, имен столбцов и имен таблиц в системных таблицах
- Переименование или удаление таблиц

### Разработка реляционных баз данных

Прочитав главу 2, вы должны понимать, как создавать таблицы и какой код для этого нужен. В данной главе мы подробно опишем процесс создания таблиц, поддерживающие Oracle типы данных, расскажем, как обеспечить согласованное представление информации в таблицах и как ограничить значения, которые могут иметь столбцы таблицы. Прежде всего мы исследуем глобальный вопрос разработки базы данных, изучив, как определять, какие таблицы нужны приложению и какие столбцы должны содержаться в различных таблицах.

## Общение с пользователем базы данных

Пожалуй, самой важной задачей, связанной с базами данных, является разработка таблиц, хранящих информацию некоторой организации, и поддержка ее делового регламента. Поскольку Oracle предлагает несколько возможностей для доступа к информации базы данных, вы можете разработать таблицы базы данных, хранящие связанную информацию в отдельных таблицах, но при этом отображающие разнобразные данные в одном отчете. Работая над любым проектом базы данных, вы должны понимать, какие элементы данных должны храниться в базе. Успешно разработанная база данных должна отражать деловой регламент, правила которого собраны из различных источников. Один из первых этапов разработки базы данных — точно определить, какие данные требуются. Для этого вам необходимо общаться с пользователями, которые регулярно работают с этими данными и могут их “чувствовать”. Впрочем, люди, как правило, очень слабо представляют, что же они хотят от системы базы данных. В ходе сбора предварительных данных вы должны быть настойчивыми. В частности, от вас требуется точно определить, какие данные требуется собирать; выяснить, как связаны данные между собой; и установить, как долго будут храниться данные в системе базы данных.

Определив элементы данных, вам нужно организовать их в группы или объекты. Как и ранее, частью этого процесса является определение делового регламента, управляющего группами. Рассмотрим, например, правила, окружающие сущность “клиент”. Может ли запись, связанная с одним клиентом, одновременно отражать больше одного заказа, включающего несколько позиций? С другой стороны, могут ли данные пользователя храниться в базе данных, если текущих невыполненных заказов для этого клиента нет, а последний заказ был помещен более года назад? Ответы на подобные вопросы влияют на разработку базы данных. Следовательно, в правильно разработанной базе должны отражаться все подобные правила. Определение делового регламента и внедрение соответствующих правил в структуру базы данных всегда сопряжено с проблемами, и обычно проект базы данных приходится несколько раз переделывать по мере формулирования и решения этих проблем. Некоторые проблемы имеют общие черты, и ниже мы разберем их подробнее.

## Определение требований пользователя

Определить требования пользователей можно множеством различных способов, один из которых — понаблюдать за работой компании. Определите потребности различных групп в пределах организации, соберите необходимые данные и классифицируйте их. Чтобы получить первое представление об информационных потребностях фирмы, можно поработать с документацией (электронной или бумажной). Перед формированием исходной структуры базы данных вам потребуется такая информация: данные, которые необходимо собрать; тип данных (числовые, символьные и т.д.) и их объем.

## Определение бизнес-объектов

Определив все данные, которые необходимо хранить в базе данных и которые формируют основные блоки памяти — таблицы базы данных, — вы уже сможете мысленно представить требуемые отчеты и формы. Фундамент успешной базы данных — это сбор основных данных, организация их в таблицы и определение связей между ними. Компании работают с бизнес-объектами (или сущностями), не называя их так. С точки зрения разработчика базы данных, *сущность* (или *объект*) — это человек, место или предмет, информацию о котором требуется записать и отследить. В качестве привычных примеров сущностей можно привести клиентов, сотрудников и пассажиров самолета. Каждая сущность описывается своими *атрибутами*, или *свойствами*. Сущность вместе со своими свойствами именуется *классом*. Каждая строка сущности заказчика имеет атрибуты (или характеристики), описывающие ее элементы. Атрибутом может быть имя, адрес, номер телефона и т.д. Как разработчик базы данных вы должны определить эти сущности и описывающие их атрибуты. По мере сбора информации и форм от пользователей вы можете группировать связанные данные в соответствующие сущности. В форме анкеты для приема на работу, например, большая часть контактной информации о клиенте, скорее всего, будет частью сущности *applicant*. Некоторая информация из анкеты в эту сущность не попадет — навыки, предыдущие работодатели и их адреса. Скорее всего, эти данные будут храниться в отдельных сущностях, которые, однако, будут связаны с указанной.

Реляционные базы данных хранят сущности в таблицах. Таблицы содержат столбцы, соответствующие атрибутам, определенным для сущности (один столбец — один атрибут). Каждая строка таблицы представляет *экземпляр*, или *пример*, данной сущности и содержит данные, уникальным образом описывающие одного представителя этого класса. По мере того как определяются классы данных или таблицы, выявляются и связи (или отношения) между этими таблицами. Таблицы всегда связаны с другими таблицами базы данных. В базе данных Redwood Realty, например, таблица *Customers* (“клиенты”) связана с таблицей *Properties* (“недвижимость”), в которой содержатся сведения о домах, принадлежащих данным клиентам. Таблица, содержащая информацию об агентах Redwood Realty, также связана с таблицей, хранящей информацию о лицензиях агентства. Кроме того, таблица *Agents* косвенно связана с таблицей *Customers*, поскольку каждый клиент таблицы *Customers* курируется определенным агентом по недвижимости, указанным в таблице *Agents*. Следовательно, реляционная база данных — это набор связанных и тщательно спроектированных таблиц. *Таблица* — это набор столбцов (или атрибутов), описывающих сущность.

Ассоциации, или связи, сущностей необходимо определить, поскольку они представляют деловой регламент компаний и должны быть представлены правильно. Связь двух таблиц может принадлежать к одному из трех типов: “один к одному”, “один ко многим” или “многие ко многим”. Связь “один к одному” означает, что любая данная строка таблицы А связана точно с одной строкой таблицы В, и наоборот.

Такая связь встречается не очень часто, поскольку в подобном случае две таблицы можно объединить в одну. Наиболее распространена связь “один ко многим”, при которой одна строка таблицы связана с несколькими строками другой таблицы, причем вторая таблица связана только с одной строкой первой. Примером отношения подобного типа является связь между клиентом и заказами, которые он помещает. Клиент может сделать один, или несколько заказов в компании, или ни одного, однако любой заказ может принадлежать только одному клиенту. Третий вариант связи, “многие ко многим”, существует, когда строка таблицы X связана с многими строками таблицы Y, и строка таблицы Y связана с многими строками таблицы X. В качестве примера можно привести отношения между студентами и преподавателями: студент может иметь несколько преподавателей (если ему читают несколько предметов), а любой преподаватель практически всегда читает лекции многим студентам. Отношение подобного типа не очень удобны в системах реляционных баз данных, поэтому их следует заменять отношениями “один ко многим”, когда помимо двух исходных таблиц применяется третья, содержащая ключевые столбцы из двух исходных таблиц (она описана ниже).

## Нормализация структуры

После того как вы соберете данные и сгруппируете их по таблицам, можете изучить таблицы, чтобы еще больше улучшить их структуру. Процесс нормализации данных требует расщепления данных на несколько таблиц, связанных между собой на основе содержащихся в них данных. Удобным и компактным методом представления содержимого таблиц базы данных является использование формы записи, подобной показанной на рис. 3.1, где представлены таблицы *Customers*, *Listings* и *SaleStatus* (все они — часть базы данных Redwood Realty). Для данных таблиц указывается название, а затем в скобках перечисляются имена столбцов. Столбец, имя которого подчеркнуто, является первичным ключом; как правило, поля первичного ключа указываются первыми. *Первичный ключ* единственным образом определяет строку таблицы; точно так же, как идентификатор (номер студенческого билета) единственным образом определяет студента института. Если подчеркнуто больше одного столбца (т.е. первичный ключ состоит из нескольких полей таблицы), эти поля называются *составным первичным ключом*. Отметим, что первичного ключа таблицы недостаточно для связи одной таблицы с другими. Для завершения соединения требуется *внешний ключ*, представляющий собой поле, единственным образом определяющее строку в другой таблице. Иными словами, внешний ключ в одной таблице согласуется со значением первичного ключа в другой связанной таблице. На рис. 3.1 внешний ключ *SaleStatusID* в таблице *Listings* определяет одноименный первичный ключ в таблице *SaleStatus*. Хотя в данном примере имена ключей совпадают, так бывает не всегда. (Внешний и первичный ключи определяются при создании таблиц; ниже вы сможете попрактиковаться в этом.)

```
Customers(CustomerID, FirstName, LastName, Address, City, State ,Zipcode, HomePhone, CellPhone,
WorkPhone)
Listings(ListingID, PropertyID, ListingAgentID, SaleStatusID, BeginListDate, EndListDate, AskingPrice)
SaleStatus(SaleStatusID, StatusText)
```

**Рис. 3.1.** Представление таблиц путем перечисления их названий и заголовков столбцов

Достоинством данной формы записи является то, что для ее реализации вам не требуется специальное графическое программное обеспечение. Все, что нужно, вы можете написать от руки или использовать текстовый процессор. Основной недостаток заключается в сложности показа связей между таблицами. В любом случае такая запись хороша для начала процесса нормализации, который обычно включает уменьшение числа столбцов таблицы и перенос некоторых столбцов в другую существующую или новую таблицу.

В главе 1 мы описывали значение первой, второй и третьей нормальной формы. Структуру исходной таблицы можно исследовать, обеспечив вначале представление в первой нормальной форме. Хотя подробное обсуждение нормализации в книге не рассматривается, несколько примеров ее мы все же разберем. Рассмотрим таблицы базы данных Redwood Realty. Предположим, что после беседы с сотрудниками компании Redwood Realty относительно их информационных нужд вы решили, что таблица *Customers* должна содержать обычную контактную информацию о клиенте. Кроме того, для тех клиентов, которые желают продать недвижимость, таблица *Customers* содержит подробности относительно продаваемого объекта, включая его адрес, метраж, число комнат и т.д.

Упрощенная форма подобной структуры показана на рис. 3.2. Как видите, без проблем не обошлось. Что, например, вы будете делать, если продавец решит не продавать недвижимость? База данных Redwood Realty может удалить запись, содержащую описание собственности, поскольку теперь данный объект уже не выставлен на рынке. Однако в таком случае удаляется также информация о продавце, а этот контакт в принципе желательно сохранить, ведь он может пригодиться для будущей переписки или деловых отношений. Описанная структура нарушает правила второй нормальной формы. В текущем виде таблица *Customers* хранит две группы фактов: информация о клиентах и данные о недвижимости. Хотя две данные темы *связаны* между собой, совмещать их не стоит. Выход заключается в расщеплении таблицы на две: одна — для хранения информации о клиентах, вторая — для записи данных о недвижимости. В связи с этим возникает очевидный вопрос: как поддержать связь между данными фактами, если они хранятся в различных таблицах? Ответ: добавить в таблицу со сведениями о недвижимости внешний ключ, содержащий значение одного из первичных ключей клиента. Для создания соединения генерируется запрос (выражение *SELECT*), в котором Oracle согласует внешний ключ таблицы с данными о недвижимости с первичным ключом таблицы, содержащей сведения о клиентах (это называется *операцией объединения* и подробно объясняется в главе 5).



**Рис. 3.2.** Упрощенная исходная структура таблицы *Customers* агентства недвижимости

Полный процесс нормализации базы данных хотя и очень важен, в данной книге рассматриваться не будет. Мы лишь расскажем, как с помощью Oracle создавать и запрашивать таблицы, а также как использовать другие инструменты Oracle. Если вы хотите больше узнать о нормализации — обращайтесь к литературе по разработке и нормализации баз данных. Кстати, существует множество Web-сайтов, содержащих подробную информацию о базах данных вообще и об Oracle в частности. Найти их можно с помощью Google ([www.google.com](http://www.google.com)).

## Учетные записи пользователей Oracle

Работая с персональными базами данных, подобными Microsoft Access, вы запускаете систему базы данных и либо создаете новую базу данных, либо открываете существующую. В любом случае вы, скорее всего, будете единственным пользователем базы данных, и Access не требует, чтобы вы вводили имя пользователя и пароль перед каждым сеансом. База данных Oracle позволяет большому числу пользователей одновременно работать с одной системой базы данных, которая обычно хранится на отдельном сервере. Файлы базы данных, хранимые и поддерживаемые Oracle, обеспечивают безопасность путем поддержания принадлежащих пользователям таблиц базы данных и других объектов в отдельной защищенной области. Данная область называется *схемой пользователя*. Все объекты, хранимые в схеме пользователя, называются *объектами схемы* (или *объектами базы данных*). Промышленная база данных “клиент/сервер”, подобная Oracle, требует, чтобы любой пользователь, желающий получить доступ к базе данных, предоставил “мандат” — имя пользователя и пароль. Как правило, имя пользователя и пароль выдаются администратором базы данных, АБД (Database Administrator — DBA). Если вы установили и используете версию Personal Edition базы данных Oracle на собственном компьютере, то можете изменить пароль для нескольких системных (администраторских) учетных записей, включая SYSTEM и SYSDBA. Если вы уже задали собственное имя пользователя и пароль, информация, предлагаемая в следующем разделе, все равно может вам пригодиться. Помимо информации о том, как создать новую учетную запись пользователя и пароль, в этом разделе пользователям Personal Oracle даются пошаговые инструкции по установке определенных критических привилегий объектов и системы. Данные привилегии

требуются для создания таблиц, представлений и доступа к вашей схеме. Если вы не используете Personal Oracle, можете сразу переходить к разделу “Создание таблиц”.

---

---

## Дальнейшие указания для пользователей персональных систем Oracle

Если вы установили Personal Oracle, то, скорее всего, присвоили новый пароль специальной учетной записи SYSTEM. Данную учетную запись следует использовать очень осторожно (и редко), не применяя ее для создания таблиц или любых других действий. При работе с книгой используйте “нормальную” учетную запись с ограниченными привилегиями, чтобы вы не могли нанести серьезного ущерба, введя неверную команду. Тем не менее, чтобы создать новую учетную запись пользователя и присвоить ей пароль, вам все же придется войти в систему под именем SYSTEM. Опишем этот процесс подробнее.

### Создание учетной записи пользователя

Для того чтобы создать в базе данных новую учетную запись пользователя, вы должны войти в систему как привилегированной пользователь, а затем ввести команду CREATE USER. В указанном качестве вы можете присваивать привилегии объектов и систем. Затем выйдите из системы, войдите под только что созданным “обычным” именем пользователя и спокойно занимайтесь своими делами: созданием таблиц и другими операциями, доступными в вашей схеме. Упрощенный синтаксис команды CREATE USER выглядит следующим образом:

```
CREATE USER <имя_пользователя> IDENTIFIED BY <пароль>
  [DEFAULT TABLESPACE <tablespace_name>]
  [TEMPORARY TABLESPACE <tempspace_name>]
```

В приведенной команде вместо <имя\_пользователя> подставляется новое имя пользователя, вместо <пароль> — пароль. Ниже будет предполагаться, что вы соединяетесь с базой данных в качестве привилегированного пользователя. В данном случае мы входим в систему под именем пользователя SYSTEM, по умолчанию имеющего пароль MANAGER. Вы должны использовать тот пароль, который присвоили ранее учетной записи SYSTEM.

Итак, чтобы создать в Oracle новую учетную запись пользователя, выполните следующие действия.

1. Используя SQL\*Plus, войдите в Oracle, выбрав учетную запись SYSTEM и пароль, присвоенный в процессе установки.
2. Выполните приведенную ниже команду, чтобы создать новую учетную запись пользователя. Выберите имя пользователя и пароль. Запомните их! Для примера

мы выбрали имя пользователя Chapter3 и пароль ChangeMe. Нажмите клавишу <Enter>, чтобы выполнить команду.

```
CREATE USER Chapter3 IDENTIFIED BY ChangeMe;
```

- Oracle ответит сообщением “User created”. Окно SQL\*Plus пока не закрывайте.

Далее вы временно прекращаете работу под учетной записью SYSTEM и входите, используя новую учетную запись, чтобы проверить, что все сделано правильно. Далее мы будем использовать указанные имя пользователя и пароль; если вы их изменили, то в соответствующих местах подставляйте свои варианты.

## **Вход в базу данных с использованием другого имени пользователя и пароля**

Теперь вам необходимо выйти из системной учетной записи и попытаться войти с использованием новой пары “имя пользователя–пароль”.

Чтобы войти в Oracle под другим именем пользователя (используя другую учетную запись), если у вас в текущий момент есть соединение с Oracle, выполните следующие действия.

- В командной строке SQL введите приведенную ниже команду и нажмите клавишу <Enter>. Подставьте свое имя пользователя и пароль. Имя пользователя указывается перед косой чертой, а пароль — после.

**Совет.** Имена пользователей и пароли в Oracle нечувствительны к регистру. В связи с этим большинство пользователей предпочитают строчные буквы, так как это не требует использования клавиши <Shift>.

```
CONNECT chapter3/changeme;
```

- В ответ Oracle выведет сообщение об ошибке:

```
ERROR:  
ORA-01045: user CHAPTER3 lacks CREATE SESSION privilege;  
logon denied
```

- Oracle выйдет из системной учетной записи, оставив запущенной среду SQL\*Plus, но не входя в систему под другим именем пользователя (рис. 3.3).

## **Изменение системных привилегий пользователя**

Ошибка, появившаяся на предыдущем этапе, “... lacks CREATE SESSION privilege ...” (“отсутствует привилегия CREATE SESSION”), означает, что указанные имя пользователя и пароль существуют, однако не дают права входить в систему.

The screenshot shows a Windows-style window titled "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area displays the following text:

```

SQL*Plus: Release 10.1.0.2.0 - Production on Sat Oct 14 14:13:32 2006
Copyright (c) 1982, 2004, Oracle. All rights reserved.

Connected to:
Personal Oracle Database 10g Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> CREATE USER Chapter3 IDENTIFIED BY ChangeMe;
User created.

SQL> CONNECT chapter3/changeme;
ERROR:
ORA-01045: user CHAPTER3 lacks CREATE SESSION privilege; logon denied

Warning: You are no longer connected to ORACLE.
SQL>

```

**Рис. 3.3.** Попытка войти в систему с использованием новой учетной записи без привилегий

Все, что вы сделали на предыдущем этапе, — это зарезервировали для пользователя имя `Chapter3`, но не разрешили его использовать. Помимо этого, пользователь `SYSTEM` должен разрешить соединение с базой данных, необходимое для того, чтобы новый пользователь мог создать сеанс. В Oracle данная возможность называется привилегией `CREATE SESSION`. Предоставлять данную привилегию может только администратор базы данных или другой суперпользователь. Ниже вы это сделаете, переключившись на привилегированную учетную запись `SYSTEM`.

Для того чтобы предоставить дополнительные привилегии пользователю, выполните следующие действия.

- Соединитесь с Oracle через учетную запись администратора базы данных `system`. В командной строке SQL выполните указанную ниже команду. (Для входа в систему мы задействовали имя пользователя `system` и пароль `manager`. Вы можете использовать любую учетную запись с правами администратора.)

```
CONNECT system/manager;
```

Oracle выдаст сообщение “Connected”, указывающее, что вы снова вошли в систему.

- Чтобы предоставить привилегии `CREATE SESSION` и `CREATE TABLE`, выполните следующую команду, подставив вместо `chapter3` имя нового пользователя:

```
GRANT CREATE SESSION, CREATE TABLE TO chapter3;
```

Oracle ответит “Grant succeeded”.

- Проверьте права нового пользователя, снова войдя в систему. Введите имя пользователя и пароль, как показано ниже.

```
CONNECT chapter3/changeme;
```

4. Oracle выведет сообщение “Connected”. Не закрывайте SQL\*Plus и не изменяйте учетную запись.

## Изменение паролей других пользователей

Если вы можете войти в Oracle, используя одну из администраторских учетных записей базы данных, например SYSTEM, то вы можете изменять пароль любого пользователя, зарегистрированного в системе. То же относится и к менеджеру базы данных, наблюдающему за корпоративной системой Oracle типа “клиент/сервер”. Если пользователь забудет свой пароль (или вы забудете свой), АБД может присвоить новый. Поскольку пароли шифруются с использованием одностороннего алгоритма, восстановить утерянный пароль не сможет даже администратор, однако он может дать пользователю новый пароль и вынудить изменить его при первом использовании. Данную возможность предоставляет необязательная опция PASSWORD EXPIRE после команды ALTER USER. Упрощенный вариант синтаксиса команды ALTER USER, позволяющей задать новый пароль и требующей от пользователя смены пароля после первого использования, выглядит так:

```
ALTER USER <имя_пользователя> IDENTIFIED BY <new_password>
    PASSWORD EXPIRE
```

Ниже подробно описано, как изменить пароль пользователя chapter3 на Columbus и вынудить пользователя сменить этот пароль после первого входа в Oracle.

Для того чтобы изменить пароль другого пользователя, необходимо выполните следующие действия.

1. В окне SQL\*Plus переключитесь на учетную запись SYSTEM, выполнив в командной строке SQL следующий код (предполагается, что пользователь SYSTEM по-прежнему имеет пароль MANAGER):

```
CONNECT SYSTEM/MANAGER;
```

2. Чтобы изменить пароль пользователя Chapter3 на Columbus, введите следующую команду:

```
ALTER USER Chapter3 IDENTIFIED BY Columbus PASSWORD EXPIRE;
```

Oracle ответит подтверждением “User altered”.

3. Запомните пароль, который вы ввели после “IDENTIFIED BY”, поскольку до передачи его пользователю Chapter3 кроме вас этот пароль никто не знает.
4. Введите CONNECT Chapter3/Columbus; и нажмите клавишу <Enter>, чтобы войти в Oracle, используя новый пароль. Oracle укажет, что время действия пароля истекло, и потребует, чтобы пользователь сменил пароль.

The screenshot shows the Oracle SQL\*Plus interface. The command entered is:

```
SQL> ALTER USER Chapter3 IDENTIFIED BY Columbus PASSWORD EXPIRE;
```

Followed by:

```
User altered.
```

Then, the user attempts to connect:

```
SQL> CONNECT Chapter3/Columbus;
ERROR:
ORA-28001: the password has expired
```

After changing the password:

```
Changing password for Chapter3
New password: *****
Retype new password: *****
Password changed
Connected.
SQL>
```

A brace on the right side of the terminal window groups the password change command and its confirmation, with the label "Вход в систему и изменение пароля" (Login system and password change) positioned below it.

**Рис. 3.4.** Изменение пароля другого пользователя

**Совет.** Когда вы будете выполнять п. 5, помните, что пароли *чувствительны к регистру*, поэтому при вводе его оба раза одинаково используйте прописные и строчные буквы. После изменения пароль снова становится нечувствительным к регистру.

- После запроса “New password:” введите новый пароль, *Sherlock*, нажмите клавишу <Enter>, еще раз введите пароль (очень внимательно) и нажмите клавишу <Enter>. Если оба раза вы набрали одинаковую строку символов (включая регистр), то Oracle ответит сообщением “Password changed”, а затем “Connected” (рис. 3.4.). Не закрывайте SQL\*Plus.

Если вы не можете войти в Oracle с привилегиями администратора — обратитесь к администратору базы данных, чтобы он присвоил вам новый пароль. С другой стороны, личные пароли стоит время от времени менять из соображений безопасности.

## Изменение собственного пароля

Любой пользователь может поменять собственный пароль, не обращаясь к администратору базы данных. Войдя в Oracle посредством SQL\*Plus или iSQL\*Plus, введите команду **PASSWORD**. После этого SQL\*Plus запросит ввести старый пароль, а затем дважды указать новый. Будьте внимательны и правильно введите новый пароль, учитывая регистр.

- Ведите **PASSWORD** и нажмите клавишу <Enter>.
- Ведите старый пароль, *Sherlock*, и нажмите клавишу <Enter>.
- Ведите новый пароль, *Timecrunch* (или любой другой по вашему выбору), и нажмите клавишу <Enter>. Повторите это еще раз.
- Появится ответ “Password changed”, подтверждающий, что вы изменили пароль. SQL\*Plus не закрывайте.

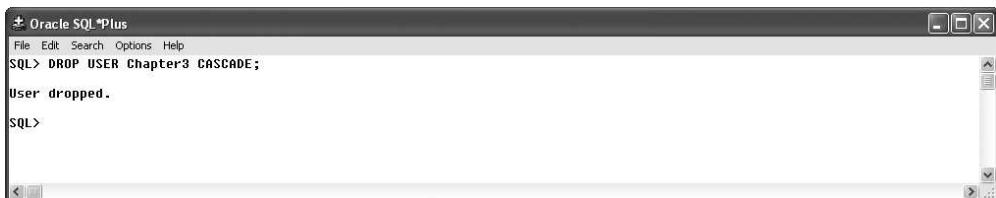


Рис. 3.5. Удаление учетной записи пользователя

## Удаление учетной записи пользователя

Для логического завершения данного сеанса удалим из системы учетную запись Chapter3. Чтобы удалить учетную запись пользователя, вы должны войти в систему с привилегиями, равными правам учетной записи SYSTEM. “Рядовой” пользователь не может удалить из системы другого пользователя (к счастью).

Для удаления учетной записи применяется команда DROP USER, синтаксис, который приведен ниже.

```
DROP USER <имя_пользователя> [CASCADE]
```

Итак, чтобы удалить пользователя, необходимо выполнить следующие действия.

1. Соединитесь с базой данных от имени пользователя SYSTEM.

```
CONNECT SYSTEM/MANAGER;
```

2. Чтобы удалить учетную запись Chapter3, введите DROP USER Chapter3 CASCADE; и нажмите клавишу <Enter>. Наберитесь терпения. Для удаления учетной записи пользователя и поиска по базе данных всех объектов его схемы потребуется несколько секунд. Вскоре Oracle ответит “User dropped” (рис. 3.5).
3. Введите Exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus.

## Создание таблиц

Таблицы базы данных — это структуры, в которых хранятся все данные и которые являются основой базы данных. Чтобы создать новую таблицу, необходимо задать ее имя, имя каждого столбца, тип данных, содержащихся в столбце, а также размер поля данных. Помимо этого, вы можете задать ограничения (или условия) на значения, которые может содержать поле. Например, вы хотите, чтобы поле Gender таблицы сотрудников могло иметь только значение M или F (прописными буквами) или чтобы столбец Title таблицы Agents или базы данных Redwood Realty содержал только значение “Broker” или “Salesperson”. Другой тип ограничения

запрещает использовать незаполненное поле (поле со значением NULL), требует от пользователя обязательного ввода значения. (Подробнее ограничения рассмотрены в следующем разделе.)

Имена таблиц Oracle должны соответствовать следующим правилам.

- Содержать от 1 до 30 символов.
- Начинаться с символа.
- Содержать буквы, цифры и специальные символы \$, \_ (подчеркивание) и #.
- Не содержать пробелов.
- Не дублировать имя другого объекта, которым владеет тот же пользователь (в текущей схеме).
- Не являться зарезервированным словом Oracle.
- Не являться словом DUAL (имя фиктивной таблицы).

Имена таблиц нечувствительны к регистру. Следовательно, имена Customer, customer и CuStOmEr идентичны в любом выражении SQL.

На имена с разделителями накладывается меньше ограничений, чем на описанные выше обычные имена. *Именами с разделителями* называются имена объектов (таблицы, столбцы и т.п.), помещенные в двойные кавычки, например "CreditLimit". Хотя сами кавычки в базе данных не хранятся, все остальные символы записываются точно так, как они представлены в выражении SQL. Данные имена чувствительны к регистру, поэтому "CreditLimit-и CREDITLIMIT — это разные имена. При этом "CREDITLIMIT-и CreditLimit являются одинаковыми именами, поскольку Oracle представляет прописными буквами все имена, не взятые в двойные кавычки. В именах с разделителями допускается использование пробелов, кроме того, данные имена могут совпадать с зарезервированными словами SQL. Тем не менее мы не рекомендуем использовать в именах с разделителями зарезервированные слова SQL и пробелы. В любом случае советуем решить более важный вопрос согласованного использования: либо обычные имена, либо имена с разделителями.

Как говорилось в главе 2, для создания таблицы применяется выражение CREATE TABLE. Напомним, что упрощенная версия его синтаксиса выглядит следующим образом:

```
CREATE TABLE [schema.]table_name
(column_name type [NOT NULL]
 [CONSTRAINT constraint_definition DEFAULT expression]
 [,column_name type [NOT NULL]
 [CONSTRAINT constraint_definition DEFAULT expression]...]
)
[PRIMARY KEY (column_name [,column_name ...])]
[FOREIGN KEY (column_name [,column_name ...]) REFERENCES foreign_table (column_name [,column_name ...])]
```

В данном коде `table_name` — это имя, присваиваемое таблице, `column_name` — имя, соотносимое со столбцом, после которого идет один из допустимых типов данных, например, числа, даты или символы. (Подробнее типы данных Oracle описаны в следующем разделе.) Значение, следующее после `DEFAULT`, не позволяет вставлять в столбцы значения `NULL`, вводя вместо них значения, заданные по умолчанию. `Constraint_definition` задает определение условия (или ограничения) на столбец. (Ограничения и примеры их использования описаны ниже.) Полный синтаксис команды `CREATE TABLE` гораздо больше, чем показано выше, подробное его описание можно найти в справочнике SQL Reference от Oracle Corporation.

## Типы данных Oracle

Изучая синтаксис команды `CREATE TABLE`, обратите внимание на то, что каждый столбец имеет собственный тип данных, указанный после имени столбца. *Тип данных* задает (или ограничивает), данные какого типа может содержать поле. Приведливость базы данных Oracle относительно информации, которая может храниться в столбцах, объясняется двумя причинами. Во-первых, зная тип данных, Oracle может более эффективно распределять место под таблицы. Во-вторых, это позволяет Oracle находить ошибки. Если вы попытаетесь записать строку символов в поле заработной платы (которое должно содержать числовые значения), Oracle выдаст сообщение об ошибке и запретит операцию — до того, как она сможет нанести ущерб. Типы данных Oracle можно разбить на четыре группы: символьные, числовые, дата/время и изображения. Чаще всего вам придется сталкиваться с символьными, числовыми данными и датой/временем.

### Символьные данные

В полях с символьными данными хранятся арифметические значения, содержащие текст и числа, не используемые в расчетах. В качестве примеров можно привести идентификационные номера студентов, номера деталей и почтовые индексы. Краткая сводка по символьным типам данных Oracle приведена в табл. 3.1.

`CHAR` и `NCHAR` содержат символьные данные фиксированной длины и данные, образованные символами национальных языков. *Символьные данные фиксированной длины* означают, что поля различных записей имеют одинаковую длину. Например, поле `CHAR` удобно использовать для хранения идентификационных номеров студентов или почтового индекса, поскольку они всегда имеют одинаковую длину. Синтаксис символьных данных фиксированной длины представляется следующим образом:

```
<fieldname> CHAR(<length>)
```

Например, поле почтового индекса в таблице `Customer` вы можете объявить так:

```
Zipcode CHAR(5)
```

**ТАБЛИЦА 3.1.** Символьные типы данных Oracle

<i>Тип</i>	<i>Код Oracle</i>	<i>Описание</i>
Fixed	CHAR(size)	Символьные данные фиксированной длины, образованные size символами, дополненными пробелами. Максимальный размер — 2 000 байт
Fixed National	NCHAR(size)	То же, что и CHAR, только позволяет хранить символы национальных языков максимальной длиной 2 000 байт
Variable	VARCHAR2(size)	Символьные данные переменной длины, образованные size символами. Максимальный размер — 4 000 байт
Variable National	NVARCHAR2(size)	Символьные данные переменной длины, образованные size символами национального языка. Максимальный размер — 4 000 байт
Memo	LONG	Символьные данные переменной длины до 2 Гбайт. (Вместо данного типа рекомендуется использовать тип данных CLOB.)

Другим популярным и универсальным типом данных является VARCHAR2, позволяющий хранить до 4 000 символов. Размер *символьных данных переменной длины* может быть разным для различных записей. Например, фамилия одного сотрудника может иметь длину 20 символов, а другого — всего из 5. Записи переменной длины экономят место, поскольку их не нужно дополнять в конце пробелами, чтобы довести до максимальной длины. Тип NVARCHAR2 похож на VARCHAR2, однако позволяет использовать символы языков с большим и сложным алфавитом. Поле VARCHAR2 объявляется с использованием следующего синтаксиса:

```
<fieldname> VARCHAR2(maximum_size)
```

Большинство символьных полей в базе данных Redwood Realty объявляются как поля переменной длины с символами национальных языков, чтобы в них можно было записывать самые разнообразные данные. Поле LastName в таблице Agents Redwood Realty объявляется следующим образом:

```
LastName NVARCHAR2(30)
```

Данное объявление поддерживает имена переменной длины до 30 символов. Поля Memo (объявляются с помощью команды LONG) используются не очень часто, вместо них рекомендуется применять тип данных CLOB (подробнее о нем — ниже).

**ТАБЛИЦА 3.2.** Числовые типы данных Oracle

<i>Тип</i>	<i>Обозначение Oracle</i>	<i>Описание</i>
INT, INTEGER, SMALLINT	NUMBER(38)	Целое число с точностью до 38 цифр
Фиксированной точности	NUMBER(p,s)	Число переменной длины. Значение p равно максимальному числу цифр, значение s — максимальному числу цифр справа от десятичной запятой
FLOAT, DOUBLE PRECISION	NUMBER	Число с плавающей запятой с точностью до 38 цифр

**Числовые данные**

Для представления всей числовой информации в Oracle применяется тип данных NUMBER. Числовые данные позволяют хранить числа со знаком, с фиксированной или плавающей запятой, имеющие 38 цифр. (Любая из этих 38 цифр может находиться либо до десятичной запятой, либо после нее.) Поле NUMBER следует использовать в любом столбце таблицы, над которым предполагается выполнять арифметические действия. Перечень числовых данных Oracle и их краткое описание приведено в табл. 3.2. В общем, Oracle позволяет использовать объявления INTEGER, INT, SMALLINT, DOUBLE PRECISION, FLOAT, REAL, DEC, DECIMAL и NUMERIC. Тем не менее все они превращаются в объявление данных типа NUMBER. Итак, для объявления числовых полей используется следующий синтаксис:

```
<fieldname> NUMBER [([ precision,] [ scale ])]
```

В этом выражении *precision* (“точность”) означает общее число цифр, хранимых в числе, а *scale* (“масштаб”) — число цифр справа от десятичного разделителя. Поскольку в первое число входят все цифры знака, оно всегда больше второго или равно ему. Рассмотрим, например такое объявление:

```
AmountDue NUMBER(7,2)
```

Оно позволяет хранить числа до 99 999,99 — всего семь цифр, две из них располагаются справа от десятичной точки. Объявление NUMBER(4,4) задает число, у которого четыре цифры располагаются справа от десятичной точки и ни одной слева.

Величина типа Integer представляет собой целое число, не имеющее дробной части (т.е. в объявлении NUMBER параметр scale опускается). Рассмотрим, например, следующий код:

```
AnnualSalary NUMBER(6)
```

Он задает целочисленное поле, максимальное значение которого — целое число 999 999, а минимальное — -999 999. Если вы попытаетесь записать число со

значением `precision`, большим заданного, Oracle выдаст ошибку. Если, например, в рассмотренное выше поле `AnnualSalary` вы введете значение, подобное 23986.59, Oracle округлит его до ближайшего целого числа (в данном случае до 23987).

*Число фиксированной точности* содержит заданное число десятичных знаков, и для них указываются и точность, и масштаб. Например, атрибуты `latitude` (“широта”) и `longitude` (“долгота”) домов, перечисленных в таблице `Properties` Redwood Realty, содержат числа, аргумент `precision` которых равен 8, а аргумент `scale` — 5, т.е. положение тома в пространстве определяется значениями широты и долготы вида 183.5456. Столбец `latitude`, например, объявляется так:

```
Latitude NUMBER(8, 5)
```

Он может представлять и положительные, и отрицательные значения, указывающие соответственно места к северу и к югу от экватора.

*Числа с плавающей запятой* содержат переменное число десятичных знаков. Цифры могут находиться с любой стороны десятичной запятой, в том числе допустимы числа без цифр слева от запятой и числа без цифр справа от запятой. Для объявления чисел с плавающей запятой используется команда `NUMBER` без скобок и чисел в них. Числа с плавающей запятой широко применяются в научной среде, например, для представления чисел, очень близких к нулю, или очень больших чисел. В качестве примера из базы данных Redwood Realty можно привести поле, содержащее размер комиссионных риэлторов в процентах, — например, 4,125.

## Информация о дате и времени

Для записи значений даты и времени в Oracle доступны три типа данных: `DATE`, `INTERVAL` и `TIMESTAMP`. Тип данных `DATE` хранит информацию о дате и времени, причем время округляется до ближайшей полной секунды. Oracle поддерживает данные в диапазоне от первого января 4712 года до н.э. до 31 декабря 4712 года н.э. Близким “родственником” данного типа является тип `TIMESTAMP`, содержащий год, месяц, день и время. В отличие от типа данных `DATE`, этот тип хранит также доли секунды (например, 17-ОСТ-06 11.02.24.00 AM). `INTERVAL` хранит временной интервал между двумя моментами времени (представляемыми с помощью соответствующих типов данных). Перечень типов данных Oracle для даты/времени и их краткое описание приведены в табл. 3.3.

По умолчанию столбцы `DATE` имеют формат `DD-MON-YY`, где `DD` — двузначная дата; `MON` — трехбуквенное сокращенное название месяца; `YY` — последние две цифры года. Время по умолчанию представляется как `HH:MI:SS a.m.` — часы, минуты и секунды в 12-часовой форме записи. Если при вводе данных в столбец `DATE` вы опустите время, Oracle установит его равным `12:00:00 A.M.` Если вы при вводе времени вы не укажете дату, то по умолчанию она будет задана как первый день текущего месяца. Тип данных `DATE` объявляется следующим образом:

**ТАБЛИЦА 3.3.** Типы данных даты и времени Oracle

<i>Тип</i>	<i>Обозначение Oracle</i>	<i>Описание</i>
Дата и время с точностью до полной секунды	DATE	Дата и время в пределах столетия, четыре цифры года, месяц, день, час, минута и секунда
Временной интервал в годах	INTERVAL YEAR(p) TO MONTH	Время в годах и месяцах, где точность (p) указывает число цифр в поле данных даты/времени YEAR
Временной интервал в днях	INTERVAL DAY(p) TO SECOND(f)	Время в днях, часах, минутах и секундах, где p — максимальное число цифр в дне; f — число цифр дробной части поля секунд
Дата и время с точностью до долей секунды	TIMESTAMP(p)	Хранит дату и время с точностью до долей секунды. Число десятичных знаков задается в скобках (p)

```
HireDate DATE
```

Данные типа DATE хранятся в нескольких полях базы данных Redwood Realty, например, в полях HireDate и BirthDate таблицы Agents.

Данные типа TIMESTAMP содержат значения с более точным времененным компонентом — до долей секунды. Данные такого типа можно применять для записи точного времени (на основе системных часов Oracle) изменения таблицы или отслеживания любых срочных модификаций таблиц базы данных. Данный тип идеально подходит для записи очень точных значений времени. (Подробнее об использовании данных типа TIMESTAMP речь пойдет в главе 4.) Объявление столбца данных TIMESTAMP выглядит следующим образом:

```
UpdateTableDateTime TIMESTAMP(2)
```

Значение в круглых скобках (точность) задает число десятичных знаков, приемлемых для хранения долей секунды. Если опустить это значение, то Oracle будет использовать шесть позиций, заданных по умолчанию.

Тип данных INTERVAL YEAR TO MONTH позволяет хранить временной промежуток в годах и месяцах в виде положительного или отрицательного значения. Если к определенному значению даты/времени добавить положительный промежуток времени, мы получим дату после указанного момента. И наоборот, прибавление к данной дате отрицательного промежутка дает момент времени, предшествующий этой дате. Значения указанного типа хранятся в следующие форме.

+ или - <elapsed years>-<elapsed months>

где + или - означает положительный или отрицательный промежуток времени, после чего следует пара чисел. Значение <elapsed years> представляет годы,

а <elapsed months> — месяцы промежутка времени. Между данными числами ставится дефис. Например, промежуток времени +06-09 представляет положительный срок в шесть лет и девять месяцев. Объявление промежутка времени выглядит так:

```
TimeSinceLastOfficeVisit INTERVAL YEAR(2) TO MONTH
```

В данном случае указывается, что годы представляются двумя цифрами (например, 02 вместо 2002). Отметим, что поля подобного типа могут оказаться полезными, например, при записи прошедшего и оставшегося времени работы над длительным проектом.

Данные типа INTERVAL DAY TO SECOND записывают прошедшее время, выраженное в часах, минутах и секундах. Формат этих данных выглядит следующим образом:

```
+ или - <days>:<hours>:<minutes>:<seconds>
```

Например, дата +06 07:26:51.00 представляет положительное прошедшее время, которое составляет 6 дней, 7 часов, 26 минут и 51 секунду. Ниже показано, как можно объявить столбец данных типа INTERVAL DAY TO SECOND.

```
TimeTrials INTERVAL DAY TO SECOND
```

Теперь вам необходимо разобраться в некоторых из описанных типов данных и понять, как они отображаются на экране, когда вы этого требуете. Выполните описанные ниже действия, чтобы запустить сценарий и изучить некоторые числовые типы данных и типы данных даты/времени. Как обычно, вы можете выбирать — использовать SQL\*Plus или iSQL\*Plus. В данном случае файл сценария рекомендуется запускать с помощью интерфейса iSQL\*Plus, поскольку некоторые столбцы будут довольно широкими и из-за этого не так удачно отобразятся в SQL\*Plus. Упомянутый файл сценария создает таблицу DemoDataTypes, помещает в нее одну строку, вызывает команду SELECT, чтобы отобразить на экране строку, и удаляет таблицу, чтобы вернуть базу данных в исходное состояние.

Итак, прежде всего найдите на диске файл Ch03DataTypes.sql и запишите полный путь к нему. Чтобы изучить некоторые типы данных, доступные в Oracle, выполните следующие действия.

1. Запустите iSQL\*Plus и войдите в Oracle, используя имя пользователя и пароль.
2. Щелкните на кнопке Load Script, щелкните на кнопке Browse рядом с текстовым окном File, выберите Look из списка и дойдите до папки, содержащей Ch03DataTypes.sql. Выберите Ch03DataTypes.sql и щелкните на кнопке Open.
3. Щелкните на кнопке iSQL\*Plus Load, чтобы загрузить сценарий в текстовое окно Workspace. Прокрутите код и изучите его. В начале сценария идет команда CREATE TABLE, создающая три типа числовых полей, после чего идут три типа полей даты/времени.



Рис. 3.6. Выбранные типы данных Oracle

- Щелкните на кнопке **Execute**, чтобы запустить сценарий. Появятся два информационных сообщения, единственная строка таблицы и команда `Table dropped` (рис. 3.6).
- Если хотите, можете распечатать Web-страницу, чтобы изучить ее позже.
- Щелкните на гиперссылке `Logout` вверху страницы, чтобы выйти из *iSQL\*Plus*, а затем закройте браузер.

Первые три отображенных значения представляют собой одно и то же число, записанное с использованием различных числовых типов данных. Внимательно изучите отличия. Следующие два поля — это значения даты/времени. Обратите внимание на различие между `DATE` и `TIMESTAMP` в части, представляющей доли секунды. Наконец, в последнем столбце записан промежуток времени, представляющий почти 326 дней.

## Данные изображения

Для хранения бесструктурных данных, например, фотографий сотрудников, звуков или бинарных файлов, подобных документам Word или Excel, необходимо включ-

**ТАБЛИЦА 3.4.** Типы данных изображения

<i>Обозначение Oracle</i>	<i>Описание</i>
BLOB	Binary LOB — бинарные, неструктурированные данные для очень больших объектов (размером до 128 терабайт)
CLOB	Character LOB — символьные данные для очень больших объектов (размером до 128 терабайт)
NCLOB	National Character LOB — символьные данные переменной длины в формате Unicode, имеющие размер до 128 терабайт

чить один из LOB-типов данных Oracle (Large Object — “большой объект”). Краткие сведения об этих типах данных представлены в табл. 3.4.

Для того чтобы столбец таблицы имел один из типов данных, представленных на рис. 3.4, необходимо использовать объявление

```
<imageitemname> BLOB
```

Вместо BLOB вы можете подставить любой другой тип данных большого размера. Длину задавать не нужно, поскольку Oracle автоматически выделяет достаточно места для хранения объекта. В качестве иллюстрации использования описанных данных рассмотрим следующий пример. Расширенный вариант базы данных Redwood Realty может включать столбец с фотографиями домов, выставленных на продажу (данный столбец логично поместить в таблицу Listings, в которой перечисляется продаваемая недвижимость).

## Создание таблицы с использованием SQL\*Plus

Основной таблицей агентства Redwood Realty является Customers. Ее строки содержат контактную информацию о клиентах, включая их имя и фамилию, адрес и номер телефона. Имена столбцов таблицы Customers, их тип данных и длина показаны в табл. 3.5. Саму таблицу вы создадите чуть позже. Однако в любом случае перед созданием любой таблицы в вашей схеме вы должны очистить ранее созданные таблицы Redwood Realty, чтобы они не конфликтовали с новыми.

**Совет.** Удаление предыдущих вариантов — это первое, что вы делаете, начиная работу с новой главой. Используйте для этого указанный ниже файл сценария, который очищает базу данных Redwood Realty от всех таблиц.

Прежде чем выполнять описанные ниже действия, запишите полный путь к файлу сценария Ch03InitializeRedwood.sql (он понадобится вам ниже).

Для того чтобы инициализировать базу данных Redwood Realty, выполните следующие действия.

**ТАБЛИЦА 3.5.** Имена столбцов и типы данных таблицы Customers

Имя столбца	Тип данных	Максимальная длина	Специальные условия
CustomerID	Integer		Первичный ключ
FirstName	Символы Unicode, строка переменной длины	30	Не нуль
LastName	Символы Unicode, строка переменной длины	30	Не нуль
Address	Символы Unicode, строка переменной длины	40	
City	Символы Unicode, строка переменной длины	30	
State	Символы Unicode, строка переменной длины	20	
Zipcode	Символы Unicode, строка переменной длины	20	
HomePhone	Символы Unicode, строка переменной длины	20	
CellPhone	Символы Unicode, строка переменной длины	20	
WorkPhone	Символы Unicode, строка переменной длины	20	

1. Запустите SQL\*Plus: щелкните на кнопке Start (Пуск), выберите All Programs (Все программы), затем OracleOraDb10\_home1, Application Development и SQL Plus. (Для удобства вы можете создать ярлык к SQL Plus на рабочем столе, чтобы иметь возможность быстро вызывать этот интерфейс.) Откроется диалоговое окно LogOn.
2. Введите свое имя пользователя в текстовом окне UserName, в поле Password введите пароль и щелкните на кнопке OK. (Предполагается, что вы используете систему Personal Oracle. В противном случае вам может потребоваться заполнить поле Host String.)
3. Введите следующий код и нажмите клавишу <Enter>, чтобы выполнить файл сценария:
 

```
START <путь>\Ch03InitializeRedwood.sql
```
4. Никакие сообщения не появятся, поскольку в сценарии есть команда на их подавление. Теперь ваша схема инициализирована.
5. Не закрывайте SQL\*Plus.

Далее необходимо создать таблицу *Customers*. Все символьные поля определяются с типом данных NVARCHAR2, что позволяет использовать символы Unicode и строки переменной длины. Единственным другим типом данных в этой таблице является INTEGER, с его помощью определяются числа с фиксированной запятой, являющиеся уникальными идентификаторами клиентов в таблице.

Для того чтобы создать таблицу *Customers*, выполните следующие действия.

1. Убедитесь, что вы по-прежнему соединены с Oracle посредством клиента SQL\*Plus.
2. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
3. Введите следующий код, нажимая клавишу <Enter> в конце каждой строки для перехода на следующую строку. На забудьте ввести скобки и закрывающую точку с запятой в последней строке.

**Совет.** В определениях столбцов вам не обязательно добавлять пробелы в начале каждой строки. Кроме того, прописные буквы применяются по желанию. Мы использовали их, чтобы отличать ключевые слова Oracle от имен, определяемых пользователем. Чтобы сэкономить время, вы можете набирать все строчными буквами — на результат это не повлияет.

```
CREATE TABLE Customers
  (CustomerID INTEGER,
   FirstName NVARCHAR2(30),
   LastName NVARCHAR2(30),
   Address NVARCHAR2(40),
   City NVARCHAR2(30),
   State NVARCHAR2(20),
   Zipcode NVARCHAR2(20),
   HomePhone NVARCHAR2(20),
   CellPhone NVARCHAR2(20),
   WorkPhone NVARCHAR2(20) );
```

4. Если потребуется, нажмите клавишу <Enter>, чтобы отправить команду CREATE TABLE SQL\*Plus для проверки и Oracle — для выполнения. При появлении ошибок введите Edit в командной строке SQL, исправьте ошибки в редакторе Notepad (Блокнот). Затем выберите в нем File (Файл), Save (Сохранить) и закройте его. После перезаписи измененной команды нажмите клавишу / в окне SQL\*Plus, чтобы очистить экран. Потом нажмите клавишу <Enter> чтобы выполнить правильную команду. Oracle ответит сообщением “Table created”.
5. Введите Describe Customers и нажмите клавишу <Enter>, чтобы показать на экране, как Oracle определяет только что созданную таблицу (рис. 3.7). Обратите внимание на то, что вместо вашего объявления INTEGER для поля CustomerID Oracle использовал объявление NUMBER(38).
6. Не закрывайте SQL\*Plus.

The screenshot shows the Oracle SQL\*Plus interface. In the command window, the following SQL code is entered:

```
SQL> CREATE TABLE Customers
  2  (CustomerID INTEGER,
  3   FirstName NVARCHAR2(30),
  4   LastName NVARCHAR2(30),
  5   Address NVARCHAR2(40),
  6   City NVARCHAR2(30),
  7   State NVARCHAR2(20),
  8   Zipcode NVARCHAR2(20),
  9   HomePhone NVARCHAR2(20),
 10   CellPhone NVARCHAR2(20),
 11   WorkPhone NVARCHAR2(20)
 12 );
```

After executing the command, the message "Table created." is displayed.

Then, the DESCRIBE command is run:

```
SQL> DESCRIBE Customers
```

The resulting table description is:

Name	Null?	Type
CUSTOMERID		NUMBER(38)
FIRSTNAME		NVARCHAR2(30)
LASTNAME		NVARCHAR2(30)
ADDRESS		NVARCHAR2(40)
CITY		NVARCHAR2(30)
STATE		NVARCHAR2(20)
ZIPCODE		NVARCHAR2(20)
HOMEPHONE		NVARCHAR2(20)
CELLPHONE		NVARCHAR2(20)
WORKPHONE		NVARCHAR2(20)

Annotations with arrows point from the explanatory text to the corresponding parts of the code and the output:

- An arrow points from the text "Команда SQL, создающая таблицу Customers" to the CREATE TABLE statement.
- An arrow points from the text "Вот так Oracle создает таблицу" to the DESCRIBE output table.

Рис. 3.7. Создание таблицы с использованием интерфейса SQL\*Plus

## Добавление комментариев к таблицам и столбцам

Oracle хранит информацию обо всех ваших объектах (таблицах, представлениях, столбцах и т.д.) в наборе собственных системных таблиц. Например, в одной системной таблице Oracle хранит данные-комментарии о таблице и столбцах. Для добавления подобного комментария применяется команда SQL COMMENT. Добавляемые комментарии не влияют на содержимое таблиц или столбцов, к которым они привязаны. Комментарии — это небольшая часть документации, которая касается цели и использования таблицы или ее столбцов. Особенно они полезны людям, которые не знакомы со всеми таблицами базы данных, но которым требуется быстро разобраться, для чего нужны эти таблицы и как они связаны между собой.

Чуть позже вы будете добавлять комментарии к созданной ранее таблице *Customers*, а затем отображать эти комментарии, обращаясь к одной из системных таблиц Oracle. Упрощенный синтаксис команды COMMENT, используемой для добавления комментария к таблице, выглядит следующим образом:

```
COMMENT ON TABLE <tablename> IS '<text>'
```

Для того чтобы удалить комментарий из таблицы, с помощью команды COMMENT вводится пустая строка.

Итак, чтобы добавить комментарий к таблице, выполните следующие действия.

1. Запустив SQL\*Plus и войдя в базу данных Oracle, введите CLEAR SCREEN, чтобы очистить экран.
2. Введите следующую команду SQL и нажмите клавишу <Enter>, чтобы выполнить ее:

The screenshot shows the Oracle SQL\*Plus interface. In the command window, the following commands are entered and executed:

```

SQL> COMMENT ON TABLE Customers IS 'People who want to buy or sell a property';
Comment created.

SQL> COLUMN comments FORMAT A40 WORD_WWRAPPED
SQL> SELECT table_name, comments
  2  FROM user_tab_comments
  3 WHERE table_name = 'CUSTOMERS';

TABLE_NAME          COMMENTS
-----              -----
CUSTOMERS          People who want to buy or sell a
                           property

```

**Рис. 3.8.** Создание и отображение комментария к таблице

```
COMMENT ON TABLE Customers IS
'People who want to buy or sell a property';
```

Oracle выведет строку “Comment created”.

3. Чтобы отобразить комментарий на экране, выполните указанные ниже команды SQL\*Plus и SQL. Не забудьте ввести CUSTOMERS прописными буквами, иначе команда не будет выполнена.

```
COLUMN comments FORMAT A40 WORD_WWRAPPED
SELECT table_name, comments
FROM user_tab_comments
WHERE table_name = 'CUSTOMERS';
```

Результат выполнения данной команды показан на рис. 3.8.

4. Ведите **Exit**, чтобы выйти из Oracle, и закройте диалоговое окно SQL\*Plus.

Подобным образом можно определить комментарии для столбца. Например, чтобы поместить комментарий в столбец **State** таблицы **Customers**, необходимо выполнить команду

```
COMMENT ON COLUMN Customers.State IS
'The state should ALWAYS be California';
```

Здесь вы указываете имя таблицы, а через точку за ним — имя поля (**State**). Имя таблицы задается для того, чтобы не путать одноименные столбцы в разных таблицах схемы. Чтобы отобразить на экране комментарии к столбцам таблицы **Customers**, необходимо выполнить команду

```
SELECT *
FROM user_col_comments
WHERE table_name = 'CUSTOMERS';
```

Результат выполнения предыдущего запроса показан на рис. 3.9.

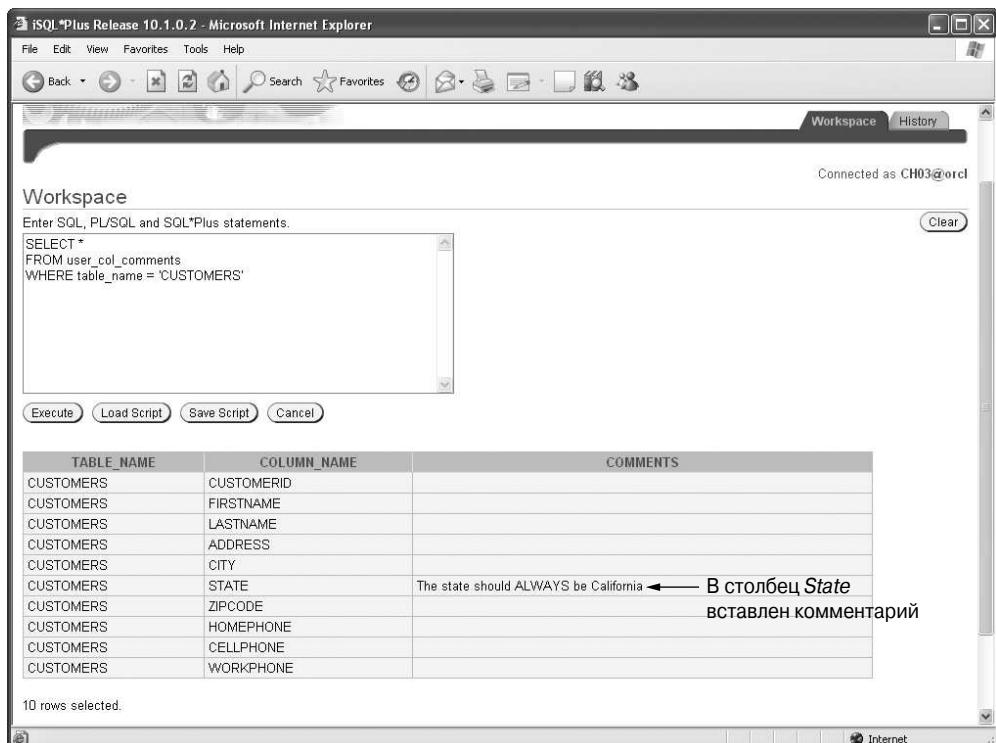


Рис. 3.9. Пример комментария к столбцу

## Определение и использование ограничений

Созданная выше таблица *Customers* содержит правильно определенные имена и типы данных столбцов. Oracle не может просто хранить данные. Таблицы базы должны быть такими, чтобы содержащиеся в них данные были точными и согласованными, а также поддерживали целостность с другими (связанными) таблицами.

### Условия

SQL определяет ряд условий целостности и домена, налагающие определенные правила на таблицы, которые гарантируют, что данные будут оставаться согласованными и точными. *Ограничения* (или *условия*) целостности — это правила, применяемые к таблицам и столбцам таблиц и ограничивающие значения, которые вводятся в таблицы. Точнее, условия целостности определяют, какие столбцы являются первичными ключами (*PRIMARY KEY*), а какие — внешними (*FOREIGN KEY ... REFERENCES*).

Напомним, что первичный ключ, состоящий из одного или нескольких столбцов, единственным образом определяет строку таблицы; внешний ключ — это столбец, идентичный первичному ключу другой таблицы. Условия домена определяют конкретные значения данных или диапазоны значений, допустимые в тех столбцах, с которыми связаны условия. Таким образом может задаваться, что значение столбца не может быть пустым (NOT NULL), должно быть уникальным среди других значений этого столбца (UNIQUE), или что в столбце допустимы только выбранные диапазоны значений (CHECK). Например, условие домена может утверждать, что в столбце Gender (“пол”) может располагаться значение M (“male” — мужской) или F (“female” — женский). Условие на диапазон может указывать, что значения в столбце CommissionRate (“комиссионные”) должны лежать в диапазоне от 1 до 6 процентов и не могут отсутствовать (NOT NULL). Условия можно налагать на отдельные столбцы, отдельные таблицы или ряд таблиц. Условия на столбцы можно указывать сразу после объявления имени столбца и типа данных, кроме того, некоторые или все условия на столбцы можно задать в конце команды CREATE TABLE непосредственно перед закрывающей правой скобкой. Условия на таблицы должны указываться в конце команды CREATE TABLE.

## Именование условий

Итак, SQL предлагает пять условий: CHECK, FOREIGN KEY, NOT NULL, PRIMARY KEY и UNIQUE. Все условия в схеме пользователя должны иметь уникальные имена (имена можно определять самостоятельно или позволить Oracle генерировать их автоматически). Если все условия вы решили именовать самостоятельно, то следует разработать соглашения относительно именования и придерживаться их в дальнейшем. Ниже мы разберем процедуру, позволяющую создавать уникальные и информативные имена условий. Любое соглашение, установленное для именование условий, называется соглашением об именовании условий. В данной книге мы будем придерживаться соглашения, что все имена условий целостности имеют такую форму:

```
{pk|fk}_<primary_key_table_name>_<foreign_key_table_name>
```

Например, столбец первичного ключа в таблице Customer будет иметь имя pk\_customer, где pk — сокращение от “primary key” (первичный ключ). Подобным образом столбец внешнего ключа (foreign key — fk) таблицы Agents, соответствующий первичному ключу таблицы LicenseStatus, будет именоваться fk\_licensesstatus\_agents (вначале таблица, содержащая первичный ключ, затем таблица, содержащая внешний ключ). Как вы помните, Oracle хранит имена объектов в верхнем регистре, если не взять имя в кавычки (что в любом случае делать не рекомендуется), поэтому использование прописных/строчных букв несущественно.

Условия домена мы будем именовать, придерживаясь следующей общей формы:

```
{nn|un|cc}_<tablename>_<columnname>
```

**ТАБЛИЦА 3.6.** Префиксы условий и примеры названий условий

Тип условия	Префикс условия	Пример имени условия
CHECK	ck	ck_customers_zipcode
FOREIGN KEY	fk	fk_customers_properties
NOT NULL	nn	nn_agents_lastname
PRIMARY KEY	pk	pk_agentid
UNIQUE	un	un_contactreason_description

Например, nn\_agents\_lastname — это имя, которое вы присваиваете условию домена NOT NULL столбца LastName таблицы Agents. Отметим, что из-за ограничения длины 30 символами некоторые имена приходится укорачивать. Например, ck\_custagentlist\_commrat — сокращенное имя, которое присваивается для проверки условия на столбец CommissionRate таблицы CustAgentList. Принятые нами соглашения относительно именования условий приведены в табл. 3.6, более подробную информацию по этой теме можно найти в Интернете.

## Определение условий

Прежде чем приступать к созданию условий, рассмотрим подробнее отдельные классы условий, начав с условий целостности.

### Условия на первичный ключ

*Условие на первичный ключ* гарантирует, что все значения столбца, в котором оно объявлено, не пустые (NOT NULL) и уникальные (UNIQUE). Любая таблица должна иметь один первичный ключ, состоящий из одного или нескольких столбцов. Если первичный ключ состоит из нескольких столбцов (*составной первичный ключ*), уникальной должна быть комбинация значений из всех этих столбцов, в пределах одного столбца значения могут повторяться. Первичный ключ, представленный одним столбцом, определяется либо как условие на столбец, либо как условие на таблицу. Составной первичный ключ определяется как условие на таблицу.

Чтобы определить условие на столбец первичного ключа, в конец определения этого столбца добавляется фраза PRIMARY KEY. Например, требуется, чтобы столбец CustomerID таблицы Customers был ее первичным ключом. В таком случае в использованное ранее объявление CREATE TABLE необходимо добавить фразу

```
CustomerID INTEGER PRIMARY KEY
```

Точно так же можно определить имя первичного ключа, используя следующую форму:

```
CustomerID INTEGER CONSTRAINT pk_customers PRIMARY KEY
```

Здесь мы придерживаемся принятого соглашения относительно именования условий — `pk`, подчеркивание, имя таблицы (все — строчными буквами). Первичный ключ можно определить и как условие на таблицу, поместив его в конец выражения `CREATE TABLE` после объявления всех полей. Например, в команде `CREATE TABLE`, налагающей на таблицу условие, которое определяет первичный ключ, последние три строки выглядят так:

```
...
WorkPhone NVARCHAR2(20),
CONSTRAINT pk_customers PRIMARY KEY (CustomerID)
);
```

Составной первичный ключ определяется в конце выражения `CREATE TABLE`. Например, первичный ключ таблицы `CustAgentList` базы данных Redwood Realty состоит из столбцов `CustomerID`, `AgentID`, `ListingID` и `ContactDate`. Объявление `PRIMARY KEY` этой таблицы выглядит следующим образом:

```
...
CommissionRate NUMERIC(4,4),
CONSTRAINT pk_CustAgentList PRIMARY KEY (CustomerID, AgentID,
ListingID, ContactDate));
```

Обратите внимание на то, что при представлении первичного ключа в виде условия на столбец, а не на таблицу, столбец первичного ключа не называется. Это объясняется тем, что в этом случае фраза является частью объявления столбца. Тем не менее в объявлении первичного ключа через условие на таблицу имена столбцов указываются.

## Условие на внешний ключ

Условия на внешние ключи связаны с условиями на первичные ключи. Если условия на первичные ключи обеспечивают целостность данных в таблице и позволяют единственным образом определить любую строку, *условие на внешний ключ* задает, что указанное поле должно существовать в виде первичного ключа в другой таблице (это называется *условие на ссылку*). Обратившись к метафоре “таблицы с внешними ключами являются детьми строк таблиц с первичными ключами”, можно сказать, что внешние ключи гарантируют наличие “родителей” у “детей”. Распространенным примером подобных отношений является связь между таблицей, содержащей заказы, и таблицей, включающей отдельные позиции этих заказов. Поддержание целостности таблицы базы данных означает сохранение связи первичного ключа с внешним. При наличии условия на ссылку вы не можете удалить поле первичного ключа, не удалив вначале все строки, ссылающиеся на строки этого поля посредством внешних ключей.

Внешний ключ, так же как и первичный, можно объявить либо в объявлении столбца (условие на столбец), либо независимо от него в конце команды `CREATE TABLE` (условие на таблицу). Упрощенный синтаксис объявления внешнего ключа в виде условия на столбец выглядит следующим образом:

```
<column-name> <datatype> [CONSTRAINT <constraint-name>]
REFERENCES <tablename> [(<column-name>)
    [ON DELETE {CASCADE | SET NULL}]]
```

Необязательное уточнение ON DELETE задает, что делать при удалении родительских строк: CASCADE указывает, что при удалении родительской строки удаляются и все строки-потомки; если опустить опцию CASCADE, Oracle не даст удалить родительскую строку при наличии в дочерней таблице зависящих от нее строк.

Ниже приведено несколько примеров внешних ключей, заданных в виде условий на столбец OwnerID, ссылающийся на первичный ключ CustomerID в таблице Customers.

```
OwnerID INTEGER CONSTRAINT fk_customers_properties (1)
    REFERENCES Customers (CustomerID)
OwnerID INTEGER REFERENCES Customers (CustomerID) (2)
OwnerID INTEGER REFERENCES Customers (3)
```

В первом примере мы присваиваем имя условию, таблице и столбцу, к которому оно относится. Во втором примере имя условия не задается. Третий пример представляет самый короткий способ задания внешнего ключа для столбца OwnerID — без упоминания столбца CustomerID. Если после имени таблицы, на которую ссылается внешний ключ, вы не указываете имя первичного ключа, то Oracle автоматически подставляет вместо него поле первичного ключа указанной таблицы. Отметим, что имена внешнего и первичного ключей не обязательно должны совпадать, однако типы данных и длина двух полей обязательно должны быть одинаковыми. Как правило, внешние ключи ссылаются именно на первичные ключи таблицы, однако они могут быть связаны и с уникальными полями других таблиц. Употребляющееся в таких случаях условие UNIQUE рассмотрено ниже в данном разделе.

Добавление внешнего ключа в виде *условия на таблицу* происходит подобно описанному выше, только соответствующая команда включается в конец выражения CREATE TABLE и содержит фразу “FOREIGN KEY”. Упрощенная форма объявления внешнего ключа в виде условия на таблицу представлена ниже.

```
[CONSTRAINT <constraint-name>] FOREIGN KEY (<column-name>)
REFERENCES <primary-key-tablename> [(<primary-key-column-name>)]
[ON DELETE {CASCADE | SET NULL}]
```

В представленном коде <column-name> определяет столбец, являющийся внешним ключом, <primary-key-tablename> — имя таблицы, содержащей соответствующий первичный ключ, а <primary-key-column-name> определяет столбец, содержащий этот первичный ключ. При необходимости для определения внешнего ключа вы можете записать условие на таблицу. Например, в приведенной ниже команде внешний ключ LicenseStatusID связывается с одноименным столбцом таблицы LicenseStatus.

```
CREATE TABLE Agents
(AgentID INTEGER PRIMARY KEY,
...
```

```
CONSTRAINT fk_licensestatus_agents FOREIGN KEY (LicenseStatusID)
    REFERENCES LicenseStatus(LicenseStatusID)
);
```

**Совет.** До того как вы сможете создать внешний ключ, ссылающийся на другую таблицу, необходимо создать эту таблицу и задать для нее условие PRIMARY KEY.

## Условия “не нуль”

Условие NOT NULL не позволяет, чтобы значения столбца были пустыми (или равными NULL). Другими словами, для любого столбца, содержащего условие NOT NULL, вы должны задать значение либо предусмотреть значения по умолчанию, которые будут подставлены в поле, если вы не зададите значение явно. NOT NULL можно задавать только как условие на столбец. Используйте его очень осторожно, поскольку при вводе в таблицу новой записи значения некритичных столбцов известны не всегда. Со столбцами первичного ключа условие NOT NULL соотнесено всегда, поскольку эти поля по определению не могут иметь нулевое значение. Хотя это и не является необходимостью, для большинства внешних ключей условие NOT NULL задается неявно, поскольку эти столбцы почти всегда указывают на существующий столбец внешнего ключа в другой таблице. Кроме того, при желании в объявлении NOT NULL можно задать имя условия (в данной книге мы этого не делали).

Среди первых столбцов таблицы *Properties* базы данных Redwood Realty есть два, значения которых не могут отсутствовать, — это *Address* и *City*. Чтобы база данных Oracle заполняла соответствующие поля в обязательном порядке, добавьте опцию NOT NULL в объявление этих столбцов:

```
CREATE TABLE Properties
(PropertyID INTEGER PRIMARY KEY,
 ...
Address NVARCHAR2(30) NOT NULL,
City NVARCHAR2(30) NOT NULL,
State NVARCHAR2(20),
...)
```

Приведенная команда NOT NULL гарантирует, что для любой недвижимости задан ее адрес и город, а значение в столбце *State* не является обязательным. При желании вы можете присвоить имя условиям NOT NULL, однако столбцы, содержащие условие NOT NULL, и так отличаются от других столбцов при использовании команды DESCRIBE <tablename>. На рис. 3.10 показан пример столбцов таблицы *Orders* базы данных Coffee Merchant, явно помеченных “NOT NULL”. В данном случае ORDERID является столбцом первичного ключа, следовательно, ему автоматически присваивается условие NOT NULL.

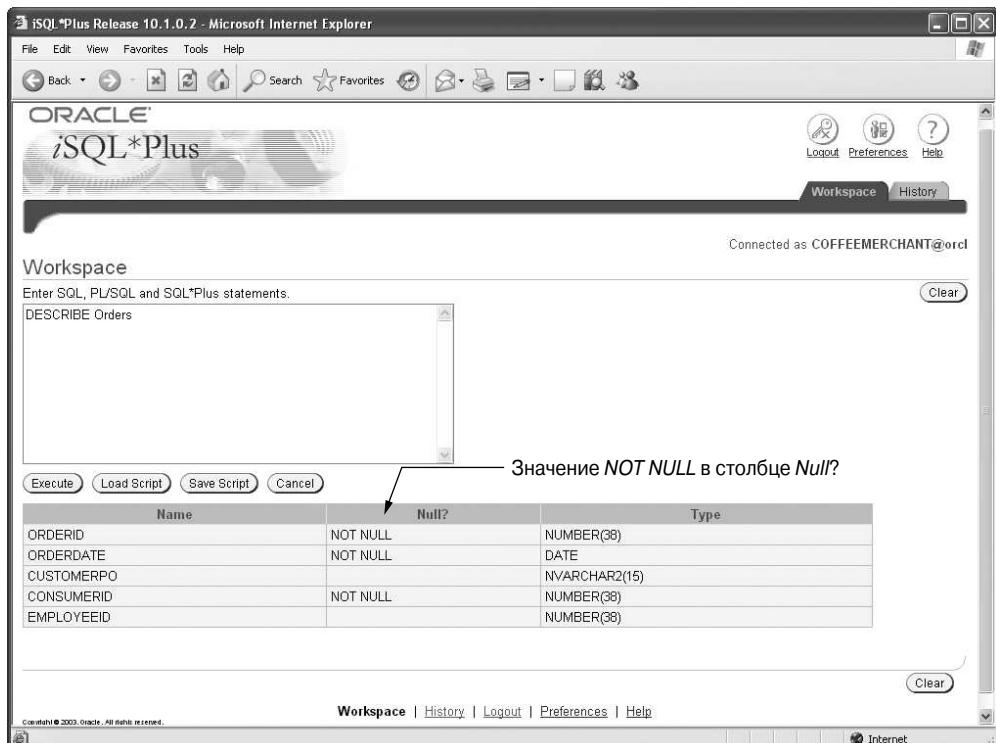


Рис. 3.10. Отображение условий NOT NULL с использованием команды DESCRIBE

## Условия уникальности

Условие UNIQUE задает, что все значения в некотором столбце должны встречаться только по одному разу, — повтор значений не допускается. Все первичные ключи получают это условие автоматически (первичные ключи обязательно должны быть уникальными). Основное различие условий PRIMARY KEY и UNIQUE заключается в том, что UNIQUE допускает пустые значения. Задавать условие UNIQUE можно либо как условие на столбец, либо как условие на таблицу. В качестве примеров столбцов, которые должны иметь условие UNIQUE, можно привести столбцы с номерами телефонов или страховок социального обеспечения (если они не являются первичными ключами). Например, таблица LicenseStatus базы данных Redwood Realty, содержащая номера лицензий и описание состояния лицензии, включает уникальные текстовые значения-характеристики (например, "Licenses" и "Revoked"), описывающие состояние лицензии риэлтора. Чтобы гарантировать, что данные значения уникальны (не дублируются), создавая данную таблицу, можно включить в определение столбца StatusText условие UNIQUE. После этого Oracle проследит за тем, чтобы любое значение, вводимое любым пользователем в столбец описания StatusText, было

единственным в своем роде. Ниже приведен пример задания условия `UNIQUE` в виде ограничения на столбец.

```
CREATE TABLE LicenseStatus
  (LicenseStatusID INTEGER PRIMARY KEY,
   StatusText NVARCHAR2(25) UNIQUE);
```

Рассмотрим теперь задание условия `UNIQUE` в виде ограничения на таблицу.

```
CREATE TABLE MovieCollection
  (MovieID INTEGER PRIMARY KEY,
   DirectorName VARCHAR2(30) NOT NULL,
   CONSTRAINT un_directorname_moviecoll UNIQUE(DirectorName));
```

Одно из основных отличий условия на столбец от условия на таблицу заключается в том, что только один столбец таблицы может иметь ограничение `UNIQUE`, заданное в виде условия на столбец. Если же вы используете задание условий на таблицу, ограничение `UNIQUE` можно будет задать для нескольких столбцов.

## Условия проверки

Условие `CHECK` инициирует применение к столбцу логического выражения, результат которого должен быть равен `true` для всех значений искомого столбца. Данное условие является потрясающим средством внедрения правил, ограничивающих возможные значения столбца. Допустим, например, что в базу данных Redwood Realty необходимо ввести следующее правило: ставка комиссионных может изменяться от 1 до 6 процентов — ни больше и ни меньше. Точно так же можно указать, что столбец `Gender` может содержать только `M` или `F` (прописными буквами). Задавая условие `CHECK`, будьте очень осторожны: после того как оно определено и таблица заполнена, изменить данное условие очень сложно. Условие `CHECK` должно быть таким, чтобы результат его применения был равен `true` или `false`. Для записи условий, которые Oracle проверяет для каждого нового значения, вводимого в столбец с `CHECK`, вы можете использовать операторы отношений, логические операции и операции над множествами (описаны ниже). Синтаксис данной команды выглядит следующим образом:

```
CONSTRAINT <constraint-name> CHECK (<logical expression>)
```

Например, чтобы ограничить значения столбца `Title` таблицы `Agents` записями “`Broker`” или “`Salesperson`”, вы можете включить в команду `CREATE TABLE` следующее условие `CHECK`:

```
CREATE TABLE Agents
  ...
  HomePhone NVARCHAR2(20),
  Title NVARCHAR2(20) CONSTRAINT ck_agents_title
    CHECK ((Title = 'Broker') OR (Title = 'Salesperson')),
  ...
```

Писать правила, ограничивающие значения заданным диапазоном, достаточно просто. Например, указанная ниже команда позволяет величине `ComRate` изменяться в диапазоне от 1 до 6 процентов включительно. В данном примере мы не присваивали

**ТАБЛИЦА 3.7.** Имена столбцов, типы данных и условия таблицы ContactReason

Имя столбца	Тип данных	Длина	Условие
ContactReason	NVARCHAR2	15	PRIMARY KEY
Description	NVARCHAR2	50	

имя условию, поэтому Oracle сгенерирует внутреннее имя — уникальное и непонятное для нас.

```
...
    ComRate NUMBER(4, 4) CHECK (ComRate >= 0.01 AND ComRate < 0.06),
...

```

Если вы позже попытаетесь ввести в столбец ComRate значение, равное, например, 0.08, Oracle выдаст сообщение об ошибке:

```
ORA-02290: check constraint(<yourschema>.SYS_<checkname>) violated
```

В данном выражении вместо <yourschema> будет стоять ваше имя пользователя, а вместо <checkname> — сгенерированное Oracle имя условия, соотнесенное с ограничением CHECK на значение ComRate.

## Создание условий на столбцы и таблицы

Ниже рассказывается, как создать условия на столбцы и таблицы, используя интерфейсные инструменты базы данных Oracle SQL\*Plus и iSQL\*Plus. Если вы по каким-то причинам предпочитаете использовать одно из этих средств — вы можете это делать. В книге оба интерфейса рассматриваются попеременно только для того, чтобы проиллюстрировать как можно больше концепций. Итак, вначале вам нужно запустить программу SQL\*Plus и войти в Oracle. После этого требуется создать несколько таблиц Redwood Realty и соотнести различные условия с выбранными полями. Прежде всего вы создадите таблицу ContactReason. После заполнения значениями строки этой таблицы будут содержать текстовую фразу, указывающую причину контакта с брокером (покупка, продажа, случайно), и первичный ключ для каждой строки. На будущее запомните, что при создании таблиц, содержащих внешние ключи, ссылающиеся на другие таблицы, вначале рекомендуется создавать таблицы, на которые идет ссылка и которые не содержат внешних ключей. Иногда это не возможно. В таком случае нужно либо откладывать определение внешнего ключа на будущее, либо изменять таблицы после их создания. Подробнее обе темы (отсрочка условий и изменение таблиц) рассмотрены ниже.

Подробная информация о таблице ContactReason, которую вам требуется создать (имена столбцов, характеристики столбцов и условия), приведена в табл. 3.7.

Для того чтобы создать таблицу ContactReason, выполните следующие действия.

1. Если необходимо, запустите SQL\*Plus и войдите в базу данных.

The screenshot shows the Oracle SQL\*Plus interface. In the command window, the following SQL code is entered:

```
SOL> CREATE TABLE ContactReason
  2  (ContactReason NVARCHAR2(15),
  3   Description NVARCHAR2(50),
  4   CONSTRAINT pk_ContactReason PRIMARY KEY (ContactReason)
  5 );
Table created.
```

Below the code, the command `DESCRIBE ContactReason` is run, displaying the table structure:

Name	Null?	Type
CONTACTREASON	NOT NULL	NVARCHAR2(15)
DESCRIPTION		NVARCHAR2(50)

A callout bubble points to the right side of the primary key constraint line in the code with the text "Введите указанную команду SQL".

Рис. 3.11. Определение первичного ключа

2. В SQL\*Plus введите команды, показанные на рис. 3.11 (номера строк не вводите — SQL\*Plus добавит их автоматически). Не забывайте нажимать клавишу <Enter>, чтобы перейти на следующую строку.
3. Ведите ; (точка с запятой) в последней строке, а затем нажмите клавишу <Enter>, чтобы выполнить команду CREATE TABLE.

**Совет.** Если Oracle генерирует сообщение об ошибке, вы можете выделить команду CREATE TABLE полностью (без номеров строк, запроса командной строки SQL и сообщений Oracle), выбрать Edit, затем Copy, открыть программу Блокнот и выбрать из меню Edit (Правка) команду Paste (Вставить), скопировав текст в Блокнот. Исправьте опечатки, а затем повторите процесс в обратном порядке: скопируйте текст из Блокнота в буфер, переключитесь на SQL\*Plus и вставьте исправленный код. Нажмите клавишу <Enter>, чтобы выполнить этот код.

4. Ведите `DESCRIBE ContactReason` и нажмите клавишу <Enter>, чтобы отобразить на экране структуру новой таблицы (рис. 3.11). Обратите внимание на то, что столбец `ContactReason` содержит условие NOT NULL, привязанное автоматически, поскольку этот столбец является первичным ключом.

Выполняя описанные выше действия, вы определили и назвали первичный ключ в виде условия на таблицу (поскольку он задается после определения обоих столбцов). Столбцы и условия таблицы Agents перечислены в табл. 3.8. Внимательно ознакомьтесь с ней, чтобы лучше понять описанные ниже действия.

Для того чтобы создать таблицу Agents, выполните следующие действия.

1. Ведите `CLEAR SCREEN` и нажмите клавишу <Enter>, чтобы очистить экран.
2. Ведите и выполните команду CREATE TABLE, показанную на рис. 3.12. Если потребуется — исправьте опечатки в Блокноте и скопируйте поправленный код в SQL\*Plus.

**ТАБЛИЦА 3.8.** Структура таблицы Agents

Имя столбца	Тип данных	Длина	Условие
AgentID	INTEGER		PRIMARY KEY
FirstName	NVARCHAR2	30	NOT NULL
LastName	NVARCHAR2	30	NOT NULL
HireDate	DATE		
BirthDate	DATE		
Gender	NVARCHAR2	10	Единственные допустимые значения — “M” и “F”
WorkPhone	NVARCHAR2	20	
CellPhone	NVARCHAR2	20	UNIQUE
HomePhone	NVARCHAR2	20	
Title	NVARCHAR2	20	
TaxID	NVARCHAR2	20	
LicenseID	NVARCHAR2	20	
LicenseDate	DATE		
LicenseExpire	DATE		
LicenseStatusID	INTEGER		

3. Введите DESCRIBE Agents и нажмите клавишу <Enter>, чтобы перечислить столбцы и условия NOT NULL.

Возможно, вы отметили, что при выполнении команды DESCRIBE отображается только условие NOT NULL. Как же проверить, что остальные условия действительно наложены на таблицу? Читайте дальше!

## Присвоение столбцам значений по умолчанию

Одной из крайне полезных особенностей SQL является возможность задания для столбца значений по умолчанию — величин, которые вводятся в столбец таблицы, если пользователь не заполнил какое-то поле. Чтобы указать, что столбец имеет значение по умолчанию, в команду CREATE TABLE добавляется следующий код:

```
<column-name> <column-type> DEFAULT <expression>
```

Значение по умолчанию, выше обозначенное как <expression>, может быть константой, выражением с математическими действиями и использованием большинства функций SQL (за некоторыми исключениями), а также множеством других объектов. В простейшем случае значение по умолчанию является константой.

В описанном ниже примере вы создадите таблицу Listings, в которой перечисляется вся продаваемая недвижимость, и наложите условие, задающее первичный ключ (Primary Key — PK), условие NOT NULL и два значения по умолчанию. Благо-

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE Agents
  2  (AgentID INTEGER CONSTRAINT pk_agents PRIMARY KEY,
  3   FirstName NVARCHAR2(30) CONSTRAINT nn_agents_fname NOT NULL,
  4   LastName NVARCHAR2(30) CONSTRAINT nn_agents_lname NOT NULL,
  5   HireDate DATE,
  6   BirthDate DATE,
  7   Gender NVARCHAR2(10) CONSTRAINT ck_agents_gender
  8           CHECK ((Gender = 'M') OR (Gender = 'F')),
  9   WorkPhone NVARCHAR2(20),
 10  CellPhone NVARCHAR2(20) CONSTRAINT un_agents_cellphone UNIQUE,
 11  HomePhone NVARCHAR2(20),
 12  Title NVARCHAR2(20),
 13  TaxID NVARCHAR2(20),
 14  LicenseID NVARCHAR2(20),
 15  LicenseDate DATE,
 16  LicenseExpire DATE,
 17  LicenseStatusID INTEGER
 18 );
Table created.

SQL>

```

**Рис. 3.12.** Создание таблицы Agents с условиями

даря этим значениям пользователь, вводящий данные с помощью операции INSERT, может пропустить ввод двух значений. Если во время ввода пользователь не укажет явно значение поля BeginListDate,строенная функция Oracle SYSDATE, автоматически подставляющая текущую дату и время, автоматически заполнит это поле. Кроме того, если значение EndListDate не будет указано явно, вместо него подставится значение “текущая дата плюс 180 дней”. Таким образом, оба указанных столбца будут иметь значение в любом случае — либо их укажет пользователь, либо они будут сгенерированы автоматически.

Для того чтобы создать таблицу Listings, дополнить ее условиями и значениями по умолчанию, выполните следующие действия.

1. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
2. Введите команду CREATE TABLE, показанную на рис. 3.13. Для набора выражения SQL можете использовать Блокнот, а затем скопировать готовый текст в SQL\*Plus.
3. Выполните команду CREATE TABLE, нажав клавишу <Enter> после завершающей точки с запятой. Oracle ответит строкой “Table created”. Исправьте опечатки, если они есть, и выполните правильную команду. Okno SQL\*Plus не закрывайте.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE Listings
  2  (ListingID      INTEGER CONSTRAINT pk_listings PRIMARY KEY,
  3  PropertyID     INTEGER CONSTRAINT nn_listings_propertyid NOT NULL,
  4  ListingAgentID INTEGER NOT NULL,
  5  SaleStatusID   INTEGER,
  6  BeginListDate  DATE    DEFAULT SYSDATE,
  7  EndListDate    DATE    DEFAULT SYSDATE + 180,
  8  AskingPrice    NUMERIC(9)
  9  );
Table created.

SQL>

```

Рис. 3.13. Задание значений по умолчанию для выбранных столбцов таблицы

Обратите внимание на то, что мы не задали имя для условия NOT NULL в столбце ListingAgentID. В подобных случаях Oracle автоматически генерирует уникальное имя.

## Изменение таблицы и ее условий

После создания таблицы иногда требуется изменить какой-то ее атрибут. Для внесения этих изменений применяется команда ALTER TABLE с уточнением DROP, MODIFY, ADD или RENAME. В число допустимых преобразований входят добавление или удаление столбцов, изменение типа данных или длины столбца, а также изменение значений по умолчанию, заданных для столбцов. Помимо этого, вы можете удалять, добавлять, включать, отключать и переименовывать условия таблицы. Команда ALTER TABLE позволяет избежать неприятных проблем, связанных с созданием таблиц в определенном порядке, чтобы не нарушать целостности на уровне ссылок (внешних ключей). Для иллюстрации сказанного на рис. 3.14 показаны две таблицы, ссылающиеся друг на друга посредством внешних ключей. Проблема заключается в том, что непонятно, какую из таблиц создавать первой. Вы не можете создать таблицу A, поскольку она содержит ссылку через внешний ключ на таблицу B, которая еще не создана. По аналогичной причине нельзя создать таблицу B. Проблема решается с помощью команды ALTER TABLE — вначале вы создаете обе таблицы, а затем указываете внешние ключи.

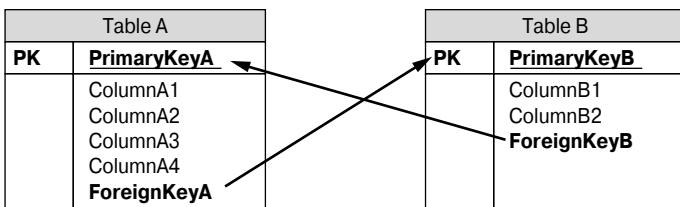


Рис. 3.14. Таблицы, перекрестно ссылающиеся друг на друга посредством внешних ключей

## Добавление, включение или выключение условий

Любое из условий, включенных в команду CREATE TABLE, можно опустить и добавить в таблицу позже с помощью команды ALTER TABLE ... ADD. Следует помнить, что условие активизируется автоматически в момент создания. Если вы желаете создать условие, но не активизировать его сразу, используйте команду ALTER TABLE ADD CONSTRAINT ... DISABLE, которая налагает условие и тут же его отключает. *Отключенным* называется условие, получившее имя, но не введенное в действие. Отключить существующее условие можно и с помощью команды ALTER TABLE ... DISABLE CONSTRAINT. Позже любое условие можно включить, выполнив команду ALTER TABLE ... ENABLE CONSTRAINT. Естественно, чтобы включить или отключить условие, необходимо знать его имя. Допустим, при создании условия вам нужно временно его отключить. Для этого применяется команда ALTER TABLE с указанием имени отключаемого условия. После этого добавляется опция DISABLE CONSTRAINT. Например, если для поля **PropertyID** в таблице **Listings** вы создали условие на внешний ключ, то не сможете вводить данные в эту таблицу, если поля первичного ключа, на которое ссылается указанный внешний ключ, еще нет или если оно не содержит данных. Чтобы решить эту проблему, условие на внешний ключ поля **PropertyID** можно отключить, заполнить таблицу **Listings**, затем таблицу **LicenseStatus**, на которую указывает внешний ключ, и активизировать условие на внешний ключ. Благодаря этому вы не нарушаете условий целостности на уровне ссылок при заполнении таблицы **Listings**. Ниже приведена команда, отключающая условие на внешний ключ (поле **PropertyID** таблицы **Listings**) до тех пор, пока вы явно не активизируете его.

```
ALTER TABLE Listings DISABLE CONSTRAINT fk_propertieslistings
```

Ниже показаны общие формы команд ALTER TABLE, позволяющие добавлять новое условие, активизировать или деактивизировать существующее условие. Обратите внимание на то, что условие NOT NULL требует дополнения MODIFY CONSTRAINT.

```
ALTER TABLE <table-name>
    ADD CONSTRAINT <constraint-name> <constraint-type>
ALTER TABLE <table-name>
    MODIFY CONSTRAINT <constraint-name> NOT NULL
```

```
ALTER TABLE <table-name>
    ENABLE CONSTRAINT <constraint-name>
ALTER TABLE <table-name>
    DISABLE CONSTRAINT <constraint-name>
```

Создавая таблицу *Listings*, вы не задали внешние ключи на другие таблицы, связанные с данной. На текущий момент в базу данных уже помещена одна из этих таблиц — *Agents*. Следовательно, можно вспомнить, что поле *ListingAgentID* таблицы *Listings* является внешним ключом, указывающим на столбец первичного ключа *AgentID* в таблице *Agents*. Зададим это условие.

Итак, чтобы добавить в таблицу *Listings* условие на внешний ключ, выполните следующие действия.

1. Введите следующий код и нажмите клавишу <Enter> после точки с запятой:

```
ALTER TABLE Listings
    ADD CONSTRAINT fk_agents_listings FOREIGN KEY (ListingAgentID)
        REFERENCES Agents (AgentID)
        ON DELETE CASCADE;
```

Oracle ответит подтверждением “Table altered”.

2. Создавая таблицу *Customers*, вы не указали столбец, являющийся первичным ключом. Чтобы исправить это упущение, необходимо задействовать команду *ALTER TABLE*. Кроме того, к столбцам *FirstName* и *LastName* следует добавить условие *NOT NULL*, чтобы соответствующие значения нельзя было опустить (для добавления условий *NOT NULL* к существующим столбцам таблицы применяется команда *MODIFY*).

Для того чтобы добавить условие, задающее первичный ключ таблицы *Customers*, выполните следующие действия.

1. Выполните следующую команду (не забудьте нажать клавишу <Enter> в конце второй строки):

```
ALTER TABLE Customers
    ADD CONSTRAINT pk_customers PRIMARY KEY(CustomerID);
```

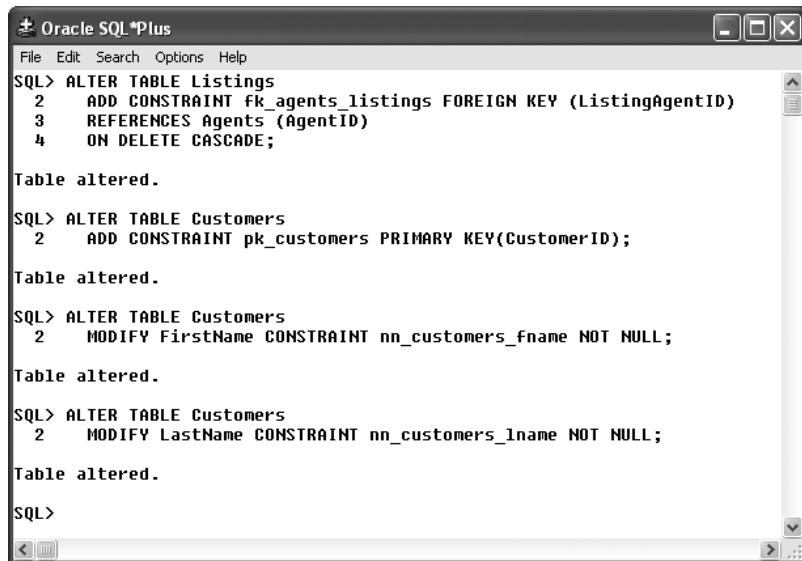
Oracle ответит строкой-подтверждением “Table altered”.

2. Чтобы добавить условие *NOT NULL* к столбцу *FirstName*, выполните следующую команду:

```
ALTER TABLE Customers
    MODIFY FirstName CONSTRAINT nn_customers_fname NOT NULL;
```

3. Чтобы добавить условие *NOT NULL* к столбцу *LastName*, выполните следующую команду:

```
ALTER TABLE Customers
    MODIFY LastName CONSTRAINT nn_customers_lname NOT NULL;
```



The screenshot shows a window titled "Oracle SQL\*Plus". Inside, four SQL commands are entered and executed:

```

SQL> ALTER TABLE Listings
  2  ADD CONSTRAINT fk_agents_listings FOREIGN KEY (ListingAgentID)
  3  REFERENCES Agents (AgentID)
  4  ON DELETE CASCADE;

Table altered.

SQL> ALTER TABLE Customers
  2  ADD CONSTRAINT pk_customers PRIMARY KEY(CustomerID);

Table altered.

SQL> ALTER TABLE Customers
  2  MODIFY FirstName CONSTRAINT nn_customers_fname NOT NULL;

Table altered.

SQL> ALTER TABLE Customers
  2  MODIFY LastName CONSTRAINT nn_customers_lname NOT NULL;

Table altered.

SQL>

```

**Рис. 3.15.** Добавление условия с помощью команды ALTER TABLE

Четыре использованных команды ALTER TABLE и реакция Oracle на них показаны на рис. 3.15.

## Удаление или переименование условий

Если условие базы данных больше не требуется, его можно удалить, использовав команду ALTER TABLE с уточнением DROP. Чтобы удалить условие, можно явно указать его имя или не упоминать его вообще. Например, условие NOT NULL на столбец LastName таблицы Customers можно отбросить с помощью команды

```
ALTER TABLE Customers DROP CONSTRAINT nn_customers_lname;
```

Подобным образом вы можете удалить первичный ключ таблицы Listings, введя любую из нижеприведенных команд.

```
ALTER TABLE Listings DROP PRIMARY KEY;
```

или

```
ALTER TABLE Listings DROP CONSTRAINT pk_listings;
```

Если для условий вы выбрали неудачные имена, можете изменить их, использовав все ту же команду ALTER TABLE. Ниже показано, как переименовать внешний ключ (столбец ListingAgentID) с fk\_agents\_listings на fk\_agents\_list.

```
ALTER TABLE Listings RENAME CONSTRAINT fk_agents_listings
  TO fk_agents_list;
```

Как правило, сгенерированные системой (бессмысленные) имена условий удобно изменять на более значимые. В следующем разделе вы узнаете, как отобразить на экране имена условий, и сможете посмотреть, какие имена генерирует система Oracle. Потренируйтесь переименовывать условия на более удачные варианты. (*Подсказка.* Одним из “кандидатов” на переименование является условие NOT NULL в таблице Listings.)

## Изменение заданного значения по умолчанию или типа данных

В числе прочих изменений, которые можно внести после выполнения команды CREATE TABLE, допускается и модификация *значения по умолчанию* с помощью команды ALTER TABLE. Например, чтобы изменить значение по умолчанию, заданное для столбца BeginListDate таблицы Listings, необходимо ввести команду.

```
ALTER TABLE Listings MODIFY BeginListDate DEFAULT SYSDATE -2;
```

Если вам требуется удалить значение по умолчанию, переменной DEFAULT следует присвоить значение NULL:

```
ALTER TABLE Listings MODIFY BeginListDate DEFAULT NULL;
```

Выполняя указанную команду, вы удаляете значение по умолчанию: в данном случае Oracle больше не будет самостоятельно вводить значение столбца BeginListDate, если пользователь опустит его в процессе ввода данных.

Как вы уже, наверное, поняли, после создания таблицы с помощью команды CREATE TABLE вы можете изменить практически все. Ниже приведено еще несколько характеристик столбца, которые можно модифицировать посредством выражения ALTER TABLE.

- Тип данных.
- Значение по умолчанию.
- Размер столбца для типа данных, предполагающего явное задание размера.
- Точность числового столбца.

Например, приведенный ниже код изменяет точность столбца AskingPrice с первоначального значения NUMBER(9) на NUMBER(12, 2), разрешая использовать на одну значащую цифру больше и две цифры после десятичной запятой:

```
ALTER TABLE Listings MODIFY AskingPrice NUMBER(12, 2);
```

Поскольку описанные в данной главе команды модификации таблиц (CREATE TABLE и ALTER TABLE) относятся к выражениям языка определения данных (Data Definition Language — DDL), результат их выполнения вступает в силу сразу же и его нельзя отменить с помощью команды ROLLBACK. Если вы все же намерены отменить команду изменения таблицы, необходимо ввести команду ALTER TABLE, явно обращающую изменения.

The screenshot shows the Oracle SQL\*Plus interface. The command `ALTER TABLE Customers ADD ContactDate DATE DEFAULT SYSDATE;` is entered and executed, resulting in the message `Table altered.`. A DESCRIBE command is then run, displaying the table structure with the new column added at the end:

Name	Null?	Type
CUSTOMERID	NOT NULL	NUMBER(38)
FIRSTNAME	NOT NULL	NUARCHAR2(30)
LASTNAME	NOT NULL	NUARCHAR2(30)
ADDRESS		NUARCHAR2(40)
CITY		NUARCHAR2(30)
STATE		NUARCHAR2(20)
ZIPCODE		NUARCHAR2(20)
HOMEPHONE		NUARCHAR2(20)
CELLPHONE		NUARCHAR2(20)
WORKPHONE		NUARCHAR2(20)
<b>CONTACTDATE</b>		<b>DATE</b>

Добавленный столбец *ContactDate*

Рис. 3.16. Добавление столбца к таблице

## Добавление, удаление и переименование столбцов

Одной из прекрасных особенностей реляционной базы данных является возможность быстрого добавления или удаления столбцов, когда это требуется. Точно так же легко можно переименовывать столбцы. Ниже мы добавим в таблицу *Customers* столбец *ContactDate* (как отмечалось выше, новый столбец становится в конец таблицы). Столбец *ContactDate* имеет тип данных DATE, и в нем содержится дата первого контакта с клиентом; по умолчанию — это текущая дата, поставляемая функцией SYSDATE.

Для того чтобы добавить столбец в таблицу, выполните следующие действия.

1. В открытом окне SQL\*Plus введите следующий код и нажмите клавишу <Enter>:
 

```
ALTER TABLE Customers
      ADD ContactDate DATE DEFAULT SYSDATE;
```
2. Введите DESCRIBE *Customers* и нажмите клавишу <Enter>, чтобы проверить, что база данных Oracle добавила столбец (рис. 3.16).

Процедура удаления столбца также проста. Тем не менее здесь нужно быть очень внимательным: удалив столбец непустой таблицы, вы не сможете никак его восстановить. Единственная возможность отмены этой операции: заново добавить столбец и вручную восстановить данные в нем (с помощью команды INSERT), но это слишком

The screenshot shows the Oracle SQL\*Plus interface. In the command window, the following commands were entered:

```

SQL> ALTER TABLE Customers
2   DROP COLUMN HomePhone;

Table altered.

SQL> DESCRIBE Customers
Name          Null?    Type
-----        -----
CUSTOMERID    NOT NULL NUMBER(38)
FIRSTNAME     NOT NULL NVARCHAR2(30)
LASTNAME      NOT NULL NVARCHAR2(30)
ADDRESS        NVARCHAR2(40)
CITY          NVARCHAR2(30)
STATE         NVARCHAR2(20)
ZIPCODE       NVARCHAR2(20)
CELLPHONE     NVARCHAR2(20)
WORKPHONE    NVARCHAR2(20)
CONTACTDATE   DATE

```

Рис. 3.17. Удаление столбца из таблицы

долго и неудобно. Впрочем, поскольку ни одна из таблиц базы данных Redwood Realty еще не заполнена данными, вы можете без последствий потренироваться в удалении столбцов.

Для того чтобы удалить столбец из таблицы, выполните следующие действия.

1. Введите следующую команду, чтобы удалить столбец HomePhone (не забудьте нажать клавишу <Enter>, чтобы выполнить ее):

```
ALTER TABLE Customers
DROP COLUMN HomePhone;
```

2. Чтобы отобразить на экране модифицированную структуру таблицы, выполните команду DESCRIBE Customers (рис. 3.17).

Если вам требуется одновременно удалить несколько столбцов, опустите ключевое слово “COLUMN” и перечислите через запятую имена столбцов:

```
ALTER TABLE Customers
DROP (BirthDate, Gender, WorkPhone);
```

Используя команду ALTER TABLE с уточнением RENAME COLUMN, вы можете изменить имя столбца.

**Предупреждение.** Если вы переименовываете столбец, все объекты, ссылающиеся на него, становятся недействительными.

Предположим далее, что вам требуется переименовать столбец Title таблицы Agents на AgentTitle. Для этого вам потребуется следующее выражение ALTER TABLE:

```
ALTER TABLE Agents RENAME COLUMN Title TO AgentTitle;
```

## Отметка столбцов как неиспользуемых и удаление их

Вместо того чтобы удалять нежелательные, неиспользуемые или ненужные столбцы, вы можете пометить их как неиспользуемые. В чем разница между удалением и присвоением статуса неиспользуемого? Неиспользуемый столбец просто не виден пользователям Oracle. Команда DESCRIBE не открывает такие столбцы, кроме того, к ним невозможно формировать запросы. По сути, это столбцы-невидимки. Их альтернативой является физическое удаление столбцов, когда отбрасываются все содержащиеся в них данные. Если удалять большие таблицы, то это может существенно сказаться на производительности, поскольку стирание данных и перезапись строк таблицы может потребовать значительного времени. Это не очень удобно в середине рабочего дня, когда с базой данных Oracle работает множество пользователей. Присвоение столбцу статуса неиспользуемого требует гораздо меньше времени. Пометить столбец как неиспользуемый гораздо быстрее, кроме того, при этом существует возможность отмены данного действия, тогда как команда удаления необратима. Упрощенная форма команды присвоения столбца статуса неиспользуемого приведена ниже. Вторая команда позволяет пометить сразу несколько столбцов ( обратите внимание на отсутствие в ней слова COLUMN).

```
ALTER TABLE <table-name> SET UNUSED COLUMN <column-name>;  
ALTER TABLE <table-name> SET UNUSED (<col-name1>, ... ,<col-namen>)
```

Позже вы можете выполнить команду ALTER TABLE с уточнением DROP COLUMN или DROP UNUSED COLUMNS, чтобы физически удалить столбец (столбцы) из базы данных. В использовании этой команды полезно попрактиковаться. Для этого мы “скроем” столбцы LicenseID, LicenseDate и LicenseExpire таблицы Agents.

Для того чтобы пометить столбцы как неиспользуемые, выполните следующие действия.

1. Если необходимо, запустите SQL\*Plus и войдите в Oracle.
2. Введите DESCRIBE Agents, чтобы посмотреть, какие столбцы содержит указанная таблица. Ближе к концу перечня вы увидите названия LicenseID, LicenseDate и LicenseExpire.
3. Введите следующую команду ALTER TABLE и нажмите клавишу <Enter>, чтобы выполнить ее:

```
ALTER TABLE Agents  
SET UNUSED (LicenseID, LicenseDate, LicenseExpire);
```

Столбцы, которые будут помечены как неиспользуемые

```

SQL> DESCRIBE Agents
Name          Null?    Type
-----        -----
AGENTID      NOT NULL NUMBER(38)
FIRSTNAME    NOT NULL NVARCHAR2(30)
LASTNAME     NOT NULL NVARCHAR2(30)
HIREDATE     DATE
BIRTHDATE    DATE
GENDER       NVARCHAR2(10)
WORKPHONE   NVARCHAR2(20)
CELLPHONE   NVARCHAR2(20)
HOMEPHONE   NVARCHAR2(20)
TITLE        NVARCHAR2(20)
TAXID        NVARCHAR2(20)
LICENSEID    NVARCHAR2(20)
LICENSEDATE  DATE
LICENSEEXPIRE DATE
LICENSESTATUSID NUMBER(38)

SQL> ALTER TABLE Agents
2  SET UNUSED (LicenseID, LicenseDate, LicenseExpire);
Table altered.

SQL> DESCRIBE Agents
Name          Null?    Type
-----        -----
AGENTID      NOT NULL NUMBER(38)
FIRSTNAME    NOT NULL NVARCHAR2(30)
LASTNAME     NOT NULL NVARCHAR2(30)
HIREDATE     DATE
BIRTHDATE    DATE
GENDER       NVARCHAR2(10)
WORKPHONE   NVARCHAR2(20)
CELLPHONE   NVARCHAR2(20)
HOMEPHONE   NVARCHAR2(20)
TITLE        NVARCHAR2(20)
TAXID        NVARCHAR2(20)
LICENSESTATUSID NUMBER(38)

SQL>

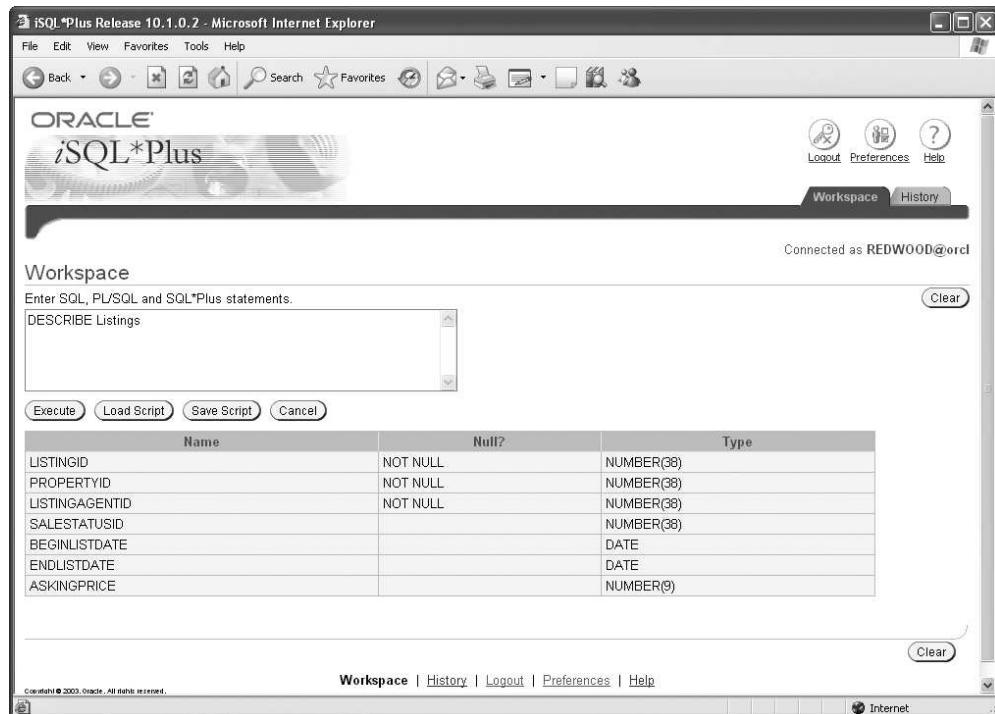
```

Рис. 3.18. Присвоение некоторым столбцам статуса неиспользуемых

- Снова введите DESCRIBE Agents (и нажмите клавишу <Enter>), чтобы удостовериться, что три указанных столбца больше не отображаются (рис. 3.18).
- Ведите Exit и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus (далее до конца главы вы будете использовать iSQL\*Plus).

## Отображение имен, структур

Добавляя информацию в таблицу или изменения структуру этой таблицы, вы можете просмотреть имена и типы данных столбцов, выполнив команду DESCRIBE <tablename>. При выполнении этой команды отображается единственное условие — NOT NULL. В частности, не показывается, какие столбцы являются первичными и внешними ключами, только если оба ключа не дополнены условием NOT NULL. Например, на рис. 3.19 показан результат выполнения команды DESCRIBE Listings. Обратите внимание на то, что для первых трех столбцов указывается условие “NOT NULL”, но не указывается, что ListingID — это первичный, а ListingAgentID — внешний ключ.



**Рис. 3.19.** Описание таблицы Listings

Чтобы просмотреть информацию об условиях, наложенных на таблицу, и именах созданных таблиц, необходимо отправить запрос к таблицам словаря данных Oracle. Словарь данных содержит таблицы с информацией о собственных таблицах и всех объектах базы данных Oracle. Словарь данных формируется всякий раз, когда АБД создает новую базу данных. Любые изменения таблиц пользователя или других объектов отражаются в словаре данных. Например, в таблице словаря данных хранятся имена столбцов для таблицы некоторого пользователя плюс информация по типу данных. Команда DESCRIBE просто запрашивает данную системную “таблицу с информацией о таблицах”, чтобы извлечь требуемые имена столбцов и типы данных.

Словарь данных принадлежит привилегированной учетной записи Oracle SYS. Как правило, пользователи и администраторы базы данных не выполняют вставку, обновление или удаление записей в словаре данных, поскольку это может нарушить базу данных. Вместо того чтобы напрямую взаимодействовать с таблицами, пользователи могут просматривать информацию словаря данных через *представления базы данных* (database views). Представление – это объект, отображающий подмножество строк и столбцов таблицы базы данных. Хотя представления выглядят как обычные (базовые) таблицы, они ограничивают то, что пользователь может извлечь из базовых

таблиц. Посредством представлений информация извлекается точно так же, как и при использовании таблиц, — с помощью команды `SELECT`. Поскольку представления словаря данных определяют, какие строки и столбцы отображаются в представлении, они предлагают широкий контроль за тем, кто может их использовать и что отображается с их помощью, т.е. реализуют разновидность защиты “принцип необходимого знания” (“need to know”), недоступной для базовых таблиц. Представления словаря данных можно разбить на три группы: `DBA`, `USER` и `ALL`. Представления `DBA` позволяют пользователям с правами администраторского уровня видеть всю информацию об объектах (таблицах, представлениях и т.д.), принадлежащих текущему пользователю, — в схеме этого пользователя. Представления группы `ALL` позволяют любому пользователю обращаться к объектам в схеме, а также к объектам, доступ к которым разрешен для данного пользователя, — обычно это объекты других пользователей. (Хотя подробно команду `SELECT` вы изучите в главе 5, а создание собственных представлений — в главе 6, ниже вы немного попрактикуетесь использовать команду `SELECT` для просмотра своих объектов в базе данных.)

## Просмотр собственных таблиц базы данных

Упрощенный синтаксис команды, предназначеннной для извлечения информации об объектах, которые вы создали или к которым имеете доступ, выглядит следующим образом:

```
SELECT view_columnname1, ..., view_columnnamek
  FROM groupprefix_objectname;
```

где `view_columnname` — имена столбцов, которые требуется извлечь, а `groupprefix` — `USER`, `DBA` или `ALL`; `objectname` — тип объектов, которые требуется исследовать (в частности, это может быть `TABLES` или `CONSTRAINTS`). Таким образом, имена таблиц можно извлечь с помощью команды

```
SELECT table_name
  FROM user_tables;
```

где `table_name` — один из столбцов, хранимых в представлении словаря данных `user_tables` и совпадающий с именем принадлежащей вам таблицы. Приведенное представление ограничивает извлекаемые имена таблиц только таблицами, созданными в вашей схеме (т.е. таблицы, к которым база данных Oracle предоставила вам разрешение на чтение, не считаются). Например, приведенная ниже команда `SELECT` отображает имена всех представлений, которые вы создали или право на доступ к которым вы имеете (несколько тысяч объектов).

```
SELECT view_name FROM all_views;
```

Лучший способ понять команду — попрактиковаться в ее использовании. Для этого вы можете задействовать *iSQL\*Plus* или *SQL\*Plus* — как вам будет удобнее.

Для того чтобы отобразить на экране имена своих таблиц, выполните следующие действия.

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT table_name
  2  FROM user_tables;

TABLE_NAME
-----
CONTACTREASON
BIN$ewBwUgbiSqmN3UadqCihSA==$0
BIN$Pc7YyijQ9KjFW6hKpta4w==$0
CUSTOMERS
LISTINGS
AGENTS

6 rows selected.

SQL> SELECT table_name
  2  FROM user_tables
  3  WHERE table_name NOT LIKE '%BIN%';

TABLE_NAME
-----
AGENTS
CONTACTREASON
CUSTOMERS
LISTINGS

```

} Имена созданных таблиц

**Рис. 3.20.** Отображение имен таблиц с использованием представления словаря данных

1. Запустите SQL\*Plus и войдите в Oracle, используя свое имя пользователя и пароль.
2. Введите следующую команду, нажимая клавишу <Enter> в конце каждой строки:

```
SELECT table_name
  FROM user_tables;
```

Oracle отобразит четыре созданных вами таблицы (имена временных таблиц могут не отображаться).

3. Запустите программу Блокнот, чтобы вы могли отредактировать и сохранить команду. Переключитесь на SQL\*Plus, выделите команду SELECT, включая точку с запятой, а затем нажмите комбинацию клавиш <Ctrl+C>, чтобы скопировать ее в буфер.
4. Переключитесь на Блокнот и нажмите комбинацию клавиш <Ctrl+V>, чтобы вставить команду. Измените ее следующим образом:

```
SELECT table_name
  FROM user_tables
 WHERE table_name NOT LIKE '%BIN%';
```

5. Введите слово “BIN” прописными буквами и поставьте в начале и в конце него знаки процента.
6. Сохраните файл как <имя>Ch3GetTableNames.sql. Скопируйте команду в SQL\*Plus и закройте программу Блокнот.
7. В окне SQL\*Plus нажмите клавишу <Enter>, чтобы выполнить скопированную команду. Oracle вернет имена таблиц (без временных таблиц), как показано на рис. 3.20.

Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)
PCT_FREE		NUMBER
PCT_USED		NUMBER
INI_TRANS		NUMBER
MAX_TRANS		NUMBER
INITIAL_EXTENT		NUMBER
NEXT_EXTENT		NUMBER
MIN_EXTENTS		NUMBER
MAX_EXTENTS		NUMBER
PCT_INCREASE		NUMBER
FREELISTS		NUMBER
FREELIST_GROUPS		NUMBER
LOGGING		VARCHAR2(3)
BACKED_UP		VARCHAR2(1)
NUM_ROWS		NUMBER
BLOCKS		NUMBER
EMPTY_BLOCKS		NUMBER
Avg_Space		NUMBER
CHAIN_CNT		NUMBER
Avg_Row_Len		NUMBER
Avg_Space_FreeList_Blocks		NUMBER
Num_FreeList_Blocks		NUMBER
Degree		VARCHAR2(10)
Instances		VARCHAR2(10)
Cache		VARCHAR2(5)
Table_Lock		VARCHAR2(8)
Sample_Size		NUMBER
Last_Analyzed		DATE
Partitioned		VARCHAR2(3)
Iot_Type		VARCHAR2(12)
Temporary		VARCHAR2(1)
Secondary		VARCHAR2(1)
Nested		VARCHAR2(3)
Buffer_Pool		VARCHAR2(7)
Row_Movement		VARCHAR2(8)
Global_Stats		VARCHAR2(3)
User_Stats		VARCHAR2(3)
Duration		VARCHAR2(15)
skip_Corrupt		VARCHAR2(8)

Рис. 3.21. Некоторые столбцы представления user\_tables

Для того чтобы узнать, какая еще информация содержится в представлении user\_tables, выполните команду DESCRIBE user\_tables. В данном представлении насчитывается 47 столбцов, однако для большинства пользователей интерес представляет лишь несколько из них. Часть столбцов представления показана на рис. 3.21.

Ряд объектов базы данных, из которых вы можете извлекать информацию, приведен в табл. 3.9. Чтобы извлечь эту информацию, вы можете использовать команду SELECT, подобную показанной выше. Часть объектов будет рассмотрена ниже в данной главе.

Все представления словаря данных, приведенные в табл. 3.9, имеют различные столбцы. Если вам интересно, что они содержат, вначале выясните имена столбцов искомого представления, выполнив команду DESCRIBE <viewname>, а затем запишите или запомните нужные столбцы. Будьте осторожны! Некоторые представления содержат очень много строк.

**ТАБЛИЦА 3.9.** Некоторые объекты, доступные через представления словаря данных

Объект	Описание
CONS_COLUMNS	Столбцы таблицы, на которые наложены условия
CONSTRAINTS	Условия, наложенные на таблицу
INDEXES	Индексы таблицы
OBJECTS	Все объекты базы данных
SEQUENCES	Последовательности, генерирующие уникальные ключи
TAB_COLUMNS	Столбцы таблицы (USER_TAB_COLUMNS – столбцы текущего пользователя)
TABLES	Таблицы базы данных
USERS	Имена всех пользователей базы данных (ALL_USERS)
VIEWS	Представления базы данных

## Просмотр информации о столбцах

Более подробную информацию о столбцах ваших таблиц можно получить с помощью представления user\_tab\_columns. (Префикс “user” ограничивает строки – в ответ на запрос возвращаются только те из них, которых принадлежат вашей схеме.) Чтобы получить эту информацию, необходимо ввести команду SELECT, которая обращается к представлению словаря данных user\_tab\_columns. Если потребуется, вы можете просмотреть столбцы этого представления, выполнив команду DESCRIBE user\_tab\_columns. Ниже мы рассмотрим, как извлечь информацию о созданной ранее таблице Listings. Итак, убедитесь, что приложение SQL\*Plus запущено и вы по-прежнему соединены с базой данных Oracle.

Для того чтобы извлечь дополнительную информацию о таблице, выполните следующие действия.

1. Введите нижеприведенный код (как обычно), нажимая клавишу <Enter> в конце каждой строки. Не забудьте набрать имя LISTINGS прописными буквами. Результат выполнения этой команды представлен на рис. 3.22.

```
CLEAR SCREEN
COLUMN column_name FORMAT A15
COLUMN data_type FORMAT A9
COLUMN nullable FORMAT A8
SELECT column_name, data_type, data_length,
       data_precision, data_scale, nullable
  FROM user_tab_columns
 WHERE table_name = 'LISTINGS';
```

2. Введите CLEAR COLUMNS и нажмите клавишу <Enter>, чтобы удалить форматирование столбцов.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> COLUMN column_name FORMAT A15
SQL> COLUMN data_type FORMAT A9
SQL> COLUMN nullable FORMAT A8
SQL> SELECT column_name, data_type, data_length,
2      data_precision, data_scale, nullable
3  FROM user_tab_columns
4 WHERE table_name = 'LISTINGS';

COLUMN_NAME          DATA_TYPE DATA_LENGTH DATA_PRECISION DATA_SCALE NULLABLE
-----              -----        -----           -----           -----
LISTINGID            NUMBER      22             0   N
PROPERTYID           NUMBER      22             0   N
LISTINGAGENTID       NUMBER      22             0   N
SALESTATUSID         NUMBER      22             0   Y
BEGINLISTDATE        DATE        7               0   Y
ENDLISTDATE          DATE        7               0   Y
ASKINGPRICE          NUMBER      22             9   0   Y

7 rows selected.

SQL>

```

**Рис. 3.22.** Отображение информации о столбцах таблицы

**ТАБЛИЦА 3.10.** Некоторые столбцы представления `user_constraints`

Столбец	Описание
owner	Владелец условия (имя пользователя)
constraint_name	Имя условия
constraint_type	Тип: P, R, C, U, V или O
table_name	Таблица, содержащая условие
status	ENABLED либо DISABLED

Обратите внимание на неожиданную длину данных. Для чисел длина данных указывает максимальное число цифр, а в столбце `DATA_PRECISION` указывается, что столбец `AskingPrice` таблицы пользователя имеет максимальную девятизначную длину. Столбец `NULLABLE` указывает, допустимы или нет значения `NULL`.

## Просмотр условий на таблицы и столбцы

Ранее вы добавили к таблице несколько условий. Иногда такие условия требуется просмотреть. В связи с этим следует знать, что Oracle предлагает несколько представлений, предоставляющих информацию об объектах вашей схемы. Одно из этих представлений, именуемое `user_constraints`, содержит информацию об условиях, наложенных на таблицы. Для примера в табл. 3.10 приведено несколько столбцов представления `user_constraints`. Хотя мы еще не очень углублялись в организацию запросов, ниже приведен запрос, позволяющий извлечь информацию об условиях на все ваши таблицы. Подробно запросы рассмотрены в главе 5, а их ближайшие “родственники” — представления — описаны в главе 6.

На первый взгляд, столбец `constraint_type` (см. табл. 3.10) может выглядеть непонятным. Значения, заполняющие этот столбец, определяют тип наложенного

условия: P обозначает первичный ключ (primary key); R — внешний ключ (foreign key); C — условие проверки (check) или “не нуль”; U — уникальное значение (unique); V — опцию проверки определения представления; O — представление только для чтения. Наиболее важными из перечисленных являются коды P, R, C и U.

Ниже показано, как, используя представление `user_constraints`, извлечь информацию об условиях, определенных для ваших таблиц.

Для того чтобы просмотреть условия, наложенные на таблицы, выполните следующие действия.

1. В окне SQL\*Plus введите следующие строки, нажимая клавишу <Enter> в конце каждой из них (не забудьте набрать слово `BIN` прописными буквами и поставить после него знак %):

```
CLEAR SCREEN
SET PAGESIZE 20
COLUMN table_name FORMAT A15
COLUMN constraint_введите FORMAT A10
SELECT table_name, constraint_type, constraint_name, status
FROM user_constraints
WHERE table_name NOT LIKE 'BIN%'
ORDER BY table_name;
```

Oracle отобразит тринацать условий, наложенных на таблицы, отсортировав их по названию таблицы (рис. 3.23).

2. Обновите переменную среды SQL\*Plus `PAGESIZE` и снимите форматирование со столбцов, выполнив следующие команды SQL\*Plus:

```
SET PAGESIZE 14
CLEAR COLUMNS
```

Обратите внимание на условие со странным именем, начинающимся с `SYS` (возможно, ваш вариант названия отличается от показанного на рисунке). Здесь мы снова сталкиваемся с автоматическим присвоением имен, которое выполнила база данных Oracle, когда вы не задали явно название одного из условий на таблицу `Listings`. Как определить, какому условию вы не присвоили имя? Для ответа на этот вопрос необходимо изучить еще одно представление словаря данных, именуемое `user_cons_columns` и содержащее более подробную информацию об условиях, наложенных на столбцы. Тремя важнейшими столбцами данного представления являются `table_name`, `column_name` и `constraint_name`.

Для того чтобы просмотреть условия, наложенные на определенную таблицу, выполните следующие действия.

1. Введите в SQL\*Plus нижеприведенный код, нажимая клавишу <Enter> в конце каждой строки. Не забудьте в последней строке ввести `LISTINGS` прописными буквами, поскольку Oracle хранит имена объектов в верхнем регистре (если вы не возьмете имя в одинарные кавычки).

The screenshot shows the Oracle SQL\*Plus interface with the following command history and output:

```

SQL> SET PAGESIZE 20
SQL> COLUMN table_name FORMAT A15
SQL> COLUMN constraint_type FORMAT A10
SQL> SELECT table_name, constraint_type, constraint_name, status
  2  FROM user_constraints
  3 WHERE table_name NOT LIKE 'BIN%'
  4 ORDER BY table_name;

```

TABLE_NAME	CONSTRAINT	CONSTRAINT_NAME	STATUS
AGENTS	C	CK_AGENTS_GENDER	ENABLED
AGENTS	C	NN_AGENTS_LNAME	ENABLED
AGENTS	C	NN_AGENTS_FNAME	ENABLED
AGENTS	U	UN_AGENTS_CELLPHONE	ENABLED
AGENTS	P	PK_AGENTS	ENABLED
CONTACTREASON	P	PK_CONTACTREASON	ENABLED
CUSTOMERS	C	NN_CUSTOMERS_LNAME	ENABLED
CUSTOMERS	P	PK_CUSTOMERS	ENABLED
CUSTOMERS	C	NN_CUSTOMERS_FNAME	ENABLED
LISTINGS	R	FK_AGENTS_LISTINGS	ENABLED
LISTINGS	P	PK_LISTINGS	ENABLED
LISTINGS	C	SYS_C007154	ENABLED
LISTINGS	C	NN_LISTINGS_PROPERTYID	ENABLED

13 rows selected.

SQL>

Рис. 3.23. Вывод на экран перечня условий

```

CLEAR SCREEN
COLUMN table_name FORMAT A15
COLUMN column_name FORMAT A15
COLUMN constraint_name FORMAT A25
SELECT table_name, column_name, constraint_name
FROM user_cons_columns
WHERE table_name = 'LISTINGS';

```

2. Выдите из SQL\*Plus, набрав exit и нажав клавишу <Enter>.

Выполнив указанные действия, вы должны увидеть то же, что и на рис. 3.24. Обратите внимание на то, что условие с непонятным именем присвоено столбцу ListingAgentD таблицы Listings. Полученных данных недостаточно для полной определенности, но это условие — либо CHECK, либо NOT NULL (на самом деле — последнее).

Если вы хотите собрать воедино информацию из обоих представлений, чтобы получить более полную картину наложения условий на столбцы, то потребуется более сложный запрос, записанный в файле сценария Ch03TableConstraintsISP.sql, запускать который мы пока что не рекомендуем. Отметим, что просмотр столбцов иногда лучше выглядит в iSQL\*Plus, поскольку этот интерфейс хорошо выравнивает столбцы и предотвращает переход текста на новые строки. Исходя из этого в следующем примере рекомендуем использовать iSQL\*Plus.

Для того чтобы просмотреть все условия, наложенные на ваши таблицы и столбцы, выполните следующие действия.

1. Запустите iSQL\*Plus и войдите в базу данных.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> COLUMN table_name FORMAT A15
SQL> COLUMN column_name FORMAT A15
SQL> COLUMN constraint_name FORMAT A25
SQL> SELECT table_name, column_name, constraint_name
  2  FROM user_cons_columns
  3 WHERE table_name = 'LISTINGS';

TABLE_NAME      COLUMN_NAME      CONSTRAINT_NAME
LISTINGS        LISTINGAGENTID  SVS_C006186
LISTINGS        LISTINGID       PK_LISTINGS
LISTINGS        PROPERTYID     NN_LISTINGS_PROPERTYID
LISTINGS        LISTINGAGENTID  FK_AGENTS_LISTINGS

SQL>

```

**Рис. 3.24.** Просмотр условий, наложенных на столбцы таблицы Listings

2. Щелкните на кнопке Load Script, затем на кнопке Browse, расположенной рядом с текстовым окном File, и дойдите до папки, содержащей файл Ch03TableConstraintsISP.sql.
3. Щелкните на названии Ch03TableConstraintsISP.sql в диалоговом окне Choose file, затем щелкните на кнопке Open и Load, чтобы загрузить файл сценария и отобразить его в текстовом окне Workspace.
4. Щелкните на кнопке Execute, чтобы запустить сценарий. В ответ Oracle отобразит всю информацию по условиям, включая имена таблиц и столбцов, а также типы, имена и состояния условий (рис. 3.25).
5. Окно iSQL\*Plus не закрывайте.

Если вы предпочтете использовать SQL\*Plus, запустите альтернативный файл сценария Ch03TableConstraints.sql, который содержит дополнительный код, форматирующий столбцы для лучшего отображения в среде SQL\*Plus.

## Просмотр комментариев к таблицам и столбцам

Наличие комментариев, касающихся таблицы или ее столбцов, может очень помочь, если вы забудете, для чего нужен тот или иной объект. Ранее в этой главе вы добавляли комментарии к таблице Customers и столбцу State этой таблицы. Для просмотра этих комментариев применялась команда SELECT. В Oracle имена таблиц и представлений, а также все комментарии, которые могут в них содержаться, хранятся в представлении словаря данных user\_tab\_comments (при этом отмечаются даже таблицы, вообще не содержащие комментариев). Как можно догадаться из названия, представление all\_tab\_comments содержит имена владельцев, таблиц и комментарии для всех таблиц и представлений базы данных. Данная таблица содержит тысячи строк, поэтому, отправляя запрос, будьте готовы получить массу информации.

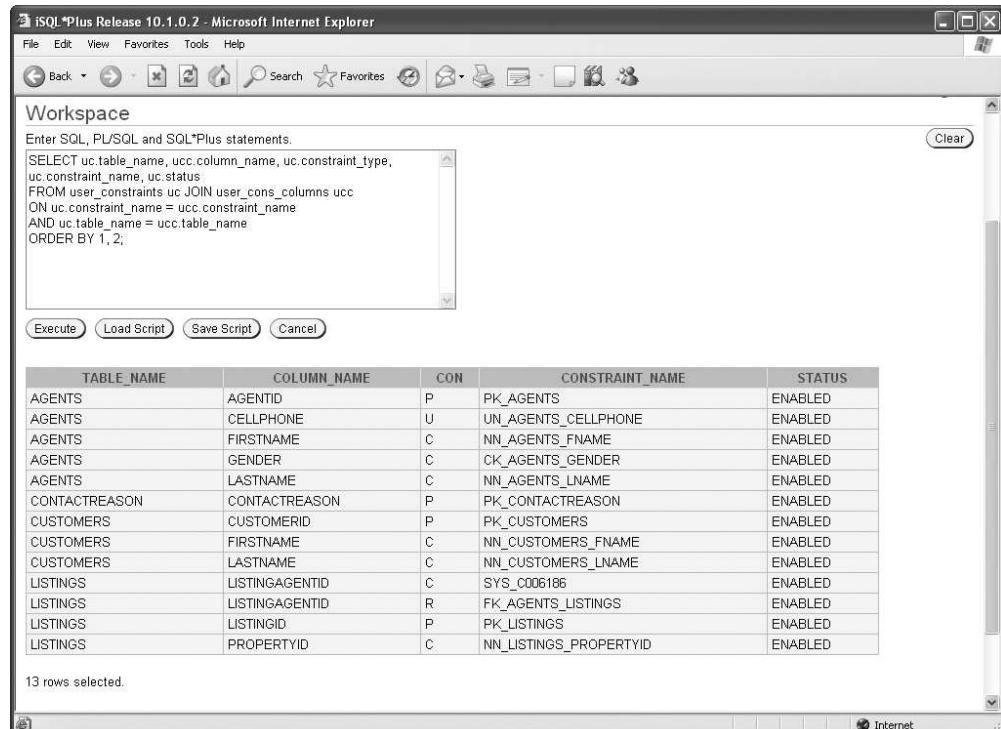


Рис. 3.25. Вывод на экран более полной информации об условиях

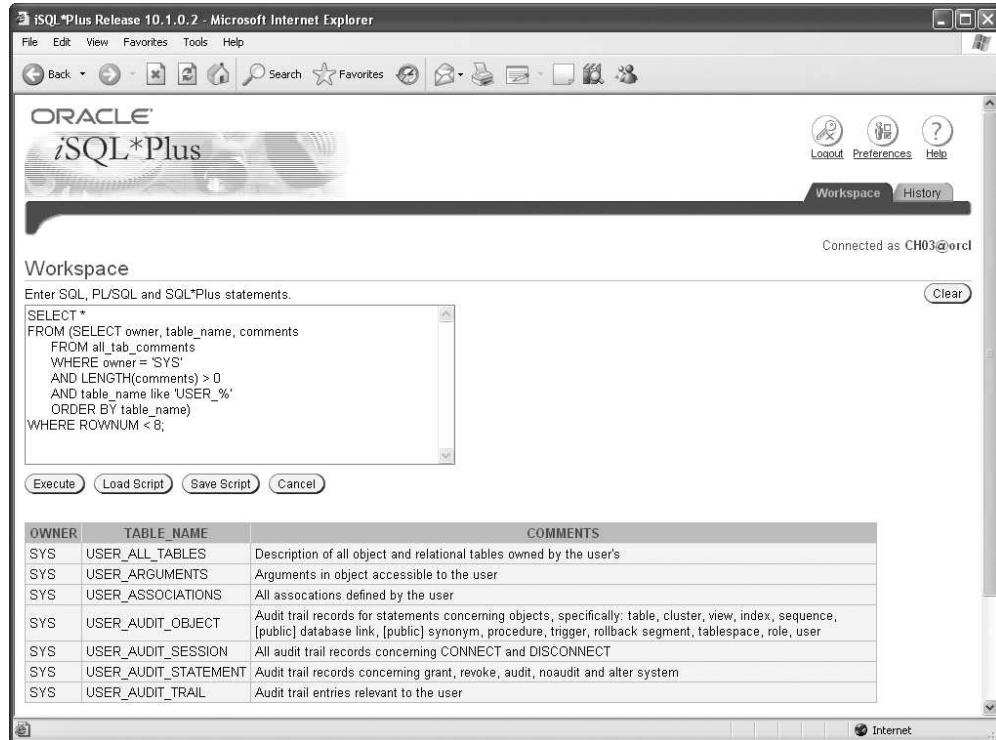
Напомним, что комментарии к любой из ваших таблиц отображаются на экране с помощью следующей команды:

```
SELECT *
  FROM user_tab_comments
```

Точно так же вы можете просмотреть комментарии, соотнесенные с столбцами таблиц:

```
SELECT *
  FROM user_col_comments
```

Добавив необязательное условие WHERE после любой из указанных выше команд SELECT, вы ограничите список извлекаемых и отображаемых комментариев. При желании вы можете изучить ряд комментариев, хранимых в учетной записи SYS, обладающей администраторскими правами доступа, и отобразить выбранные комментарии к таблицам. Поскольку соответствующий запрос может вернуть порядка нескольких тысяч строк, мы написали файл сценария, который вы можете загрузить и выполнить, когда вам будет удобно. Запрос, содержащийся в этой файле (самый сложный из всех, что встречались вам до этого момента), гарантирует возврат всего семи строк. Если вы решите использовать SQL\*Plus вместо iSQL\*Plus, запустите альтернативный сценарий Ch03TableComments.sql.



**Рис. 3.26.** Семь строк с комментариями таблицы из схемы SYS

Для того чтобы отобразить на экране комментарии к избранным таблицам учетной записи SYS, выполните следующие действия.

1. В окне iSQL\*Plus щелкните на кнопке Load Script, затем выберите Browse. Найдите папку, содержащую файл Ch03TableCommentsISP.sql, щелкните на кнопках Open и Load. Содержимое сценария появится в текстовом окне Workspace.
2. Щелкните на кнопке Execute, чтобы просмотреть результаты запроса (рис. 3.26).
3. Щелкните на гиперссылке Logout, чтобы выйти из базы данных, а затем закройте окно браузера.

## Удаление, восстановление и переименование таблиц

В SQL существуют выражения, позволяющие удалять (DROP) или переименовывать (RENAME) таблицы. Обе операции могут привести к тому, что другие объекты схемы, ссылающиеся на удаленные или переименованные таблицы, станут недействительными.

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> DROP TABLE Agents;
DROP TABLE Agents
*
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys

SQL>

```

Рис. 3.27. Попытка выполнения команды `DROP TABLE`, порождающая сообщение об ошибке

## Удаление таблицы

Если таблица базы данных вам больше не нужна, ее можно удалить, выполнив команду `DROP`. Будьте осторожны, поскольку на удаляемую таблицу могут ссылаться другие таблицы, представления и пользователи. Помните, что заново создать и заполнить таблицу может быть довольно сложно. При удалении таблицы из базы данных удаляется и содержащаяся в ней информация, а также структура таблицы. Oracle при этом удаляет все строки, информацию о таблице из словаря данных, а также все индексы, присоединенные процедуры и условия таблицы. В версиях Oracle, предшествовавших 10g, таблица удалялась безвозвратно, а занимаемое ею пространство освобождалось под другие цели. Начиная с Oracle 10g удаляемая таблица переносится в корзину, благодаря чему ее можно потом восстановить. Если вы точно знаете, что таблицу восстанавливать не придется, не помещайте ее в корзину, а добавьте в код необязательную команду `PURGE`. В целом синтаксис команды `DROP TABLE` выглядит так:

```
DROP TABLE <tablename> [CASCADE CONSTRAINTS] [PURGE]
```

Напомним, что от удаляемой таблицы могут зависеть другие таблицы, ссылающиеся на нее посредством внешних ключей. Например, если вы удалите таблицу `Agents`, Oracle выведет сообщение об ошибке, поскольку столбец `ListingAgentID` таблицы `Listings` ссылается на столбец `AgentID` таблицы `Agents` (рис. 3.27). Правила целостности на уровне ссылок не позволяют удалить родительскую таблицу, на которую ссылается одна или несколько таблиц. Решить эту проблему можно тремя способами: отключить или удалить отдельные условия, задающие внешние ключи, применив команду `ALTER TABLE` ко всем зависимым таблицам; сначала удалить все таблицы, посредством внешних ключей ссылающиеся на таблицу, которую вы собираетесь удалить; попросить Oracle удалить условия на внешние ключи, использовав опцию `CASCADE CONSTRAINTS`. Если вы выберете третий вариант, Oracle вначале удалит все условия, определяющие внешние ключи, которые ссылаются на удаляемую таблицу, а после этого удалит саму таблицу. Отметим, что перед удалением таблицы, на которую могут ссылаться другие, вы можете изучить представления словаря данных `all_dependencies` или `user_dependencies`.

**Предупреждение.** Направляя запрос к представлению словаря данных `all_dependencies`, будьте внимательны, — оно содержит более 60 000 строк!

Далее вы попробуете удалить таблицу `Agents`, затем добавите опцию `CASCADE CONSTRAINTS` и выполните модифицированную команду.

Для того чтобы удалить таблицу, выполните следующие действия.

1. Если необходимо, запустите SQL\*Plus и войдите в базу данных Oracle.
2. Выполните следующую команду:

```
DROP TABLE Agents;
```

Oracle выведет сообщение об ошибке (рис. 3.27).

3. Введите и выполните исправленную команду:

```
DROP TABLE Agents CASCADE CONSTRAINTS;
```

Oracle удалит таблицу и отобразит подтверждающее сообщение “Table dropped”.

## Восстановление удаленной таблицы

Если вы используете Oracle 10g или более позднюю версию и не используете опцию `PURGE` в команде `DROP TABLE`, то удаленную таблицу можно восстановить из корзины, выполнив команду `FLASHBACK TABLE`. Кроме того, Oracle восстановит все присоединенные процедуры, условия и индексы, соотнесенные с восстанавливаемой таблицей. Условия целостности на уровне ссылок, связывавшие восстанавливаемую таблицу с другими таблицами, не возобновляются. Упрощенный синтаксис команды, возвращающей таблицу к состоянию, предшествовавшему выполнению команды `DROP TABLE`, выглядит так:

```
FLASHBACK TABLE <tablename> TO BEFORE DROP;
```

Удалив таблицу, вы можете исследовать корзину, запросив представление словаря данных `user_recyclebin`, или же выполнить команду `SHOW RECYCLEBIN`, которая отображает некоторые столбцы представления словаря данных `RECYCLEBIN`.

Чтобы “заглянуть” в корзину, а затем восстановить удаленную таблицу, выполните следующие действия.

1. Введите следующую команду и нажмите клавишу `<Enter>`, чтобы изучить содержимое корзины:

```
COLUMN введите FORMAT A10  
SELECT original_name, object_name, type  
FROM user_recyclebin;
```

В ответ Oracle отобразит несколько строк, в том числе строку, значение аргумента `original_name` которой равно `Agents` (разумеется, остальные имена в вашем случае могут отличаться от показанных на рис. 3.28).

The screenshot shows the Oracle SQL\*Plus interface. The command line displays:

```

SQL> COLUMN type FORMAT A10
SQL> SELECT original_name, object_name, type
2  FROM user_recyclebin;

```

ORIGINAL_NAME	OBJECT_NAME	TYPE
BIN\$Sa0zw73IyT/WyDHO6FFc7yg==\$0	BIN\$NvUT6EUR3Q6+NMSIGBBHbHy6w==\$1	INDEX
BIN\$33ZtvBwyS/i+mxBtULPUaQ==\$0	BIN\$gEn8VbZKQuaxDss9dbUWQ==\$1	INDEX
AGENTS	BIN\$nyapMrJuTwSjj9IoFdstvw==\$0	TABLE

```

SQL> FLASHBACK TABLE Agents TO BEFORE DROP;
Flashback complete.
SQL>

```

**Рис. 3.28.** Восстановление удаленной таблицы

2. Чтобы восстановить таблицу Agents, выполните следующую команду:

```
FLASHBACK TABLE Agents TO BEFORE DROP;
```

Oracle ответит сообщением “Flashback complete”.

## Очистка корзины

Используя команду PURGE, корзину можно очистить целиком либо удалить отдельные ее элементы. Ниже приведен упрощенный синтаксис команды удаления одной таблицы.

```
PURGE TABLE <tablename>
```

Команда полной очистки корзины записывается следующим образом:

```
PURGE RECYCLEBIN
```

Для того чтобы очистить корзину, выполните следующие действия.

1. Введите следующую команду и нажмите клавишу <Enter>:

```
PURGE RECYCLEBIN;
```

Oracle отобразит подтверждающее сообщение “Recycle bin purged”.

## Переименование таблиц

Для переименования таблиц и других объектов базы данных (например, представлений и синонимов) применяется команда RENAME; вы также можете использовать команду ALTER TABLE. В любом случае Oracle переименует таблицу и автоматически наложит на нее все условия целостности, добавит индексы и связи с другими объектами, которые были соотнесены со старым именем таблицы. Тем не менее некоторые объекты, ссылающиеся на старую таблицу, например запрограммированные процедуры (глава 7), функции (глава 7) и представления (глава 6), больше работать не будут. Переименование таблицы осуществляется с помощью команды RENAME:

реименовывая таблицу, будьте предельно внимательны, поскольку это может иметь нежелательные последствия для вашей базы данных. Команда переименования имеет следующий синтаксис:

```
RENAME <old-tablename> TO <new-tablename>
```

## Создание новых таблиц на основе существующих

Еще один способ создания таблицы — “позаимствовать” определение, включая имена столбцов и типы данных, у существующей таблицы. “Родительская” таблица может принадлежать вашей или чужой схеме, к которой у вас есть право доступа SELECT. При выполнении команды CREATE TABLE AS с необязательной конструкцией SELECT создается таблица, структура которой идентична “родительской” таблице. После этого в новую таблицу копируется содержимое выбранных столбцов старой. Упрощенная форма синтаксиса команды CREATE TABLE AS выглядит таким образом:

```
CREATE TABLE <new-tablename> AS  
SELECT *  
FROM <old-tablename>
```

С помощью описанной команды вы можете создать в своей схеме точный дубликат таблицы. Предположим, например, что вам требуется создать резервную копию таблицы Listings на случай, если что-то случится с оригиналом.

Для того чтобы создать новую таблицу на основе существующей, выполните следующие действия.

1. В окне SQL\*Plus введите следующую команду и нажмите клавишу <Enter>:

```
CREATE TABLE NewListings AS  
SELECT *  
FROM Listings;
```

Oracle ответит сообщением “Table created”.

2. Чтобы проверить структуру новой таблицы, выполните указанную ниже команду SQL\*Plus.

```
DESCRIBE NewListings
```

Результаты выполнения двух команд должны дать результат, показанный на рис. 3.29.

3. Введите exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus.

```

SQL> CREATE TABLE NewListings AS
2  SELECT *
3  FROM Listings;
Table created.

SQL> DESCRIBE NewListings
Name          Null?    Type
-----        -----   -----
LISTINGID      NOT NULL NUMBER(38)
PROPERTYID     NOT NULL NUMBER(38)
LISTINGAGENTID NOT NULL NUMBER(38)
SALESTATUSID   NUMBER(38)
BEGINLISTDATE  DATE
ENDLISTDATE   DATE
ASKINGPRICE   NUMBER(9)

```

Рис. 3.29. Создание новой таблицы из уже существующей

## Резюме

Разработка реляционной базы данных, которая бы соответствовала потребностям ваших клиентов или превышала их, требует решения с этими клиентами следующих вопросов: как работает искомая организация, какие данные необходимо хранить для ее функционирования, и т.п. Для поддержания требуемого функционирования можно разрабатывать различные формы и отчеты. Естественным форматом хранения данных являются таблицы, состоящие из столбцов и строк. Каждая строка описывает одну сущность того “факта”, который представлен таблицей. Каждая строка таблицы единственным образом определяется первичным ключом, который также обеспечивает связь данной таблицы с другими таблицами. Столбец одной таблицы, совпадающий с первичным ключом другой (родительской) таблицы, называется внешним ключом дочерней таблицы. Подобные отношения служат для обеспечения целостности на уровне ссылок, которая запрещает удалять строку родительской таблицы без удаления связанных с ней строк дочерней таблицы.

Любой столбец таблицы может содержать данные только одного типа; всего в Oracle определены четыре основных типа данных: символьные, числовые, дата и время, изображения. Если при определении таблицы вы забыли упомянуть какой-то столбец, то можете ввести его позже с помощью команды ALTER TABLE. Данная команда позволяет модифицировать столбцы, в том числе изменять масштаб или точность числовых столбцов, присваивать значения по умолчанию и добавлять условия. Условиями называются правила, соотносимые с таблицей: ограничение значений, которые могут храниться в столбце; требование уникальности значений в пределах столбца; запрет на пропуск значения, а также присвоение столбцам статуса первичного или внешнего ключа, реализующего правила целостности.

Работая с базой данных Oracle, можно выбирать из двух интерфейсов: командной строки и Web-интерфейса. Используя интерфейс командной строки SQL\*Plus, вы вво-

дите команду и нажимаете клавишу <Enter>, чтобы выполнить ее немедленно. Web-интерфейс iSQL\*Plus действует через ваш браузер, отправляя на выполнение команды SQL и SQL\*Plus и возвращая в браузер аккуратно отформатированные столбцы.

С помощью различных представлений Oracle вы можете извлекать информацию, касающуюся имен таблиц, условий на столбцы и таблицы, комментариев к столбцам и таблицам, а также другие данные. Используя команду `DROP TABLE`, вы можете удалить таблицу из базы данных и (по желанию) очистить корзину, освободив занимаемое этой таблицей место. Находящуюся в корзине удаленную таблицу можно восстановить с помощью команды `FLASHBACK`. Для очистки корзины Oracle применяется команда `PURGE RECYCLEBIN`.

## Основные термины

- Условие проверки
- Класс
- Составной первичный ключ
- Условие
- Соглашение об именовании констант
- Тип данных
- Имена с разделителями
- Отключенное условие
- Условия домена
- Сущность
- Символ фиксированной длины
- Число фиксированной точности
- Восстановление
- Число с плавающей запятой
- Условие “внешний ключ”
- Целое число
- Условие целостности
- NOT NULL
- Словарь данных Oracle
- Точность
- Первичный ключ
- Условие “первичный ключ”
- Условие-ссылка

- Масштаб
- Объект схемы
- Таблица
- Условие уникальности
- Схема пользователя
- Символ переменной длины
- Представление

## Повторение пройденного материала

### Истина или ложь?

1. После задания имени пользователя и пароля изменить пароль невозможно.
2. Столбец, объявленный с типом данных VARCHAR2(30), всегда имеет длину 30 символов. Если строка короче этой длины, Oracle дополняет ее пробелами до 30 символов.
3. Условия на столбцы необязательны, однако они позволяют реализовать такие правила делового регламента, как максимальное и минимальное допустимые значения.
4. Условие “внешний ключ” служит для обеспечения целостности на уровне ссылок.
5. Наибольшее число, которое можно записать в столбце с объявленным типом данных NUMBER(7, 2), равно 99,9999.

### Заполнить пропущенное

1. По соглашениям, принятым в книге, имя условия “внешний ключ” начинается с \_\_\_\_\_, за которым следует символ подчеркивания.
2. \_\_\_\_\_ (два слова) позволяет единственным образом идентифицировать строку.
3. Условие \_\_\_\_\_ ограничивает значения, которые могут храниться в столбце.
4. Место временного хранения удаленных таблиц перед окончательным стиранием из базы данных Oracle называется \_\_\_\_\_.
5. Объект, отображающий подмножество строк и таблиц базы данных, называется \_\_\_\_\_.

### Варианты ответов

1. Если вы обладаете привилегиями администратора базы данных, то можете создать новое имя пользователя и задать для него пароль. С помощью какой команды это сделать?

- a) MODIFY USER <имя\_пользователя> as identified by <пароль>
  - б) CONNECT <имя\_пользователя>/<пароль>
  - в) CREATE USER <имя\_пользователя>/<пароль>
  - г) CREATE USER <имя\_пользователя> IDENTIFIED BY <пароль>
  - д) Все варианты неверные.
2. С помощью какого из приведенных ниже объявлений типов данных задается столбец *Quantity*, содержащий целые числа, максимальное значение которых равно 96 785?
- а) Quantity NUMBER(38)
  - б) Quantity NUMBER(7,2)
  - в) Quantity NUMBER(96785)
  - г) Quantity NUMBER(5)
3. Данное условие гарантирует, что значения определенного столбца данной таблицы не будут дублироваться.
- а) PRIMARY KEY
  - б) FOREIGN KEY
  - в) CHECK
  - г) UNIQUE
  - д) Правильные варианты а) и г).
4. Объект базы данных *user\_tables* находится в словаре данных Oracle и называется \_\_\_\_.
- а) Представлением.
  - б) Таблицей.
  - в) Таблицей корзины.
  - г) Условием.
  - д) Все варианты неправильные.
5. При удалении таблицы с использованием данной опции Oracle не помещает таблицу в корзину. Вместо этого таблица удаляется безвозвратно, а место, занимаемое ею, освобождается.
- а) CASCADE CONSTRAINTS
  - б) DUMP
  - в) PURGE
  - г) FLASHBACK
  - д) NO RECYCLE

## Упражнения

### 1. Readwood Realty

Роберт Стирлинг (Robert Stirling), руководитель отделения Redwood Realty, требует, чтобы вы удалили таблицу из базы данных агентства и создали новую. Новая таблица *CustAgentList* будет связана с таблицами *Agents*, *Customers*, *ContactReason* и *Listings*. На текущий момент одни упомянутые таблицы существуют, другие — нет. Таблица *CustAgentList* содержит следующие поля: *CustomerID*, *AgentID*, *ListingID*, *ContactDate*, *ContactReason*, *BidPrice* и *CommissionRate*. Первичный ключ данной таблицы составной, он образован первыми четырьмя из перечисленных столбцов. Первые три столбца также являются внешними ключами, указывающими на соответствующие таблицы. Четвертый внешний ключ — *ContactReason*. Столбец *BidPrice* может быть пустым. Если же в него вводится значение, то оно должно лежать в диапазоне от 90 000 до 800 000. Значение *CommissionRate* (если оно задано) должно составлять от 2 до 6 процентов включительно.

1. Запустите SQL\*Plus и войдите в Oracle. Вместе с SQL\*Plus можете использовать программу Блокнот, если вы хотите записать команды в виде файла сценария (или если придется вносить исправления).
2. Введите следующий код, нажимая клавишу <Enter> в конце каждой строки. Обратите внимание на то, что *дополнительные* пробелы не требуются — мы использовали их для улучшения читаемости команд; чтобы сократить время набора, можете их не вводить. Вместо <путь> введите нужный путь, а вместо <имя> — свое имя.

```

REM Drop the таблица NewListings
DROP TABLE NewListings CASCADE CONSTRAINTS;
COLUMN введите FORMAT A10
SPOOL C:\<путь>\<имя>Ch3Redwood.txt
SHOW USER
SELECT original_name, object_name, type
FROM user_recyclebin;
SPOOL OFF
REM Permanently eliminate the NewListings
PURGE RECYCLEBIN;
DROP TABLE CustAgentList CASCADE
CONSTRAINTS PURGE;
CREATE TABLE CustAgentList
  (CustomerID INTEGER NOT NULL,
   AgentID INTEGER NOT NULL,
   ListingID INTEGER NOT NULL,
   ContactDate DATE NOT NULL,
   ContactReason NVARCHAR2(15),
   BidPrice NUMERIC(9) CONSTRAINT
   cc_cuaglistBidPrice
     CHECK ((BidPrice >= 90000) AND (BidPrice <= 800000)),
   CommissionRate NUMERIC(4,4) CONSTRAINT
   cc_cuaglistCommRate
     CHECK (CommissionRate Between 0.02

```

```
    AND 0.06),
CONSTRAINT pk_CustAgentList
    PRIMARY KEY (CustomerID, AgentID,
ListingID, ContactDate));
```

3. Введите следующий код, нажимая клавишу <Enter> в конце каждой строки. С помощью команды ALTER TABLE вы добавляете условия в уже созданную таблицу.

```
ALTER TABLE CustAgentList
ADD CONSTRAINT fk_CustAgentList_Cust
FOREIGN KEY (CustomerID)
REFERENCES Customers (CustomerID) ON
DELETE CASCADE;
ALTER TABLE CustAgentList
ADD CONSTRAINT fk_CustAgentList_Agent
FOREIGN KEY (AgentID)
REFERENCES Agents (AgentID) ON DELETE
CASCADE;
ALTER TABLE CustAgentList
ADD CONSTRAINT fk_CustAgentList_Listing
FOREIGN KEY (ListingID)
REFERENCES Listings (ListingID) ON DELETE
CASCADE;
ALTER TABLE CustAgentList
ADD CONSTRAINT fk_CustAgentList_Contact
FOREIGN KEY (ContactReason)
REFERENCES ContactReason (ContactReason)
ON DELETE CASCADE;
```

4. Введите следующий код, чтобы отобразить на экране созданные условия. Результат будет добавляться в буферный файл, поэтому не забудьте указать то же имя файла, что и в строке 4 п. 2.

```
COLUMN table_name FORMAT A15
COLUMN column_name FORMAT A15
COLUMN constraint_name FORMAT A25
SPOOL C:\<путь>\<имя>Ch3Redwood.txt
APPEND
DESCRIBE CustAgentList
SELECT table_name, column_name,
constraint_name
FROM user_cons_columns
WHERE table_name = 'CUSTAGENTLIST';
SPOOL OFF
CLEAR COLUMNS
```

5. Введите exit и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus.
6. Откройте программу Блокнот, выберите File (Файл), затем Open (Открыть). Найдите папку, в которой хранится буферный файл <имя>Ch3Redwood.txt, выберите <имя>Ch3Redwood.txt, а затем щелкните на кнопке Open (Открыть).
7. Переместите точку ввода в крайнее левое положение первой строки и нажмите клавишу <Enter>, чтобы открыть новую строку. Перейдите на пустую строку и введите свое имя и фамилию, чтобы файл можно было идентифицировать.

8. Из меню File (Файл) выберите Print (Печать), чтобы напечатать буферный файл. Закройте Блокнот и щелкните на кнопке Yes (Да), когда потребуется подтвердить запись изменений.

## 2. Coffee Merchant

Администратор базы данных компании Coffee Merchant требует, чтобы вы создали две таблицы и наложили несколько условий на их столбцы. При выполнении данной работы вы будете использовать SQL\*Plus. Администратор хочет, чтобы вы создали таблицу `Inventory`, которая будет содержать названия сортов чая и кофе для продажи; также он попросил создать таблицу `Countries`, которая бы содержала идентификационные номера и названия всех стран мира. Заполненная таблица `Inventory` будет содержать внешние ключи для каждого сорта чая и кофе, указывающие на страну — производителя.

Набирая и выполняя команды SQL\*Plus и SQL, вы можете делать ошибки. Возможно, лучшим вариантом будет набор текста по отдельным командам в программе Блокнот с последующим копированием его в SQL\*Plus. Просто аккуратно исправляйте ошибки и вводите правильные выражения в SQL\*Plus. “Хроника” ваших ошибок будет представлена в буферном файле. Откройте этот файл на шаге 9 представленной ниже последовательности действий и удалите из него фрагменты с ошибками. Итак, вначале вам необходимо найти и запустить файл сценария `Ch03InitializeCoffee.sql`, который очистит базу данных Coffee Merchant, чтобы вы могли начать работу с чистого листа.

1. Запустите SQL\*Plus и войдите в Oracle.
2. Введите SPOOL<путь>\<имя> Ch3Coffee.txt и нажмите клавишу <Enter>. Не забудьте подставить вместо <путь> правильный путь (например, C:\Temp), а вместо <имя> — свое имя.
3. Введите следующий код:

```
COLUMN введите FORMAT A10
SHOW USER
DROP TABLE States;
SELECT original_name, object_name, type
FROM user_recyclebin;
PURGE RECYCLEBIN;
```

4. Создайте таблицу `Countries`, набрав следующие команды и нажав клавишу <Enter> в конце каждой строки:

```
CREATE TABLE Countries
(CountryID INTEGER CONSTRAINT pk_countries
PRIMARY KEY,
CountryName NVARCHAR2(40) CONSTRAINT
nn_CountriesCountryName NOT NULL);
```

5. Создайте таблицу Inventory, набрав следующие команды и нажав клавишу <Enter> в конце каждой строки (последняя строка заканчивается круглой скобкой и точкой с запятой):

```
CREATE TABLE Inventory
  (InventoryID INTEGER CONSTRAINT
  pk_inventory PRIMARY KEY,
  Name NVARCHAR2(40) NOT NULL,
  Price NUMBER(6,2) CONSTRAINT cc_InvPrice
    CHECK (Price >=3 AND PRICE <= 400),
  OnHand INTEGER CONSTRAINT cc_InvOnHand
    CHECK (OnHand BETWEEN -1000 AND 25000),
  Description NVARCHAR2(500),
  ItemBведите NVARCHAR2(1) CONSTRAINT
  cc_InvItemType
    CHECK (ItemType = 'C' OR ItemType = 'T'),
  CountryID INTEGER CONSTRAINT
  fk_CountriesInventory
    REFERENCES Countries(CountryID) ON
    DELETE CASCADE);
```

6. Введите следующий код, чтобы просмотреть заданные вами имена условий, а также имена, автоматически присвоенные Oracle (как обычно, нажимайте клавишу <Enter> в конце каждой строки):

```
DESCRIBE Inventory
DESCRIBE Countries
COLUMN table_name FORMAT A15
COLUMN column_name FORMAT A15
COLUMN constraint_name FORMAT A25
SELECT table_name, column_name,
constraint_name
FROM user_cons_columns WHERE table_name =
'INVENTORY';
SPOOL OFF
CLEAR COLUMNS
```

7. Введите `exit`, чтобы выйти из SQL\*Plus и Oracle.
8. Откройте программу Блокнот, из меню `File` (Файл) выберите `Open` (Открыть). Найдите папку, в которой хранится буферный файл <имя>Ch3Coffee.txt, выберите <имя> Ch3Coffee.txt, а затем щелкните на кнопке `Open` (Открыть).
9. Из меню `File` (Файл) выберите `Print` (Печать), чтобы напечатать буферный файл. Закройте программу Блокнот.

### 3. Rowing Ventures

Руководство Rowing Ventures желает, чтобы данные, вводимые в различные таблицы и составляющие информацию о регате, были согласованными; значения, введенные в выбранные столбцы находились в разумных пределах, а в родственных таблицах правильно задавались внешние ключи. Перед началом инициали-

зируйте таблицы базы данных Rowing Ventures, найдя и запустив файл сценария Ch03InitializeRowing.sql. Затем откройте программу Блокнот и введите все команды, требуемые для выполнения поставленной задачи. Запишите этот файл как <имя>Ch3Rowing.sql. После этого создайте две таблицы — BoatCrew и Person. Таблица BoatCrew содержит три столбца: BoatID, PersonID и Position. Столбцы BoatID и PersonID содержат целочисленные значения и вместе образуют составной первичный ключ таблицы BoatCrew. Ни один из столбцов не может быть пустым. Столбец Position содержит одну цифру от 0 до 8, причем 0 обозначает рулевого, а номера с 1 по 8 — положение, в котором располагаются гребцы. PersonID представляет собой внешний ключ, дублирующий столбец PersonID таблицы Person. Третий столбец, Position, может быть пустым. Поскольку столбец PersonID ссылается на еще не созданную таблицу, вначале следует определить таблицу Person, а затем с помощью команды ALTER TABLE определить внешний ключ таблицы BoatCrew.

Таблица Person содержит шесть столбцов: PersonID, LastName, FirstName, Gender, Weight и DateOfBirth. PersonID, целочисленное значение без заданного масштаба, представляет собой первичный ключ. LastName, FirstName и Gender представляются строками символов Unicode, максимальная длина которых равна 50. LastName и FirstName не должны быть пустыми, остальные поля могут не иметь заданного значения. Значениями поля Gender могут быть только M и F (прописными буквами), значение по умолчанию (подставляемое, если явно не было задано другое) равно F. (*Подсказка.* Введите значение по умолчанию перед заданием условия на столбец.) Поле Weight содержит любое целочисленное значение (масштаб не задается) от 85 до 250. В таблицу BoatCrew добавьте комментарий “Position 0 indicates the coxswain” (“Значению Position = 0 соответствует положение рулевого”). Задайте две таблицы, придерживаясь описанных выше требований. После этого откройте буферный файл и запишите следующую информацию: ваше имя пользователя (SHOW USER); описания двух таблиц; комментарии таблицы BoatCrew (*Подсказка.* Чтобы ограничить отображаемые результаты, используйте команду WHERE table\_name = 'BOATCREW'); список условий для созданных таблиц. (*Подсказка.* Чтобы ограничить показ только условиями, наложенными на две созданные таблицы, обратитесь с запросом к представлению user\_constraints, добавив ограничение WHERE table\_name IN ('BOATCREW', 'PERSON')). Распечатайте буферный файл и файл сценария, записанный с помощью программы Блокнот.

## 4. Broadcloth Clothing

Администратор базы данных Broadcloth Clothing желает, чтобы данные, вводимые в таблицу Customer, содержали необходимые условия и типы данных. Запустите iSQL\*Plus, а затем найдите, загрузите и выполните файл сценария Ch03InitializeBroadcloth.sql. (Запуск этого сценария удаляет все таблицы Broadcloth Clothing, которые остались после работы с предыдущими главами.) На сообщения

**ТАБЛИЦА 3.11.** Столбцы таблицы Customer и соотнесенные с ними условия

<i>Имя столбца</i>	<i>Тип данных</i>	<i>Длина</i>	<i>Условие</i>
CustomerID	Integer		Первичный ключ; значение по умолчанию — 0
CompanyName	String	Переменная; до 50 символов NOT NULL	
City	String	Переменная; до 50 символов NOT NULL	
Nation	String	Переменная; до 50 символов (нет)	
ContactID	Integer		(нет)
BaseCurrency	String	Переменная; до 50 символов (нет)	

об ошибках, подобные “table or view does not exist” (“таблица или представление на существует”), не обращайте внимания. Создайте таблицу Customer со столбцами и условиями, показанными в табл. 3.11. Всем условиям присвойте значимые имена. Создав таблицы, напечатайте Web-страницу iSQL\*Plus с сообщением “Table created”.

К столбцу BaseCurrency добавьте комментарий “Currency appropriate for home country” (“Валюта страны — производителя”). Отобразите условия только для таблицы Customers. *Подсказка.* Используйте представление user\_constraints и добавьте команду WHERE table\_name = 'CUSTOMER', чтобы ограничить отображение условия из таблицы Customer. Напечатайте полученную Web-страницу. Отобразите комментарии только для столбцов таблицы Customer (см. подсказку выше). Выполните команду DESCRIBE Customer и напечатайте полученную Web-страницу.



# ГЛАВА 4

## Изменение данных и наблюдение за таблицей

**В этой главе...**

- Вставка строк в таблицы
- Ввод данных в таблицы, содержащие правила области и целостности
- Перенос информации из других таблиц базы данных
- Обновление данных в одном или нескольких столбцах
- Использование подстановочных переменных в файле сценария SQL
- Удаление одной, нескольких или всех строк таблицы
- Усечение таблицы
- Запуск и завершение транзакций базы данных
- Наблюдение за операциями вставки, обновления и удаления с помощью обработчиков событий

### Вставка строк в таблицы

В главе 3 вы узнали, как использовать DDL-выражения Oracle для создания и удаления таблиц, а также именования условий. Выполняя команду CREATE TABLE, Oracle немедленно создает объект базы данных. Напомним, что это является особенностью всех DDL-выражений: для их вступления в действие команда COMMIT не требуется. В этой главе вы узнаете, как использовать выражения DML (Data Manipulation Language — язык манипулирования данными), чтобы вставлять, обновлять и удалять записи в таблицах базы данных. Выполняя DML-команды, вы можете периодич-

ски вводить команду управления транзакциями (Transaction Control — ТС) COMMIT, фиксируя внесенные изменения в базу данных. Иногда для просмотра внесенных изменений потребуется использовать команду запроса SELECT, которая подробно рассмотрена в главах 5 и 6. Команды запросов позволяют пользователям просматривать информацию, содержащуюся в базе данных, и находить ответы на различные вопросы (например: “Сколько наших клиентов живет в Калифорнии?”). Команды DML, выполняемые для вставки, обновления и удаления данных из таблиц, иногда называются *запросами действия* (action queries), поскольку они изменяют информацию, записанную в таблицах.

Запросы действия изменяют таблицы, причем при неправильной формулировке эти изменения могут нанести непоправимый урон и привести к потере данных. Позже вы узнаете, как наблюдать за запросами действия, фиксируя, кто их запустил и какие были значения данных до и после выполнения запроса. Для этого применяются удобные инструменты, именуемые *обработчиками событий* (trigger), — программы, автоматически запускаемые Oracle при выполнении команды INSERT, UPDATE или DELETE в указанной таблице базы данных.

Для того чтобы инициализировать таблицы базы данных, используемые в данной главе, вам потребуется запустить созданный нами сценарий. Напомним, что файл сценария представляет собой текстовый документ, содержащий команды SQL и SQL\*Plus. Чтобы запустить файл сценария с помощью SQL\*Plus, необходимо выполнить команду START, указав путь и имя файла. В данном случае путем является любой допустимый путь Windows, который не содержит пробелов. Напомним также, что вместо слова START вы можете подставить @.

Для инициализации базы данных Redwood Realty, которую мы будем использовать в главе 4, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в базу, используя свое имя пользователя и пароль, введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
2. Найдите файл сценария Ch04InitializeRedwood.sql и запишите путь Windows к нему. (Файл сценария создает две таблицы и налагает условия на выбранные столбцы таблицы.)
3. Введите приведенную ниже строку и нажмите клавишу <Enter>, чтобы запустить сценарий (вместо <путь> подставьте свой путь).

```
@C:\<путь>Ch04InitializeRedwood.sql
```

Тот же результат будет получен после выполнения команды START C:\<путь> Ch04InitializeRedwood, в которой опускается расширение .sql и вместо START подставляется @.

4. Чтобы убедиться, что созданы две таблицы Redwood Realty, введите и выполните следующие команды, отображающие на экране структуры таблиц:

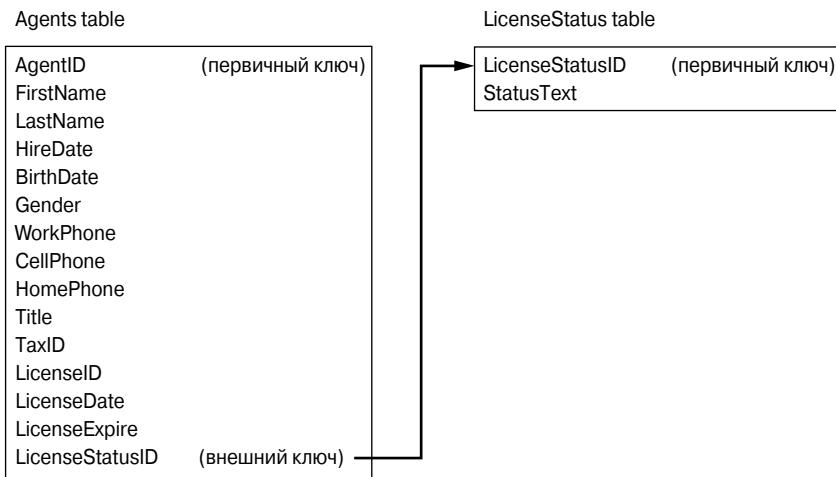


Рис. 4.1. Структура двух связанных таблиц Redwood Realty

```
DESCRIBE LicenseStatus
DESCRIBE Agents
```

Сценарий, запущенный на предыдущих этапах, удаляет все таблицы Redwood Realty и создает таблицы LicenseStatus и Agents. Обе таблицы не содержат ни одной строки данных, поскольку вы будете добавлять их в процессе работы над главой. Таблица Agents содержит информацию о брокерах и риэлторах компании Redwood Realty, включая их имена, контактную информацию, дату рождения, дату найма, а также состояние лицензии на работу с недвижимостью. Один из столбцов таблицы Agents, LicenseStatusID, является внешним ключом, совпадающим с соответствующим столбцом первичного ключа таблицы LicenseStatus. Помимо первичного ключа LicenseStatusID, таблица LicenseStatus содержит полное описание типа лицензии, полученной риэлтором или брокером. Каждому первичному ключу соответствует подробное текстовое описание. На рис. 4.1 показаны имена столбцов таблицы, а также связи первичных/внешних ключей. В табл. 4.1 показаны условия, наложенные на таблицу Agents. (Единственное условие, которое наложено на таблицу LicenseStatus, — это заданный столбец первичного ключа LicenseStatusID.) Условия внедряются в командах CREATE TABLE, запущенных при выполнении описанной выше процедуры. Если вы пожелаете изучить соответствующие команды DDL и SQL\*Plus, откройте файл сценария Ch04InitializeRedwood.sql в программе Блокнот и распечатайте его.

В этой главе вы будете добавлять, удалять и модифицировать строки обеих таблиц, чтобы разобраться в работе SQL-выражений INSERT, UPDATE и DELETE. Изучая данные выражения, вы столкнетесь с сообщениями об ошибках, генерируемыми Oracle, и посмотрите, как Oracle обеспечивает целостность на уровне области и ссылок.

**ТАБЛИЦА 4.1.** Условие и структура таблицы Agents

Столбец	Описание	Тип данных	Условие (условия)
AgentID	Уникальный номер, присвоенный агенту	INTEGER	Первичный ключ
Gender	Пол агента	NVARCHAR2 (10)	Проверка: допускается только значение “M”, “m”, “F” или “f”
Title	Внутреннее обращение к сотруднику	NVARCHAR2 (20)	Проверка: допускается только значение “salesperson” (“продавец”) или “broker” (“брюкер”) (возможна любая комбинация прописных и строчных букв)
License-StatusID	Код, представляющий состояние лицензии агента по работе с недвижимостью	INTEGER	Внешний ключ; значение должно быть таким же, как в соответствующем столбце первичного ключа таблицы LicenseStatus (LicenseStatusID)

Для добавления новых строк в таблицу базы данных применяется команда `INSERT`, с помощью которой вы можете задавать следующее:

- таблицу, в которую вставляется строка;
- список столбцов, в которые нужно вставить значения;
- список значений, вставляемых в заданные столбцы.

Общий синтаксис команды `INSERT` выглядит так:

```
INSERT INTO <table-name> [(column, column,...)]
    VALUES (expression, expression, ...);
```

В данном коде `<table-name>` обозначает таблицу, в которую вставляются значения. Если значения задаются для всех столбцов, имена столбцов можно не перечислять. Если же какие-то столбцы вы хотите пропустить, то необходимо задать и имена столбцов, и вставляемые значения. В любом случае необходимо указать значение первичного ключа и всех остальных столбцов, на которые наложено условие `NOT NULL` (включая столбец внешнего ключа). Если в определении таблицы для каких-то столбцов заданы значения по умолчанию, в команде `INSERT` значения для этих столбцов можно пропустить.

Если вы решили ввести значения во все столбцы таблицы, то имена столбцов в выражении `INSERT` перечислять не обязательно. При этом, однако, необходимо перечислить значения для каждого столбца таблицы в том же порядке, в котором столбцы указывались в команде `CREATE TABLE`. Если значение столбца неизвестно, используйте кодовое слово `NULL`. Например, чтобы вставить в таблицу `LicenseStatus` три строки, используйте следующие команды:

```
INSERT INTO LicenseStatus VALUES (1001, 'Licensed');
INSERT INTO LicenseStatus VALUES (1004, 'Deceased');
INSERT INTO LicenseStatus VALUES (1003, NULL);
```

Поскольку в приведенном коде названия столбцов не указываются, Oracle помещает первое значение в столбец `LicenseStatusID`, а второе значение (строку) — в столбец `StatusText`. Третья команда иллюстрирует случай, когда вы не знаете описание кода 1003: Oracle вводит в столбец значение `NULL`, указывая, что значение столбца `StatusText`, соответствующее `LicenseStatusID = 1003`, на данный момент неизвестно. Поскольку столбец `StatusText` содержит данные типа `NVARCHAR2` (строка символов), любой текст следует помещать в одинарные кавычки. Напомним, что любой текст, помещенный в одинарные кавычки, чувствителен к регистру. Кроме того, если вы захотите ввести строку символов, содержащую одну одинарную кавычку, вам необходимо будет использовать две одинарные кавычки. Например, допустимым является значение `'Realtor''s license suspended'`. Будьте внимательны и используйте при записи текстовых строк две одинарные кавычки, а не одну двойную.

Если вам требуется ввести значения только в некоторые выбранные поля таблицы, используйте форму `INSERT INTO`:

```
INSERT INTO <table-name> (column, column, ... )
    VALUES (expression, expression, ... );
```

В данном варианте команды `INSERT` необходимо задать имена столбцов, в которые вводятся значения, а также соответствующий набор значений (выражений) после оператора `VALUES`. *Порядок* имен столбцов в команде `INSERT` не обязательно должен быть таким же, как в команде `CREATE TABLE`. Тем не менее последовательность значений в списке `VALUES` должна соответствовать порядку имен столбцов в `INSERT INTO`. Например, ниже показано, как можно вставить данные в таблицу `Agents`. Единственное значение, которое должно указываться обязательно, — первичный ключ (столбец `AgentID`).

```
INSERT INTO AGENTS(AgentID, LastName, FirstName)
    VALUES (12345, 'Frockmeister', 'Robert');
INSERT INTO AGENTS(LastName, TaxID, LicenseStatusID, AgentID)
    VALUES ('Kellog', '1234-rtu', NULL, 40335);
INSERT INTO AGENTS(AgentID, Gender, LastName, BirthDate)
    VALUES(21334, 'f', 'Pennypacker', To_Date('04-JUN-1979'));
```

Во всех предыдущих выражениях столбцы идут не в том порядке, как задавалось при определении таблицы. Тем не менее указанные значения вставляются в соответ-

ствующие столбцы. Самостоятельно добавлять информацию в таблицу вы попробуете немного позже.

## Задание списка столбцов

Выполнив указанные ниже действия, вы увидите, что произойдет, если попытаться вставить строку, содержащую внешний ключ, в пустую или не полностью заполненную таблицу.

Для того чтобы вставить строку в таблицу Agents, выполните приведенные ниже действия.

- Если вы вышли из SQL\*Plus и Oracle, запустите SQL\*Plus и войдите в Oracle.
- Откройте Блокнот, чтобы вы могли легко модифицировать выражения SQL, которые придется набрать далее. Прежде всего запишите на диск файл Ch4<имя>Insert41.sql, а затем введите в Блокноте следующий код:

```
INSERT INTO Agents (AgentID, FirstName, LastName)
VALUES (10235, 'Tobias', 'Carling');
```

- Скопируйте данные строки SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить их. Oracle добавит новую строку (рис. 4.2).
- Переключитесь на Блокнот и введите следующую команду INSERT, скопируйте ее в SQL\*Plus, а затем выполните:

```
INSERT INTO Agents (LastName, AgentID, Gender, Title, FirstName)
VALUES ('O''Malley', 14556, 'M', 'Salesperson', 'Sean');
```

**Совет.** После буквы “О” в фамилии O’Malley наберите две одинарные кавычки. Помните, что любые две последовательные одинарные кавычки, набранные в строке, которая сама расположена внутри кавычек, база данных Oracle представляет одинарной кавычкой.

## Условия целостности

Прежде чем вы сможете добавлять строки, которые содержат внешний ключ, ссылающийся на первичный ключ другой таблицы, необходимо обязательно определить значения этой таблицы. Например, вы не можете добавить в таблицу Agent строку со значением 1006 в поле столбца LicenseStatusID, если таблица LicenseStatus еще не имеет строки, имеющей в столбце первичного ключа значение 1006. Решить подобную проблему взаимоотношений между первичным и внешним ключами можно минимум двумя способами. Во-первых, сначала можно заполнить таблицу, содержащую первичный ключ, на который ссылается внешний ключ другой таблицы. Во-вторых, можно указать NULL в поле внешнего ключа и отложить заполнение

The screenshot shows the Oracle SQL\*Plus interface. The title bar reads "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following SQL session:

```

SQL*Plus: Release 10.1.0.2.0 - Production on Mon Nov 13 11:32:24 2006
Copyright (c) 1982, 2004, Oracle. All rights reserved.

Connected to:
Personal Oracle Database 10g Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> INSERT INTO Agents (AgentID, FirstName, LastName)
  2   VALUES (10235, 'Tobias', 'Carling');
1 row created.

SQL> INSERT INTO Agents (LastName, AgentID, Gender, Title, FirstName)
  2   VALUES ('O''Malley', 14556, 'M', 'Salesperson', 'Sean');
1 row created.

SQL>

```

**Рис. 4.2.** Вставка строк в таблицу Agents

столбца первичного ключа. Второй подход означает, что позже вам придется обновить столбец внешнего ключа для всех строк после того, как вы заполните все строки таблицы с первичным ключом. Как правило, поступают следующим образом: вначале заполняют все таблицы, содержащие первичные ключи, на которые ссылаются другие таблицы посредством внешних ключей, а затем заполняют все остальные таблицы. Для иллюстрации сказанного попробуем вставить в таблицу Agent строку, нарушающую условие целостности на уровне ссылок, — ее внешний ключ указывает на пустую таблицу LicenseStatus.

Для того чтобы попробовать вставить строку, нарушающую условие целостности на уровне ссылок, выполните следующие действия.

- Сохраните текущее содержимое текстового файла, открытого в Блокноте. Из меню **File** (Файл) выберите **New** (Создать), чтобы создать новый файл. Запишите этот файл как Ch4<имя>IntegrityTest42.sql (вместо <имя> подставьте свое имя.)
- Ведите следующий код SQL в Блокноте, скопируйте его в диалоговое окно SQL\*Plus, а затем выполните:

```
INSERT INTO Agents (AgentID, LastName, LicenseStatusID, Gender)
  VALUES (14883, 'Fernandez', 1001, 'F');
```

Oracle генерирует сообщение об ошибке, поскольку значения 1001 в таблице LicenseStatus не существует. Поскольку родительского первичного ключа, который бы соответствовал внешнему ключу Fernandez, не существует, Oracle выдает ошибку.

- Переключитесь на программу Блокнот и введите NULL, заменив 1001 в третьей строке после оператора VALUES. Скопируйте модифицированную команду INSERT в SQL\*Plus и выполните ее. В результате Oracle вставит в таблицу правильную строку (рис. 4.3).

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO Agents (AgentID, LastName, LicenseStatusID, Gender)
  2  VALUES (14883, 'Fernandez', 1001, 'F');
INSERT INTO Agents (AgentID, LastName, LicenseStatusID, Gender)
*
ERROR at line 1:
ORA-02291: integrity constraint (CH04.FK_AGENTSLICENSESTATUSID) violated -
parent key not found

SQL> INSERT INTO Agents (AgentID, LastName, LicenseStatusID, Gender)
  2  VALUES (14883, 'Fernandez', NULL, 'F');

1 row created.

SQL>

```

**Рис. 4.3.** Нарушение условия целостности исправлено

Теперь модифицированное выражение `INSERT` выполняется правильно, поскольку в поле, которое является внешним ключом, вы вставили `NULL`. Из-за этого Oracle не проверяет, существует ли соответствующее значение первичного ключа в таблице `LicenseStatus`. Полностью проблема еще не решена, поскольку в конечном счете вам потребуется связать две таблицы между собой. Для этого вначале нужно будет заполнить таблицу `LicenseStatus`. После этого потребуется модифицировать все значения `NULL` в столбце `LicenseStatusID`, заменив их правильными значениями `LicenseStatusID`, согласующимися с величинами в столбце первичного ключа таблицы `LicenseStatus`.

## Пропуск списка столбцов

Взятый в скобки список имен столбцов таблиц в команде `INSERT INTO` можно пропускать при выполнении следующих условий: (1) вы указали значения для *всех* столбцов таблицы; (2) указанные значения идут в том же порядке, что и определения столбцов в команде `CREATE TABLE`. Возможность вставки строк в таблицу, не требующая указания имен столбцов, может сильно сократить объем набираемого кода. Фактически данный метод использован в большинстве файлов сценариев, написанных нами для заполнения таблиц. Поскольку столбец `LicenseStatusID` таблицы `Agents` зависит от значений одноименного столбца в таблице `LicenseStatus`, заполнение таблицы `LicenseStatus` является первоочередной задачей.

Итак, чтобы вставить значения в таблицу, не задавая список столбцов, выполните следующие действия.

1. Убедитесь, что вы вошли в `SQL*Plus`, а затем введите указанный ниже код. Как и ранее, используйте Блокнот для редактирования и копирования команд в `SQL*Plus`. В конце данной последовательности действий сохраните файл Блокнота на диске под уникальным именем.

```
INSERT INTO LicenseStatus VALUES (1001, 'Licensed');
INSERT INTO LicenseStatus VALUES (1002, 'Licensed NBA');
INSERT INTO LicenseStatus VALUES (1003, 'Canceled Officer');
```

2. Oracle подтвердит, что три строки вставлены.
3. Найдите файл Ch04InsertLS.sql, откройте его в Блокноте, скопируйте все содержимое в SQL\*Plus и выполните все команды INSERT. (Вы также можете запустить его как файл сценария — с помощью команды START.) Oracle добавит 13 строк в таблицу LicenseStatus.
4. В SQL\*Plus введите Exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus.
5. Откройте Web-браузер, а затем запустите iSQL\*Plus. Войдите в iSQL\*Plus, используя свое имя пользователя и пароль. (Если хотите, можете продолжать работать с SQL\*Plus. Просто мы считаем, что iSQL\*Plus лучше подходит для отображения столбцов таблиц, так как данный интерфейс втискивает поля всех столбцов в одну линию, поэтому каждая строка таблицы занимает одну строчку на экране; кроме того, iSQL\*Plus выравнивает значения под названием столбца.)
6. Введите SELECT \* FROM LicenseStatus в текстовом окне Workspace и щелкните на кнопке Execute, чтобы просмотреть содержимое таблицы LicenseStatus (рис. 4.4).
7. После просмотра результатов щелкните на гиперссылке Logout и закройте браузер.

Далее вам нужно добавить оставшиеся строки в таблицу Agents. Вместо того чтобы выполнять целый набор команд INSERT INTO, вы можете запустить файл сценария, который сам заполнит все оставшиеся строки.

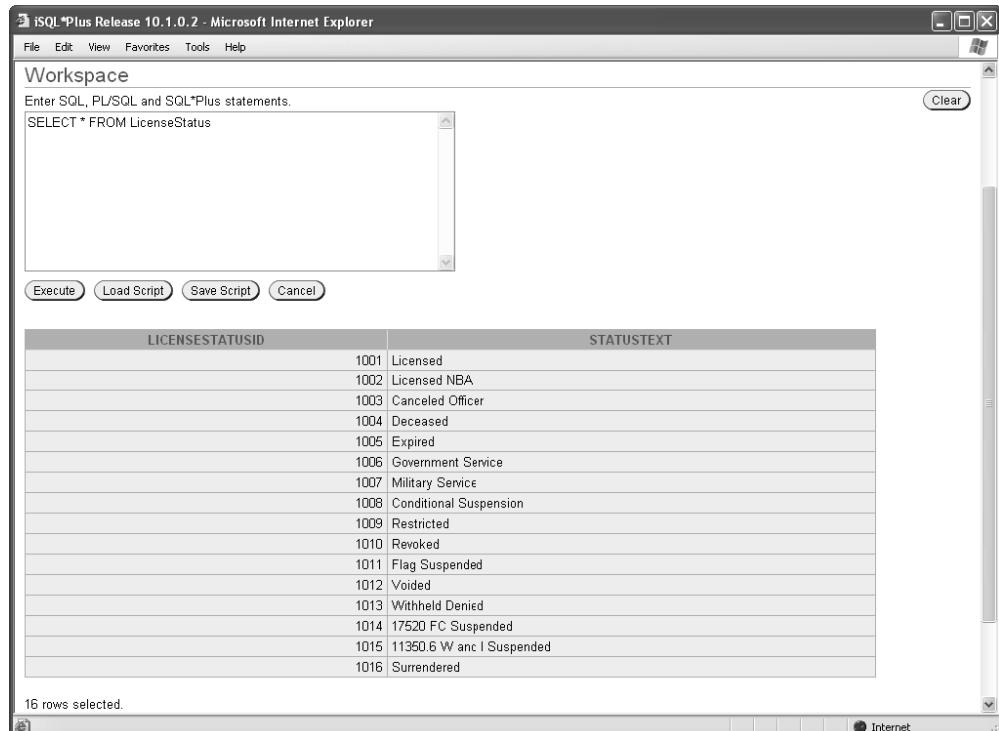
Для того чтобы заполнить оставшиеся строки таблицы Agents, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в систему базы данных, используя свое имя пользователя и пароль.
2. Найдите файл сценария Ch04InsertAgents.sql, запишите или запомните путь к нему, а затем введите и выполните указанную ниже команду START, добавляющую строки в таблицу Agents. Не забудьте подставить путь к файлу сценария. Результат выполнения команды должен быть таким, как показано на рис. 4.5.

**Совет.** При использовании команд START или @ суффикс .sql файла сценария можно опускать. Oracle знает, что файлы .sql — это файлы сценариев.

```
START C:\Ch04InsertAgents
```

3. Oracle отобразит ряд сообщений “1 row created”, указывающих, что строки вставлены в таблицу. При желании вы можете изучить файл сценария, открыв его в Блокноте.



**Рис. 4.4.** Вывод на экран содержимого таблицы LicenseStatus

4. Чтобы отобразить выбранные столбцы таблицы Agents, выполняются следующие выражения SQL\*Plus и SQL:

```
CLEAR SCREEN
SET PAGESIZE 35
COLUMN FirstName FORMAT A10
COLUMN LastName FORMAT A10
COLUMN Gender FORMAT A6
SELECT AgentID, FirstName, LastName, Gender
FROM Agents
ORDER BY LastName;
```

Результаты выполнения команды должны выглядеть так, как показано на рис. 4.5.

## Вставка даты и времени

Когда вы вводите в таблицу числовые данные, заключать их в одинарные кавычки не нужно. Напомним, однако, что символьные данные и информация о дате/времени должны заключаться в одинарные кавычки. Независимо от типа данных, вводимых посредством выражения INSERT, Oracle записывает все значения во внутреннем двоичном представлении. Выражения SQL, извлекающие данные из таблицы, переводят эту информацию в форму, подходящую для печати.

The screenshot shows the Oracle SQL\*Plus interface with the following command history and output:

```

SQL> SET PAGESIZE 35
SQL> COLUMN FirstName FORMAT A10
SQL> COLUMN LastName FORMAT A10
SQL> COLUMN Gender FORMAT A6
SQL> SELECT AgentID, FirstName, LastName, Gender
  2  FROM Agents
  3  ORDER BY LastName;

```

AGENTID	FIRSTNAME	LASTNAME	GENDER
15293	Lora	Allee	F
10235	Tobias	Carling	F
13555	Belinda	Chong	F
10429	Elizabeth	Dahlen	
12211	Cornelis	Dann	M
14883		Fernandez	F
14447	Jackson	Flamenbaum	M
15233	David	Gagnon	M
14599	Barbara	Herring	F
14677	James	Kellogg	M
15521	Patricia	Lewis	F
14556	Sean	O'Malley	M
12875	Essi	Okindo	M
12715	Nancy	Piperova	F
12963	Lee	Reed	F
12499	Clair	Robinson	F
11775	Cecilia	Romero	F
10497	Ramanathan	Rowe	M
15249	Tim	Schutz	M
13649	Ricki	Selby	F
10849	Heather	Sheibani	F
12381	Daniel	Silverburg	M
13853	Stanislaw	Soltwedel	M
14117	Tim	St-Onge	M
13771	Jessica	Taylor	F
12765	Edwin	Townsend	M
10913	Bruce	Voss	M
14601	Sindisiwe	Weber	M
15061	Christine	Williams	F
10041	Kai		M

30 rows selected.

SOL>

Рис. 4.5. Отображение выбранных столбцов таблицы Agents

По умолчанию для ввода даты в Oracle применяются форматы (*модели формата*) DD-MON-YY и DD-MON-YYYY. Например, чтобы заполнить в записях двух агентов поля AgentID и BirthDate, вы можете использовать следующие команды INSERT (для иллюстрации мы использовали оба формата данных по умолчанию):

```

INSERT INTO Agents (AgentID, BirthDate) VALUES (99887, '12-OCT-1970')
INSERT INTO Agents (AgentID, BirthDate) VALUES (55678, '08-JUN-82')

```

Обратите внимание на то, что в обоих случаях поля даты берутся в одинарные кавычки.

Если двузначное представление года принадлежит диапазону от 0 до 49, то считается, что дата относится к XXI веку. Если используются двузначные числа, превышающие 50, считается, что даты относятся к XX веку. Например, дата '20-OCT-35' представляет 20 октября 2035 года, а дата '04-JUL-51' — 4 июля 1951 года. Чтобы избежать возможных ошибок, при вводе дат год рекомендуется представлять четырьмя цифрами.

При вводе или извлечении информации о дате или времени можно задействовать модель формата, определяющую представление входной или выходной информации. *Модель формата* — это буквенно-цифровая строка, задающая, каким образом значение отображается на экране или интерпретируется при вводе в базу данных. На внутреннее представление данных модели формата не влияют. Из сказанного по-

**ТАБЛИЦА 4.2.** Типичные символы форматов модели, используемых для представления дат

<i>Символ формата модели</i>	<i>Отображаемое на экране или вводимое значение</i>
MONTH	FEBRUARY
Month	February
MM	02
DD	15
DDD	251
DAY	WEDNESDAY
Day	Wednesday
DY	WED
YYYY	2006
YY	06

нятно, что данные модели особенно важны при вводе или отображении кодов даты и времени. Чтобы вставить дату в поле DATE, не используя ни один из принятых по умолчанию форматов даты, ее можно представить строкой символов и использовать функцию Oracle TO\_DATE, которая преобразует символьную строку во внутренний формат, приемлемый для Oracle. Общий синтаксис функции TO\_DATE выглядит следующим образом:

```
TO_DATE('date string', 'date format model')
```

В данном выражении date string представляет код даты, взятый в одинарные кавычки, а date format model — формат представления кода даты. Например, приведенные ниже команды INSERT добавляют информацию в поля HireDate и AgentID записей агентов, используя форматы, отличающиеся от принятых по умолчанию.

```
INSERT INTO Agents (HireDate, AgentID)
    VALUES (TO_DATE('July 14, 2005', 'Month DD, YYYY'), 77777);
INSERT INTO Agents (AgentID, HireDate)
    VALUES (88888, TO_DATE('10/17/1946', 'DD/MM/YYYY'));
```

Использование пунктуации и прописных/строчных букв элементов формата даты указывают Oracle, как следует интерпретировать вводимые значения. Между элементами данных в date format model могут использоваться косая черта (/), дефис (-) и двоеточие (:), разумеется, точно так же они должны употребляться в аргументе date string. Для удобства в табл. 4.2 приведены примеры символов модели формата даты.

Теперь давайте попрактикуемся добавлять информацию об агенте в таблицу Agents, задавая константы даты и соответствующие модели формата.

Для того чтобы добавить строки в таблицу Agents, выполните следующие действия.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> INSERT INTO Agents(AgentID, HireDate, BirthDate)
  2 VALUES(23456,TO_DATE('January 12, 2004', 'Month DD, YYYY'),'10-JUN-1981');
1 row created.

SQL> INSERT INTO Agents(HireDate, AgentID)
  2 VALUES(TO_DATE('2004/02/03','YYYY/MM/DD'),56789);
1 row created.

SQL> SELECT AgentID, BirthDate, HireDate
  2 FROM Agents WHERE AgentID > 16000;
  AGENTID BIRTHDATE HIREDATE
----- ---------
  23456 10-JUN-81 12-JAN-04
  56789      03-FEB-04

SQL>

```

Рис. 4.6. Ввод дат в таблицу

- Если необходимо, запустите SQL\*Plus и войдите в систему, используя свое имя пользователя и пароль. Откройте программу Блокнот.
- Наберите в Блокноте следующий код SQL, скопируйте его в диалоговое окно SQL\*Plus, а затем выполните (если потребуется, исправьте опечатки в Блокноте, затем скопируйте исправленную команду в SQL\*Plus и выполните ее):

```

INSERT INTO Agents(AgentID, HireDate, BirthDate)
VALUES(23456,TO_DATE('January 12, 2004', 'Month DD, YYYY'),
'10-JUN-1981');
INSERT INTO Agents(HireDate, AgentID)
VALUES(TO_DATE('2004/02/03','YYYY/MM/DD'),56789);

```

- Если необходимо, нажмите клавишу <Enter>, чтобы выполнить второе выражение SQL. Oracle ответит двумя выражениями “1 row created”.
- Выведите добавленные строки на экран, выполнив простое выражение SELECT (подробный разбор выражений SELECT представлен в главе 5):

```

SELECT AgentID, BirthDate, HireDate
FROM Agents WHERE AgentID > 16000;

```

Результаты выполнения трех введенных выражений SQL показаны на рис. 4.6. У вас должна получиться точно такая же картина. Не выходя из SQL\*Plus, поскольку в следующем разделе мы продолжим работать с этим интерфейсом.

Ввод времени очень похож на ввод даты, только в данном случае применяются элементы формата модели HH, MI, SS, AM и PM, обозначающие часы, минуты, секунды и время до или после полудня (AM или PM можно писать с точками или без них). Например, чтобы вставить время (например, момент начала гонок), необходимо выполнить следующую команду INSERT:

```

INSERT INTO BoatRace(RaceID, BeginTime)
VALUES(1237, TO_DATE('08:20:25 AM', 'HH:MI:SS AM'));

```

Запись моментов времени бывает важна в базах данных, подобных базе Rowing Ventures, где основной информацией является время прохождения лодкой трассы — разность между временем, когда лодка финишировала, и моментом начала гонок.

## Вставка данных из других таблиц базы данных

Команду `INSERT` можно использовать с необязательным оператором `SELECT`, который позволяет выбирать данные из других таблиц. Соответствующий подзапрос может возвращать нуль, один или несколько строк из другой таблицы, вставляя их в целевую таблицу. Число столбцов и типы данных столбцов исходной и целевой таблиц должны совпадать. Несмотря на это, соответствующие столбцы исходной и целевой таблиц могут и не совпадать. Например, вы можете вставить выбранные строки из таблицы `NevadaRealtors` в таблицу `Agents`. Предполагая, что поля первичного ключа из исходной таблицы `NevadaRealtors` не противоречат столбцам `AgentID` таблицы `Agents`, содержащими информацию о существующих агентах, вы можете ввести приведенную ниже команду `INSERT` с подзапросом `SELECT` и скопировать строки из таблицы `NevadaRealtors`, соответствующие риэлторам родом из Уиннемакки.

```
INSERT INTO Agents(AgentID, FirstName, LastName, BirthDate,
                   LicenseID)
  SELECT RealtorID, Fname, Lname, DOB, LicNumb
    FROM NevadaRealtors
   WHERE LOWER(HomeCity) = 'winnemucca';
```

Oracle исследует таблицу `NevadaRealtors` и извлечет из нее перечисленные столбцы, для которых значение `HomeCity` исходной таблицы (переведенное в строчные буквы) совпадает с `winnemucca`. Столбцы исходной таблицы извлекаются и вставляются в соответствующую целевую таблицу (`Agents`) по одной строке за раз. Если из таблицы `NevadaRealtors` будет извлечено 14 строк, Oracle вставит их все в таблицу `Agents`. Если вы желаете вставить все строки исходной таблицы — не используйте оператор `WHERE`. Немного поэкспериментировав с данной формой команды `INSERT`, вы почувствуете, как она работает, и оцените ее удобства.

Для того чтобы добавить строки в таблицу `Agents` из другой таблицы, выполните следующие действия.

- Найдите файл сценария `Ch04NevadaRealtors.sql`, запишите путь к нему, а затем введите и выполните следующую команду (не забудьте подставить свой путь к сценарию):

```
START C:\Ch04NevadaRealtors
```

- Ведите в `SQL*Plus` нижеприведенный код. Разумеется, сначала вы можете запустить Блокнот, набрать в нем команду и скопировать ее в `SQL*Plus`. Нажмите клавишу `<Enter>`, чтобы выполнить выражение.

The screenshot shows the Oracle SQL\*Plus interface. The command line displays:

```

SQL> START C:\Ch04\NevadaRealtors
SQL> INSERT INTO Agents(AgentID, FirstName, LastName, BirthDate, LicenseID)
  2   SELECT RealtorID, Fname, Lname, DOB, LicNumb
  3   FROM NevadaRealtors
  4   WHERE LOWER(HomeCity) = 'winnemucca';

5 rows created.

SQL> SELECT AgentID, FirstName, LastName
  2   FROM Agents WHERE AgentID > 515432;

```

The output shows the inserted data:

AGENTID	FIRSTNAME	LASTNAME
515602	Doc	Bishop
518444	Bill	Jeffords
516344	Jim	Case
516476	John L.	Peterson
515894	Craig	Cardoza

SQL>

Рис. 4.7. Вставка строк из другой таблицы

```

INSERT INTO Agents(AgentID, FirstName, LastName, BirthDate,
                   LicenseID)
SELECT RealtorID, Fname, Lname, DOB, LicNumb
FROM NevadaRealtors
WHERE LOWER(HomeCity) = 'winnemucca';

```

Если все будет нормально, Oracle выведет на экран сообщение “5 rows created”.

- Чтобы отобразить на экране вставленные строки, введите и выполните следующую SQL-команду SELECT:

```

SELECT AgentID, FirstName, LastName
  FROM Agents WHERE AgentID > 515432;

```

Результаты выполнения пп. 2 и 3 показаны на рис. 4.7.

- Чтобы завершить данное упражнение, удалите таблицу NevadaRealtors, выполнив следующую команду SQL:

```
DROP TABLE NevadaRealtors CASCADE CONSTRAINTS PURGE;
```

Oracle удаляет таблицу, высвобождает занимаемое ею место и выдает подтверждающее сообщение “Table dropped”.

- Ведите Exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus.

## Создание и использование последовательностей

(sequence) называется объект базы данных, генерирующий ряд уникальных целых чисел, которые часто применяются для задания уникальных первичных ключей для новых записей, добавляемых в таблицу. Всякий раз, когда вы используете последовательность при добавлении новой строки в таблицу, число, создаваемое последовательностью, гарантированно будет новым, так что никакие две строки таблицы не получат одинакового номера. Благодаря последовательностям пользователи, вводящие данные, могут не выдумывать самостоятельно уникальные идентификаторы, поручив это дело базе данных. Ценность автоматической генерации уникальных идентификаторов становится даже более очевидной, если рассмотреть, что может произойти, когда информацию одновременно вводят в базу данных десятки или сотни пользователей (например, заказы на закупку). Для каждого нового вводимого заказа необходимо гарантировать, что введенный первичный ключ отличается от всех остальных чисел, присваиваемых другими членами команды. Последовательность решает эту задачу, если при вводе данных с помощью команды `INSERT` все пользователи указывают имя одной и той же последовательности. Именованная последовательность мгновенно раздает разные целые числа всем пользователям, затребовавшим это. Поскольку максимальное целое число, генерируемое последовательностью, равно  $10^{27}$ , для создания уникальных идентификаторов данных чисел вам точно хватит.

### Создание последовательности

Чтобы создать последовательность, применяется команда SQL `CREATE SEQUENCE`, причем одновременно вы можете иметь столько последовательностей с уникальными именами, сколько требует конкретная база данных. Синтаксис команды `CREATE SEQUENCE` выглядит следующим образом:

```
CREATE SEQUENCE <sequence name>
[START WITH <value>]
[INCREMENT BY <value>]
[ {MAXVALUE <value> | NOMAXVALUE} ]
[ {MINVALUE <value> | NOMINVALUE} ]
[ {CYCLE | NOCYCLE} ]
[ {ORDER | NOORDER} ]
[ {CACHE <value> | NOCACHE} ];
```

Все фразы в квадратных скобках являются необязательными, а фразы в фигурных скобках означают, что вы должны выбрать один из перечисленных элементов.

Подобно другим объектам в базе данных, все последовательности в вашей схеме пользователя должны иметь уникальные имена. Чтобы отличать последовательности от других объектов, рекомендуем именовать их с использованием суффикса `_seq`. Оператор `START WITH` позволяет задавать начальное значение последовательности. Если вы его не укажете, Oracle начнет последовательность с единицы. Оператор `INCREMENT BY` указывает положительное или отрицательное число, добавляя которое

к последнему значению последовательности, Oracle получит следующее значение. Последовательности могут идти в восходящем (положительное значение-инкремент) или нисходящем (отрицательное значение-инкремент) порядке, причем значения последовательности могут быть отрицательными. Значения MINVALUE и MAXVALUE задают соответственно минимальное и максимальное значения последовательности. Бессмысленно задавать MINVALUE при положительных, а MAXVALUE — при отрицательных значениях INCREMENT BY. Значение CYCLE/NOCYCLE определяет, должна ли база данных Oracle продолжать выдавать значения последовательности по достижении максимума или минимума. CYCLE задает, что последовательность автоматически запускается снова, а NOCYCLE (значение по умолчанию) — что после достижения максимального или минимального значения Oracle не должна генерировать новые числа. ORDER задает, что целые числа должны распределяться в порядке поступления запросов на них: в ответ на первый запрос выдается ближайшее свободное число, в ответ на второй — следующее и т.д. По умолчанию задано значение NOORDER. CACHE/NOCACHE задает, может ли Oracle заблаговременно сгенерировать ряд уникальных значений до того момента, как они потребуются, и записать их в памяти для быстрой выдачи в ответ на запрос. Значение NOCACHE (применяющееся по умолчанию) означает, что предварительно группа чисел генерироваться не будет. Пожалуй, самое важное, что вы должны помнить о последовательностях, это то, что, после того как значение из последовательности было сгенерировано и присвоено, аналогичное значение не может быть сгенерировано, пока последовательность не начнет повторяться (что будет очень не скоро).

Ниже вы создадите последовательность, которую можно будет использовать для создания уникального первичного ключа при вводе новой записи (агента) в таблицу Agents. Используя данную последовательность вместе с командой INSERT, вы гарантируете, что в любом случае будет присваиваться уникальный первичный ключ. Новую последовательность чисел мы начнем с 654321, поскольку желательно, чтобы всем новым агентам присваивались шестизначные идентификационные номера, которые были бы больше любого из использованных ранее идентификаторов.

Для того чтобы создать последовательность и присвоить ей имя, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в систему, используя свое имя пользователя и пароль. Откройте программу Блокнот и используйте ее в качестве редактора. Скопируйте набранный в Блокноте код в SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить эти выражения в SQL\*Plus. Если потребуется, исправьте ошибки и повторите требуемые действия.
2. В SQL\*Plus введите CLEAR SCREEN и нажмите клавишу <Enter>.
3. Наберите в SQL\*Plus следующий код:

```
CREATE SEQUENCE AgentID_seq
```

```
START WITH 654321
INCREMENT BY 2 NOCACHE NOCYCLE;
```

4. Нажмите клавишу <Enter>, чтобы выполнить набранную команду. Oracle ответит строкой подтверждения “Sequence created”.

Чтобы Oracle перечислила все последовательности, созданные в вашей схеме, необходимо отобразить соответствующие столбцы представления словаря данных `USER_SEQUENCES`.

Для того чтобы отобразить на экране имена последовательностей и подробности, касающиеся представителей этого списка, выполните следующие действия.

1. Введите и выполните следующий код, чтобы отобразить на экране информацию о столбцах `USER_SEQUENCES`:

```
DESCRIBE user_sequences
```

2. Введите и выполните следующее выражение:

```
SELECT sequence_name, min_value, max_value, last_number
FROM user_sequences;
```

3. Изучите результат, полученный на последнем этапе и показанный на рис. 4.8. Обратите внимание на то, что значение `MAXVALUE` для последовательности `AGENTID_SEQ` (во внутреннем представлении Oracle записывает имена объектов прописными буквами) равно  $10^{27}$  — огромной величине!

## Использование значений последовательности в качестве первичных ключей

Определив последовательность `AgentID_seq`, вы можете вызывать ее для создания уникальных целых чисел, требуемых для решения какой-либо задачи. Ниже с помощью данной последовательности вы будете создавать первичные ключи для информации о каждом новом агенте, добавляемом в таблицу `Agents`. Для обращения к значениям последовательности используются два псевдостолбца: `NEXTVAL` и `CURRVAL`. На первый взгляд псевдостолбец Oracle кажется полем из таблицы базы данных, но на самом деле это команда, возвращающая значение. `NEXTVAL` (“следующее значение”) — это псевдостолбец, который в действительности генерирует значение последовательности. После того как это значение создано, оно записывается в псевдостолбце `CURRVAL` (“текущее значение”). Данный псевдостолбец очень полезен, поскольку вы можете обратиться к нему и посмотреть, какое значение сгенерировала база данных, а затем, например, заполнить поле внешнего ключа родственной таблицы с помощью только что сгенерированного значения. Чтобы с помощью последовательности создать новое (уникальное) целое число, используйте приведенную ниже команду `INSERT`.

The screenshot shows the Oracle SQL\*Plus interface. The command line displays the creation of a sequence named AgentID\_seq with a start value of 654321, increment by 2, and no cache or cycle. It then describes the user\_sequences to show the sequence's characteristics like min\_value (1), max\_value (1.0000E+27), and last\_number (654321). Finally, it selects the sequence\_name, min\_value, max\_value, and last\_number from the user\_sequences table.

```

SQL> CREATE SEQUENCE AgentID_seq
  2  START WITH 654321
  3  INCREMENT BY 2 NOCACHE NOCYCLE;

Sequence created.

SQL> DESCRIBE user_sequences
Name          Null?    Type
-----        -----
SEQUENCE_NAME      NOT NULL VARCHAR2(30)
MIN_VALUE          NUMBER
MAX_VALUE          NUMBER
INCREMENT_BY       NOT NULL NUMBER
CYCLE_FLAG         VARCHAR2(1)
ORDER_FLAG         VARCHAR2(1)
CACHE_SIZE         NOT NULL NUMBER
LAST_NUMBER        NOT NULL NUMBER

SQL> SELECT sequence_name, min_value, max_value, last_number
  2  FROM user_sequences;

SEQUENCE_NAME      MIN_VALUE  MAX_VALUE LAST_NUMBER
-----            -----
AGENTID_SEQ        1         1.0000E+27  654321

SQL>

```

**Рис. 4.8.** Создание последовательности и вывод на экран ее характеристик

<sequencename>.NEXTVAL

Для того чтобы обратиться к последнему сгенерированному значению, используйте следующий синтаксис команды SQL:

<sequencename>.CURRVAL

Далее в таблицу Agents необходимо вставить три новые строки, каждая из которых представляет информацию о трех новых агентах по недвижимости. В данных примерах с целью экономии времени для каждого агента вы будете заполнять данными лишь несколько столбцов.

Чтобы вставить новую строку, используя последовательность AgentID\_seq, необходимо выполнить следующие действия.

1. Введите и выполните следующие команды INSERT:

```

INSERT INTO Agents (AgentID, FirstName, LastName)
  VALUES (AgentID_seq.NEXTVAL, 'Angelica', 'Francis');
INSERT INTO Agents (AgentID, FirstName, LastName)
  VALUES (AgentID_seq.NEXTVAL, 'Hershel', 'Wickstrom');
INSERT INTO Agents (AgentID, FirstName, LastName)
  VALUES (AgentID_seq.NEXTVAL, 'Bryan', 'Wilson');

```

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT AgentID, FirstName, LastName
2  FROM Agents
3  WHERE AgentID > 123456
4  ORDER BY AgentID;

AGENTID FIRSTNAME          LASTNAME
-----  -----
515602 Doc                 Bishop
515894 Craig                Cardoza
516344 Jim                  Case
516476 John L.              Peterson
518444 Bill                 Jeffords
654321 Angelica            Francis
654323 Hershel              Wickstrom
654325 Bryan                Wilson

8 rows selected.

SQL> SELECT AgentID_seq.CURRVAL
2  FROM dual;

CURRVAL
-----
654325

SQL>

```

**Рис. 4.9.** Просмотр текущего (CURRVAL) и присвоенных значений последовательности

2. Введите следующий код, чтобы отобразить на экране введенные в таблицу строки. Изучите столбец первичного ключа. Согласуются ли присвоенные первичные ключи с созданной выше последовательностью?

```

SELECT AgentID, FirstName, LastName
FROM Agents
WHERE AgentID > 123456
ORDER BY AgentID;

```

3. Введите следующий код, чтобы отобразить на экране значение псевдостолбца CURRVAL для последовательности AgentID\_seq:

```

SELECT AgentID_seq.CURRVAL
FROM DUAL;

```

DUAL — это определенная Oracle таблица, которая содержит одну строку и один столбец. Она применяется, когда требуется отобразить на экране значение (например, результат расчетов или дату), не являющееся столбцом какой-либо таблицы. Подробнее об использовании таблицы DUAL рассказывается в главе 5.

Результаты выполнения команд SELECT на этапах 2 и 3 показаны на рис. 4.9. Как видите, первичные ключи, сгенерированные для столбца AgentID последовательностью AgentID\_seq, равны 654321, 654323 и 654325 — как и ожидалось. Согласитесь — последовательности очень удобны при создании первичных ключей.

## Изменение и удаление последовательностей

Настройки любой последовательности можно изменить с помощью команды ALTER SEQUENCE. Любые изменения последовательности применяются к значениям, генерируемым после модификации. Команда ALTER SEQUENCE имеет следующие ограничения: значение START WITH нельзя изменять, и любые изменения не могут сделать недействительными существующие значения последовательности. Например, вы не можете изменить последовательность, понизив MAXVALUE до значения, меньшего уже сгенерированного числа последовательности. Упрощенный синтаксис команды ALTER SEQUENCE выглядит так:

```
ALTER SEQUENCE <sequencename>
[ INCREMENT BY <value> ]
[ {MAXVALUE <value> | NOMAXVALUE} ]
[ {MINVALUE <value> | NOMINVALUE} ]
[ {CYCLE | NOCYCLE} ]
[ {ORDER | NOORDER} ]
[ {CACHE <value> | NOCACHE} ];
```

Если последовательность вам больше не нужна, вы можете удалить ее с помощью команды DROP SEQUENCE. Синтаксис этой команды довольно прост:

```
DROP SEQUENCE <sequencename>;
```

Для иллюстрации сказанного мы удалим созданную в данном разделе последовательность AgentID\_seq.

Итак, чтобы удалить последовательность, выполните следующие действия.

1. Введите и выполните следующую команду SQL:

```
DROP SEQUENCE AgentID_seq;
```

Oracle удалит последовательность и выдаст подтверждающее сообщение “Sequence dropped”.

2. Из SQL\*Plus не выходите.

---

---

## Обновление данных

Данные о покупателях, продавцах и недвижимости, фигурирующие в таблицах базы данных Redwood Realty, изменяются довольно часто, а статус различных домов закономерно меняется с “продается” на “продано”. Риэлторы компании Redwood Realty обновляют свои лицензии на работу с недвижимостью, т.е. время от времени потребуется обновлять значения в столбце LicenseExpire. Большинство таблиц в базе данных любого коммерческого предприятия обновляется постоянно. Уменьшение наличных товаров на складе магазина должно отражаться в базе данных, основываясь на мгновенных, часовых или дневных продажах. Информация о людях, продуктах и со-

**ТАБЛИЦА 4.3.** Операторы отношений Oracle

<i>Оператор отношения</i>	<i>Значение</i>
=	Равно
<> или !=	Не равно
>	Больше
<	Меньше
<=	Меньше или равно
>=	Больше или равно

бытиях требует модификации при изменении текущих условий. Для отражения меняющейся ситуации в столбцах таблиц базы данных применяется команда SQL UPDATE.

## Команда обновления UPDATE

Для модификации существующих данных в SQL применяется команда UPDATE. Используя одну команду UPDATE, одновременно допускается модификация одного или нескольких столбцов одной таблицы. Общий синтаксис команды UPDATE, иногда имеемой запросом действия (action query), выглядит следующим образом:

```
UPDATE <tablename>
SET <columnname1> = <expression1>
    , <columnname2> = <expression2>, ...
[WHERE <condition>];
```

В данном коде <tablename> задает таблицу, значения которой необходимо обновить, <columnname> — это имя столбца таблицы, а <expression> — значение или выражение, которое дает значение того же типа, что указан в определении столбца. Необязательный оператор WHERE задает условие поиска и ограничивает изменения строками отобранных столбцов. Если вы опустите эту команду, будут обновлены все столбцы. Аргумент <condition> состоит из имен столбцов, операторов отношения, значений и логических операторов. При обработке условия WHERE должно получаться значение true или false. Операторы отношения Oracle, подобно своим аналогам в других системах баз данных и языках программирования, состоят из значений, приведенных в табл. 4.3.

Логические операторы Oracle показаны в табл. 4.4. Точно такие же операторы применяются и в других базах данных.

Запись, представляющая агента Тобиаса Карлинга (Tobias Carling), содержит большой набор значений NULL, поскольку в момент ввода первоначальной информации (см. рис. 4.2) вы не знали, как заполнить поля Title, LicenseDate и т.д. Чтобы предоставить информацию для отсутствующих столбцов этой записи, необходимо написать команду UPDATE, так задав условия на столбцы FirstName и LastName, чтобы изменить только требуемую строку и заполнить или модифицировать в ней только

**ТАБЛИЦА 4.4.** Логические операторы Oracle

Логический оператор	Значение	Пример команды <i>WHERE</i>
AND	True, если оба условия дают значение true; false – в противном случае	WHERE State = 'MN' AND Gender = 'M'
OR	True, если любое из значений равно true; false – в противном случае	WHERE LicenseStatusID = 1001 OR LicenseStatusID = 1002
NOT	Выражение отрицания	WHERE NOT State = 'NE'
IN	True, если значение равно одному из ряда перечисленных	WHERE City IN('Arcata', 'Fortuna', 'Orick')
LIKE	Групповое выражение, позволяющее задавать условия, подобные “не имеет значения”	WHERE LastName LIKE 'Mc%'
BETWEEN ... AND ...	True, если значение принадлежит указанному диапазону (включительно)	WHERE SqFt BETWEEN 1500 AND 2000

требуемые поля. Ниже показано, как заполнить поля, соответствующие столбцам *HireDate*, *BirthDate*, *Gender* и *Title*. После этого вы можете отобразить строку, соответствующую агенту Тобиасу Карлингу, и проверить модификации визуально.

Для того чтобы обновить одну строку с помощью выражения *UPDATE*, выполните следующие действия.

1. В окне SQL\*Plus введите нижеприведенный код, который отформатирует часть столбцов и отобразит часть строк, соответствующих записи *Carling*. Запишите данный набор команд, чтобы вы могли выполнить их после завершения данного упражнения. Как обычно, для редактирования и записи команд можете использовать программу Блокнот, а затем скопировать их в SQL\*Plus и выполнить.

```
CLEAR SCREEN
COLUMN FirstName FORMAT A10
COLUMN LastName FORMAT A10
COLUMN Gender FORMAT A6
SELECT FirstName, LastName, HireDate, BirthDate, Gender, Title
FROM Agents
WHERE FirstName = 'Tobias' AND LastName = 'Carling';
```

**Совет.** Не забудьте набрать имя и фамилию точно так, как показано выше, – с первыми прописными буквами. В противном случае Oracle не найдет и не сможет отобразить соответствующую строку.

2. Чтобы единственным образом определить строку, подлежащую изменению, в операторе WHERE команды UPDATE также необходимо задать имя и фамилию агента. Этого достаточно, поскольку маловероятно, чтобы в небольшой таблице были представлены два человека (записи) с одинаковыми именами и фамилиями. Введите и выполните следующую команду:

```
UPDATE Agents
SET HireDate = TO_DATE('12/19/2000', 'MM/DD/YYYY'),
    BirthDate = TO_DATE('10/19/1975', 'MM/DD/YYYY'),
    Gender = 'M',
    Title = 'SALESPERSON'
WHERE FirstName = 'Tobias' AND LastName = 'Carling';
```

Oracle выведет строку “1 row updated”.

3. Чтобы проверить внесенные изменения, введите указанную ниже команду SELECT, отображающую на экране строку Carling. Просто вставьте в SQL\*Plus код, написанный на этапе 1, или перенаберите его и выполните еще раз.

```
SELECT FirstName, LastName, HireDate, BirthDate, Gender, Title
FROM Agents
WHERE FirstName = 'Tobias' AND LastName = 'Carling';
```

Как видите (рис. 4.10), база данных Oracle обновила четыре поля, которые вы модифицировали при выполнении п. 2.

4. SQL\*Plus не закрывайте.

Условия UPDATE могут влиять на несколько строк. Если опустить оператор WHERE, Oracle применит все указанные изменения ко всем записям таблицы. Поэтому, будьте очень осторожны, используя команду UPDATE без WHERE. Ошибки могут обойтись очень дорого. Кстати, если вы сделали ошибку и хотите отменить предыдущую команду, сделайте это с помощью операции ROLLBACK. Подробности, касающиеся эффектов команды ROLLBACK, ее использования, а также обзор противоположной операции COMMIT приводятся в разделе “Транзакции базы данных” далее в этой главе.

Лора Элли (Lora Allee), один из брокеров компании Redwood Realty, обратила внимание на то, что значения поля Title для недавно добавленных записей таблицы Agents пусты. В частности, она указала, что записи агентов, идентификаторы (AgentID) которых равны или превышают 515602, должны иметь в столбце Title значение “Salesperson” (“продавец”). Лора просит вас модифицировать только это поле, но для всех указанных агентов. Как вы измените значения одного столбца для нескольких строк? Желательно, чтобы при решении подобных задач у вас была какая-то характеристика (значение для одного или нескольких столбцов), которая единственным образом определяет только нужные строки. В данном случае в качестве такого значения используется значение AgentID, а для решения поставленной задачи применяется команда UPDATE с уточнением WHERE.

The screenshot shows the Oracle SQL\*Plus interface. The command line displays an UPDATE statement for the Agents table, changing the Title for the row where FirstName = 'Tobias' AND LastName = 'Carling' from 'SALESPERSON' to 'Salesperson'. The output shows the update was successful (1 row updated), and a SELECT statement is run to verify the change.

```

SQL> COLUMN FirstName FORMAT A10
SQL> COLUMN LastName FORMAT A10
SQL> COLUMN Gender FORMAT A6
SQL> SELECT FirstName, LastName, HireDate, BirthDate, Gender, Title
  2  FROM Agents
  3 WHERE FirstName = 'Tobias' AND LastName = 'Carling';

FIRSTNAME LASTNAME HIREDATE BIRTHDATE GENDER TITLE
----- ----- ----- ----- -----
Tobias      Carling

SQL> UPDATE Agents
  2  SET HireDate = TO_DATE('12/19/2000', 'MM/DD/YYYY'),
  3    BirthDate = TO_DATE('10/19/1975', 'MM/DD/YYYY'),
  4    Gender = 'M',
  5    Title = 'SALESPERSON'
  6 WHERE FirstName = 'Tobias' AND LastName = 'Carling';

1 row updated.

SQL> SELECT FirstName, LastName, HireDate, BirthDate, Gender, Title
  2  FROM Agents
  3 WHERE FirstName = 'Tobias' AND LastName = 'Carling';

FIRSTNAME LASTNAME HIREDATE BIRTHDATE GENDER TITLE
----- ----- ----- ----- -----
Tobias      Carling   19-DEC-00 19-OCT-75 M     SALESPERSON
SQL>

```

**Рис. 4.10.** Обновление нескольких столбцов для одной строки

Итак, чтобы обновить значение одного столбца для нескольких указанных строк, необходимо выполнить следующие действия.

1. В окне SQL\*Plus наберите и выполните следующий код:

```

UPDATE Agents
SET Title = 'Salesperson'
WHERE AgentID >= 515602;

```

Oracle ответит сообщением “8 rows updated”.

В настоящий момент все значения в столбце `Title` представлены прописными и строчными буквами, — например, используются названия `Salesperson` и `Broker`. Вы решили, что будет лучше, если все значения будут представлены только прописными буквами. Мы можем взять текущее значение столбца и модифицировать его с использованием команды `UPDATE`. В данном случае мы используем встроенную функцию Oracle `UPPER`, которая переведет значения столбца `Title` в верхний регистр. Записывая команду `UPDATE` без уточнения `WHERE`, мы модифицируем все строки.

Чтобы модифицировать столбец для всех строк, необходимо выполнить следующие действия.

+ Oracle SQL*Plus		
File Edit Search Options Help		
FIRSTNAME	LASTNAME	TITLE
Tobias	Carling	SALESPERSON
Sean	O'Malley	SALESPERSON
	Fernandez	
FIRSTNAME	LASTNAME	TITLE
Kai		SALESPERSON
Elizabeth	Dahlen	SALESPERSON
Ramanathan	Rove	SALESPERSON
Heather	Sheibani	SALESPERSON
Bruce	Voss	SALESPERSON
Cecilia	Romero	SALESPERSON
Cornelis	Dann	SALESPERSON
Danial	Silverburg	SALESPERSON
Clair	Robinson	SALESPERSON
Nancy	Piperova	SALESPERSON
FIRSTNAME	LASTNAME	TITLE
Edwin	Townsend	SALESPERSON
Essi	Okindo	BROKER
Lee	Reed	SALESPERSON
Stanislaw	Soltwedel	SALESPERSON
Belinda	Chong	SALESPERSON
Ricki	Selby	BROKER
Jessica	Taylor	BROKER
Tim	St-Onge	SALESPERSON
Jackson	Flamenbaum	SALESPERSON
Barbara	Herring	SALESPERSON
Sindisiwe	Weber	SALESPERSON
FIRSTNAME	LASTNAME	TITLE
James	Kellogg	SALESPERSON
Christine	Williams	SALESPERSON
David	Gagnon	BROKER
Lora	Allee	BROKER
Tim	Schutz	SALESPERSON
Patricia	Lewis	BROKER

40 rows selected.  
SQL>

< > ...

Рис. 4.11. Обновление нескольких строк с помощью одного выражения

1. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
  2. Наберите и выполните приведенную ниже команду.
- ```
UPDATE Agents
SET Title = UPPER>Title);
```
- Oracle ответит сообщением “40 rows updated”.
3. Выполните следующий код, чтобы отобразить на экране выбранные столбцы всех строк и просмотреть изменения.
- ```
SELECT FirstName, LastName, Title
FROM Agents;
```

Обратите внимание на то, что теперь значения в столбце Title состоят из прописных букв (см. рис. 4.11).

## Введение в структуру CASE

Существует довольно много ситуаций, когда обновляемое значение столбца зависит от столбцов таблицы. Например, вы пожелали модифицировать ставку комиссионных риэлтора, основываясь на времени (числе лет) работы риэлтора в компании. Другой пример: вы пожелали вставить или изменить комиссионное вознаграждение брокера агентства, основываясь на предлагаемой цене продажи дома. Подобная логика типа `if-then-else`, применяющаяся, когда необходимо изменить столбец, не являющийся константой, реализована в структуре CASE. Ее можно использовать везде, где Oracle допускает применение выражений SQL. Общий синтаксис структуры CASE выглядит следующим образом:

```
CASE
    WHEN <condition1> THEN <result1>
    WHEN <condition2> THEN <result2>
    ...
    [ELSE <default result>]
END
```

Oracle проверяет каждое условие WHEN, начиная с первого. Далее база данных последовательно переходит к следующему условию WHEN, пока не найдет условие, которое дает значение `true`. Если Oracle найдет условие, равное `true`, она возвращает результат, следующий после оператора THEN, и пропускает оставшиеся операторы WHEN/THEN. Если ни одно из условий не дало значений `true`, то Oracle возвращает значение, следующее после ELSE. Тем не менее, если оператора ELSE нет и ни одно из предшествовавших условий WHEN не дало значения `true`, то команда CASE возвращает значение NULL.

Например, приведенная ниже команда SQL обновляет гипотетический столбец, именуемый `CommissionRate`. Новое значение столбца основывается на значении в столбце `Paygrade` продавца и операторе SET, использующем структуру CASE для выбора одного из четырех возможных значений обновленной величины `CommissionRate`. Значение, в которое переходит `CommissionRate`, зависит от значения в столбце `Paygrade`.

```
UPDATE AgentPayGrade
SET CommissionRate =
CASE
    WHEN Paygrade = 1 THEN 0.06
    WHEN Paygrade = 2 THEN 0.075
    WHEN Paygrade = 3 THEN 0.08
    ELSE 0.09
END
WHERE PositionType = 'SALESPERSON';
```

Если значение столбца `Paygrade` для какой-то строки равно 1, то `CommissionRate` получает значение 0.06; если `Paygrade` равно 2, то `CommissionRate` становится равным 0.075, и т.д.

## Обновление данных с использованием структуры CASE

Хотя структура CASE может применяться не только в команде UPDATE, особенно она полезна при изменении значений столбцов на основе условий, задаваемых полями строки, относящимися к другим столбцам. Предположим, например, что кто-то из руководства Redwood Realty обнаружил, что сроки окончания действия лицензий агентов неверны. Допустим, что срок действия лицензий, полученных после 1 января 2002 года, равен пяти годам (начиная с LicenseDate). Лицензии, выданные между 1 января 2000 г. и 31 декабря 2001 г., становятся недействительными через 4 года (начиная с LicenseDate). Срок действия всех остальных лицензий остается таким же, как указано в базе данных. Чтобы обработать необходимые изменения, вы можете использовать одну команду UPDATE со структурой CASE, “разветвляющейся” на три случая. Ниже показано, как для решения той же задачи применяется встроенная функция Oracle ADD\_MONTHS, имеющая следующий синтаксис:

```
ADD_MONTHS(<date column>, <months>)
```

Аргумент <date column> представляет собой столбец, содержащий код даты, а второй аргумент, <months>, — это целое число, представляющее количество месяцев, которое необходимо прибавить к первому аргументу. Чтобы добавить пять лет к некоторому столбцу, вместо второго аргумента потребуется указать значение 60 (5 лет по 12 месяцев в году).

Итак, чтобы обновить столбец, используя структуру CASE, выполните следующие действия.

- Если потребуется, запустите SQL\*Plus, войдите в систему, используя свое имя пользователя и пароль, и найдите файл сценария Ch04ListAgents.sql. В SQL\*Plus введите START C:\Ch04ListAgents и нажмите клавишу <Enter>, чтобы выполнить сценарий. Не забудьте подставить свой путь вместо C:\. Файл сценария отобразит строки из таблицы Agents, упорядоченные по возрастанию значения LicenseDate (рис. 4.12).
- Запустите Блокнот, наберите следующий код SQL, скопируйте его в SQL\*Plus и выполните:

```
UPDATE Agents
SET LicenseExpire = CASE
    WHEN LicenseDate >='01-JAN-02'
        THEN ADD_MONTHS(LicenseDate, 60)
    WHEN LicenseDate >='01-JAN-00'
        THEN ADD_MONTHS(LicenseDate, 48)
    ELSE LicenseExpire
END;
```

- Если Oracle выдаст сообщения об ошибках, исправьте код в Блокноте и повторите процесс копирования/выполнения, пока не получите сообщение “40 rows updated”, подтверждающее успешное выполнение операции.

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area displays a table with four columns: FIRSTNAME, LASTNAME, LICENSEDATE, and LICENSEEXPIRE. The data consists of 27 rows of agent information. The bottom of the window shows the message "27 rows selected." and the prompt "SQL>".

FIRSTNAME	LASTNAME	LICENSEDATE	LICENSEEXPIRE
Bruce	Voss	27-AUG-95	01-SEP-98
Lora	Allee	22-NOV-95	01-DEC-98
Essi	Okindo	26-APR-96	01-MAY-99
Belinda	Chong	22-JAN-97	01-FEB-00
Kai		30-JAN-97	01-FEB-00
Stanislaw	Soltwedel	04-MAY-97	01-JUN-00
Christine	Williams	03-AUG-97	01-SEP-00
Tim	St-Onge	25-JUN-98	01-JUL-01
Danial	Silverburg	24-OCT-98	01-NOV-01
Jackson	Flamenbaum	16-FEB-99	01-MAR-02
Ramanathan	Rowe	06-APR-99	01-MAY-02
Cecilia	Romero	02-MAY-99	01-JUN-02
Jessica	Taylor	23-OCT-99	01-NOV-02
Lee	Reed	20-FEB-00	01-MAR-03
David	Gagnon	14-MAR-00	01-APR-03
Cornelis	Dann	10-SEP-00	01-OCT-03
James	Keillogg	06-JUN-01	01-JUL-04
Ricki	Selby	23-DEC-01	01-JAN-05
Nancy	Piperova	27-MAY-02	01-JUN-05
Barbara	Herring	18-SEP-02	01-OCT-05
Heather	Sheibani	27-NOV-02	01-DEC-05
Tim	Schutz	17-MAR-03	01-APR-06
Elizabeth	Dahlen	22-DEC-04	01-JAN-08
Edwin	Townsend	26-MAR-06	01-APR-09
Clair	Robinson	29-MAR-06	01-APR-09
Sindisiwe	Weber	21-FEB-07	01-MAR-10
Patricia	Lewis	29-JUN-07	01-JUL-10

Рис. 4.12. Избранные строки таблицы Agents перед обновлением

- Повторно выполните приведенную ниже команду, подставив свой путь к файлу сценария. Не забудьте нажимать клавишу <Enter> в конце строки.

```
START C:\Ch04ListAgents
```

- Сценарий отобразит список, показанный на рис. 4.12, только с обновленными значениями столбца LicenseExpire для указанных строк (рис. 4.13). Проверьте, что обновление прошло успешно, изучив результаты и сравнив их с показанными на рис. 4.12 и 4.13. Обратите внимание на то, что дата окончания срока действия лицензии LicenseExpire для записи David Gagnon равна “14-MAR-04”, т.е. ровно четыре года после даты выдачи LicenseDate.
- Ведите Exit, нажмите клавишу <Enter>, чтобы выйти из Oracle, и закройте клиент SQL\*Plus.

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area displays a table with four columns: FIRSTNAME, LASTNAME, LICENSEDATE, and LICENSEEXPIRE. The data consists of 27 rows, each representing an agent's information. The table is styled with alternating row colors. At the bottom of the table, a message "27 rows selected." is displayed. Below the table, there is a SQL prompt "SQL>" followed by a scroll bar and other standard window controls.

FIRSTNAME	LASTNAME	LICENSEDATE	LICENSEEXPIRE
Bruce	Voss	27-AUG-95	01-SEP-98
Lora	Allee	22-NOV-95	01-DEC-98
Essi	Okindo	26-APR-96	01-MAY-99
Belinda	Chong	22-JAN-97	01-FEB-00
Kai		30-JAN-97	01-FEB-00
Stanislaw	Soltwedel	04-MAY-97	01-JUN-00
Christine	Williams	03-AUG-97	01-SEP-00
Tim	St-Onge	25-JUN-98	01-JUL-01
Danial	Silverburg	24-OCT-98	01-NOV-01
Jackson	Flamenbaum	16-FEB-99	01-MAR-02
Ramanathan	Rowe	06-APR-99	01-MAY-02
Cecilia	Romero	02-MAY-99	01-JUN-02
Jessica	Taylor	23-OCT-99	01-NOV-02
Lee	Reed	20-FEB-00	20-FEB-04
David	Gagnon	14-MAR-00	14-MAR-04
Cornelis	Dann	10-SEP-00	10-SEP-04
James	Kellogg	06-JUN-01	06-JUN-05
Ricki	Selby	23-DEC-01	23-DEC-05
Nancy	Piperova	27-MAY-02	27-MAY-07
Barbara	Herring	10-SEP-02	10-SEP-07
Heather	Sheibani	27-NOV-02	27-NOV-07
Tim	Schutz	17-MAR-03	17-MAR-08
Elizabeth	Dahlen	22-DEC-04	22-DEC-09
Edwin	Townsend	26-MAR-06	26-MAR-11
Clair	Robinson	29-MAR-06	29-MAR-11
Sindisiwe	Weber	21-FEB-07	21-FEB-12
Patricia	Lewis	29-JUN-07	29-JUN-12

27 rows selected.

SQL>

Рис. 4.13. Избранные строки таблицы Agents после обновления

## Подстановочные переменные

Когда вам требуется обновить несколько строк таблицы, набирать команду UPDATE десять–двадцать раз может быть неудобно. Oracle позволяет избавиться от подобной рутинной работы с помощью подстановочных переменных. Помимо этого, подстановочные переменные полезны в файлах сценариев SQL, где с их помощью можно изменить любую часть команды SQL во время выполнения. Другими словами, подстановочные переменные позволяют создать более универсальное выражение SQL, которое откладывает все уточнения на момент выполнения, когда вы заполняете все пропущенные фрагменты.

*Подстановочные переменные* (substitution variables) получили свое имя благодаря тому, что во время выполнения выражений SQL вместо них подставляются значения. Когда вы запускаете выражение SQL, содержащее подстановочные переменные, Oracle предлагает вам ввести вместо этих переменных значения. Ниже на примере команды UPDATE показано, как подстановочные переменные могут облегчить вашу работу.

Предположим, что вам потребовалось добавить столбец AnnualBonus в таблицу Agents, а затем обновить значение этого столбца, указав годовую премию для

**ТАБЛИЦА 4.5.** Годовая премия, основанная на стаже работы сотрудника

<i>Стаж работы (лет)</i>	<i>Размер годовой премии (доллары)</i>
12 или меньше	0
от 13 до 24	500
от 25 до 48	700
от 49 до 72	1 000
73 или больше	1 500

всех сотрудников. Размер годовой премии зависит от времени работы сотрудника в Redwood Realty. Следовательно, чтобы заполнить пустой столбец AnnualBonus, требуется несколько команд UPDATE. Помимо того, что данный процесс потребует очень много времени, при вводе нескольких десятков команд UPDATE возможны ошибки. Используя подстановочные переменные, мы существенно уменьшаем объем однообразной работы. Команда UPDATE, использующая подстановочные переменные для модификации столбца AnnualBonus, приведена ниже.

```
UPDATE Agents
SET AnnualBonus = &BonusAmount
WHERE HireDate > &MinHireDate;
```

Две подстановочные переменные, использованные в предыдущей команде (&BonusAmount и &MinHireDate), указывают SQL\*Plus запросить у вас необходимые значения при выполнении команды в SQL\*Plus. В ответ вы вначале набираете значение &BonusAmount и нажимаете клавишу <Enter>. После этого SQL\*Plus запрашивает значение &MinHireDate, вы вводите его и нажимаете клавишу <Enter>. SQL\*Plus помещает два введенных значения в выражение SQL и отправляет модифицированную команду в SQLPlus, после чего ее выполняет Oracle. Если вы захотите обновить другие строки таблицы Agents, введите косую черту (/) и нажмите клавишу <Enter>. SQL\*Plus повторно запросит у вас значения для двух подстановочных переменных, а затем отправит команду Oracle для выполнения. Далее мы попрактикуемся в ее использовании на более сложном примере.

Роберт Стирлинг (Robert Stirling) хочет, чтобы вы модифицировали таблицу Agents, добавив в нее столбец AnnualBonus, и обновили размер годовой премии, основываясь на стаже работы сотрудников в компании Redwood. Схема распределения премий показана в табл. 4.5.

Чтобы рассчитать, как долго сотрудник проработал в Redwood Realty, можно использовать две функции Oracle: SYSDATE и MONTHS\_BETWEEN. Функция SYSDATE возвращает текущую дату (по часам вашего компьютера), а MONTHS\_BETWEEN — число месяцев между двумя датами. Общая форма этой функции выглядит так:

```
MONTHS_BETWEEN(<date1>, <date2>)
```

Функция вычисляет, сколько месяцев прошло между двумя датами, и возвращает эту величину в виде действительного числа (включающего дробную часть). Если

первая дата соответствует более позднему моменту времени, чем вторая, то функция возвращает положительную величину. В противном случае она выдает отрицательное значение. Чтобы рассчитать период времени, в течение которого сотрудник работает на Redwood Realty, функцию MONTHS\_BETWEEN необходимо использовать следующим образом:

```
MONTHS_BETWEEN(SYSDATE, HireDate)
```

Данная команда рассчитывает число месяцев, прошедших с даты найма до текущего дня. Ниже для иллюстрации мы используем вымышленную дату, чтобы полученный вами результат не зависел от текущей даты.

Для того чтобы добавить столбец и обновить его содержимое с помощью подстановочных переменных, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle, введите CLEAR SCREEN и нажмите клавишу <Enter>.
2. Запустите программу Блокнот, а затем сразу сохраните пустой файл как Ch04<имя>SubVars.sql, чтобы использовать его в дальнейшем. (Вместо <имя> подставьте свое имя.)
3. Введите следующий код в Блокноте:

```
ALTER TABLE Agents
    ADD(AnnualBonus NUMBER(5) DEFAULT 0);
```

4. Скопируйте команду в SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить ее. Oracle ответит сообщением “Table altered”.
5. Переключитесь на программу Блокнот и введите приведенную ниже команду UPDATE после команды Alter Table. После этого сохраните файл в программе Блокнот, скопируйте команду UPDATE (только ее) в SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить ее. Закройте Блокнот.

```
UPDATE Agents
    SET AnnualBonus = &BonusAmount
    WHERE MONTHS_BETWEEN('01-JAN-2006', HireDate)
        BETWEEN &LowMonths AND &HighMonths;
```

В ответ SQL\*Plus запросит у вас значение BonusAmount.

6. Введите 500 и нажмите клавишу <Enter>. Oracle отобразит старое и новое значения.
7. Введите 13, нажмите клавишу <Enter>, наберите 24 и нажмите клавишу <Enter>. Oracle ответит строкой “3 rows updated” (рис. 4.14).

Как видите, интерфейс SQL\*Plus подставил 500, 13 и 24 вместо трех подстановочных переменных и передал полученную команду Oracle. После этого база данных обновила три строки, основываясь на модифицированной команде UPDATE:

The screenshot shows the Oracle SQL\*Plus interface. The menu bar includes File, Edit, Search, Options, Help. The command line shows:

```
SQL> ALTER TABLE Agents
  2   ADD (AnnualBonus NUMBER(5) DEFAULT 0);

Table altered.

SQL> UPDATE Agents
  2   SET AnnualBonus = &BonusAmount
  3 WHERE MONTHS_BETWEEN('01-JAN-2006', HireDate)
  4   BETWEEN &LowMonths AND &HighMonths;
Enter value for bonusamount: 500
old  2:   SET AnnualBonus = &BonusAmount
new  2:   SET AnnualBonus = 500
Enter value for lowmonths: 13
Enter value for highmonths: 24
old  4:   BETWEEN &LowMonths AND &HighMonths
new  4:   BETWEEN 13 AND 24

3 rows updated.

SQL>
```

Рис. 4.14. Запуск команды SQL, содержащей подстановочные переменные

```
UPDATE Agents
  SET AnnualBonus = 500
WHERE MONTHS_BETWEEN('01-JAN-2006', HireDate)
      BETWEEN 13 и 24;
```

SQL\*Plus отобразит старые и новые строки SQL\*Plus, а также номера строк, в которые были подставлены значения. Поскольку вы собираетесь выполнить по-добный ввод еще трижды, можете запретить вывод на экран подтверждения в виде сопоставления старых и новых строк, выполнив команду SQL\*Plus SET VERIFY OFF. Поскольку команда SQL по-прежнему находится во внутреннем буфере данного интерфейса, для ее повторного вызова достаточно ввести косую черту (/) и нажать клавишу <Enter>.

Вообще, чтобы выполнить команду, находящуюся в буфере SQL, использовав различные подстановочные значения, необходимо выполнить следующие действия.

1. В окне SQL\*Plus введите SET VERIFY OFF и нажмите клавишу <Enter>.
2. Ведите / и нажмите клавишу <Enter>, чтобы еще раз выполнить команду SQL, находящуюся в буфере.
3. Ведите 700, нажмите клавишу <Enter>, введите 25, нажмите клавишу <Enter>, введите 48 и нажмите клавишу <Enter>.
4. Ведите / и нажмите клавишу <Enter>; введите 1000, нажмите клавишу <Enter>, введите 49, нажмите клавишу <Enter>, введите 72 и нажмите клавишу <Enter>.
5. Ведите / и нажмите клавишу <Enter>; введите 1500, нажмите клавишу <Enter>, введите 73, нажмите клавишу <Enter>, введите 678 и нажмите клавишу <Enter> (рис. 4.15). (Введенное вами значение 678 является просто произвольным большим значением, означающим “очень долго”.)

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> UPDATE Agents
  2  SET AnnualBonus = &BonusAmount
  3 WHERE MONTHS_BETWEEN('01-JAN-2006', HireDate)
  4 BETWEEN &LowMonths AND &HighMonths;
Enter value for bonusamount: 500
Enter value for lowmonths: 13
Enter value for highmonths: 24
3 rows updated.

SQL> SET VERIFY OFF
SQL> /
Enter value for bonusamount: 700
Enter value for lowmonths: 25
Enter value for highmonths: 48
4 rows updated.

SQL> /
Enter value for bonusamount: 1000
Enter value for lowmonths: 49
Enter value for highmonths: 72
4 rows updated.

SQL> /
Enter value for bonusamount: 1500
Enter value for lowmonths: 73
Enter value for highmonths: 678
13 rows updated.

SQL>

```

**Рис. 4.15.** Обработка подстановочных переменных

6. Введите SET VERIFY ON, чтобы восстановить настройки, принятые по умолчанию.
  7. Изучите модификации таблицы Agents, набрав и выполнив в SQL\*Plus следующий код:
- ```

CLEAR SCREEN
SELECT LastName, HireDate, AnnualBonus
FROM Agents
WHERE HireDate IS NOT NULL
ORDER BY HireDate;

```
8. Обратите внимание на то, что величина премии AnnualBonus изменяется в зависимости от времени работы. Это очевидно благодаря сортировке по значению HireDate (рис. 4.16).
  9. Введите Exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus.

Обратите внимание на то, что годовая премия (AnnualBonus) агента Робинсон (Robinson) равна нулю. Значение в соответствующей строке не было обновлено, поскольку данная сотрудница проработала в компании меньше 13 месяцев, если считать, что текущей датой является 1 января 2006 года. Значение равно нулю, поскольку именно таким оно являлось по умолчанию и не изменялось в процессе обработки таблицы.

Альтернативный способ определения подстановочных переменных предлагает команда SQL\*Plus ACCEPT, которая содержит необязательную фразу PROMPT, отобра-

The screenshot shows the Oracle SQL\*Plus interface with the following content:

```

File Edit Search Options Help
SQL> SELECT LastName, HireDate, AnnualBonus
  2  FROM Agents
  3  WHERE HireDate IS NOT NULL
  4  ORDER BY HireDate;

```

| LASTNAME   | HIREDATE  | ANNUALBONUS |
|------------|-----------|-------------|
| Chong      | 30-AUG-95 | 1500        |
| Okiindo    | 20-DEC-95 | 1500        |
| Williams   | 08-JAN-96 | 1500        |
| Soltwedel  | 04-MAY-96 | 1500        |
| Voss       | 05-JUN-96 | 1500        |
| Ailee      | 14-JUL-96 | 1500        |
| St-Onge    | 03-OCT-96 | 1500        |
| Rove       | 13-JUL-97 | 1500        |
| Silverburg | 05-SEP-97 | 1500        |
| Taylor     | 21-SEP-97 | 1500        |
|            | 29-JUN-98 | 1500        |

| LASTNAME   | HIREDATE  | ANNUALBONUS |
|------------|-----------|-------------|
| Gagnon     | 17-JAN-99 | 1500        |
| Romero     | 01-NOV-99 | 1500        |
| Flamenbaum | 08-FEB-00 | 1000        |
| Reed       | 18-SEP-00 | 1000        |
| Carling    | 19-DEC-00 | 1000        |
| Dann       | 02-SEP-01 | 1000        |
| Selby      | 05-JAN-02 | 700         |
| Kellogg    | 05-FEB-02 | 700         |
| Piperova   | 24-DEC-02 | 700         |
| Sheibani   | 18-JUL-03 | 700         |
| Herring    | 07-DEC-03 | 0           |

| LASTNAME | HIREDATE  | ANNUALBONUS |
|----------|-----------|-------------|
|          | 12-JAN-04 | 500         |
|          | 03-FEB-04 | 500         |
| Schutz   | 21-JUN-04 | 500         |
| Dahlen   | 23-MAY-05 | 0           |
| Lewis    | 30-MAY-06 | 0           |
| Weber    | 25-AUG-06 | 0           |
| Townsend | 12-SEP-06 | 0           |
| Robinson | 16-OCT-06 | 0           |

30 rows selected.

Рис. 4.16. Отображение на экране обновленной таблицы

жающую запрос и ожидающую, пока пользователь введет значение. Команда ACCEPT подобна подстановочным переменным, использованным на предыдущих этапах, однако она позволяет вам контролировать порядок, в котором появляются запросы. Поскольку выражения ACCEPT часто используются с записанными процедурами и выражениями PL/SQL, которые рассматриваются в главе 7, мы отложим их обсуждение до вышеизданной главы.

## Удаление строк и усечение таблиц

Когда вам требуется удалить одну строку, несколько или все строки таблицы, применяется команда DELETE (или запрос действия DELETE). При удалении строк из таблицы сама таблица из базы данных не удаляется (для этого используется команда DROP TABLE). Пожалуй, DELETE — это простейшая команда SQL из всех, что вы можете написать. В упрощенном виде ее синтаксис имеет следующий вид:

```
DELETE [FROM] [schema.]<tablename> | <view>
[WHERE <condition>];
```

Обратите внимание на то, что в команде `DELETE` не упоминаются имена столбцов, поскольку она удаляет строки полностью. Хотя оператор `FROM` является необязательным, лучше его все же включать. Вместо аргумента `schema` подставляется имя владельца таблицы или представления. (Представления будут рассмотрены в главе 6.) Необязательный оператор `WHERE` идентифицирует удаляемые строки, т.е. строки, удовлетворяющие условию, заданному аргументом `<condition>`. Будьте осторожны: команда `DELETE` не только простая, но и очень опасная. Это объясняется тем, что в команде достаточно просто случайно опустить оператор `WHERE`, и Oracle удалит все строки заданной таблицы. Представьте, какими будут последствия, если, например, кто-то случайно удалит все имена и адреса клиентов из базы данных Redwood Realty! К счастью, случайно удаленные строки можно восстановить с помощью команды SQL `ROLLBACK`, которая описана ниже, в разделе “Транзакции базы данных”. Как правило, если вы попытаетесь удалить строки, на которые ссылаются другие таблицы (посредством связи первичного и внешнего ключей), Oracle не удалит строки, а сгенерирует вместо этого сообщение об ошибке. В этом состоит вся природа *целостности на уровне ссылок* (*referential integrity*) — она защищает строки, не позволяя их “осиротить”. Создавая таблицу, вы можете наложить на столбец внешнего ключа необязательное условие `ON DELETE CASCADE`. Благодаря этому удаление будет проходить цепочкой, начинающейся с удаляемой строки и переходящей на первичный ключ с последующим удалением всех зависимых строк в других таблицах. Именно такого рода связи существуют между таблицами `LicenseStatus` и `Agents`. Таблица `Agents` содержит столбец внешнего ключа `LicenseStatusID`, который через первичный ключ ссылается на строку в таблице `LicenseStatus`. Если вы собрались удалить строку таблицы `LicenseStatus`, содержащую первичный ключ со значением 1001, Oracle автоматически выполнит операцию удаления по всем строкам таблицы `Agents`, значение внешнего ключа которых равно 1001. Если данная возможность вам не нужна, опустите команду `ON DELETE CASCADE`, когда будете задавать поле внешнего ключа.

## Удаление выбранных строк

Изучая материал данной главы, вы уже несколько раз добавляли новые строки в таблицу `Agents`. Пришло время удалить часть из них.

Для того чтобы удалить строки из таблицы, выполните следующие действия.

1. Запустите SQL\*Plus и войдите в Oracle (если требуется), введите `CLEAR SCREEN` и нажмите клавишу `<Enter>`.
2. Выполните следующую команду `DELETE`:

```
DELETE FROM Agents
WHERE FirstName IS NULL AND LastName IS NULL;
```

Oracle ответит строкой-подтверждением “2 rows deleted”. Две удаленные строки характеризуются пустыми полями имени и фамилии (FirstName и LastName равны null). Теперь этих строк больше нет.

3. Введите и выполните следующую команду:

```
DELETE FROM Agents  
WHERE AgentID = 515602;
```

Как и ранее, Oracle сообщает, что одна строка была удалена.

4. Введите и выполните следующую команду:

```
DELETE FROM Agents  
WHERE Title = 'Salesperson';
```

На последнем этапе Oracle выдает сообщение “0 rows deleted”. Обычно это означает, что команда DELETE имеет правильный синтаксис, но ни одна строка не удовлетворяет заданным условиям. Это, в свою очередь, указывает на наличие лiteralного значения с ошибкой или неверного оператора отношения. В связи с этим напомним, что ранее вы обновили столбец Title и теперь он содержит только прописные буквы.

Роберт Стирлинг вынужден временно уменьшить число агентов в штате фирмы. Он решает оправить во временный отпуск агентов, проработавших в компании Redwood Realty меньше всех, т.е. сотрудников, нанятых в 2006 году. Напишите запрос действия DELETE, который удаляет строки, удовлетворяющие указанному критерию.

Для того чтобы удалить выбранные строки, основываясь на значении HireDate, выполните следующие действия.

1. Введите и выполните следующую команду:

```
DELETE FROM Agents  
WHERE HireDate >= TO_DATE('2006/01/01', 'YYYY/MM/DD');
```

Oracle удалит четыре строки.

2. Удалите строки, в которых значение HireDate неизвестно (равно NULL). Введите и выполните следующую команду:

```
DELETE FROM Agents  
WHERE HireDate IS NULL;
```

На этот раз Oracle удалит девять строк (рис. 4.17), т.е. в таблице имелось девять строк, столбец HireDate которых содержит значение NULL.

3. SQL\*Plus не закрывайте.

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays several SQL commands being executed:

```

SQL> DELETE FROM Agents
2 WHERE FirstName IS NULL AND LastName IS NULL;
2 rows deleted.

SQL> DELETE FROM Agents
2 WHERE AgentID = 515602;
1 row deleted.

SQL> DELETE FROM Agents
2 WHERE Title = 'Salesperson';
0 rows deleted.

SQL> DELETE FROM Agents
2 WHERE HireDate >= TO_DATE('2006/01/01', 'YYYY/MM/DD');
4 rows deleted.

SQL> DELETE FROM Agents
2 WHERE HireDate IS NULL;
9 rows deleted.

SQL>

```

Рис. 4.17. Удаление выбранных строк таблицы

## Удаление всех строк

Удалить все строки, не затронув структуру таблицы, можно двумя способами. Если вы опустите оператор `WHERE` в команде `DELETE`, то из таблицы будут удалены все строки. Например, приведенная ниже команда удаляет все строки из таблицы `Agents`, оставляя ее пустой.

```
DELETE FROM Agents;
```

Удаление строк из большой таблицы, содержащей тысячи строк, является “дорогой” операцией, поскольку Oracle поддерживает копии всех удаленных транзакций на случай, если вам потребуется отменить внесенные изменения. Если таблица большая, этот процесс может оказаться длительным и потребовать много места, пока вы не завершите операцию удаления с помощью ТС<sup>1</sup>-выражения `COMMIT`, описанного ниже в данной главе.

Иногда бывает нужно удалить все записи из таблицы, причем точно известно, что восстанавливать их не потребуется. Допустим, например, что вы создали новую таблицу, взяв устаревшие записи из другой таблицы. Или, например, вы переместили данные из таблицы в несколько других таблиц, чтобы нормализовать базу данных, но при этом таблица еще нужна для ввода данных. В таких случаях вместо команды `DELETE` можно использовать выражение `TRUNCATE TABLE`. При выполнении этой команды Oracle удаляет все данные из таблицы, в том числе не сохраняет информацию, необходимую для отмены этих данных. Когда SQL Server *усекает* (*truncate*) таблицу, записи удаляются безвозвратно, однако структура таблицы остается. Общий вид команды `TRUNCATE TABLE` достаточно прост:

<sup>1</sup>Transaction Control – управление транзакциями.

The screenshot shows the Oracle SQL\*Plus interface. The menu bar includes File, Edit, Search, Options, and Help. The command line area contains the following SQL statements:

```
SQL> ALTER TABLE Agents
  2  DISABLE CONSTRAINT fk_AgentsLicenseStatusID;
Table altered.

SQL> TRUNCATE TABLE Agents;
Table truncated.

SQL>
```

Рис. 4.18. Усечение таблицы Agents

```
TRUNCATE TABLE <tablename>;
```

Если на таблицу наложены условия, задающие внешние ключи (без использования опции ON DELETE CASCADE), вы не можете усечь таблицу. Вначале вам следует отключить условие, используя команду ALTER TABLE. Например, вы можете удалить внешний ключ таблицы Agents, выполнив команду

```
ALTER TABLE Agents
DISABLE CONSTRAINT fk_AgentsLicenseStatusID;
```

Для примера мы удалим все строки из таблицы Agents, используя команду TRUNCATE TABLE.

Для того чтобы изменить условие целостности на уровне ссылок, а затем удалить все строки таблицы, выполните следующие действия.

1. В окне SQL\*Plus выполните команду

```
ALTER TABLE Agents
DISABLE CONSTRAINT fk_AgentsLicenseStatusID;
```

2. Проведите усечение таблицы, выполнив код

```
TRUNCATE TABLE Agents;
```

Oracle ответит сообщением “Table truncated” (рис. 4.18).

## Слияние строк

Команда SQL MERGE позволяет соединять строки исходной таблицы с целевой таблицей, объединяя в себе функции команд INSERT, UPDATE и DELETE. Синтаксис выражения MERGE выглядит следующим образом:

```
MERGE INTO <target-tablename>
USING <source-tablename or query>
ON (<condition>)
[WHEN MATCHED THEN UPDATE <set-clause>
 [DELETE <condition>]]
[WHEN NOT MATCHED THEN INSERT <insert-clause>];
```

Типичной ситуацией, в которой удобно использовать команду MERGE, является обновление таблицы с данными о товарах с указанием новых цен и количества единиц товара согласно таблице транзакций. Новый товар, указанный в таблице транзакций, вставляется в основную таблицу товаров. При этом согласно записям таблицы транзакций, соответствующим товарам основной таблицы, обновляются столбцы количества и цены основной таблицы товаров. Разумеется, все, что нам нужно, можно реализовать, добавляя, удаляя и модифицируя строки с помощью DML<sup>2</sup>-команд INSERT, UPDATE и DELETE. Несмотря на это, удобнее воспользоваться командой MERGE, собрав в одном выражении все, что потребовало бы нескольких DML-команд. Поскольку MERGE соединяет в себе несколько выражений UPDATE и INSERT, ее иногда неформально называют “upsert” (UPdate + inSERT). Ниже приведен пример, иллюстрирующий работу этой команды.

Оператор MERGE INTO задает имя таблицы, которую Oracle дополняет строками. Оператор USING ON определяет две задействованные таблицы (исходную и целевую) и два столбца (поля двух первичных ключей), которые будут сравниваться на предмет поиска соответствующих значений. Оператор WHEN MATCHED задает, что делать при выполнении условия USING ON, т.е. когда первичные ключи совпадают. Оператор WHEN NOT MATCHED задает, что предпримет Oracle, когда строка исходной таблицы не совпадет с соответствующей строкой целевой таблицы. В большинстве случаев строка, для которой нет соответствия, просто вставляется в целевую таблицу.

Таблица LicenseStatus содержит два столбца и шестнадцать строк, показанных на рис. 4.19 (под заголовком “Целевая таблица LicenseStatus”). Таблица UpdateLicenseStatus (под заголовком “Исходная таблица UpdateLicenseStatus” на рис. 4.19) содержит новые строки, которые необходимо добавить в таблицу LicenseStatus, а также существующие строки (для которых значения ключа LicenseStatusID имеют соответствующие аналоги в таблице LicenseStatus), которые обновят значения, существующие в таблице LicenseStatus (см. “Таблица LicenseStatus после выполнения команды MERGE” на рис. 4.19). Ниже показана команда MERGE, при выполнении которой целевая таблица LicenseStatus обновляется с использованием таблицы UpdateLicenseStatus.

```
MERGE INTO LicenseStatus LS
USING UpdateLicenseStatus ULS
    ON (LS.LicenseStatusID = ULS.LicenseStatusID)
    WHEN MATCHED THEN UPDATE SET LS.StatusText = ULS.StatusText
    WHEN NOT MATCHED THEN INSERT (LS.LicenseStatusID, LS.StatusText)
        VALUES (ULS.LicenseStatusID, ULS.StatusText);
```

В данном коде LS и ULS являются псевдоименами. В базах данных *псевдоименами* (alias) называются альтернативные имена таблицы или столбца. Псевдоимена

---

<sup>2</sup>Data Manipulation Language — язык манипулирования данными.

Исходная таблица UpdateLicenseStatus

|      |                       |
|------|-----------------------|
| 1004 | Passed Away           |
| 1005 | Expired--Fee Not Paid |
| 1101 | License Probationary  |
| 1105 | License Pending       |

Целевая таблица LicenseStatus

|      |                           |
|------|---------------------------|
| 1001 | Licensed                  |
| 1002 | Licensed NBA              |
| 1003 | Canceled Officer          |
| 1004 | Deceased                  |
| 1005 | Expired                   |
| 1006 | Government Service        |
| 1007 | Military Service          |
| 1008 | Conditional Suspension    |
| 1009 | Restricted                |
| 1010 | Revoked                   |
| 1011 | Flag Suspended            |
| 1012 | Voided                    |
| 1013 | Withheld Denied           |
| 1014 | 17520 FC Suspended        |
| 1015 | 11350.6 W and I Suspended |
| 1016 | Surrendered               |

Таблица UpdateLicenseStatus после применения команды MERGE

|             |                              |
|-------------|------------------------------|
| 1001        | Licensed                     |
| 1002        | Licensed NBA                 |
| 1003        | Canceled Officer             |
| 1004        | <b>Passed Away</b>           |
| 1005        | <b>Expired--Fee Not Paid</b> |
| 1006        | Government Service           |
| 1007        | Military Service             |
| 1008        | Conditional Suspension       |
| 1009        | Restricted                   |
| 1010        | Revoked                      |
| 1011        | Flag Suspended               |
| 1012        | Voided                       |
| 1013        | Withheld Denied              |
| 1014        | 17520 FC Suspended           |
| 1015        | 11350.6 W and I Suspended    |
| 1016        | Surrendered                  |
| <b>1101</b> | <b>License Probationary</b>  |
| <b>1105</b> | <b>License Pending</b>       |

Обновленные  
столбцы

Вставленные  
строки

**Рис. 4.19.** Пример использования команды MERGE

часто определяются и используются для сокращения имени таблицы или столбца при последующем обращении к нему. В предыдущем примере псевдоимя LS короче, чем LicenseStatus, используемое в четырех местах, а ULS — гораздо короче UpdateLicenseStatus, которое также четырежды используется в команде.

Команда MERGE указывает Oracle обновить выбранные поля, если первичный ключ исходной таблицы совпадает с первичным ключом целевой таблицы. Оператор WHEN NOT указывает Oracle вставить строку исходной таблицы, если ее первичный ключ не имеет аналога в целевой таблице. Если вы хотите дополнительно удалить строки из целевой таблицы, то в оператор WHEN MATCHED необходимо включить команду DELETE.

Стирлинг Леонард (Stirling Leonard) желает, чтобы вы обновили две строки и вставили две новые строки в таблицу LicenseStatus. Обновить необходимо строки с первичными ключами 1004 и 1005 и значениями Passed Away и Expired – Fee Not Paid поля LicenseText. Помимо этого, необходимо вставить две строки — вторую и третью записи под заголовком “Исходная таблица UpdateLicenseStatus” (см. рис. 4.19). Их первичные ключи, 1101 и 1105, не имеют аналогов в таблице LicenseStatus, поэтому Oracle добавляет эти две строки в таблицу LicenseStatus.

Для того чтобы создать исходную таблицу UpdateLicenseStatus, которая позже будет задействована в операции MERGE, выполните следующие действия.

1. В окне SQL\*Plus очистите экран (введите CLEAR SCREEN и нажмите клавишу <Enter>). Наберите и выполните нижеприведенную команду, чтобы создать исходную таблицу, содержащую строки, обновляющие таблицу LicenseStatus. Как обычно, вы можете отредактировать команды в программе Блокнот, а затем скопировать их в SQL\*Plus.

```
CREATE TABLE UpdateLicenseStatus
  (LicenseStatusID INTEGER PRIMARY KEY,
   StatusText NVARCHAR2(25));
```

Oracle выведет сообщение, что операция прошла успешно: “Table created”.

2. Заполните таблицу UpdateLicenseStatus набрав и выполнив следующие команды INSERT (в конце каждой строки нажмайтесь клавишу <Enter>):

```
INSERT INTO UpdateLicenseStatus VALUES (1004, 'Passed Away');
INSERT INTO UpdateLicenseStatus VALUES (1005,
  'Expired--Fee Not Paid');
INSERT INTO UpdateLicenseStatus VALUES (1101,
  'License Probationary');
INSERT INTO UpdateLicenseStatus VALUES (1105, 'License Pending');
```

После каждой команды INSERT Oracle будет отвечать сообщением “1 row created”.

Получив исходную таблицу, содержащие строки, которые нужно модифицировать и вставить в таблицу UpdateLicenseStatus, вы можете написать команду MERGE.

Для того чтобы выполнить команду MERGE, обновляющую и добавляющую строки в целевую таблицу, выполните следующие действия.

1. Внимательно введите следующий код в программе Блокнот, скопируйте его в SQL\*Plus и выполните (если потребуется, исправьте ошибки в программе Блокнот и выполните модифицированную команду):

```
MERGE INTO LicenseStatus LS
USING UpdateLicenseStatus ULS
  ON (LS.LicenseStatusID = ULS.LicenseStatusID)
WHEN MATCHED THEN
  UPDATE SET LS.StatusText = ULS.StatusText
WHEN NOT MATCHED THEN
  INSERT (LS.LicenseStatusID, LS.StatusText)
  VALUES (ULS.LicenseStatusID, ULS.StatusText);
```

Oracle ответит строкой-подтверждением “4 rows merged” (рис. 4.20).

2. Выведите на экран строки обновленной таблицы LicenseStatus, выполнив команду

```
SELECT *
  FROM LicenseStatus
 ORDER BY LicenseStatusID;
```

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE TABLE UpdateLicenseStatus
  2  (LicenseStatusID INTEGER PRIMARY KEY,
  3   StatusText      NVARCHAR2(25)
  4 );
Table created.

SQL> INSERT INTO UpdateLicenseStatus VALUES (1004, 'Passed Away');

1 row created.

SQL> INSERT INTO UpdateLicenseStatus VALUES (1005, 'Expired--Fee Not Paid');

1 row created.

SQL> INSERT INTO UpdateLicenseStatus VALUES (1101, 'License Probationary');

1 row created.

SQL> INSERT INTO UpdateLicenseStatus VALUES (1105, 'License Pending');

1 row created.

SQL> MERGE INTO LicenseStatus LS
  2  USING UpdateLicenseStatus ULS
  3  ON (LS.LicenseStatusID = ULS.LicenseStatusID)
  4  WHEN MATCHED THEN
  5    UPDATE SET LS.StatusText = ULS.StatusText
  6  WHEN NOT MATCHED THEN
  7    INSERT (LS.LicenseStatusID, LS.StatusText)
  8    VALUES (ULS.LicenseStatusID, ULS.StatusText);

4 rows merged.

SQL>

```

**Рис. 4.20.** Использование команды MERGE для модификации таблицы

3. Сравните полученный список со списком, показанным на рис. 4.19. В конце должны появиться две новые строки, первичные ключи которых равны 1101 и 1105. Строки с первичными ключами 1004 и 1005 должны содержать модифицированные значения.
4. Удалите уже ненужную таблицу `UpdateLicenseStatus`, выполнив команду

```
DROP TABLE UpdateLicenseStatus;
```
5. Выходите из Oracle и SQL\*Plus, набрав `Exit` и нажав клавишу <Enter>.

## Транзакции базы данных

Важной концепцией, которую следует понимать при использовании выражений языка манипулирования данными (Data Manipulation Language — DML), является *транзакция базы данных* — группа выражений, представляющих логический блок работы. Это один или несколько неделимых наборов выражений SQL, которые либо все должны выполниться правильно и зафиксироваться после этого, либо отмениться всем блоком. Для иллюстрации данной концепции обычно используют события, происходящие, когда вы оплачиваете счет чеком на имя кого-то, имеющего счет в вашем банке. Когда в банке обрабатывают чек, с текущего счета вычитают (записывают в дебет) необходимое количество денег, уменьшая его баланс. Одновременно банк добавляет

(записывает в кредит) такую же сумму подателю чека. Обе операции (вычитания и сложения) должны завершиться успешно, в противном случае обе транзакции необходимо аннулировать. Если этого не сделать, деньги будут потеряны где-то в киберпространстве.

Описанная ситуация — это пример двух команд UPDATE, которые необходимо сгруппировать в одну транзакцию, чтобы гарантировать либо успешное выполнение, либо отмену обеих операций. Довольно часто транзакции состоят из любого числа операций INSERT, DELETE и UPDATE. Для определения транзакций применяются выражения COMMIT, ROLLBACK и SAVEPOINT.

Когда в ходе сеанса вы выполняете DML-команды (например, INSERT или DELETE), изменения сразу видны в таблицах базы данных. Несмотря на это, Oracle не позволяет, чтобы другой *сеанс* (отдельное соединение с сервером базы данных Oracle) видел изменения или обращался к модифицированным строкам, пока изменения не зафиксированы. Помимо прочих плюсов, блокировка строк не позволяет одному пользователю изменить строку таблицы, если в это же время данную строку изменяет другой пользователь. Если бы подобный сценарий был разрешен, то изменения, внесенные в таблицу одним пользователем, могли бы сводиться на нет другим пользователем. Какой-либо явной команды запуска транзакций не существует. Транзакция начинается автоматически, когда вы входите в Oracle посредством SQL\*Plus или iSQL\*Plus, и заканчивается, когда вы фиксируете текущую транзакцию. После завершения одной транзакции автоматически начинается следующая.

## Фиксация

Команда COMMIT делает неизменными любые изменения базы данных. На время транзакции Oracle блокирует все модифицированные строки. После ввода команды COMMIT блокировка снимается, и другие пользователи могут изучать или модифицировать эти строки. Выражения языка определения данных (Data Definition Language — DDL), подобные CREATE TABLE, CREATE SEQUENCE и DROP TABLE, а также выражения языка управления данными (Data Control Language — DCL), подобные GRANT, указывают Oracle неявно выполнять команду COMMIT. Например, когда вы создаете таблицу, она фиксируется сразу после выполнения команды-определения и не требует ввода выражения COMMIT. Кроме того, Oracle неявно выполняет команду COMMIT, когда вы выходите из базы данных. Например, если с момента последней фиксации транзакции с помощью команды COMMIT вы вставили в таблицу 25 строк и обновили 12 других строк, данные изменения будут неизменными в момент вашего выхода из Oracle.

После выполнения одного или нескольких выражений DML вы можете *записать* (commit) все изменения, внесенные в ходе транзакции. Все незафиксированные ранее изменения становятся постоянными, и отменить их уже нельзя. Синтаксис соответствующей команды COMMIT прост невероятно:

```
COMMIT;
```



**Рис. 4.21.** Иллюстрация транзакций

Транзакция измеряется с последнего момента выполнения команды COMMIT в текущем сеансе до настоящего момента. Для иллюстрации на рис. 4.21 показана временная диаграмма, демонстрирующая группировку отдельных выражений DML в транзакции.

## Откат

При *откате* (rollback) базы данных вы отменяете все приостановленные изменения. Если вы начали транзакцию, которая завершилась неверно или дала неправильный результат, то вы можете выполнить откат и восстановить исходные данные. Для этого применяется команда SQL ROLLBACK, которая возвращает в исходное состояние все команды DML, относящиеся к текущей транзакции, — к началу транзакции или к точке сохранения. Команда ROLLBACK снимает блокировку со всех приостановленных изменений, заданных в ходе транзакции, и затронутых этими изменениями записей. Кроме того, данная команда завершает транзакцию и стирает все точки сохранения. Отметим, что хотя команда ROLLBACK применяется для отмены DML-выражений, она не может отменять DDL-выражения. Любое DDL-выражение (CREATE TABLE и т.д.) можно обратить вспять только с помощью подходящего выражения DROP. Синтаксис команды ROLLBACK достаточно прост:

```
ROLLBACK [WORK] [TO [SAVEPOINT] <savepoint-name>];
```

Аргумент <savepoint-name> — это имя, определенное ранее в текущей транзакции (подробнее об этом — ниже, в разделе “Точки сохранения”). Например, вы можете ввести команду ROLLBACK TO SAVEPOINT Alpha23, где Alpha23 — точка сохранения, определенная в транзакции. После этого произойдет следующее:

- изменения, внесенные в базу данных с момента заданной точки сохранения, отменяются;
- все точки сохранения, отмеченные после Alpha23, стираются;
- со всех строк и таблиц, заблокированных с момента указанной точки сохранения, снимается блокировка.

Команда ROLLBACK или COMMIT затрагивает только выражения SQL, относящиеся к текущей транзакции. Напомним, что транзакция начинается с одного из перечисленных ниже событий.

- Вы соединяетесь с Oracle и выполняете команду DML.
- Вы завершаете предыдущую транзакцию, а затем выполняете другую команду DML.

Завершением транзакции служит одно из указанных ниже событий.

- Вы выполнили команду DML, но она не была успешно завершена. Oracle выполняет команду ROLLBACK только для выражений, давших сбой.
- Вы выполняете команду COMMIT или ROLLBACK.
- Вы обычным образом отключаетесь от Oracle.
- Вы вводите DCL-выражение, например GRANT.
- Вы выполняете DDL-выражение, например CREATE TABLE.

## Точки сохранения

*Точками сохранения* называются именованные точки в пределах транзакции, которые позволяют откатывать изменения до любого заданного момента транзакции. Точки сохранения полезны, когда есть длинная транзакция и желательно, чтобы в случае ошибки можно было отменить небольшую часть, а не всю транзакцию. Это позволяет не возвращаться слишком далеко назад и не вводить повторно огромное число DML-выражений. Синтаксис команды SAVEPOINT, задающей точку сохранения, выглядит так:

```
SAVEPOINT <savepointname>;
```

Если вы выполняете команду ROLLBACK, не задавая точку сохранения, то все DML-выражения откатываются до начала транзакции и все имена точек сохранения стираются. На рис. 4.22 приведен пример транзакции, содержащей DML-выражения, команды SAVEPOINT и операцию ROLLBACK.

На рис. 4.22 начало транзакции обозначено вверху — сразу после входа в систему. Первая точка сохранения, SaveFirst, устанавливается после ввода информации по агенту с фамилией Карлинг (Carling). Далее мы добавляем информацию о брокере по фамилии Восс (Voss) и создаем вторую точку сохранения, SaveSecond. Далее название (title) Восса меняется на Salesperson (“продавец”). При выполнении команды ROLLBACK отменяются все изменения, внесенные после второй точки сохранения, т.е. Восс снова именуется брокером (Broker). После ввода команды SELECT мы видим две записи, добавленные после начала транзакции, в которых не отражены изменения, внесенные посредством команды UPDATE, а затем отмененные. Наконец,

The screenshot shows a session in Oracle SQL\*Plus. The user performs several operations:

- `INSERT INTO Agents (AgentID, LastName, Title)` (1 row created)
- `SAVEPOINT SaveFirst;` (Создание точки сохранения)
- `INSERT INTO Agents (AgentID, LastName, Title)` (1 row created)
- `SAVEPOINT SaveSecond;` (Создание точки сохранения)
- `UPDATE Agents SET Title = 'SalesPerson' WHERE AgentID = 10913;` (1 row updated)
- `SELECT LastName, Title FROM Agents;`

| LASTNAME | TITLE       |
|----------|-------------|
| Carling  | SalesPerson |
| Voss     | SalesPerson |

- `ROLLBACK TO SaveSecond;` (Создание точки сохранения)
- `Commit complete.`
- `SELECT LastName, Title FROM Agents;`

| LASTNAME | TITLE       |
|----------|-------------|
| Carling  | SalesPerson |
| Voss     | Broker      |

- `ROLLBACK TO SaveFirst;` (Откат изменений до последней точки сохранения)
- `Commit complete.`
- `SQL>`

Рис. 4.22. Использование точек сохранения в транзакции

команда `COMMIT` закрывает (фиксирует) текущую транзакцию, начинает новую транзакцию и делает постоянными два указанных действия `INSERT`.

Напомним, что в предыдущих сеансах работы с Oracle вы удаляли все строки из таблицы `Agents`, выполняя команду `TRUNCATE TABLE Agents`. Ниже вы поработаете с выражениями языка манипулирования данными `ROLLBACK` и `COMMIT`, позволяющими лучше понять выражения языка управления транзакциями (*Transactions Control Language — TCL*). Как обычно, мы рекомендуем внимательно вводить все команды в программе Блокнот, копировать их в `SQL*Plus`, а затем выполнять. Если Oracle выдаст сообщение об ошибке, исправьте команду в Блокноте, а затем скопируйте правильный вариант в `SQL*Plus` и выполните его.

Итак, чтобы выполнить DML-выражения, создать точки сохранения и отметить транзакции, выполните следующие действия.

- Запустите `SQL*Plus`, войдите в Oracle и введите следующий код `SQL*Plus`, который отформатирует вывод на экран (нажимайте клавишу `<Enter>` в конце каждой строки):

```
CLEAR SCREEN
COLUMN AgentID FORMAT 999999
```

```
COLUMN FirstName FORMAT A12
COLUMN LastName FORMAT A12
COLUMN Gender FORMAT A6
```

2. Введите и выполните следующие команды:

```
INSERT INTO Agents (AgentID, FirstName, LastName, Gender)
    VALUES (10429, 'Elizabeth', 'Dahlen', NULL);
SAVEPOINT InsertOne;
INSERT INTO Agents (AgentID, FirstName, LastName, Gender)
    VALUES (10497, 'Ramanathan', 'Rowe', 'M');
INSERT INTO Agents (AgentID, FirstName, LastName, Gender)
    VALUES (10849, 'Heather', 'Sheibani', NULL);
UPDATE Agents SET Gender = 'M' WHERE LastName = 'Dahlen';
SAVEPOINT InsertTwo;
```

Если вы все правильно ввели, Oracle ответит сообщениями “1 row created” и “Savepoint created” после каждой команды INSERT и команды SAVEPOINT соответственно.

3. Выведите на экран строки таблицы Agents, набрав и выполнив команду

```
SELECT AgentID, FirstName, LastName, Gender FROM Agents;
```

Обратите внимание на то, что в столбце Gender записи Sheibani находится значение NULL (см. рис. 4.23).

4. Введите и выполните нижеприведенные команды. Обратите внимание на изменения. Вы откатили состояние базы данных до точки сохранения InsertOne, т.е. до момента, предшествовавшего двум последним командам INSERT и команде UPDATE.

```
CLEAR SCREEN
ROLLBACK TO InsertOne;
SELECT AgentID, FirstName, LastName, Gender FROM Agents;
```

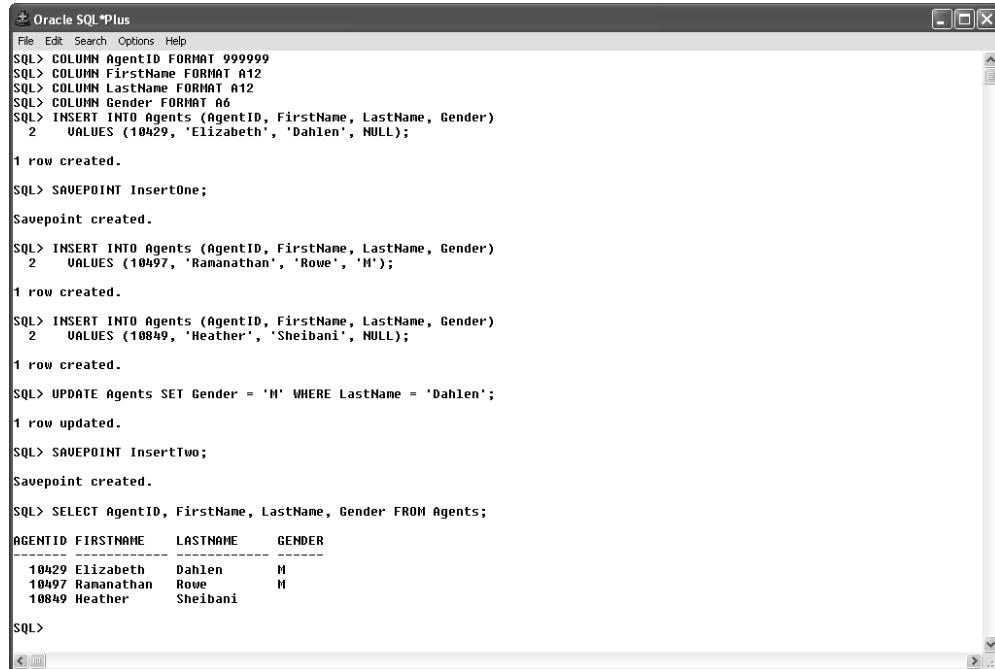
5. Зафиксируйте изменения и попытайтесь произвести откат, выполнив в SQL\*Plus следующие команды:

```
COMMIT;
SELECT AgentID, FirstName, LastName, Gender FROM Agents;
ROLLBACK;
SELECT AgentID, FirstName, LastName, Gender FROM Agents;
```

Результат выполнения приведенных команд показан на рис. 4.24. Как видите, после выполнения команды COMMIT Oracle делает все изменения постоянными, и пути назад нет.

6. Удалите все записи из таблицы Agents, введите Exit и нажмите клавишу <Enter>, чтобы выйти из Oracle. В момент, когда вы завершите сеанс, Oracle выполнит неявную команду COMMIT:

```
DELETE FROM Agents;
exit
```



The screenshot shows a window titled 'Oracle SQL\*Plus'. Inside, SQL commands are being run against a database. The commands include setting column formats, inserting rows into the 'Agents' table, creating savepoints, updating a row, and selecting data from the 'Agents' table. The output shows the creation of 1 row, the creation of savepoints, the update of 1 row, and the final state of the Agents table.

```

File Edit Search Options Help
SQL> COLUMN AgentID FORMAT 999999
SQL> COLUMN FirstName FORMAT A12
SQL> COLUMN LastName FORMAT A12
SQL> COLUMN Gender FORMAT A6
SQL> INSERT INTO Agents (AgentID, FirstName, LastName, Gender)
2  VALUES (10429, 'Elizabeth', 'Dahlen', NULL);
1 row created.

SQL> SAVEPOINT InsertOne;
Savepoint created.

SQL> INSERT INTO Agents (AgentID, FirstName, LastName, Gender)
2  VALUES (10497, 'Ramanathan', 'Rowe', 'M');
1 row created.

SQL> INSERT INTO Agents (AgentID, FirstName, LastName, Gender)
2  VALUES (10849, 'Heather', 'Sheibani', NULL);
1 row created.

SQL> UPDATE Agents SET Gender = 'M' WHERE LastName = 'Dahlen';
1 row updated.

SQL> SAVEPOINT InsertTwo;
Savepoint created.

SQL> SELECT AgentID, FirstName, LastName, Gender FROM Agents;
AGENTID FIRSTNAME LASTNAME GENDER
----- 
 10429 Elizabeth Dahlen M
 10497 Ramanathan Rowe M
 10849 Heather Sheibani

SQL>

```

Рис. 4.23. Модификация таблицы Agents

## Создание и использование обработчиков

Обработчик событий (trigger) базы данных соотносится с таблицей, представлением (мы рассмотрим его в главе 6), схемой или базой данных. Она выполняется (*срабатывает*) автоматически, когда с этой таблицей, представлением, схемой или базой данных происходит определенное событие. В данной главе вы узнаете об обработчиках событий, соотнесенных с таблицами базы данных. Обработчики событий, присоединенные к таблицам, наблюдают за выполнением DML-выражений `INSERT`, `UPDATE` или `DELETE`. Обработчики событий часто используются для наблюдения за изменениями, происходящими в определенных столбцах заданных таблиц. Для написания данных процедур применяется язык программирования Oracle — PL/SQL. Готовый обработчик событий Oracle компилирует и записывает в словаре данных.

## Введение в обработчики событий

Обработчики событий позволяют реализовывать условия целостности на уровне ссылок, не используя внешние ключи, хотя мы рекомендуем все же использовать встроенные условия целостности Oracle в виде внешних ключей. Обработчики событий особенно полезны, когда вам требуется предотвратить появление неверных записей

The screenshot shows a session in Oracle SQL\*Plus. The user has performed the following steps:

```

SQL> ROLLBACK TO InsertOne;
Rollback complete.

SQL> SELECT AgentID, FirstName, LastName, Gender FROM Agents;
AGENTID FIRSTNAME LASTNAME GENDER
----- ----- -----
10429 Elizabeth Dahlen

SQL> COMMIT;
Commit complete.

SQL> SELECT AgentID, FirstName, LastName, Gender FROM Agents;
AGENTID FIRSTNAME LASTNAME GENDER
----- ----- -----
10429 Elizabeth Dahlen

SQL> ROLLBACK;
Rollback complete.

SQL> SELECT AgentID, FirstName, LastName, Gender FROM Agents;
AGENTID FIRSTNAME LASTNAME GENDER
----- ----- -----
10429 Elizabeth Dahlen

```

**Рис. 4.24.** Завершение транзакции с помощью команды COMMIT

в таблицах, выполнить проверку допустимости значений столбца или обеспечить безопасность доступа. Допустим, например, что деловой регламент компании Redwood Realty задает, что дата выдачи лицензии `LicenseDate` должна предшествовать текущей дате, — вы легко напишете соответствующую процедуру. Точно так же обработчик событий может гарантировать, что дата окончания срока действия лицензии должна соответствовать моменту, предшествующему дате ее выдачи, и вы легко реализуете эту проверку перед вводом информации о новом агенте в таблицу `Agents`.

Другой популярной сферой применения обработчиков событий является отслеживание модификаций таблиц базы данных. Определив обработчики событий для некоторых таблиц, Oracle может регистрировать данные, касающиеся любых модификаций этих таблиц. Когда пользователь Oracle выполняет команду `INSERT`, `UPDATE` или `DELETE`, обработчик событий, присоединенный к соответствующей таблице, может записывать информацию в контрольной таблице. Информация, хранимая в контрольной таблице, может включать имя пользователя Oracle, модифицирующего таблицу, имя таблицы, тип модификации таблицы, значения столбцов до и после операции, а также метки даты и времени. Это очень полезно, если требуется отследить, кто и когда пытается без авторизации открыть таблицу и модифицировать ее.

Обработчики событий могут срабатывать до, после или вместо операции DML. Обработчик событий, срабатывающий до события, может собрать данные о строке перед началом модификации, а обработчик, срабатывающий после события, может получить новые, модифицированные значения строки. Обработчик, срабатывающий вместо события DML, полезен, когда требуется запретить выбранные модификации таблицы.

Код, определяющий обработчик событий, может запускаться один раз для каждой задействованной строки или один раз за все время выполнения команды (независимо от числа затронутых строк). Обработчик событий, срабатывающий для каждой строки, называется *обработчиком событий на уровне строки* (row-level trigger), а обработчик, срабатывающий один раз на все событие DML при любом числе затронутых строк, называется *обработчиком событий на уровне выражений* (statement-level trigger). Например, если вы удалили двенадцать строк таблицы, а с операцией удаления соотнесен обработчик на уровне строки, он запустится двенадцать раз. Если же обработчик определялся на уровне выражения, он сработает лишь один раз.

Чтобы определить обработчик событий, используйте выражение CREATE TRIGGER, которое имеет следующий общий синтаксис:

```
CREATE [OR REPLACE] TRIGGER <trigger-name>
{BEFORE | AFTER | INSTEAD OF} <trigger-event> ON <table_name>
[ FOR EACH ROW [WHEN <condition>] ]
[DECLARE
    <declaration statements>]
BEGIN
    <trigger-body>
END;
```

В данной команде CREATE указывает, что обработчик событий является новым, а REPLACE указывает, что он заменяет существующий обработчик. Уточнение REPLACE является необязательным, однако его следует использовать только для модификации обработчика событий. При его применении Oracle заменяет существующий обработчик с указанным именем. Тем не менее, если вы создадите обработчик событий, а позже замените его процедурой, определенной для другой таблицы, Oracle сгенерирует ошибку. Вернемся к синтаксису команды CREATE TRIGGER. Аргумент <trigger name> — это имя обработчика событий. BEFORE или AFTER задает, когда срабатывает обработчик — до или после события. INSTEAD OF указывает, что обработчик запускается вместо события DML, однако данную опцию можно использовать только для представлений базы данных (подробнее об этом — в главе 6).

**Совет.** Мы предлагаем использовать следующие имена обработчиков событий: <table-name>\_{bi|bu|bd|ai|au|ad}\_trg, где <table-name> — имя таблицы, trg — сокращение от “trigger” (“обработчик событий”), bi — “before insert” (“перед INSERT”), bu — “before update” (“перед UPDATE”), ai — “after insert” (“после INSERT”) и т.д. Например, обработчик событий AFTER UPDATE, присоединенный к таблице Agents, будет называться Agents\_au\_trg. Имена обработчиков должны быть уникальными для всех обработчиков схемы, однако они не обязательно должны быть уникальными относительно других объектов. Другими словами, два обработчика событий не могут иметь одинаковое имя, однако таблица и обработчик вполне могут именоваться одинаково (хотя это и не рекомендуется).

Вернемся к синтаксису. Аргумент `<trigger event>` задает событие, вызывающее срабатывание процедуры; это может быть `INSERT`, `UPDATE`, `DELETE` или любая комбинация данных команд, разделенная операцией “`OR`”. Аргумент `<table-name>` обозначает имя таблицы, к которой присоединен обработчик событий. Опция `FOR EACH ROW` указывает, что задается обработчик, присоединенный на уровне строк, т.е. он будет срабатывать для *каждой* задействованной строки. Если вы пропустите эту опцию, получится процедура, присоединенная на уровне выражений. Необязательная фраза “`WHEN <condition>`” содержит булево выражение, ограничивающее фактическое выполнение тела присоединенного обработчика событий. Аргумент `<trigger-body>` состоит из кода PL/SQL, в котором, собственно, и реализован обработчик событий.

Существует несколько ограничений, которые следует рассмотреть перед созданием обработчиков событий. Перечислим некоторые из них. Процедура или функция, присоединенная к объекту, не может выполнять выражения управления транзакциями (`COMMIT`, `ROLLBACK` и `SAVEPOINT`). Любое выражение, выполненное обработчиком событий, является частью транзакции. Действия присоединенной процедуры фиксируются или откатываются вместе с событием, вызвавшим срабатывание обработчика.

В теле обработчика событий вы можете обращаться к значениям в любом столбце текущей записи как до, так и после выполнения обработчика событий. У каждого столбца, модифицируемого обработчиком, имеются два *корреляционных имени* (согласно relation name), которые содержат старое и новое значения столбца. Данные имена доступны только для обработчиков событий на уровне строк. Корреляционное имя `:OLD` (двоеточие и слово “`OLD`”) содержит имя столбца перед выполнением выражения SQL, вызывающего срабатывание обработчика событий. Подобным образом, `:NEW` (двоеточие и слово “`NEW`”) содержит новое значение для столбца, получаемое им после выполнения выражения SQL, вызывающего срабатывание обработчика событий. Предположим, например, что вы определили обработчик событий `BEFORE UPDATE` для таблицы `Agents`. Если вы примените выражение `UPDATE` к таблице `Agents`, сработает обработчик событий `BEFORE UPDATE`. В теле этой процедуры доступны корреляционные значения `:NEW` и `:OLD` для каждого обновляемого столбца. Если вы обновляете столбец `LicenseExpire`, то можете обратиться к значению, предшествующему обновлению, как к `:OLD.LicenseExpire`, а к предлагаемому (обновленному) значению — как к `:NEW.LicenseExpire`. Значения корреляционных имен `OLD` и `NEW` для команд `INSERT`, `UPDATE` и `DELETE` приведены в табл. 4.6.

## **Создание и использование обработчика событий BEFORE**

Выше в этой главе (см. раздел “Создание и использование последовательностей”) вы создали последовательность `Agent_ID`, поставляющую первичные ключи для таблицы `Agents`. Хотя последовательности чрезвычайно удобны, когда при вставке

**ТАБЛИЦА 4.6.** Значения, доступные в теле обработчика событий  
через корреляционные имена NEW OLD

| <i>Выражение SQL</i> | <i>Корреляционное значение</i> | <i>имя</i>                                                              |
|----------------------|--------------------------------|-------------------------------------------------------------------------|
| INSERT               | NEW                            | Значение, которое присваивается столбцу оператором, начавшим транзакцию |
|                      | OLD                            | NULL                                                                    |
| UPDATE               | NEW                            | Значение, которое присваивается столбцу оператором, начавшим транзакцию |
|                      | OLD                            | Последнее из зафиксированных перед текущей транзакцией значений столбца |
| DELETE               | NEW                            | NULL                                                                    |
|                      | OLD                            | Последнее из зафиксированных перед текущей транзакцией значений столбца |

новых строк требуется генерировать первичные ключи, следует помнить, что в каждой команде `INSERT` вы должны запрашивать новое число последовательности, т.е. последовательности не соотносятся автоматически со столбцом первичного ключа какой-либо таблицы. Данную досадную проблему можно решить с помощью обработчика событий.

### Создание обработчика событий BEFORE

Создав нужный обработчик событий, вы сможете вводить новых агентов Redwood Realty в таблицу `Agents`, не заботясь о вводе значения `AgentID`. Соответствующее число последовательности будет предоставлено обработчиком событий `BEFORE INSERT`. Итак, для начала вы заново создадите последовательность, удаленную ранее в этой главе.

Для того чтобы удалить последовательность и воссоздать ее заново, выполните следующие действия.

1. Запустите `SQL*Plus`, войдите в `Oracle`, введите и выполните следующие команды:

```
CLEAR SCREEN
DROP SEQUENCE AgentID_seq;
CREATE SEQUENCE AgentID_seq
    START WITH 12345
    INCREMENT BY 2;
```

Возможно, после команды `DROP` `Oracle` сгенерирует сообщение об ошибке, поскольку вы уже удалили указанную последовательность. Не обращайте на это внимания — никакого вреда это не принесет, просто так мы гарантируем, что вы начнете с чистого листа, т.е. с определения последовательности. Помимо этого, база данных `Oracle` должна выдать сообщение “Sequence created”.

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following SQL commands:

```

SQL> DROP SEQUENCE AgentID_seq;
DROP SEQUENCE AgentID_seq
*
ERROR at line 1:
ORA-02289: sequence does not exist

SQL> CREATE SEQUENCE AgentID_seq
 2   START WITH 10001
 3   INCREMENT BY 2;
Sequence created.

SQL> CREATE OR REPLACE TRIGGER Agents_bi_trg
 2   BEFORE INSERT ON Agents
 3   FOR EACH ROW
 4   BEGIN
 5     SELECT AgentID_seq.NEXTVAL INTO :NEW.AgentID FROM DUAL;
 6   END;
 7 /
Trigger created.

SQL>

```

**Рис. 4.25.** Создание последовательности и обработчика событий BEFORE INSERT

- Очень внимательно введите нижеприведенный код, нажимая клавишу <Enter> в конце каждой строки. Обратите внимание на завершающую косую черту (/). В конце этой строки также нажмите клавишу <Enter>. Косая черта указывает Oracle скомпилировать и сохранить процедуру.

```

CREATE OR REPLACE TRIGGER Agents_bi_trg
BEFORE INSERT ON Agents FOR EACH ROW
BEGIN
  SELECT AgentID_seq.NEXTVAL INTO :NEW.AgentID FROM DUAL;
END;
/

```

Oracle отметит, что операция прошла успешно, выдав сообщение “Trigger created” (рис. 4.25).

**Совет.** Если Oracle выдаст сообщение об ошибке “Warning: Trigger created with compilation errors” (“Предупреждение: обработчик событий скомпилирован с ошибками”), значит, вы неверно набрали строку SELECT. Исправьте ошибки и еще раз выполните весь набор команд, приведенный в п. 2.

**Совет.** Чтобы отобразить на экране ошибки компиляции обработчика событий, введите команду

```
SHOW ERRORS TRIGGER <trigger-name>
```

**Совет.** Вместо <trigger-name> подставьте имя обработчика событий.

- SQL\*Plus не закрывайте.

Разберемся, какой обработчик событий мы создали. Как видно из имени (`Agents_bi_trg`), процедура относится к классу BEFORE INSERT. Это означает, что процедура срабатывает при выполнении команды `INSERT INTO Agents` — непосредственно перед тем, как эта команда добавит строку в таблицу `Agents`. Код обработчика событий помещен между словами BEGIN и END. Он просто выбирает следующее значение последовательности (`AgentID_seq.NEXTVAL`) и помещает его в столбец `AgentID` с корреляционным именем :`NEW`. Oracle поддерживает корреляционные имена `NEW` и `OLD` в теле обработчика событий всегда, когда выполняется команда `INSERT` или `UPDATE`, вызывающая срабатывание обработчика событий. В опции `SELECT` упоминается универсальная таблица Oracle `DUAL`, поскольку вы не можете достать значение “из воздуха” — вы должны сослаться на таблицу. Получив новый уникальный первичный ключ и сохранив его в строке `NEW`, Oracle берет значения, предоставленные пользователем в команде `INSERT`, а также корреляционные значения строки `NEW` и вставляет их в таблицу `Agents`.

## Проверка обработчика событий BEFORE

Теперь проверим обработчик событий, введя три новые строки в таблицу `Agents` (в текущий момент — пустую).

Для того чтобы вставить строки в таблицу и вызвать определенный выше обработчик событий, выполните следующие действия.

1. Используя SQL\*Plus, введите и выполните следующие команды `INSERT`:

```
INSERT INTO Agents(LastName, FirstName)
    VALUES('Marcoux', 'Kai');
INSERT INTO Agents(LastName, FirstName)
    VALUES('Silverburg', 'Danial');
INSERT INTO Agents(LastName, FirstName)
    VALUES('Taylor', 'Jessica');
```

После каждого выражения `INSERT` Oracle ответит сообщением “1 row created”.

2. Ведите и выполните следующую команду `SELECT`, чтобы отобразить на экране только что вставленные три строки в таблицу. Первичные ключи для этих строк подставляет обработчик событий, определенный в предыдущем разделе.

```
SELECT AgentID, FirstName, LastName
FROM Agents;
```

Результаты выполнения команды `SELECT` показаны на рис. 4.26. Обратите внимание на значения первичного ключа, которые автоматически вставляются обработчиком событий, — 12345, 12347 и 12349.

3. Ведите `exit` и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть окно SQL\*Plus. При этом вы автоматически фиксируете все изменения в базе данных, внесенные после входа в базу данных Oracle.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> DROP SEQUENCE AgentID_seq;
Sequence dropped.

SQL> CREATE SEQUENCE AgentID_seq
  2  START WITH 12345
  3  INCREMENT BY 2;
Sequence created.

SQL> CREATE OR REPLACE TRIGGER Agents_bi_trg
  2  BEFORE INSERT ON Agents
  3  FOR EACH ROW
  4  BEGIN
  5    SELECT AgentID_seq.NEXTVAL INTO :NEW.AgentID FROM DUAL;
  6  END;
  7 /

Trigger created.

SQL> INSERT INTO Agents(LastName, FirstName) VALUES('Marcoux', 'Kai');
1 row created.

SQL> INSERT INTO Agents(LastName, FirstName) VALUES('Silverburg', 'Danial');
1 row created.

SQL> INSERT INTO Agents(LastName, FirstName) VALUES('Taylor', 'Jessica');
1 row created.

SQL> SELECT AgentID, FirstName, LastName
  2  FROM Agents;
  AGENTID FIRSTNAME      LASTNAME
-----  -----
    12345 Kai            Marcoux
    12347 Danial         Silverburg
    12349 Jessica        Taylor
SQL>

```

**Рис. 4.26.** Отображение строк таблицы Agents с первичными ключами, полученными из обработчика событий

## Создание обработчика событий AFTER, наблюдающего за операциями с таблицей

Хотя Oracle предлагает богатые средства наблюдения, вы можете создать и собственные инструменты. Определяя обработчики событий UPDATE, INSERT или DELETE на выбранных таблицах, вы можете регистрировать любые модификации этих таблиц. Информация о модификации данных (так называемый *контрольный журнал* — audit trail) хранится в одной или нескольких контрольных таблицах, структуру которых определяете вы. Типичные данные контрольного журнала — это дата и время модификации, природа модификации (обновление, удалении или вставка), имя пользователя, которые производят модификацию, а также старые и новые значения.

Роберт Стирлинг желает, чтобы вы отслеживали все изменения UPDATE таблицы Agents (но не препятствовали им). Когда любой пользователь, имеющий право доступа к таблице Agents, применяет команду UPDATE к таблице Agents, Роберт требует, чтобы база данных Oracle записывала в контрольный журнал информацию: событие (в данном случае — INSERT), имя пользователя, который вносит изменения, а также дату и время изменения. Кроме того, он хочет, чтобы в журнал занеслись значения семи столбцов таблицы Agents: AgentID, FirstName, LastName,

HireDate, BirthDate, Gender и Title. Хотя полный контрольный журнал обычно включает все столбцы таблицы, находящейся под наблюдением, пока нам будет достаточно записи семи указанных столбцов. Позже Роберт может попросить вас добавить в контрольный журнал и другие столбцы, а также написать обработчики событий, запускающиеся для команд INSERT и DELETE.

## Создание контрольной таблицы, регистрирующей модификации таблицы

Чтобы выполнить указание Роберта, вам потребуется вначале создать новый контрольный журнал, который будет содержать записываемую информацию об изменениях таблицы Agents. После этого вам придется создать обработчик событий, срабатывающий при каждом выполнении команды UPDATE применительно к таблице Agents. Начнем же мы с создания контрольной таблицы AuditAgents.

Для того чтобы создать контрольную таблицу, в которой будет храниться выбранная информация об изменениях таблицы Agents, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle, а затем введите и выполните команду CLEAR SCREEN.
2. Запустите программу Блокнот, создайте выражения SQL в Блокноте, а затем скопируйте их в SQL\*Plus. Выполните сценарий SQL\*Plus. Если потребуется, исправьте ошибки в SQL\*Plus и модифицируйте команду.
3. Ведите следующий код в программе Блокнот, скопируйте его в SQL\*Plus и выполните, чтобы создать таблицы AuditAgents:

```
CREATE TABLE AuditAgents
(AuditEvent NVARCHAR2(6),
BeforeAfter NVARCHAR2(6),
AuditUsername NVARCHAR2(30),
AuditDate DATE,
AgentID INTEGER,
FirstName NVARCHAR2(30),
LastName NVARCHAR2(30),
HireDate DATE,
BirthDate DATE,
Gender NVARCHAR2(1),
Title NVARCHAR2(11));
```

4. Запишите файл в программе Блокнот под уникальным именем, например <имя> CreateAudit.sql. Не забудьте выбрать из списка Save as type (Тип файла) значение All Files (Все файлы).

## Создание обработчика событий AFTER

Далее вы создадите обработчик событий, срабатывающий, когда к таблице Agents применяется команда UPDATE. Этот обработчик может запускаться после операции

обновления и должен выполняться для каждой обновляемой строки.

Для того чтобы присоединить обработчик событий AFTER UPDATE к таблице Agents, выполните следующие действия.

1. Введите следующий код в программе Блокнот, затем скопируйте его в SQL\*Plus:

```
CREATE OR REPLACE TRIGGER Agents_au_trg
AFTER UPDATE ON Agents
FOR EACH ROW
BEGIN
    INSERT INTO AuditAgents
        VALUES ('Update', 'Before', user, sysdate,
                :OLD.AgentID, :OLD.FirstName, :OLD.LastName,
                :OLD.HireDate, :OLD.BirthDate, :OLD.Gender, :OLD.Title);
    INSERT INTO AuditAgents
        VALUES ('Update', 'After', NULL, NULL,
                :NEW.AgentID, :NEW.FirstName, :NEW.LastName,
                :NEW.HireDate, :NEW.BirthDate, :NEW.Gender, :NEW.Title);
END;
```

2. В SQL\*Plus нажмите клавишу <Enter>, чтобы перейти на новую строку под “END”, введите косую черту (/) и нажмите клавишу <Enter>, чтобы скомпилировать код. Oracle ответит сообщением “Trigger created” (рис. 4.27). При наличии ошибок внимательно проверьте код, внесите необходимые исправления и повторите ввод.
3. Переключитесь на программу Блокнот и сохраните файл.

Обработчик событий готов. Он будет срабатывать при каждом обновлении строки таблицы Agents. Запущенная процедура записывает четыре специальных контрольных значения и семь столбцов Agents. Обработчик событий вставляет две строки: одну со значениями до операции обновления, вторую — со значениями после него.

## Проверка обработчика событий AFTER

Чтобы протестировать созданную процедуру AFTER UPDATE, следует применить к таблице Agents три выражения UPDATE. Эти команды модифицируют один из семи столбцов Agents, загруженных в контрольную таблицу, хотя вы можете обновить любой столбец. После этого необходимо отобразить содержимое таблицы AuditAgents, чтобы изучить данные, собранные при выполнении обработчика событий AFTER UPDATE. Любой строке, обновляемой в таблице Agents, должны соответствовать две строки в таблице AuditAgents. А теперь давайте проверим наш обработчик событий.

Для того чтобы модифицировать строки таблицы Agents и активизировать обработчик событий AFTER UPDATE, выполните следующие действия.

1. Введите и выполните приведенные ниже команды UPDATE. Если хотите, для набора кода можете использовать Блокнот, затем скопировать команды в SQL\*Plus и выполнить их. После этого запишите файл в Блокноте.

```

SQL> CREATE TABLE AuditAgents
  2   (AuditEvent    NVARCHAR2(6),
  3    BeforeAfter   NVARCHAR2(6),
  4    AuditUsername NVARCHAR2(30),
  5    AuditDate     DATE,
  6    AgentID      INTEGER,
  7    FirstName    NVARCHAR2(30),
  8    LastName     NVARCHAR2(30),
  9    HireDate     DATE,
 10   BirthDate    DATE,
 11   Gender       NVARCHAR2(1),
 12   Title        NVARCHAR2(11)
 13  );

Table created.

SQL> CREATE OR REPLACE TRIGGER Agents_au_trg
  2  AFTER UPDATE ON Agents
  3  FOR EACH ROW
  4  BEGIN
  5    INSERT INTO AuditAgents
  6      VALUES ('Update', 'Before', user, sysdate,
  7             :NEW.AgentID, :NEW.FirstName, :NEW.LastName,
  8             :NEW.HireDate, :NEW.BirthDate, :NEW.Gender, :NEW.Title);
  9    INSERT INTO AuditAgents
 10      VALUES ('Update', 'After', NULL, NULL,
 11             :NEW.AgentID, :NEW.FirstName, :NEW.LastName,
 12             :NEW.HireDate, :NEW.BirthDate, :NEW.Gender, :NEW.Title);
 13  END;
 14 /

Trigger created.

SQL>

```

**Рис. 4.27.** Создание контрольной таблицы и обработчика событий AFTER UPDATE

```

UPDATE Agents SET Gender = 'M' WHERE AgentID = 12345;
UPDATE Agents SET Title = 'Salesperson'
    WHERE LastName = 'Silverburg';
UPDATE Agents SET HireDate = '29-JUN-98' WHERE AgentID = 12349;

```

Oracle ответит тремя сообщениями “1 row updated”.

2. Введите и выполните одно дополнительное обновление строки, уже обновленной при выполнении п. 1.

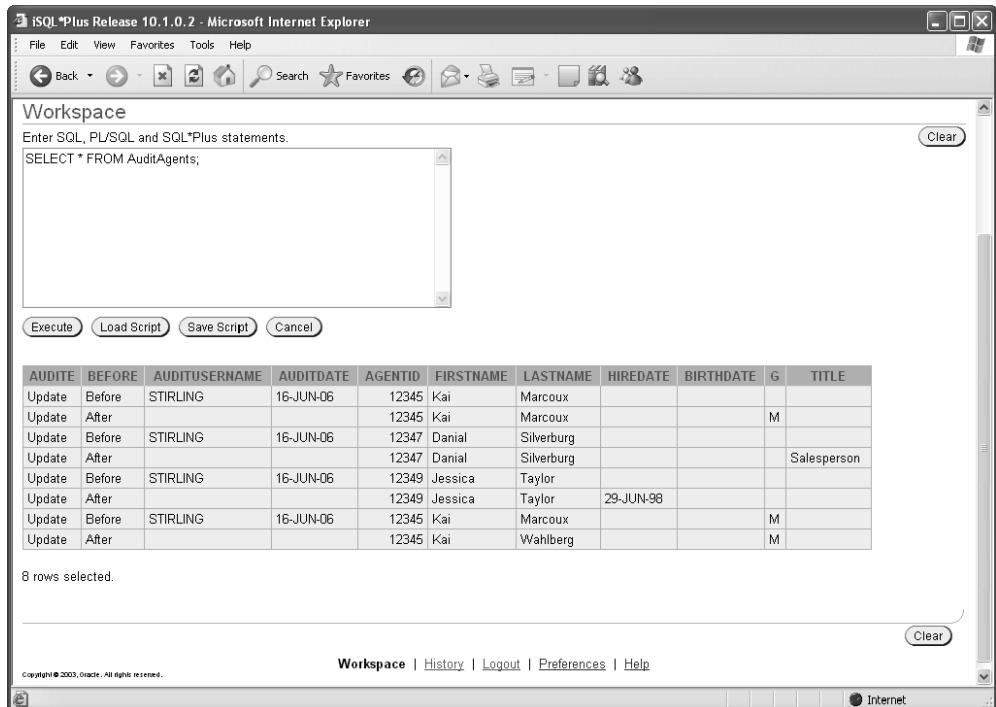
```
UPDATE Agents SET LastName = 'Wahlberg' WHERE AgentID = 12345;
```

3. Поскольку вы переключались на iSQL\*Plus, чтобы отобразить на экране столбцы контрольной таблицы, введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из Oracle, и выйдите SQL\*Plus. (По желанию, последующие упражнения можно выполнить в SQL\*Plus.)

4. Если хотите закрыть Блокнот, выберите из меню `File` (Файл) команду `Exit` (Выход).

Всего в таблицу `Agents` вы вставили четыре строки. Больше, похоже, ничего не произошло. Тем не менее ваш обработчик событий AFTER UPDATE записал четыре группы внесенных изменений и сохранил эту информацию в контрольной таблице `AuditAgents`. Далее вы временно станете аудитором базы данных и посмотрите, кто и когда модифицировал таблицу `Agents`.

Для того чтобы отобразить на экране столбец `AuditAgents` контрольной таблицы, выполните следующие действия.



**Рис. 4.28.** Отображение на экране контрольной таблицы

1. Запустите браузер, запустите iSQL\*Plus и щелкните в текстовом окне Username. Введите свое имя пользователя, нажмите клавишу <Tab>, введите свой пароль в текстовом окне Password и щелкните на кнопке Login.
2. Щелкните в текстовом окне Workspace и введите следующий код:
 

```
SELECT * FROM AuditAgents;
```
3. Щелкните на кнопке Execute. В ответ Oracle вернет восемь строк из таблицы AuditAgents (рис. 4.28).
4. Щелкните на кнопке Logout, чтобы выйти из iSQL\*Plus, и закройте браузер.

Внимательно рассмотрите рис. 4.28. Обратите внимание на строки Before и After всех модификаций. В двух первых строках отражено изменение значения поля (столбец "G" на рисунке). В третьей и четвертой строках показано, что после обновления появилось значение Title. В двух последних строках отражено изменение столбца LastName. Если вы хотите отобразить на экране время и дату изменения, можете с помощью оператора TO\_CHAR и подходящей модели формата указать, что в столбце AuditDate отображается дата и время. Например, для учета информации о дате и времени можно выполнить следующую команду:

```
SELECT TO_CHAR(AuditDate, 'MM/DD/YY HH:MM:SS');
```

**Совет.** Тот же результат вы получите, использовав вместо AFTER UPDATE обработчик событий BEFORE UPDATE. В данном случае между ними нет никакой разницы.

## Создание и использование обработчиков событий на уровне выражений

Все обработчики событий, с которыми вы ознакомились в данной главе, относятся к процедурам на уровне строк, т.е. они срабатывают для каждой затронутой строки при наступлении события, на которое настроен обработчик событий. Напомним, что обработчик событий на уровне выражения срабатывает один раз для события, независимо от числа затронутых строк таблицы. В данном разделе мы рассмотрим пример того, как внешне правильный обработчик событий, присоединенный на уровне строки, генерирует ошибку Oracle. Абсолютно правильный код мы получим, заменив ее эквивалентной процедурой, присоединенной на уровне выражения. Задача обработчика событий проста: при каждом применении команды INSERT или DELETE к таблице Agents вы требуете от Oracle отображать на экране сообщение, указывающее число строк, оставшихся в таблице. Ниже вы создадите процедуру AFTER, реагирующую на INSERT и DELETE, так что сможете отслеживать оба события. Напомним, что к таблице Agents мы уже присоединили обработчик событий BEFORE INSERT и AFTER UPDATE. Вообще, Oracle разрешает задавать для одной таблицы несколько присоединенных процедур, срабатывающих в ответ на определенное событие (INSERT и т.д.), однако Oracle не может гарантировать, в каком порядке будут срабатывать одноранговые обработчики событий, определенные для одного типа DML-выражения. В нашем случае это допустимо, поскольку корректная работа обработчиков событий не зависит от порядка их выполнения.

Давайте поэкспериментируем с обработчиком событий на уровне строк, который отображает число строк в таблице Agents, когда кто-либо применяет к таблице Agents команду INSERT или DELETE. Процедура будет запускаться автоматически сразу после модификации. Для проверки процедуры мы используем SQL\*Plus, выводя результаты на экран (консоль) с помощью пакета DBMS\_OUTPUT, который позволяет отправлять сообщения от обработчиков событий на экран (с помощью процедур PUT и PUT\_LINE). Например, приведенное ниже тело обработчика событий отображает обновленное значение столбца LastName таблицы Agents.

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE('New LastName value: ' || :NEW.LastName);  
END;
```

Чтобы просмотреть результат выполнения процедуры на экране, необходимо запустить следующую команду SQL\*Plus:

```
SET SERVEROUTPUT ON
```

В обработчике событий, который вы собираетесь написать, будет использовано кое-что новое — переменная. Внутренняя переменная представляет собой времен-

ную ячейку памяти, которая существует только во время выполнения обработчика событий. В данной переменной будет храниться целое число, представляющее собой счетчик строк таблицы Agents. Переменная объявляется в необязательном заголовке обработчика событий, который начинается со слова DECLARE и заканчивается словом BEGIN. После BEGIN начинается тело обработчика событий.

Для того чтобы создать обработчик событий, отображающий на экране счетчик строк при выполнении команды INSERT или UPDATE, выполните следующие действия.

1. Запустите SQL\*Plus. Затем введите и выполните следующие команды (которые очищают экран и настраивают вывод информации). В конце каждой строки нажмайте клавишу <Enter>. Вы можете также использовать программу Блокнот в качестве редактора, а затем скопировать готовую команду в SQL\*Plus.

```
CLEAR SCREEN
SET SERVEROUTPUT ON
```

2. Введите следующий код, скопируйте его в SQL\*Plus и нажмите клавишу <Enter> после слова END:

```
CREATE OR REPLACE TRIGGER Agents_adai_trg
AFTER INSERT OR DELETE ON Agents
FOR EACH ROW
DECLARE V_RowCount NUMBER;
BEGIN
    SELECT COUNT(*) INTO V_RowCount FROM Agents;
    DBMS_OUTPUT.PUT_LINE(V_RowCount || ' rows in Agents');
END;
```

3. В SQL\*Plus введите косую черту (/) и нажмите клавишу <Enter>, чтобы скомпилировать обработчик событий (рис. 4.29). Oracle ответит строкой “Trigger created”.

Не закрывайте программу Блокнот с кодом обработчика событий, поскольку чуть позже он вам понадобится. Далее мы проверим обработчик событий, чтобы посмотреть, как он работает.

Чтобы протестировать обработчик событий, выполните следующие действия.

1. В SQL\*Plus введите и выполните следующую команду DELETE, чтобы вызвать обработчик событий:

```
DELETE FROM Agents WHERE Lastname = 'Silverburg';
```

Oracle выведет сообщение об ошибке.

2. Отмените операцию удаления, набрав и выполнив команду ROLLBACK;

3. Переключитесь на программу Блокнот и подготовьтесь к изменению кода обработчика событий.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE TRIGGER Agents_adai_trg
 2  AFTER INSERT OR DELETE ON Agents
 3  FOR EACH ROW
 4  DECLARE
 5      V_RowCount NUMBER;
 6  BEGIN
 7      SELECT COUNT(*) INTO V_RowCount FROM Agents;
 8      DBMS_OUTPUT.PUT_LINE(V_RowCount || ' rows in Agents');
 9  END;
10 /
Trigger created.

SQL>

```

**Рис. 4.29.** Ошибка выполнения обработчика событий

Как видите, с нашей присоединенной процедурой что-то не так, поскольку появилось сообщение “... AGENTS is mutating, trigger/function may not see it” (“... таблица AGENTS мутировала, обработчик событий/функция может ее не видеть”). Мутировавшей таблицей (mutating table) называется таблица, модифицированная выражением DML, которое инициировало событие. Выражения SQL в теле обработчика событий (например, SELECT в предыдущем примере) не могут считать или модифицировать таблицу, принадлежащую к инициирующему выражению (в частности, это относится к самой таблице, инициирующей запуск обработчика событий). В результате обработчик событий пытается изменить или изучить что-то, что уже изменилось. Oracle это запутывает. Что делать Oracle, когда обработчик событий, присоединенный на уровне строк, готовится запуститься для второй строки команды удаления, затрагивающей несколько строк? Предполагает ли приложение, что база данных должна рассматривать таблицу в состоянии, предшествовавшем изменению? Или следует полагать, что первая операция удаления уже произошла? Похоже, что мы столкнулись с парадоксом. Проблему можно решить, переведя обработчик событий с уровня строк на уровень выражений.

Для того чтобы модифицировать и перекомпилировать обработчик событий, выполните следующие действия.

1. В программе Блокнот удалите строку “FOR EACH ROW” из тела обработчика событий, запишите код под выбранным именем (используйте суффикс .sql) и скопируйте определение процедуры в SQL\*Plus.
2. Введите косую черту (/) и нажмите клавишу <Enter>, чтобы перекомпилировать обработчик событий.
3. Еще раз протестируйте обработчик событий, набрав и выполнив следующий код SQL\*Plus:

```
DELETE FROM Agents WHERE Lastname = 'Silverburg';
```

На этот раз обработчик событий работает без ошибок. Обратите внимание на сообщение “2 rows in Agents”, появившееся непосредственно под командой

The screenshot shows the Oracle SQL\*Plus interface with the following session history:

```

SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE TRIGGER Agents_adai_trg
  2  AFTER INSERT OR DELETE ON Agents
  3  FOR EACH ROW
  4  DECLARE
  5    U_RowCount NUMBER;
  6  BEGIN
  7    SELECT COUNT(*) INTO U_RowCount FROM Agents;
  8    DBMS_OUTPUT.PUT_LINE(U_RowCount ||' rows in Agents');
  9  END;
 10 /
Trigger created.

SQL> DELETE FROM Agents WHERE Lastname = 'Silverburg';
DELETE FROM Agents WHERE Lastname = 'Silverburg'
*
ERROR at line 1:
ORA-04011: table CH04.AGENTS is mutating, trigger/function may not see it
ORA-06512: at "CH04.AGENTS_ADAI_TRG", line 4
ORA-04088: error during execution of trigger 'CH04.AGENTS_ADAI_TRG'

SQL> ROLLBACK;
Rollback complete.

SQL> CREATE OR REPLACE TRIGGER Agents_adai_trg
  2  AFTER INSERT OR DELETE ON Agents
  3  DECLARE
  4    U_RowCount NUMBER;
  5  BEGIN
  6    SELECT COUNT(*) INTO U_RowCount FROM Agents;
  7    DBMS_OUTPUT.PUT_LINE(U_RowCount ||' rows in Agents');
  8  END;
 9 /
Trigger created.

SQL> DELETE FROM Agents WHERE Lastname = 'Silverburg';
2 rows in Agents
1 row deleted.
SQL>

```

On the right side of the screenshot, there is a vertical text block: "Результат выполнения присоединенной процедуры на уровне выражения". This text corresponds to the output of the trigger execution, specifically the message "2 rows in Agents" and "1 row deleted".

**Рис. 4.30.** Правильное выполнение обработчика событий, присоединенного на уровне выражения

DELETE. Данное сообщение процедуры, присоединенной на уровне выражения, указывает, что после операции удаления остались две строки. После этого Oracle отображает сообщение “1 row deleted” (рис. 4.30).

4. Введите `COMMIT` и нажмите клавишу `<Enter>`, чтобы сделать изменения постоянными.
5. Введите `Exit` и нажмите клавишу `<Enter>`, чтобы выйти из `SQL*Plus`, поскольку в следующем разделе вы будете использовать `iSQL*Plus`.

## Просмотр, изменение и удаление обработчиков событий

Информация об обработчиках событий (как и данные о других объектах Oracle) хранится в словаре данных. Извлечь данную информацию можно из представления словаря данных `user_triggers`. Более понятный список всех обработчиков событий имеется в представлении `all_triggers`, хотя данный список также является и более длинным — он содержит тысячи символов. (Например, столбец `column_name` имеет длину 4000 символов.) Столбцы, которые можно извлечь из представления `user_triggers`, представлены в табл. 4.7.

**ТАБЛИЦА 4.7.** Столбцы, извлекаемые из представления словаря данных user\_triggers

| Имя столбца       | Тип данных     | Значение                                                                  |
|-------------------|----------------|---------------------------------------------------------------------------|
| Trigger_Name      | VARCHAR2(30)   | Имя обработчика событий                                                   |
| Trigger_Type      | VARCHAR2(16)   | Тип обработчика событий                                                   |
| Triggering_Event  | VARCHAR2(227)  | Событие, вызывающее срабатывание присоединенной процедуры                 |
| Table_Owner       | VARCHAR2(30)   | Пользователь, владеющий таблицей, на которую ссылается обработчик событий |
| Base_Object_Type  | VARCHAR2(16)   | Тип объекта, на который ссылается обработчик событий                      |
| Table_Name        | VARCHAR2(30)   | Таблица, на которую ссылается обработчик событий                          |
| Column_Name       | VARCHAR2(4000) | Столбец, на который ссылается обработчик событий                          |
| Referencing_Names | VARCHAR2(128)  | Имена псевдонимов OLD и NEW                                               |
| When_Clause       | VARCHAR2(4000) | Условие WHEN обработчика событий                                          |
| Status            | VARCHAR2(8)    | Метка, указывающая, включен или отключен обработчик событий               |
| Description       | VARCHAR2(4000) | Описание обработчика событий                                              |
| Action_Type       | VARCHAR2(11)   | Тип действия обработчика событий                                          |
| Trigger_Body      | LONG           | Код, содержащийся в теле обработчика событий                              |

## Отображение информации об обработчике событий

Некоторые из указанных столбцов нам малоинтересны. Несмотря на это, столбцы trigger\_name, trigger\_type, triggering\_event, table\_name и status весьма важны. Рассмотрим их внимательнее.

Для того чтобы отобразить на экране выбранную информацию об обработчике событий, выполните следующие действия.

1. Запустите iSQL\*Plus (вы можете также использовать SQL\*Plus), войдите в Oracle, используя свое имя пользователя и пароль, и щелкните в текстовом окне Workspace.
2. В текстовом окне Workspace введите следующий код:

```
SELECT Trigger_Name, Triggering_Event, Trigger_Type, Table_Name,
       Status
  FROM user_triggers;
```

Указанная информация отобразится на экране для трех созданных выше обработчиков событий (рис. 4.31).

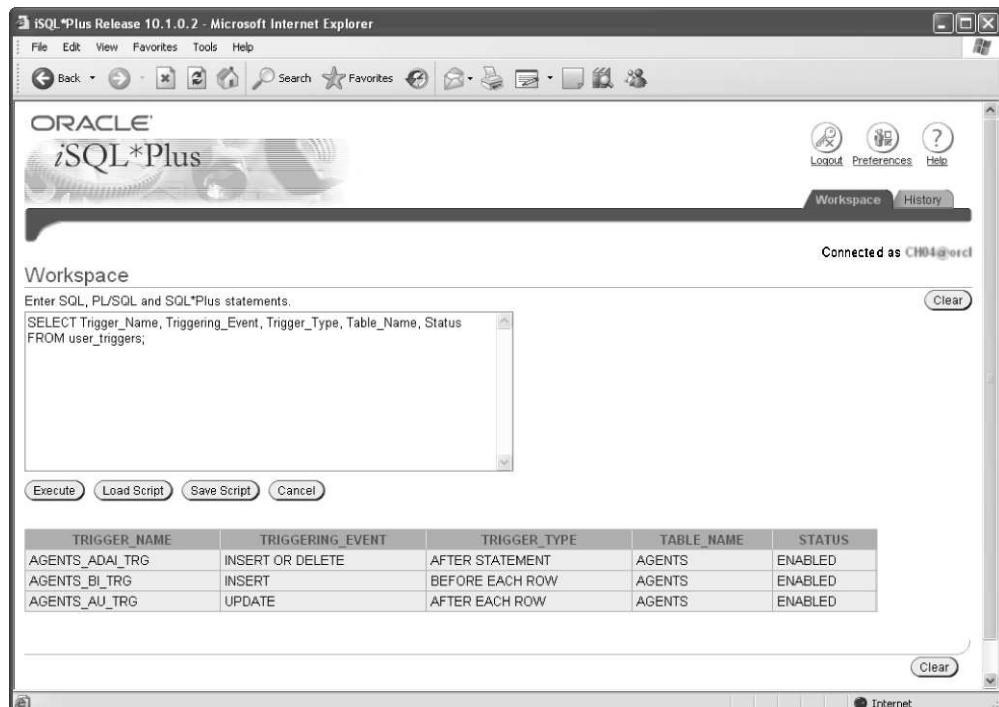


Рис. 4.31. Вывод на экран информации об обработчике событий

- Если вы используете *iSQL\*Plus*, щелкните на гиперссылке *Logout*, чтобы выйти из Oracle, и закройте браузер. Если вы используете интерфейс *SQL\*Plus*, не закрывайте его — скоро мы к нему вернемся.

## Включение и выключение обработчиков событий

Когда вы создаете и компилируете обработчик событий, он включается автоматически. Если вам нужно временно отключить его, это можно сделать с помощью выражения *ALTER TRIGGER*. Когда обработчик событий выключается, он по-прежнему считается определенным, но при этом является неактивизированным и не будет срабатывать при наступлении соответствующего события. Синтаксис команды *ALTER TRIGGER* выглядит следующим образом:

```
ALTER TRIGGER [schema.]<trigger-name> {ENABLE | DISABLE};
```

В данном коде аргумент *schema* — это необязательное имя схемы. Для любого обработчика событий, который вы создали (соответственно владельцем которого вы являетесь), имя схемы можно опускать. Если же у вас есть право удаления обработчиков событий, принадлежащих другой схеме, вы должны указывать уточняющее имя схемы. Для примера можете отключить обработчик событий *Agents\_au\_trg*,

который вы создали выше (соответственно вы являетесь его владельцем), чтобы он больше не срабатывал. Для этого выполните следующую команду SQL:

```
ALTER TRIGGER Agents_au_trg DISABLE;
```

Если позже вы решите возобновить работу обработчика событий, выполните код

```
ALTER TRIGGER Agents_au_trg ENABLE;
```

## Удаление обработчиков событий

Если вы уверены, что один или несколько обработчиков событий вам больше никогда не потребуется, вы можете целиком удалить их и их определения. Подобно другим объектам базы данных, обработчики событий удаляются с использованием команды `DROP`, которая имеет такой синтаксис:

```
DROP TRIGGER [schema.]<trigger-name>
```

Поскольку нам больше не потребуется ни один из обработчиков событий, созданных в данной главе, мы удалим их все. Если вы забыли имя какой-либо процедуры, запустите приведенную выше команду (запрос) `SELECT`, которая отобразит на экране имена всех обработчиков событий.

Итак, чтобы удалить созданные вами обработчики событий, выполните следующие действия.

1. Если необходимо, запустите `SQL*Plus`, а затем введите и выполните следующие команды:

```
DROP TRIGGER Agents_au_trg;  
DROP TRIGGER Agents_bi_trg;  
DROP TRIGGER Agents_adai_trg;
```

После каждой команды `Oracle` выведет сообщение “Trigger dropped”.

2. Чтобы убедиться, что все обработчики событий удалены, введите и выполните команду

```
SELECT Trigger_Name from user_triggers;
```

`Oracle` ответит сообщением “no rows selected”, подтверждающим, что вы удалили все обработчики событий.

3. Введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из `Oracle`, и закройте `SQL*Plus`.

## Резюме

К выражениям языка манипулирования данными (Data Manipulation Language – DML) `Oracle` относятся команды `INSERT`, `UPDATE` и `DELETE`. Команда `SQL INSERT` позволяет добавлять в таблицу новые строки. Используя ее, вы задаете необязательный список имен столбцов и соответствующий список значений. Всего должно быть столько зна-

чений (включая NULL), сколько указано имен столбцов, причем значения должны идти в таком же порядке, в котором перечислены столбцы. Тем не менее порядок имен столбцов может отличаться от заданного при определении таблицы. Все столбцы, пропущенные при выполнении команды INSERT, получают значения по умолчанию (если оно было задано) или NULL. Если вы выполняете команду INSERT для таблицы, содержащей внешний ключ, соответствующий первичному ключу другой таблицы, следует быть очень внимательным. Как правило, первой должна заполняться таблица, на которую идет ссылка. Возможен также вариант временного отключение условия (внешнего ключа) или вставки значения NULL в столбец внешнего ключа. Для ввода информации о дате применяется стандартная форма записи — DD-MON-YY в одинарных кавычках, также можно задействовать функцию TO\_DATE с моделью формата, согласующейся с форматом вводимых данных.

Последовательностью называется объект базы данных, генерирующий ряд уникальных чисел, удобных в роли первичных ключей. Чтобы создать именованную последовательность, задается необязательное начальное число и инкремент, после чего можно обращаться к псевдостолбцу NEXTVAL, содержащему значение следующего уникального числа. Псевдостолбец CURRVAL содержит последнее сгенерированное значение. Для отображения на экране значения псевдостолбца или результата расчетов применяется таблица Oracle Dual, состоящая из одной строки и одного столбца.

Команда UPDATE (запрос действия) используется для модификации данных. С ее помощью можно обновить один или несколько столбцов одной таблицы. Необязательный оператор WHERE содержит логическое выражение, состоящее из операторов отношений, логических операторов, констант и ссылок на столбцы. Логические выражения ограничивают число затрагиваемых строк, а пропустив оператор WHERE, вы примените изменения ко всем строкам таблицы. Для задания нескольких альтернативных значений для столбца применяется структура CASE. Подстановочные переменные Oracle используются для подстановки значений в выражения SQL в процессе выполнения сценария, когда Oracle запрашивает у вас их реальные значения.

Запрос действия DELETE удаляет строки из таблицы. Необязательный оператор WHERE позволяет выбирать, какие строки будут удалены. Если его пропустить, будут удалены все строки, однако сама таблица останется. Чтобы безвозвратно удалить все строки таблица, применяется команда TRUNCATE TABLE (структура таблицы при этом остается).

Команда MERGE объединяет в себе функции операторов INSERT, UPDATE и DELETE и применяется для слияния строк исходной таблицы с целевой. Слияние происходит следующим образом: если исходная и целевая таблицы имеют строки с одинаковыми первичными ключами, строка исходной таблицы заменяет собой соответствующую строку целевой. Строки целевой таблицы, не имеющие аналогов в исходной, удаляются. В целом действие команды регламентируется условиями WHEN MATCHED и WHEN NOT MATCHED.

В число выражений управления транзакциями базы данных входят COMMIT, ROLLBACK и SAVEPOINT. Команда COMMIT фиксирует выполненные действия DML, ROLLBACK отменяет некоторые или все действия DML, сделанные после начала транзакции. Транзакции начинаются с входа в Oracle, когда вы вводите DDL-выражение (например, CREATE TABLE) или команду COMMIT, закрывающую предыдущую транзакцию. Точки сохранения позволяют устанавливать метки, к которым можно выполнить откат, и вернуть базу данных в состояние, предшествовавшее точке сохранения.

Обработчиком событий называется блок кода, соотнесенный с таблицей и выполняющийся (срабатываящий) до или после применения к таблице определенного выражения DML. Обработчики событий очень удобны для привязывания к базе данных правил делового регламента (например, проверки значения перед изменением или введением приемлемого диапазона значений). Обработчики событий можно определить и не для DML-выражений, однако обычно эти процедуры определяются для событий INSERT, UPDATE и DELETE (или любой их комбинации). Обработчики событий могут запускаться до или после выполнения выражения DML. Наиболее распространенной областью их применения является создание контрольных таблиц, в которых регистрируется информация относительно внесенных изменений. В любой момент обработчик событий можно отключить, снова включить или удалить.

## Основные термины

- Запрос
- Запрос действия
- Корреляционное имя
- Модель формата
- Мутирующая таблица
- Обработчик событий
- Обработчик событий, присоединенный на уровне выражения
- Обработчик событий, присоединенный на уровне строки
- Откат
- Подстановочная переменная
- Последовательность
- Псевдоним
- Псевдостолбец
- Сеанс
- Срабатывание процедуры
- Таблица Dual

- Транзакция
- Усечение
- Фиксация

## Повторение пройденного материала

### Истина или ложь?

1. Выражение UPDATE всегда изменяет значение таблицы, причем отменить эти изменения нельзя.
2. Когда вы вставляете строку в таблицу, вам не нужно задавать пару “имя столбца–значение” для каждого столбца таблицы.
3. Усечение таблицы незначительно отличается от удаления всех ее строк. Удаление строк является обратимой операцией, а усеченные строки нельзя восстановить с помощью отката.
4. COMMIT делает постоянным любое изменение базы данных.
5. В теле обработчика событий BEFORE INSERT (FOR EACH ROW) ваш код может обращаться и не к NULL значениям, хранимым в корреляционных значениях OLD и NEW для каждого столбца во вставляемой строке.

### Заполнить пропущенное

1. Команда INSERT содержит столбец и список значений, если значение неизвестно, вместо него указывается \_\_\_\_\_.
2. Чтобы вставить дату, включая ее формат модели, используйте функцию Oracle \_\_\_\_\_.
3. Параметр \_\_\_\_\_ выражения CREATE SEQUENCE устанавливает наименьшее значение, генерируемое в последовательности.
4. После выполнения команды SQL \_\_\_\_\_ никакие изменения базы данных обратить уже нельзя.
5. Чтобы обработчик событий срабатывал перед модификацией столбца License-Date таблицы Agents, в определении процедуры необходимо задать \_\_\_\_\_ \_\_\_\_\_. Если требуется, чтобы обработчик срабатывал только один раз независимо от числа задействованных строк, в его определении необходимо пропустить слова \_\_\_\_\_ \_\_\_\_\_.

### Варианты ответов

1. Какая из указанных ниже команд INSERT правильная (используйте знания о таблице Agents)?

- a) `INSERT INTO Agents(AgentID, LastName) (96743, 'Gilbert')`
  - б) `INSERT INTO Agents VALUES (96743, 'Gilbert')`
  - в) `INSERT INTO Agents(AgentID, LastName) VALUES ('96743', 'Gilbert')`
  - г) `INSERT INTO Agents(AgentID, LastName) VALUES (96743, 'Gilbert')`
2. Какое из следующих выражений DELETE является неправильным?
- а) `DELETE FROM Agents`
  - б) `DELETE FROM Agents WHERE AgentID = 12345`
  - в) `DELETE Agents WHERE LastName = 'Ellison'`
  - г) `DELETE FROM Agents WHERE FirstName = 'Leonard' AND LastName = 'Smythe'`
3. Какая фраза при выполнении команды MERGE INTO предшествует оператору INSERT, когда требуется добавить строки, пропущенные в целевой таблице?
- а) WHEN MATCHED THEN UPDATE
  - б) WHEN NOT MATCHED THEN UPDATE
  - в) WHEN MATCHED THEN INSERT
  - г) WHEN NOT MATCHED THEN INSERT
4. Отдельное соединение с сервером базы данных Oracle называется
- а) Транзакцией.
  - б) Мутирующей таблицей.
  - в) Схемой.
  - г) Сеансом.
5. Oracle позволяет использовать обработчик событий BEFORE и AFTER с командами INSERT, UPDATE и DELETE. Как называется обработчик событий, позволяющий работать не с таблицами, а с представлениями?
- а) AUDIT.
  - б) INSTEAD OF.
  - в) IN VIEW OF.
  - г) IN REVERSE OF.

## Упражнения

### 1. Readwood Realty

Роберт Стирлинг Robert Stirling хочет, чтобы вы добавили в базу данных Redwood Realty таблицу *Customers*, которая будет содержать контактную информацию о клиентах, интересующихся покупкой и продажей недвижимости. Роберт подготов-

вил файл сценария, который создает таблицу и вставляет в нее исходные десять строк с записями о клиентах. После того как вы запустите этот файл сценария, вы удалите существующую последовательность `AgentID_seq`, создадите новую для генерации первичных ключей для таблицы `Customers`, которые будут использоваться при вставке новых клиентов, зададите несколько точек сохранения, вставите новых клиентов, выполните откат, удалите выбранных клиентов и создадите обработчик событий, который будет срабатывать автоматически при вставке данных о новом клиенте в таблицу `Customers`. Наконец, вы удалите обработчик событий и созданную последовательность. Продемонстрируйте свою работу, подключив буферный файл, записывающий ввод с клавиатуры и ответные действия Oracle.

Запустите SQL\*Plus и войдите в Oracle. В качестве текстового редактора используйте Блокнот, копируя готовые фрагменты кода в SQL\*Plus. Прежде всего найдите на компакт-диске файл сценария `Ch04Problem1.sql` и запишите путь к нему. Подставьте этот путь вместо аргумента `<путь>` на этапе 1, вместо `<mypath>` подставьте любой удобный для вас путь, а вместо `<имя>` — свое имя.

**Важное примечание.** Вы должны завершить данное упражнение в течение одного сеанса. Если вы выполните какие-то этапы, выйдете из Oracle, а затем вернетесь, Oracle зафиксирует внесенные изменения и некоторые команды ROLLBACK будут работать некорректно.

1. Итак, введите и выполните следующую команду:

```
START C:\<путь>\Ch04Problem1.sql
COLUMN CustomerID FORMAT 99999
COLUMN LastName FORMAT A10
COLUMN City FORMAT A10
COLUMN State FORMAT A10
COLUMN Zipcode FORMAT A7
CLEAR SCREEN
SPOOL C:\<mypath>\<имя>Ch04Redwood.txt
SHOW USER
CREATE SEQUENCE CustomerID_seq
    START WITH 34567
    INCREMENT BY 2;
```

2. Обновите строки `Customer` и задайте точку сохранения.

```
UPDATE Customers
SET City = 'Aracta', State = 'California'
WHERE Zipcode = '95570';
SAVEPOINT SaveAfterUpdate;
```

3. Вставьте запись нового клиента и задайте новую точку сохранения.

```
INSERT INTO Customers (CustomerID, FirstName, LastName)
VALUES (25599, 'Lora', 'Dietrich');
SAVEPOINT SaveAfterInsert;
```

4. Удалите записи всех клиентов, выведите их перечень (пустой) на экран, передумайте (зачем я это сделал?) и восстановите все записи. Снова выведите на экран список клиентов. Удалите записи некоторых клиентов, отобразите оставшиеся записи, восстановите все до состояния первой точки сохранения и выведите на экран выбранные столбцы.

```
DELETE FROM Customers;
SELECT CustomerID, LastName, City, State,
Zipcode FROM Customers;
ROLLBACK TO SAVEPOINT SaveAfterInsert;
SELECT CustomerID, LastName, City, State,
Zipcode FROM Customers;
DELETE FROM Customers WHERE WorkPhone IS
NULL;
SELECT CustomerID, LastName, City, State,
Zipcode FROM Customers;
ROLLBACK TO SAVEPOINT SaveAfterUpdate;
SELECT CustomerID, LastName, City, State,
Zipcode FROM Customers;
```

5. Создайте обработчик событий, который бы автоматически вставлял первичные ключи в таблицу *Customers* при добавлении записей о новых клиентах. Добавьте несколько таких записей. Отобразите их на экране.

```
CREATE OR REPLACE TRIGGER Customers_bi_trg
BEFORE INSERT ON Customers
FOR EACH ROW
BEGIN
    SELECT CustomerID_seq.NEXTVAL INTO
        :NEW.CustomerID FROM DUAL;
END;
/

INSERT INTO Customers (LastName, FirstName,
City, State, Zipcode)
VALUES ('Chang', 'Y.F.', 'Arcata', 'CA', '95518');
INSERT INTO Customers (LastName, FirstName,
City, State, Zipcode)
VALUES ('Newman', 'Richard', 'Blue Lake', 'CA', '95525');
INSERT INTO Customers (LastName, FirstName,
City, State, Zipcode)
VALUES ('Nasser', 'Jehad', 'Eureka',
'CA', '95503');
COMMIT;
SELECT CustomerID, LastName, City, State,
Zipcode
FROM Customers ORDER BY CustomerID;
```

6. Отключите буферизацию, удалите обработчик событий, удалите последовательность (последовательности) и все таблицы, закройте буферный файл.

```
SPOOL OFF
DROP TRIGGER Customers_bi_trg;
DROP SEQUENCE CustomerID_seq;
DROP SEQUENCE AgentID_seq;
```

```

DROP TABLE Agents CASCADE CONSTRAINTS
PURGE;
DROP TABLE AuditAgents CASCADE CONSTRAINTS
PURGE;
DROP TABLE LicenseStatus CASCADE
CONSTRAINTS PURGE;
DROP TABLE Customers CASCADE CONSTRAINTS
PURGE;

```

7. Введите `exit` и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus.
8. Откройте программу Блокнот, выберите пункт меню `File` (Файл), затем `Open` (Открыть). Найдите папку, в которой записан буферный файл <имя>Ch04Redwood.txt, щелкните на его имени, а затем на кнопке `Open` (Открыть).
9. Нажмите клавишу <Enter> в первой строке буферного файла, чтобы добавить пустую строку, введите в ней свое имя и фамилию. Выберите `File` (Файл), а затем `Print` (Печать), чтобы напечатать буферный файл. Закройте программу Блокнот.

## 2. Coffee Merchant

Администратор базы данных Coffee Merchant хочет, чтобы вы создали и заполнили таблицу `Employees`, в которой хранится информация о сотрудниках Coffee Merchant. Затем вы должны создать специальную таблицу `AuditEmployees`, в которой будет храниться информацию обо всех операциях вставки, обновления или удаления, применяемых к таблице `Employees`. Для этого вы напишете обработчик событий, который будет записывать образы затронутых строк таблицы `Employees` до и после изменения. В контрольной таблице будет содержаться следующая информация: слово “`Before`” или “`After`”, имя пользователя Oracle, внесшего изменения, дата и время модификации, значение первичного ключа `EmployeeID` и полей `LastName` и `CommissionRate`. Для проверки обработчика событий вы создадите точки сохранения и будете вставлять, обновлять и удалять строки из таблицы `Employees`. После модификации таблицы вы напечатаете строки `AuditEmployees`, чтобы просмотреть информацию, которую содержит эта таблица; кроме того, вся информация будет выводиться в буферный файл, который можно распечатать и изучить.

Мы рекомендуем использовать Блокнот или любой подходящий текстовый редактор для набора кода и последующего его копирования в SQL\*Plus. Завершив работу над командой, запишите файл программы Блокнот, содержащий команды SQL\*Plus и SQL, и напечатайте его. Напечатайте также буферный файл. Не забудьте поместить в файл свое имя и фамилию, дату, номер главы и другую информацию, которая может вам потребоваться для идентификации распечатки. Если в процессе будут встречаться ошибки, можете начать с первого пункта, запуская файл инициализации `Ch04Problem2.sql`, который удалит все созданные вами объекты и обновит таблицы Coffee Merchant. Итак, введите все показанные ниже команды в программу Блокнот, скопируйте их в SQL\*Plus и выполните.

- Запустите SQL\*Plus. Войдите в Oracle, введите и выполните нижеприведенную команду, которая инициализирует таблицы Coffee Merchant и создает контрольную таблицу AuditEmployees. Определить путь к файлу Ch04Problem2.sql и замените им аргумент <путь>.

```
CLEAR SCREEN
START C:\<путь>\Ch04Problem2.sql
CREATE TABLE AuditEmployees
(BeforeAfter NVARCHAR2(6),
AuditUsername NVARCHAR2(30),
AuditDate DATE,
EmployeeID INTEGER,
LastName NVARCHAR2(30),
Commission Number(4,4)
);
```

- Создайте обработчик событий, срабатывающий после выполнения команд INSERT, UPDATE или DELETE применительно к таблице Employees. (Таблица Employees была создана и заполнена при выполнении п. 1 с помощью сценария Ch04Problem2.sql.) Не забудьте ввести косую черту (/) и нажать клавишу <Enter>, чтобы скомпилировать и записать обработчик событий.

```
CREATE OR REPLACE TRIGGER
Employees_aiaauad_trg
AFTER INSERT OR UPDATE OR DELETE ON
Employees
FOR EACH ROW
BEGIN
  INSERT INTO AuditEmployees VALUES
  ('Before', user, sysdate,
   :OLD.EmployeeID, :OLD.LastName,
   :OLD.CommissionRate);
  INSERT INTO AuditEmployees VALUES
  ('After', NULL, NULL,
   :NEW.EmployeeID, :NEW.LastName,
   :NEW.CommissionRate);
END;
/
```

- Создайте точку сохранения и вставьте три строки с информацией о сотрудниках.

```
SAVEPOINT BeforeInsert;
INSERT INTO Employees (EmployeeID,
FirstName, LastName, Gender)
VALUES (1528, 'Luca', 'Pacioli', 'M');
INSERT INTO Employees (EmployeeID,
FirstName, LastName, Gender)
VALUES (3432, 'Melinda', 'English', 'F');
INSERT INTO Employees (EmployeeID,
FirstName, LastName, Gender)
VALUES (3692, 'Steve', 'Ballmer', 'M');
```

- Создайте другую точку сохранения и обновите несколько строк с информацией о сотрудниках, используя структуру CASE.

```

SAVEPOINT BeforeUpdate;
UPDATE Employees
  SET CommissionRate =
    CASE
      WHEN CommissionRate IS NULL THEN 0.025
      WHEN CommissionRate < 0.07 THEN CommissionRate * 1.15
      WHEN CommissionRate BETWEEN 0.07 and 0.09 THEN
        CommissionRate * 1.1
      ELSE CommissionRate * 1.05
    END;

```

5. Создайте еще одну точку сохранения, удалите информацию о нескольких сотрудниках и выведите на экран число оставшихся строк. Отмените команду DELETE, зафиксируйте (COMMIT) изменения и снова отобразите число строк Employees, чтобы проверить работу команды ROLLBACK.

```

SAVEPOINT BeforeDelete;
DELETE FROM Employees WHERE Gender = 'M';
SELECT COUNT(*) FROM Employees;
ROLLBACK TO SAVEPOINT BeforeDelete;
COMMIT;
SELECT COUNT(*) FROM Employees;

```

6. Начните вывод данных SQL\*Plus в буферный файл. Отобразите на экране имя пользователя Oracle и строки контрольной таблицы. Затем отключите обработчик событий и удалите выбранные строки таблицы Employees. Отобразите число строк, отмените все изменения и отобразите число оставшихся строк.

```

SPOOL C:\Ch04\Problem2-spoolfile.txt
SHOW USER
SELECT * FROM AuditEmployees;
ALTER TRIGGER Employees_aiauad_trg DISABLE;
DELETE FROM Employees Where Gender = 'F';
SELECT COUNT(*) FROM Employees;
ROLLBACK;
SELECT COUNT(*) FROM Employees;

```

7. Выведите на экран информацию об обработчике событий, включая его имя, тип и таблицу, с которой он связана. Удалите обработчик событий и все таблицы, созданные в данном упражнении, и остановите запись в буферный файл.

```

SELECT Trigger_Name, Trigger_Type,
Table_Name FROM User_Triggers;
DROP TRIGGER Employees_aiauad_trg;
DROP TABLE AuditEmployees CASCADE
CONSTRAINTS PURGE;
DROP TABLE Employees CASCADE CONSTRAINTS
PURGE;
SPOOL OFF

```

8. Введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из Oracle и закрыть SQL\*Plus.

9. Запишите текстовый файл в программе Блокнот, содержащий все введенные команды SQL\*Plus и выражения SQL. Откройте буферный файл, нажмите клавишу <Enter>, чтобы добавить новую строку в начале файла, введите свои имя и фамилию, запишите и напечатайте буферный файл. То же самое сделайте с файлом сценария SQL\*Plus/SQL — введите имя и фамилию в первой строке, запишите и распечатайте файл.

### 3. Rowing Ventures

Есть подозрение, что кто-то неумышленно вводит неверную информацию в таблицу Person, содержащую имя, пол, вес и дату рождения участников соревнований. Согласно большинству правил регат любой рулевой, вес которого меньше определенного значения, должен дополнительно нести груз, компенсирующий малый вес. Поскольку вес рулевого (да и всех остальных гребцов тоже) имеет очень большое значение, официальные устроители регаты хотят проследить за всеми изменениями таблицы Person. Если кто-либо изменит данные о весе рулевого (увеличит их), лодка получит ощутимое преимущество. От вас требуется создать и выполнить ряд выражений SQL, выполняющих сформулированную ниже задачу.

Прежде всего запустите программу Блокнот и SQL\*Plus. Найдите файл сценария Ch04Problem3.sql, инициализирующий базу данных для этой задачи. Запустите файл сценария, выполнив команду START (после нее укажите путь к файлу сценария и его имя). Далее создайте таблицу AuditPerson, содержащую столбцы, в которые записываются имя пользователя, текущая дата/время, и столбцы таблицы Person — PersonID, Weight и DateOfBirth. Создайте обработчик событий AFTER (Person\_aiauad\_trg), срабатывающий, когда кто-либо применяет к таблице Person команду INSERT, UPDATE или DELETE. Данная процедура записывает значения таблицы Person до и после изменения в соответствующих столбцах таблицы AuditPerson. Вставьте в таблицу две строки с некоторыми значениями PersonID (более 1100), LastName, FirstName, Gender, Weight и DateOfBirth. Установите дату рождения после 1972 года. Обновите столбец Weight для мужчин (Gender = 'M'), увеличив значения на 6 процентов. Зафиксируйте все изменения. Установите точку сохранения BeforeDeleteFemale. Удалите строки таблицы Person, в которых дата рождения гребца (DateOfBirth) находится между 1971 и 1972 гг. включительно. Отмените это изменение, откатив систему до созданной точки сохранения. Запустите буферный файл, а затем отобразите содержимое файла AuditPerson, выполнив следующую команду:

```
SELECT * FROM AuditPerson;
```

Выполните на экран информацию об обработчике событий, включая его имя, тип, таблицу, за которой он наблюдает, и состояние. Удалите обработчик событий и две таблицы — AuditPerson и Person. Отключите буферизацию. Распечатайте хронику

**ТАБЛИЦА 4.8.** Таблица AuditContact

| <i>Имя столбца</i> | <i>Тип данных</i> | <i>Содержимое</i>                                                                 |
|--------------------|-------------------|-----------------------------------------------------------------------------------|
| OperationType      | VARCHAR2(6)       | Операция, применяемая к таблице: INSERT, UPDATE или DELETE                        |
| BeforeAfter        | VARCHAR(6)        | Указывает выбранные значения до ('BEFORE') или после ('AFTER') изменения          |
| Username           | VARCHAR(30)       | Имя пользователя Oracle, который внес изменения                                   |
| ModDate            | Date              | Дата и время изменения строки                                                     |
| RecordPK           | INTEGER           | Копия столбца ContactID таблицы, используемый для идентификации измененной строки |
| RecordLastName     | NVARCHAR(20)      | Значение LastName измененной или вставленной записи                               |

сессии SQL\*Plus, включая все выполненные вами выражения. Напечатайте буферный файл. Не забудьте задать в обеих распечатках свое имя и фамилию. Выйдите из SQL\*Plus и закройте SQL\*Plus.

## 4. Broadcloth Clothing

База данных Broadcloth Clothing состоит из 17 таблиц. Для АБД самой важной из них является Contact, содержащая контактную информацию, включая идентификатор клиента ContactID, его имя, номер телефона (стационарный и мобильный), адрес, адрес электронной почты и т.д. Если эта таблица будет повреждена или модифицирована кем-то, то восстановить исходные значения будет проблематично. Исходя из этого компания, безусловно, будет поддерживать резервные копии, однако АБД хочет, чтобы вы реализовали некоторые средства наблюдения, записывающие все команды INSERT, UPDATE и DELETE, применяемые к таблице Contact. Прежде всего запустите SQL\*Plus, найдите файл Ch04Problem4.sql и запустите его в SQL\*Plus, чтобы инициализировать выбранные таблицы компании Broadcloths Clothing. Затем создайте контрольную таблицу AuditContact. Ее столбцы описаны в приведенной ниже таблице.

После этого создайте три обработчика событий, записывающие изменения таблицы Contact. Не забудьте записать код для каждого обработчика событий, чтобы вы могли распечатать его и изучить на досуге. Обработчик событий contact\_bu\_trg, присоединенный на уровне строки, запускается перед обновлением и записывает две строки в контрольную таблицу — значение столбца до изменения (в столбце OperationType указывается “UPDATE”, а в столбце BeforeAfter — “BEFORE”) и значение после изменения (значение OperationType равно “UPDATE”, значение BeforeAfter — “AFTER”). Обработчик событий contact\_bi\_trg, присоединенный на

уровне строки, срабатывает перед вставкой и вводит только одну строку в таблицу AuditContact. В столбец OperationType эта процедура помещает значение “**INSERT**”, а в столбец BeforeAfter — значение “**(new)**”; остальные столбцы таблицы AuditContact заполняются именем пользователя Oracle, текущей датой/временем, первичным ключом новой записи и значением LastName. (*Подсказка.* Используйте корреляционные имена :**NEW**.) Наконец, обработчик событий contact\_bd\_trg, присоединенный на уровне строки, запускается перед удалением и вставляет в таблице AuditContact только одну строку. В столбец OperationType эта процедура помещает значение “**DELETE**”, в BeforeAfter — “**(old)**”, в остальные поля заносится имя пользователя Oracle, текущая дата/время, первичный ключ и значение LastName старой записи. (*Подсказка.* Используйте корреляционные имена :**OLD**.) Вставьте две записи (используйте первичные ключи, значения которых больше 1234, и установите только значения FirstName и LastName.) Не забудьте использовать команду DESCRIBE, чтобы просмотреть структуру таблицы Contact. Удалите все строки, значение поля в столбце Nation которых равно Pakistan или Malaysia (первые буквы — прописные). Для строк, значение PrimaryLanguage которых равно “**Chinese**” (код печатайте точно так, как показано), увеличьте значение GMTDifference на два. Сделайте все изменения постоянными. Включите буферизацию, а затем введите и выполните выражение SELECT \* FROM AuditContact;. После этого отключите буферизацию, найдите и отредактируйте его, поместив свое имя в первую строку. Запишите, а затем напечатайте буферный файл и файлы сценариев, в которых отражено, как вы создали обработчик событий. Наконец, выполните команду DROP, чтобы удалить все обработчики событий, удалить таблицы Contact и AuditContact.



# ГЛАВА 5

## Организация запросов к базе данных

В этой главе...

- Использование выражения `SELECT` для извлечения данных из одной таблицы
- Применение выражения `SELECT` для фильтрации строк и отображения уникальных значений
- Сортировка возвращаемых строк в восходящем и нисходящем порядке по возвращаемым значениям столбцов
- Формирование расчетов и создание псевдонимов столбцов
- Создание в выражениях `SELECT` символьных, числовых, специальных функций и функций преобразования SQL
- Использование агрегирующих функций SQL `avg`, `count`, `max`, `min` и `sum`
- Группировка результатов по общим элементам
- Фильтрация групп с помощью оператора `HAVING`
- Использование команд форматирования отчетов SQL\*Plus

---

### Отображение информации из одной таблицы базы данных

В предыдущих главах вы создавали таблицы, заполняли их данными, налагали условия, модифицировали и удаляли информацию из таблиц. В этой главе вы узнаете, как извлекать информацию из таблиц базы данных. Реляционные базы данных обеспе-

чивают возможность доступа к элементам данных из одной или нескольких таблиц базы данных и отображение их в двухмерном табличном формате. Используя язык структурированных запросов (Structure Query Language – SQL) и клиент SQL\*Plus, вы можете писать выражения, обеспечивающие доступ к данным и согласующиеся с промышленным стандартом организации запросов к базам данных. Используя SQL, вы задаете, где найти данные, как связать несколько таблиц, какие столбцы информации вам требуются, какой критерий фильтрации использовать для отбора строк и в каком порядке необходимо возвращать строки получившейся таблицы. Для извлечения информации из таблиц реляционных баз данных применяется команда SQL SELECT. В отличие от DML-команд, выражение SELECT никак не изменяет содержимое таблиц.

Из этой и следующей главы вы узнаете, как получить точно ту информацию, которая вам требуется. Например, вам может понадобиться узнать, какой агент Reedwood Realty в этом месяце продал недвижимости на наибольшую сумму. А возможно, вам потребуется список домов для потенциального покупателя, имеющих определенный почтовый индекс и цену, принадлежащую заданному диапазону. В общем, можно привести еще множество вопросов, ответить на которые можно посредством запросов, для написания которых применяются выражения SELECT.

## Написание выражений SELECT

Простейшая форма выражения SELECT имеет следующий синтаксис:

```
SELECT * FROM <table-name>;
```

В данном выражении `<table-name>` — это имя таблицы, строки которой вы хотите извлечь, а звездочка (\*) в списке выбора указывает, что база данных Oracle должна извлечь *все* столбцы таблицы. Столбцы отбираются в таком же порядке, как они определены в таблице. Благодаря данному специальному символу вам не нужно вручную набирать все столбцы. С другой стороны, единственное, что вы можете контролировать при использовании звездочки, — это порядок перечисления столбцов в возвращаемых строках. Команда SELECT может быть сложной и, честно говоря, пугающей. Ниже показан упрощенный синтаксис команды SELECT с представленными основными операторами.

```
SELECT [DISTINCT] <select-list>
FROM {<table-list>}
[WHERE <conditions>]
[GROUP BY <group-by-list>]
[HAVING <group-conditions>]
[ORDER BY <order-list> [ASC | DESC]]
[FOR UPDATE <for-update-options>];
```

Необязательный оператор DISTINCT подавляет вывод на экран дублирующихся строк (имеющих одинаковые значения во всех столбцах, выбранных для отображения). Аргумент `select-list` представляет список имен столбцов таблицы и выражений, разделенных запятыми. Обращаясь к столбцам таблицы, принадлежащей другой

схеме, вы используете структуру вида схема.имя-таблицы. Аргумент `table-list`, следующий после зарезервированного слова `FROM`, содержит список из одной или нескольких таблиц, из которых будут извлекаться данные. Необязательный оператор `WHERE` определяет фильтр или условие поиска, которому должны удовлетворять все возвращаемые строки таблицы (таблиц). Необязательный оператор `GROUP BY` разбивает полученный набор результатов на группы. Каждая группа формируется на основе значений во всех столбцах, перечисленных в списке `group-by-list`. Еще один необязательный оператор, `HAVING`, задает дополнительный фильтр для сгруппированных строк. (Оператор `WHERE` применяется к отдельным строкам.) Необязательный оператор `ORDER BY` сортирует строки результата согласно значениям в перечисленных столбцах. Если после имени столбца следует уточнение `ASC` или `DESC`, строки сортируются по возрастанию или по убыванию в соответствии со значениями в данном столбце. Сортировка применяется слева направо, причем вы можете задавать любое число столбцов для сортировки. Вы можете сортировать по столбцам независимо от того, появляются ли они в возвращаемом наборе столбцов или нет. Если оператор `ORDER BY` пропущен, Oracle возвращает строки в непредсказуемом порядке.

Популярной областью применения необязательной фразы `FOR UPDATE` является блокировка выбранных строк. Операторы `SELECT`, `FROM`, `WHERE`, `HAVING` и `ORDER BY` должны идти строго в указанном порядке, однако сама команда `SELECT` может занимать любое количество строк. Другими словами, база данных Oracle не обращает внимания на жесткий возврат каретки в начало следующей строки или на дополнительные пробелы между словами и фразами. Впрочем, традиционно пользователи базы данных пишут каждую фразу с новой строки.

Ниже приведен простой пример запроса у Oracle выбранных столбцов таблицы `Customers`. Фильтр `City = 'Arcata'` указывает, что нас интересуют только клиенты, проживающие в г. Арката, а оператор `ORDER BY` сортирует строки согласно значениям `LastName`, затем (если значения `LastName` одинаковы) по значениям `FirstName`. Хотя вы можете задавать ключевые слова Oracle, имена таблиц и столбцов строчными буквами, прописными буквами или строчными и прописными буквами, мы рекомендуем придерживаться одного стиля именования объектов. В данной книге зарезервированные слова SQL представляются прописными буквами, а для представления имен столбцов применяются прописные и строчные буквы. Разумеется, любой критерий, заданный в одинарных кавычках, должен буквально соответствовать записям в базе данных. (В приведенном примере название города Арката в базе данных выглядит точно так же, как значение в столбце `City` таблицы `Properties`, т.е. первая буква прописная, остальные строчные.)

```
SELECT FirstName, LastName, Address, City, State, Zipcode  
FROM Customers WHERE City = 'Arcata' ORDER BY LastName, FirstName;
```

Прежде чем продолжить далее, настроим всю базу данных Redwood Realty. На компакт-диске, прилагаемом к настоящей книге, вы можете найти файл сценария,

который создает таблицы Redwood Realty, заполняет их и устанавливает первичный и внешние ключи. Перед запуском SQL\*Plus найдите файл сценария BuildRedwood.sql в файлах, относящихся к главе 5, и запишите путь к этому файлу. Помимо этого файла вы найдете еще десять файлов сценариев, которые вызывает BuildRedwood.sql. Перепишите их все в одну папку. Придерживаясь описанных ниже действий, создайте с нуля базу данных Redwood Realty. Для выполнения файла сценария требуется время — возможно, порядка минуты, в зависимости от возможностей вашего компьютера. Этот файл сценария дает “точку отсчета”, формируя с нуля восемь таблиц, составляющих базу данных. В конце выполнения сценария на экран выводится число строк в созданных таблицах.

Для того чтобы создать базу данных Redwood Realty, выполните следующие действия.

1. Запустите SQL\*Plus и войдите в базу данных Oracle. (*Примечание:* интерфейс iSQL\*Plus в данном задании не используйте.)
2. Найдите файлы сценариев базы данных Redwood Realty в папке RedwoodRealty. Подставьте соответствующий путь вместо <путь> в приведенном ниже выражении и нажмите клавишу <Enter> в конце строки.

```
START <путь>\BuildRedwood
```

Немного подождите, и Oracle выведет сообщения, подтверждающие, что все таблицы были созданы и заполнены (рис. 5.1).

Теперь, имея базу данных Redwood Realty, вы можете запустить команду SELECT и отобразить на экране информацию из одной таблицы.

Для того чтобы отобразить на экране информацию из таблицы, выполните следующие действия.

1. Введите приведенные ниже фразы и нажмите клавишу <Enter> в конце каждой строки. Эти команды SQL\*Plus очищают экран и так форматируют столбцы, чтобы они более компактно располагались на экране.

```
CLEAR SCREEN
COLUMN FirstName FORMAT A12
COLUMN LastName FORMAT A12
COLUMN Gender FORMAT A6
COLUMN Title FORMAT A11
```

2. Введите следующий код, чтобы отобразить на экране все строки таблицы Agents (нажмите клавишу <Enter> в конце последней строки, чтобы скомпилировать и выполнить выражение SELECT):

```
SELECT FirstName, LastName, Gender, Title
FROM Agents;
```

The screenshot shows the Oracle SQL\*Plus interface. The menu bar includes File, Edit, Search, Options, Help, and a logo. The main window displays the following text:

```

* Oracle SQL*Plus
File Edit Search Options Help
Agents      table created/populated.
ContactReason table created/populated.
CustAgentList table created/populated.
Customers    table created/populated.
LicenseStatus table created/populated.
Listings     table created/populated.
Properties   table created/populated.
SaleStatus   table created/populated.
Foreign key constraints added.

Table          Rows
-----
Agents         29
ContactReason 3
CustAgentList  1,018
Customers      2,500
LicenseStatus  16
Listings       502
Properties     2,000
SaleStatus      3

SQL>

```

**Рис. 5.1.** Сообщения, генерируемые файлом сценария BuildRedwood

Результаты выполнения команды показаны на рис. 5.2. На экран выведены выбранные столбцы всех строк таблицы Agents. Порядок строк произвольный.

3. Введите exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus. В данной главе вы будете поочередно использовать SQL\*Plus и iSQL\*Plus.

## Выбор всех столбцов

Если вы хотите отобразить все столбцы из таблицы, у вас есть два варианта записи аргумента select-list команды SELECT. Вы можете перечислить после слова SELECT все столбцы, разделив их запятыми. Для длинных списков столбцов такой вариант нежелателен. Второй, более короткий путь — это использовать в аргументе select-list символ “звездочка” (\*). Основной недостаток второго подхода заключается в том, что столбцы будут перечислены в том же порядке (слева направо), в каком они идут в определении таблицы. Если это непринципиально, рекомендуем использовать более короткий вариант запроса. Попробуем сделать это с помощью

The screenshot shows the Oracle SQL\*Plus interface with three distinct SELECT statements displayed as tables:

```

SQL> COLUMN Title FORMAT A11
SQL> SELECT FirstName, LastName, Gender, Title
  2  FROM Agents;

FIRSTNAME LASTNAME GENDER TITLE
----- -----
Kai        Marcoux   M    Salesperson
Tobias    Carling   M    Salesperson
Elizabeth Dahlens  F    Salesperson
Ramanathan Rowe     M    Salesperson
Heather   Sheibani  F    Salesperson
Bruce     Voss      M    Salesperson
Cecilia   Romero   F    Salesperson
Cornelis  Dann      M    Salesperson
Danial   Silverburg M    Salesperson
Clair     Robinson  F    Salesperson
Nancy    Piperova  F    Salesperson

FIRSTNAME LASTNAME GENDER TITLE
----- -----
Edwin    Townsend  M    Salesperson
Essi     Okindo    M    Broker
Lee      Reed      F    Salesperson
Stanislaw Soltwedel M    Salesperson
Belinda  Chong     F    Salesperson
Ricki    Selby     F    Broker
Jessica  Taylor    F    Broker
Tim      St-Onge   M    Salesperson
Jackson  Flamenbaum M    Salesperson
Barbara  Herring   F    Salesperson
Sindisive Weber     M    Salesperson

FIRSTNAME LASTNAME GENDER TITLE
----- -----
James    Kellogg   M    Salesperson
Crystal Fernandez F    Salesperson
Christine Williams  F    Salesperson
David    Gagnon    M    Broker
Lora     Allee     F    Broker
Tim      Schutz    M    Salesperson
Patricia Lewis     F    Broker

```

29 rows selected.

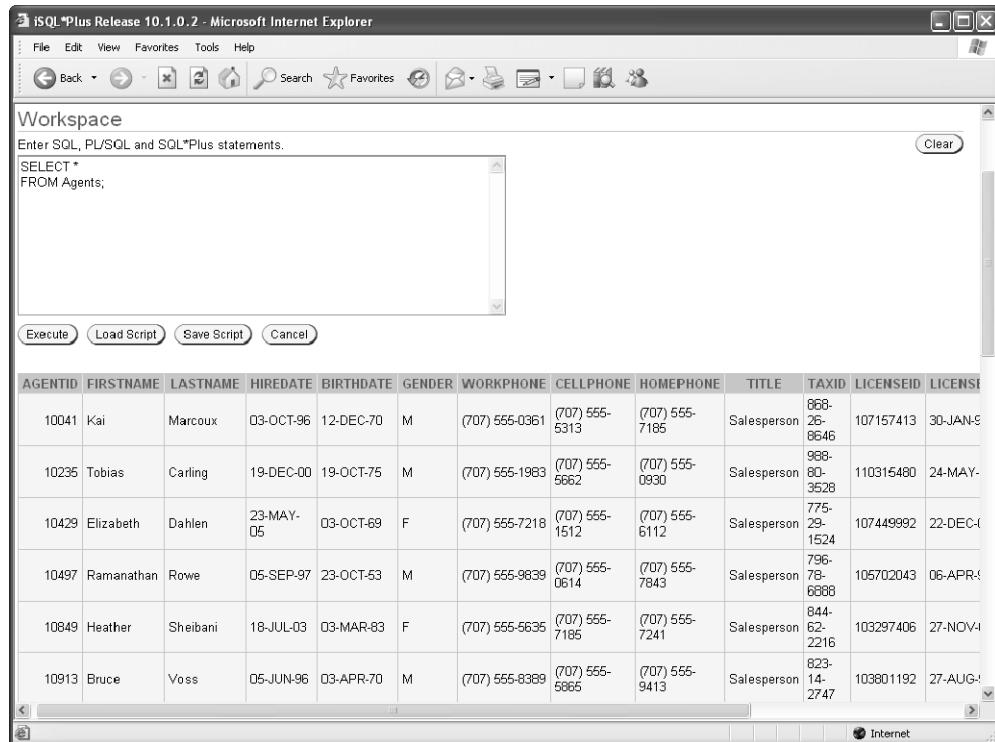
SQL>

**Рис. 5.2.** Выражение SELECT, которое возвращает информацию, касающуюся агента по недвижимости

*i*SQL\*Plus, поскольку в *i*SQL\*Plus таблица с большим числом столбцов отображается гораздо лучше, чем в SQL\*Plus.

Итак, чтобы отобразить на экране все столбцы и все строки таблицы, выполните следующие действия.

1. Запустите свой браузер, а затем *i*SQL\*Plus. Войдите в Oracle, используя свое имя пользователя и пароль.
  2. Щелкните в текстовом окне Workspace и введите следующий код. (В *i*SQL\*Plus точка с запятой, завершающая выражение, является необязательной. Мы использовали ее просто для поддержания единообразия.)
- ```
SELECT * FROM Agents;
```
3. Щелкните на кнопке Execute, чтобы выполнить выражение. Если потребуется, исправьте ошибки в текстовом окне Workspace и повторите данный этап. Oracle отобразит на экране несколько строк и все столбцы таблицы Agents (рис. 5.3). Для просмотра извлеченной информации используйте горизонтальную и вертикальную полосы прокрутки.
  4. Щелкните на кнопке Logout, чтобы выйти из Oracle, а затем закройте браузер.



**Рис. 5.3.** Вывод на экран всех столбцов таблицы Agents

**Совет.** Если в ходе работы над книгой iSQL\*Plus в перерывах между группами действий выгружает вас из Oracle, значит, вы превышаете максимальное время ожидания этого интерфейса. Следует помнить, что iSQL\*Plus измеряет время, прошедшее с момента последнего нажатия клавиши или щелчка на кнопке. Чтобы избежать принудительного выхода из системы, можно увеличить время ожидания или периодически щелкать мышью в текстовом окне Workspace. Если же это все-таки случилось, запустите iSQL\*Plus и еще раз войдите в систему базы данных.

## Использование оператора DISTINCT для отображения уникальных строк

Если перед аргументом select-list выражения SELECT указать необязательный оператор DISTINCT, то из результата будут удалены дублирующиеся строки. Дубликатами считаются строки, для которых значения во всех столбцах, возвращаемых командой SELECT, совпадают с соответствующими значениями по крайней мере одной другой строки. Вообще, дублирующиеся строки — явление очень редкое, если вы отбираете из таблицы много столбцов (особенно, если один из этих столбцов содержит

The screenshot shows the Oracle SQL\*Plus interface. The command line displays:

```
SQL> DESCRIBE CustAgentList
SQL> SELECT DISTINCT CommissionRate
  2  FROM CustAgentList;
```

The output shows the description of the CustAgentList table with columns CUSTOMERID, AGENTID, LISTINGID, CONTACTDATE, CONTACTREASON, BIDPRICE, and COMMISSIONRATE. It then lists the unique values for the COMMISSIONRATE column, which are .03, .04, .05, and .06.

**Рис. 5.4.** Извлечение уникальных строк с использованием оператора DISTINCT

жит первичные ключи). Однако если вы выбираете всего лишь несколько столбцов, оператор DISTINCT может оказаться весьма полезным. Предположим, например, что вы желаете узнать, в каком диапазоне изменяются комиссионные, которые покупатели платят агентству за услуги. Соответствующий столбец находится в таблице CustAgentList. Чтобы извлечь требуемую информацию, вы можете использовать следующий запрос:

```
SELECT CommissionRate FROM CustAgentList;
```

После этого вам придется просматривать очень длинный список (1018 строк), чтобы найти различные значения ставки комиссионных. Очевидно, вам такое не подходит. Включив в команду оператор DISTINCT, вы отсечете дублирующиеся строки, оставив только уникальные значения, т.е. в данном случае стоит использовать команду

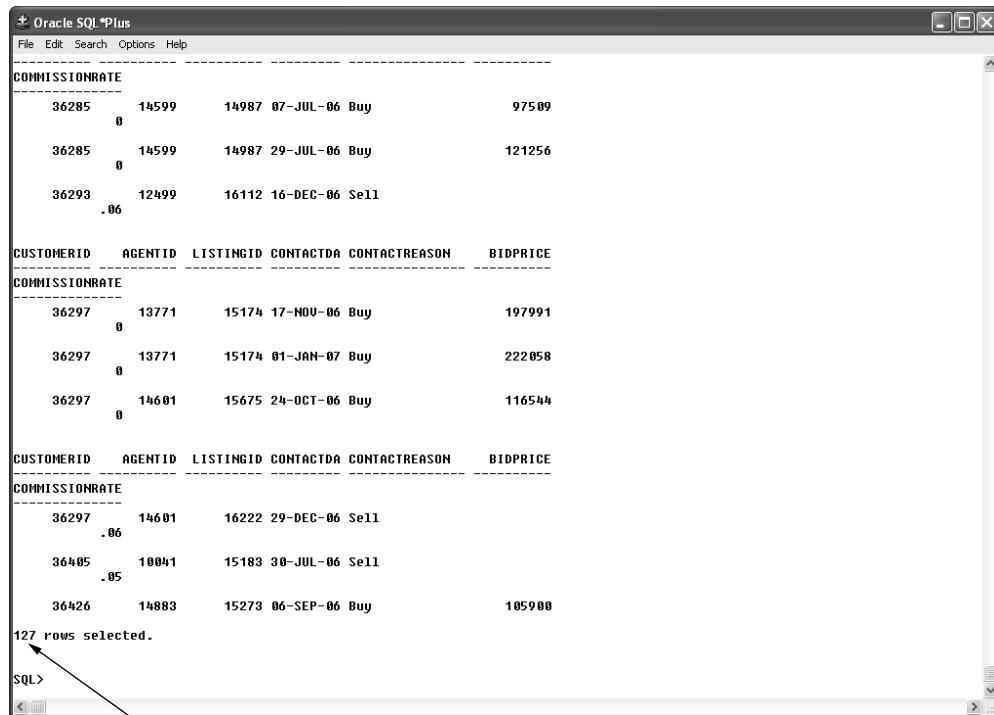
```
SELECT DISTINCT CommissionRate FROM CustAgentList;
```

В ответ на запрос в такой форме вы получите всего пять строк. Результат выполнения данной команды SELECT после выражения DESCRIBE CustAgentList показан на рис. 5.4. Просмотрев список, вы увидите, что существует всего четыре ставки комиссионных: 3, 4, 5 и 6%. (Нуль процентов в данной схеме означает покупателя.)

## Прерывание выполняемого запроса

Изучая Oracle, вы несколько раз сталкивались с выражениями SELECT, которые при запуске в SQL\*Plus выполнялись бесконечно долго. Вообще, подобные запросы могут возвращать сотни и даже тысячи строк. “Достучаться” до SQL\*Plus и прервать длительный цикл обработки достаточно просто: нажмите комбинацию клавиш <Ctrl+C>, чтобы приостановить выполнение текущей команды. Посмотрим, как это работает.

Для того чтобы прервать длительное выполнение выражения SQL в интерфейсе SQL\*Plus, выполните следующие действия.



Число строк, отображенных до прерывания

Рис. 5.5. Прерывание длинного вывода SQL\*Plus

1. Запустите SQL\*Plus, войдите в Oracle и введите следующий код. Нажмите клавишу <Enter> в конце последней строки, чтобы отправить команду Oracle для выполнения. Выражение SELECT вернет 1018 строк!

```
SELECT * FROM CustAgentList
```

2. Подождав несколько секунд, вы можете прервать работу SQL\*Plus и остановить выдачу значений. Нажмите комбинацию клавиш <Ctrl+C> (нажмите клавишу <Ctrl>, нажмите клавишу <C> и отпустите обе клавиши одновременно). SQL\*Plus остановит вывод результатов на экран, отобразит число уже выданных строк, а затем запрос командной строки SQL> (рис. 5.5).

## Использование условий поиска для фильтрации результатов

Для фильтрации строк, возвращаемых Oracle из базы данных, в команде SELECT применяется необязательный оператор WHERE. Вы уже использовали оператор WHERE в главе 4 — тогда это были команды UPDATE и DELETE. В выражении SELECT оператор WHERE применяется с той же целью. Условие поиска, следующее после WHERE,

определяет конкретные строки таблицы или таблицы, которые должна извлечь база данных Oracle. Для этого указываются выражения и значения, которые должны согласовываться с возвращаемыми строками (или оцениваться как `true`). Напомним, оператор `WHERE` имеет следующий синтаксис:

```
WHERE <column-name> <comparison-operator> <search-expression>
```

Аргумент `column-name` задает столбец таблицы, значение которого проверяет Oracle. Вместо `comparison-operator` подставляется один из операторов сравнения, который для каждой строки таблицы сравнивает значение в столбце таблицы с `search-expression`. (Операторы сравнения перечислены в табл. 4.3.) Аргумент `search-expression` часто представляет собой значение — строку символов или число. Оператор `WHERE` допускает объединение нескольких групп `"column-name/comparison-operator/search-expression"`, разделенных логическими операторами. (Логические операторы рассмотрены в главе 4, табл. 4.4.)

## Использование операторов сравнения

Операторы сравнения позволяют сравнивать два элемента и определять между ними отношения “меньше”, “равно”, “больше” или “не равно”, а также комбинации этих отношений. Ниже приведены примеры команд `WHERE`, извлекающих выбранные строки из таблицы `Customers`.

```
...WHERE City = 'Blue Lake'  
...WHERE CustomerID = 25343  
...WHERE FirstName <> 'Randy'
```

Чтобы лучше понять оператор `WHERE`, организуем в SQL\*Plus несколько запросов к таблицам базы данных Redwood Realty. Несмотря на то что мы применяем выражения `SELECT` к таблицам, содержащим несколько тысяч строк, Oracle будет возвращать относительно небольшое число строк, поскольку оператор `WHERE` задает условие поиска, которому удовлетворяет лишь сравнительно небольшое число строк.

Итак, чтобы поэкспериментировать с оператором `WHERE`, необходимо выполнить следующие действия.

1. Запустите SQL\*Plus, введите `CLEAR SCREEN` и нажмите клавишу `<Enter>`, чтобы выполнить следующую команду `SELECT`:

```
SELECT CustomerID, FirstName, LastName  
FROM Customers  
WHERE LastName = 'Nguyen';
```

Oracle вернет восемь строк, имеющих различные значения `CustomerID`.

2. Введите и выполните следующую команду:

```
SELECT OwnerID, Bedrooms, Bathrooms, Stories, SqFt  
FROM Properties  
WHERE SqFt = 2100;
```

Oracle вернет пять строк — по одной на каждый дом, показатель SqFt (площадь в квадратных футах) которого равен 2100.

3. Введите и выполните следующую команду:

```
SELECT AgentID, LastName, FirstName  
FROM Agents  
WHERE Title <> 'Salesperson';
```

Oracle вернет шесть строк с информацией об агентах, имеющих звание, отличное от Salesperson.

4. Введите и выполните следующую команду. (Значение linesize устанавливается равным 90, чтобы каждая строка таблицы помещалась в одну строку.)

```
SET Linesize 90  
SELECT *  
FROM CustAgentList  
WHERE BidPrice > 350000;
```

5. Введите и выполните следующую команду, чтобы отобразить на экране выбранные строки таблицы Listings:

```
SELECT ListingID, BeginListDate, EndListDate, AskingPrice  
FROM Listings  
WHERE BeginListDate <= '26-JUN-2006';
```

Oracle вернет строки, в которых дата выставления на продажу назначена до 27 июня 2006 г. Результаты выполнения пп. 4 и 5 показаны на рис. 5.6.

## Правила для символов и дат

Используя символы в условиях поиска оператора WHERE, помните, что прописные буквы не эквивалентны строчным. Например, приведенное ниже условие поиска WHERE в команде SELECT не вернет ни одной строки из таблицы Agents.

```
SELECT LastName  
FROM Agents  
WHERE Title = 'salesperson'
```

Это объясняется тем, что все значения Title в таблице Agents записаны строчными и прописными буквами: каждое начинается и прописной буквы, за которой следуют строчные. Это означает, что строка символов, подобная “salesperson”, не соответствует строке символов “SALESPERSON” (а также любой другой комбинации прописных и строчных букв). (Позже вы узнаете, как обойти это ограничение, используя несколько простых функций Oracle.) Кроме того, помните, что строку символов, если она является частью лiteralной константы в операторе WHERE, необходимо помещать в одинарные кавычки. И наоборот, числа в условиях поиска в кавычки *не берутся*.

```

SQL> SET Linesize 90
SQL> SELECT *
2 FROM CustAgentList
3 WHERE BidPrice > 350000;

CUSTOMERID AGENTID LISTINGID CONTACTID CONTACTREASON BIDPRICE COMMISSIONRATE
----- ----- ----- ----- ----- -----
35979 15293 15685 29-JAN-07 Buy 369836 0
40341 12381 14998 06-AUG-06 Buy 485196 0
41040 12875 15685 15-FEB-07 Buy 399745 0
25571 10849 15685 22-JAN-07 Buy 355326 0
26075 14677 16021 13-JAN-07 Buy 402402 0
31598 10897 15375 08-OCT-06 Buy 563091 0
31598 10897 15375 04-NOV-06 Buy 688638 0
32809 10429 15028 13-SEP-06 Buy 377155 0
32809 10429 15028 29-OCT-06 Buy 417423 0
48207 15233 15390 24-SEP-06 Buy 361185 0
48207 15233 15390 25-SEP-06 Buy 382746 0

11 rows selected.

SQL> --
SQL> SELECT ListingID, BeginListDate, EndListDate, AskingPrice
2 FROM Listings
3 WHERE BeginListDate <= '26-JUN-2006';

LISTINGID BEGINLIST ENDLISTDATE ASKINGPRICE
----- ----- ----- -----
14979 23-JUN-06 23-NOV-06 126000
14981 23-JUN-06 23-DEC-06 127000
14987 23-JUN-06 23-AUG-06 124950
14992 23-JUN-06 23-JUL-06 154685
14994 23-JUN-06 23-OCT-06 282500
14997 23-JUN-06 23-SEP-06 145000
14998 24-JUN-06 24-OCT-06 495000
15001 25-JUN-06 25-JUL-06 114950
15005 26-JUN-06 26-SEP-06 122000
15007 26-JUN-06 26-NOV-06 107000
15010 26-JUN-06 26-NOV-06 129950

11 rows selected.

SQL>

```

**Рис. 5.6.** Использование оператора WHERE для фильтрации строк таблицы

Возможно, вам потребуется создать условие поиска WHERE, в котором задействованы даты. Помните, что даты представляются в формате Oracle, принятом по умолчанию, — DD-MON-YY, где DD — день; MON — трехбуквенное сокращенное обозначение месяца; YY — две последние цифры года. Ранее уже говорилось, что значения YY, превышающие 50, относятся к предыдущему веку, а значения YY, меньшие или равные 50, — к текущему.

Как видно из предыдущего примера использования оператора WHERE, условия поиска вполне могут содержать даты и реляционные операторы. Предположим, что вы желаете перечислить всех агентов по недвижимости, срок действия лицензий которых истекает 1 июня 2005 года или позже. Приведенное ниже выражение SELECT выбирает строки, в которых даты LicenseExpire больше или равны '01-JUN-05'.

```

SELECT FirstName, LastName
FROM Agents
WHERE LicenseExpire >= '01-JUN-05';

```

Подобным образом вы можете перечислить всех агентов, срок действия лицензий которых истек на текущий момент, отобрав записи, значение LicenseExpire которых меньше (предшествует) текущей даты (извлекается посредством псевдостолбца SYSDATE). Для этого необходимо выполнить следующую команду SELECT:

**ТАБЛИЦА 5.1.** Реляционные операторы SQL

<i>Оператор</i>	<i>Значение</i>
BETWEEN . . . AND	Сопоставляется с диапазоном значений
IN	Сопоставляется с конечным списком допустимых значений
IS NULL	Сопоставляется со значением NULL. Является единственной возможностью использования значения NULL в условиях поиска
LIKE	Сопоставляется с шаблоном (строкой символов)

The screenshot shows the Oracle SQL\*Plus interface. The command entered is:

```
SQL> SELECT PropertyID, AskingPrice
  2  FROM Listings
  3 WHERE AskingPrice BETWEEN 212000 AND 235000;
```

The results are displayed in a grid:

PROPERTYID	ASKINGPRICE
1893	230000
1898	230000
1899	235000
1887	225000
1879	219950
1878	219000
1895	234000
1900	235000
1889	227000
1870	212000
1876	217316
1877	217500

Below the grid, the message "12 rows selected." is shown, followed by the SQL prompt "SQL>".

**Рис. 5.7.** Использование оператора BETWEEN . . . AND

```
SELECT FirstName, LastName
  FROM Agents
 WHERE LicenseExpire < SYSDATE;
```

## Использование операторов SQL

Операторы SQL, которые иногда объединяют с операторами сравнения, позволяют фильтровать или ограничивать возвращаемые строки, основываясь на согласовании строк с шаблоном, списках допустимых значений, значениях NULL или диапазонах допустимых значений. Некоторые распространенные операторы SQL представлены в табл. 5.1.

### BETWEEN

Оператор “между”, BETWEEN . . . AND, очень удобен при поиске строк, в указанных столбцах которых значения попадают в определенный диапазон значений. На рис. 5.7 показано 12 отобранных строк, в которых AskingPrice принадлежит диапазону от 212 000 до 235 000 долларов включительно. Обратите внимание, что один дом имеет цену, точно соответствующую нижней границе, а два — цены, соответствующие верхней границе.

## IN

Оператор `IN` возвращает строки, в которых значения согласуются с одним из значений, указанных в списке допустимых. Чтобы строка была возвращена в ответ на запрос, значения в столбцах таблицы должны совпадать с одним из перечисленных значений. Оператор `IN` особенно удобен, когда в качестве критерия отбора используется небольшое число различных значений. Например, вам требуется отобразить на экране информацию обо всех заказчиках, проживающих в штате Огайо, Индиана, Мичиган или Иллинойс. Проще всего данная задача решается с помощью следующего оператора `WHERE`:

```
WHERE State IN ('Ohio', 'Indiana', 'Michigan', 'Illinois')
```

Сравните это решение с эквивалентной, но существенно большей версией, использующей операторы сравнения и логические операторы (они описаны ниже):

```
WHERE State = 'Ohio' OR State = 'Indiana'  
      OR State = 'Michigan' OR State = 'Illinois'
```

## LIKE

Оператор `LIKE` с помощью групповых символов находит в строке заданные структуры символов. *Групповыми* называются специальные символы, подменяющие один или несколько символов в строке поиска. Используя в команде `WHERE` групповые символы и оператор `LIKE`, можно определить, соответствует ли какая-то строка символов столбца таблицы заданному шаблону. Для создания шаблонов поиска применяются обычные символы и два групповых символа Oracle: `%` и `_` (подчеркивание). Символ подчеркивания (`_`) согласуется с одним символом в указанной позиции, а символ процента (`%`) согласуется с любым числом символов (включая нуль) в заданной позиции. Например, чтобы перечислить всех клиентов (`Customers`), фамилия которых начинается с буквы `H`, можно использовать следующее выражение:

```
SELECT FirstName, LastName, Address, City, State  
FROM Customers  
WHERE LastName LIKE 'H%';
```

Если вы хотите найти клиентов, почтовый индекс которых начинается с `955` и заканчивается `1` независимо от четвертой цифры, то можете использовать следующий оператор `WHERE` с символом подчеркивания на месте четвертой цифры:

```
... WHERE ZipCode LIKE '955_1';
```

Если один из символов, которые вам требуется найти в столбце, — это символ процента или подчеркивания, используйте опцию `ESCAPE`. Данная опция задает символ, который, располагаясь перед групповым символом в команде `LIKE`, превращает его из обычного знака в групповой. Допустим, вам требуется найти в столбце строку, содержащую знак процента (например, “`... almost 35% of all people ...`”). Для этого вы используете команду, приведенную ниже `WHERE`, и необязательную опцию `ESCAPE`, задающую в качестве управляющего символа обратную косую черту. В данном случае использование обратной косой черты определяет первый знак процента как групповой символ, а во второй раз процент считается литеральным символом.

```
... WHERE StatusText LIKE '\%almost 35% of all people' ESCAPE '\'
```

**Совет.** В Oracle 10g появился еще более мощный набор инструментов сопоставления строк, называемый *регулярными выражениями*. Хотя данная тема в книге рассматриваться не будет, ознакомиться с ней все же стоит. Введите в любом поисковике фразу “регулярные выражения”, и вы найдете массу статей, посвященных этой теме.

#### IS NULL

Иногда необходимо написать запрос, отображающий строки со значением NULL в определенном столбце (т.е. соответствующее значение неизвестно). Например, вы можете перечислить строки (соответствующие домам), в графе YearBuilt которых указано NULL, или вывести на экран строки таблицы Agents, в которых data LicenseExpire равна NULL. Поскольку NULL обозначает отсутствие информации, вы не можете определить с помощью оператора сравнения, что значение столбца является пустым. Вместо этого вы можете использовать опцию IS NULL в операторе WHERE. Разумеется, вы также можете использовать IS NOT NULL для поиска всех значений столбца за исключением NULL. Приведенный ниже запрос перечисляет выбранные столбцы таблицы Agents, в которых значение поля LicenseExpire неизвестно.

```
SELECT FirstName, LastName  
FROM Agents  
WHERE LicenseExpire IS NULL;
```

## Использование логических операторов

Напомним (см. главу 3), что существует всего три логических оператора: AND, OR и NOT. Они являются частью выражения, входящего в оператор WHERE и ограничивающего возвращаемые строки на основе того, что два условия выполняются одновременно (AND) или выполняется одно из них (OR). Логические операторы применяются для объединения частей условий поиска. Оператор NOT инвертирует следующее за ним логическое выражение. Например, предположим, что вы желаете приобрести дом в Арката, построенный после 1999 г. и имеющий размер (SqFt) не менее 2 000 квадратных футов. Более того, в нем должно быть не меньше трех комнат и от двух до четырех санузлов. Хотя с помощью операторов сравнения вы можете наложить любое из указанных условий (например, SqFt > 2000), чтобы гарантировать выполнение *всех* условий, необходимо использовать логический оператор AND, связывающий несколько отдельных условий в единый блок.

При написании условных выражений необходимо учитывать *приоритет операторов* — порядок, в котором Oracle вычисляет подвыражения. Рассмотрим, например, следующее условное выражение WHERE:

```
WHERE City = 'Arcata' AND YearBuilt < 2004 OR Bedrooms = 3;
```

Здесь Oracle не вычисляет выражение слева направо — порядок расчетов определяется приоритетом операторов. Любой из операторов сравнения имеет более высокий приоритет, чем любой из логических операторов. Самый высокий приоритет имеют круглые скобки, поэтому выражения в скобках вычисляются прежде всего. После этого вычисляются выражения, содержащие `<`, `=`, `>` и другие комбинации операторов отношения. Наконец, рассчитываются все подвыражения с логическими операторами. В приведенном выше операторе `WHERE` первым вычисляется фраза `City = 'Arcata'` и определяется ее значение — `true` или `false` в зависимости от текущей строки таблицы. После этого вычисляется подвыражение `YearBuilt = 2004` и т.д. Наконец, Oracle вычисляет, чему равно объединение трех результатов `true/false` с помощью двух логических операторов. Из логических операторов самый высокий приоритет имеет `NOT`, поэтому он вычисляется в первую очередь. Следующий по старшинству приоритет имеет `AND`, самый низкий — `OR`. Подвыражение, содержащее операторы с равным приоритетом, вычисляется слева направо.

А теперь попрактикуемся в использовании изученных условий поиска с помощью SQL\*Plus. Как обычно, мы рекомендуем использовать программу Блокнот для набора отдельных выражений, скопировать команды в SQL\*Plus и нажать клавишу `<Enter>`, чтобы их выполнить. Таким образом, вы получите текстовый файл с записью всех команд, выполненных в SQL\*Plus.

Для того чтобы создать выражения `SELECT`, использующих различные условия поиска, выполните следующие действия.

- Если необходимо, запустите SQL\*Plus и войдите в Oracle.
- Ведите `CLEAR SCREEN` и нажмите клавишу `<Enter>`.
- Выполните следующий код, чтобы отобразить на экране адреса и другую информацию о домах в Аркатае, построенных после 1991 года и имеющих не менее 2 000 квадратных футов полезной площади:

```
SELECT Address, YearBuilt, SqFt, Bedrooms, Bathrooms
FROM Properties
WHERE City = 'Arcata' AND YearBuilt > 1991 AND SqFt >= 2000;
```

- Выполните следующий код, чтобы отобразить на экране недвижимость Блу Лейк (`Blue Lake`) или Тринидада (`Trinidad`), расположенную на улицах, названия которых заканчиваются на “Ln”. Поскольку вы не знаете, какими буквами (прописными или строчными) записывается улица в таблице базы данных, необходимо перебрать все четыре возможных варианта с помощью выражения `OR`. (Позже будет описан более простой метод решения этой задачи — с использованием функций.)

```
SELECT Address, City
FROM Properties
WHERE City IN ('Blue Lake', 'Trinidad') AND
(Address LIKE '%Ln' OR Address LIKE '%ln'
 OR Address LIKE '%LN' OR Address LIKE '%ln');
```

The screenshot shows the Oracle SQL\*Plus interface. The menu bar includes File, Edit, Search, Options, Help, and a window title bar. The main area displays a SQL query and its results.

```

SQL> SELECT SaleStatusID, BeginListDate, AskingPrice
  2  FROM Listings
  3  WHERE SaleStatusID = 101
  4    AND BeginListDate BETWEEN '01-JUN-2006' AND '30-JUN-2006'
  5    AND AskingPrice > 130000;

```

SALESTATUSID	BEGINLIST	ASKINGPRICE
101	23-JUN-06	154685
101	23-JUN-06	145000
101	24-JUN-06	405000
101	27-JUN-06	178128
101	28-JUN-06	183000
101	29-JUN-06	138950

6 rows selected.

SQL>

Рис. 5.8. Пример запроса с логическими операторами и операторами сравнения

5. Выполните следующий код, чтобы перечислить столбцы SaleStatusID, BeginListDate и SellingPrice недвижимости, выставленной на продажу (значение SaleStatusID равно 101) в июне 2006 года (BeginListDate):

```

SELECT SaleStatusID, BeginListDate, AskingPrice
FROM Listings
WHERE SaleStatusID = 101
    AND BeginListDate BETWEEN '01-JUN-2006' AND '30-JUN-2006';

```

6. Модифицируйте предыдущий запрос, добавив ограничение,— запрашиваемая цена (AskingPrice) недвижимости превышает 130 000.

```

SELECT SaleStatusID, BeginListDate, AskingPrice
FROM Listings
WHERE SaleStatusID = 101
    AND BeginListDate BETWEEN '01-JUN-2006' AND '30-JUN-2006'
    AND AskingPrice > 130000;

```

Последний запрос и выданные в качестве результата строки показаны на рис. 5.8. У вас должно получиться то же самое.

## Сортировка

До этого момента все примеры выражений SELECT возвращали строки в произвольном порядке. Чтобы проконтролировать порядок отображения строк на экране, в команде SELECT необходимо использовать необязательный оператор ORDER BY, который имеет следующий синтаксис:

```

ORDER BY {<column-name>|<result-column-number>} [ASC | DESC]
[NULLS FIRST|NULLS LAST]
[, {<column-name>|<result-column-number>} [ASC | DESC]
[NULLS FIRST|NULLS LAST]]...

```

Аргумент `column-name` представляет имя столбца в таблице (возможно, даже столбца, который не возвращается в ответ на запрос). Аргумент `result-column-number` представляет положение, которое занимает возвращаемый результат в списке столбцов (число от единицы до количества возвращаемых столбцов). `ASC` обозначает “*ascending*” (“по возрастанию”), а `DESC` — “*descending*” (“по убыванию”). `NULLS FIRST/LAST` задает, как при сортировке располагаются значения `NULL` — вверху или внизу столбца (значение по умолчанию — `LAST`). Используя имя столбца или его порядковый номер, вы можете задать для каждого столбца необязательные ключевые слова, определяющие порядок сортировки значений. Несколько столбцов, по которым выполняется сортировка, разделяются запятыми. Если вы зададите больше одного столбца, сортировка будет выполняться слева направо. Oracle упорядочивает возвращаемые строки, основываясь на первом заданном столбце сортировки. При наличии одинаковых значений в этом столбце неопределенность разрешается за счет второго столбца в группе и т.д.

При сортировке по возрастанию или по убыванию символьные столбцы или значения вычисляются слева направо. Числовые значения сортируются от меньшего к большему (по возрастанию) или от большего к меньшему (по убыванию). Для дат восходящий порядок означает “от наиболее старых дат к наиболее свежим”, а нисходящий — наоборот. По умолчанию все значения сортируются по возрастанию, а порядки сортировки нескольких столбцов не связаны между собой — вы можете задать `ASC` для одного столбца и `DESC` для другого. Оператор `ORDER BY` является последним в выражении `SELECT`, и Oracle сортирует строки перед возвратом их клиентскому компьютеру.

*Предупреждение.* Если вы используете ключевое слово `DISTINCT` для определения столбцов в команде `SELECT`, то столбец (столбцы), к которому принадлежит ключевое слово, также должен перечисляться в операторе `ORDER BY`.

Сортировка по одному или нескольким столбцам существенно облегчает нахождение нужных строк. Например, если вы постоянно обращаетесь к списку клиентов, используя фамилии клиентов, то в операторе `SELECT`, генерирующем подобный список, необходимо задать упорядочение по `Lastname`, а затем — по `FirstName` (так называемая *сортировка телефонной книги*). С другой стороны, если вы желаете выделить всю недвижимость некоторого города, а затем выбрать нужный объект в пределах города, имеет смысл выполнять сортировку именно по этим столбцам. Ниже приведена команда `SELECT`, которая сортирует недвижимость по городам (по возрастающей), затем по числу комнат (по возрастающей) и, наконец, по году постройки (по убывающей).

```
SELECT YearBuilt, City, Bedrooms, SqFt
FROM Properties
WHERE YearBuilt > 1989
AND City IN ('Fortuna', 'McKinleyville', 'Loleta')
AND SqFt >= 2000
ORDER BY City, 3, YearBuilt DESC;
```

Oracle отбирает строки, основываясь на критерии, заданном в операторе WHERE, извлекает четыре столбца из выбранных строк, сортирует ответ согласно трем столбцам, указанным в ORDER BY, и возвращает 26 строк. Обратите внимание на значение 3 в операторе ORDER BY. Таким образом в аргументе column-list указывается третий столбец (Bedrooms), являющийся вторым ключом сортировки (сразу после City). Использование чисел особенно удобно, когда в списке SELECT вы используете очень длинное выражение, поскольку обращаясь к столбцу, указывая его позицию, гораздо проще, чем повторять длинное выражение в операторе ORDER BY.

## Значения NULL и оператор ORDER BY

Если ORDER BY задает столбец, содержащий значения NULL, при сортировке данные значения по умолчанию будут помещаться в конец списка — как при сортировке по возрастанию, так и при сортировке по убыванию. Если вам требуется, чтобы значения NULL в отсортированном столбце помещались в начало списка, после указания столбца задайте необязательное условие NULLS FIRST. Например, приведенный ниже запрос при сортировке помещает значения NULL вверх списка независимо от модификатора DESC или ASC.

```
SELECT BidPrice FROM CustAgentList  
ORDER BY BidPrice DESC NULLS FIRST;
```

## Запросы типа “*n* верхних”

Так называемые запросы типа “*n* верхних” представляют собой запросы, сортирующие строки и отбирающие первые *n* наибольших (или наименьших) значений. Например, вы можете отсортировать таблицу продаж в порядке убывания величины SalesAmount, а затем отобразить десять первых строк. Таким образом вы получите десять крупнейших продаж за выбранный период. Подобным образом, отсортировав столбец SalesAmount в порядке возрастания (по умолчанию), вы можете организовать запрос “*n* верхних” и отобрать первые пять значений, определив тем самым пять худших продаж. С помощью подобных запросов можно найти последнего нанятого риэлтора, самого молодого продавца или пять лучших продуктов. Поскольку в SQL нет специальной фразы, позволяющей отбирать “*n* верхних” или “*n* нижних” значений, необходимо использовать небольшой трюк.

Чтобы сгенерировать список из “*n* верхних” позиций отобранных результатов, необходимо прежде всего отсортировать список в возрастающем порядке для наименьших (число) или старейших (даты) значений или в убывающем порядке для наибольших (числа) или самых свежих (даты) значений. После этого необходимо воспользоваться псевдостолбцом Oracle ROWNUM, чтобы вычленить столько верхних строк возвращаемого набора, сколько вам требуется. Отсортированные строки ну-

меруются псевдостолбцом ROWNUM начиная с единицы. Чтобы отобрать необходимое число верхних строк, необходимо использовать операторы сравнения (отношения) < или <=. Использование операторов > или >= псевдостолбец ROWNUM не допускает. Предположим, например, что вы хотите перечислить пять верхних значений AskingPrice из таблицы Listings, которая содержит текущий перечень домов, выставленных на продажу. Ниже приведено выражение, дающее искомый результат — пять самых дорогих домов из продаваемых в текущий момент. Обратите внимание на то, что в действительности псевдостолбец ROWNUM можно использовать в операторе WHERE, не отображая его на экране.

```
SELECT AskingPrice
  FROM (SELECT AskingPrice
           FROM Listings
          ORDER BY AskingPrice DESC)
 WHERE ROWNUM <= 5;
```

Разберем, как работает эта команда. После выражения FROM идет подзапрос (запрос внутри запроса); “самый внутренний” запрос называется *внутренним представлением* (inline view). Он выполняется самым первым и возвращает все строки таблицы Listings, отсортированные по убыванию значения AskingPrice — все 502 записи. В результате получается временная таблица (внешнее представление), к которой обращается второй запрос. В числе прочих внутренний запрос содержит псевдостолбец ROWNUM с числами от 1 до 502 (упорядоченных по возрастанию). Поскольку внутреннее представление дает строки, отсортированные по убыванию цены, первые пять позиций содержат пять самых дорогих домов. Оператор WHERE внешнего запроса, основываясь на значении ROWNUM, извлекает только первые пять строк этой таблицы.

Картина немного усложняется при наличии в возвращаемом наборе значений NULL. Вы сами это увидите ниже при рассмотрении примера использования команды SELECT. Пока же мы поэкспериментируем с оператором ORDER BY.

Для того чтобы отсортировать строки, используя оператор ORDER BY, выполните следующие действия.

- Если необходимо, запустите SQL\*Plus и войдите в Oracle. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
- Определите *семь наибольших* значений BidPrice, помещенных потенциальными покупателями домов, набрав и выполнив следующее выражение:

```
CLEAR SCREEN
SELECT BidPrice
  FROM (SELECT BidPrice FROM CustAgentList
          ORDER BY BidPrice DESC)
 WHERE ROWNUM <= 7;
```

К сожалению, Oracle возвращает много всего и никакой информации! Поскольку значение BidPrice равно NULL для домов, которыми никто не интересуется, при сортировке с аргументом DESC верхнюю часть списка занимают значения NULL.

```

SQL> SELECT BidPrice
  2  FROM (SELECT BidPrice FROM CustAgentList
  3        ORDER BY BidPrice DESC)
  4 WHERE ROWNUM <= 7;
BIDPRICE
-----
688638
563891
485196
417423
402402
399745
382746

7 rows selected.

SQL> SELECT BidPrice
  2  FROM (SELECT BidPrice FROM CustAgentList
  3        ORDER BY BidPrice DESC NULLS LAST)
  4 WHERE ROWNUM <= 7;
BIDPRICE
-----
382746
399745
402402
417423
485196
563891
688638

7 rows selected.

SQL>

```

Верхняя часть списка содержит значения NULL

Опция NULLS LAST помещает значения NULL в конец списка

**Рис. 5.9.** Результаты двух запросов типа “п первых”

3. Теперь необходимо поместить значения NULL вниз списка, чтобы вы могли отобрать “реальные” из верхней части списка. Отредактируйте запрос и добавьте фразу NULLS LAST сразу после слова DESC перед закрывающей скобкой. Выполните модифицированный запрос.

```

SELECT BidPrice
  FROM (SELECT BidPrice FROM CustAgentList
        ORDER BY BidPrice DESC NULLS LAST)
 WHERE ROWNUM <= 7;

```

Oracle вернет семь верхних значений BidPrice (рис. 5.9).

4. Перечислите фамилии агентов по недвижимости, их имена и даты рождения. Отсортируйте строки от самого младшего агента к самому старшему, заканчиваая (включительно) июлем 1969 года. Введите и выполните следующие команды SQL\*Plus и SQL:

```

CLEAR SCREEN
COLUMN LastName FORMAT A15
COLUMN FirstName FORMAT A15
SELECT LastName, FirstName, BirthDate
  FROM Agents
 WHERE BirthDate >= '01-JUL-69'
 ORDER BY BirthDate DESC;

```

5. Сравните свои результаты с рис. 5.10.

The screenshot shows the Oracle SQL\*Plus interface with the following command history and output:

```

SQL> COLUMN LastName FORMAT A15
SQL> COLUMN FirstName FORMAT A15
SQL> SELECT LastName, FirstName, BirthDate
  2  FROM Agents
  3  WHERE BirthDate >= '01-JUL-69'
  4  ORDER BY BirthDate DESC;

LASTNAME      FIRSTNAME      BIRTHDATE
Schutz        Tim            21-AUG-83  ← Самый молодой сотрудник
Sheibani      Heather        03-MAR-83
Selby         Ricki          08-MAY-89
Piperova      Nancy          21-JUN-78
Carling       Tobias         19-OCT-75
Okindo        Essi           04-MAR-74
Marcoux       Kai             12-DEC-70
Voss          Bruce          03-APR-70
Kellogg       James          02-NOV-69
Dahlen        Elizabeth      03-OCT-69
Robinson      Clair          01-JUL-69  ← Самый старый из сотрудников,
                                             родившихся после июля 1969 года

11 rows selected.

SQL> CLEAR COLUMNS
columns cleared
SQL>

```

Рис. 5.10. Сортировка строк с информацией об агентах по убыванию даты рождения

6. Введите следующий код SQL\*Plus и SQL, чтобы вывести на экран строки, используя для организации результатов три столбца: City, Bedrooms и Bathrooms. Обратите внимание на то, что по двум столбцам сортировка идет в убывающем порядке, а по одному (Bathrooms) — в возрастающем (настройка по умолчанию).

```

CLEAR SCREEN
COLUMN Address FORMAT A20
COLUMN City FORMAT A14
COLUMN SqFT FORMAT 99,999
SET PAGESIZE 25
SELECT Address, City, Bedrooms, Bathrooms, SqFT
FROM Properties
WHERE SqFT > 3000 AND Bedrooms > 4
ORDER BY City DESC, Bedrooms DESC, Bathrooms;
SET PAGESIZE 14
CLEAR COLUMNS

```

7. Сравните свои результаты с рис. 5.11.
8. Вы можете не выходить из SQL\*Plus или ввести `exit` и нажать клавишу `<Enter>`, если хотите немного отдохнуть от этого интерфейса.

## Включение расчетов в запросы

Большинство приложений баз данных отображают как информацию непосредственно из таблиц (имена и адреса), так и информацию, выводимую из других столбцов той же таблицы или других таблиц. Фактически правила нормализации таблицы запрещают хранение в таблице информации, которую можно вывести из других данных таблицы. Например, было бы неразумно хранить промежуточные суммы различных позиций

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> COLUMN Address FORMAT A20
SQL> COLUMN City FORMAT A14
SQL> COLUMN Sqft FORMAT 99,999
SQL> SET PAGESIZE 25
SQL> SELECT Address, City, Bedrooms, Bathrooms, Sqft
  2  FROM Properties
  3 WHERE Sqft > 3000 AND Bedrooms > 4
  4 ORDER BY City DESC, Bedrooms DESC, Bathrooms;

ADDRESS          CITY      BEDROOMS  BATHROOMS   SQFT
2016 Saint Maru Ln McKinleyville      5          4    3,475
3125 Beau Pre Dr McKinleyville      5          4    5,688
2236 Home Dr       Eureka        6          3    3,338
6739 Myrtle Ave    Eureka        6          4    3,548
1991 Holly St      Eureka        6          4    7,000
1921 Heather Ln    Eureka        5          3    3,429
429 W Carson St   Eureka        5          3    4,180
2704 F St          Eureka        5          3    3,250
2467 Redwood St   Eureka        5          4    3,580
4656 Myrtle Ave   Eureka        5          4    5,200
1420 Leslie Rd    Eureka        5          5    4,640
3515 M St          Eureka        5          5    3,666
620 2nd Ave        Blue Lake      5          3    3,180
902 14th           Arcata        5          3    3,380
1127 Spear Ave    Arcata        5          3    4,330
8130 W End Rd     Arcata        5          4    5,150

16 rows selected.

SQL> SET PAGESIZE 14
SQL> CLEAR COLUMNS
columns cleared
SQL>

```

**Рис. 5.11.** Разрешение неопределенности с помощью сортировки по нескольким столбцам

счета — произведение числа заказанных вещей на цену вещи. Если в строке таблицы вы храните, скажем, число проданных элементов (12), цену за элемент (5,00 долл.) и общую сумму (60,00 долл.), то эта строка содержит зависимость, не представляемую первичным ключом, — т.е. имеем нарушение правил нормализации. Более того, изменение неверно введенной цены элемента не приведет к автоматическому обновлению произведения, так что ошибка может передаваться дальше.

К счастью, SQL допускает задавать в списке столбцов **SELECT** не только имена столбцов, но и расчеты. Используя расчеты, например, цену за один тип товара (количество, умноженное на цену одного элемента), вы гарантируете, что сумма всегда будет правильной независимо от изменений либо цены за единицу товара, либо числа приобретаемых вещей. Подобным образом вы можете задать простое выражение, рассчитывающее возраст сотрудника на основе даты рождения, записанной в таблице. (Разумеется, хранить в подобной таблице возраст нерационально — ведь он постоянно изменяется.) Вы можете создавать запросы SQL, помимо столбцов таблиц, содержащих расчеты, — от простых выражений до расчетов произвольной сложности, включающих арифметические операторы и встроенные функции.

## Использование математических операторов для численных расчетов

Oracle выполняет арифметические расчеты в выражениях SQL, содержащих операторы сложения, вычитания, умножения и деления, используя при этом стандартные математические операторы +, -, \* и /. Напомним, что операторы сложения и вычи-

тания в выражении имеют низкий приоритет, а операторы умножения и деления — высокий (следовательно, эти операции вычисляются до сложения и вычитания). Самыми первыми, как обычно, вычисляются выражения в скобках.

Вы можете сформировать выражения с математическими операторами в столбцах, тип данных которых равен NUMBER, DATE или INTERVAL. Впрочем, как будет показано в следующем разделе, некоторые математические операции не имеют смысла, если применяются к данным типа DATE или INTERVAL.

При желании вы можете написать арифметические выражения, включающие имеющиеся на столбцов, константы и функции SQL, в списке, следующем после команды SELECT. От последующих столбцов или выражений они отделяются запятой. Если вам требуется простой расчет, включающий только константы или текущее время (которое хранится на сервере), вы можете запросить упомянутую ранее таблицу DUAL. Напомним, что DUAL — это таблица, состоящая из одного столбца и одной строки, благодаря которой вы можете писать и тестировать выражения. Поскольку вы не можете просто написать выражение без оператора FROM <имя\_таблицы>, для этой цели применяется таблица DUAL. Предположим, например, что для вещи стоимостью 37,85 долл. вам требуется рассчитать налог с оборота (6 процентов) и общую сумму к получению. Для этого вы можете использовать следующий простой запрос SQL:

```
SELECT 0.06 * 37.85, 1.06 * 37.85
FROM DUAL;
```

(Разумеется, для выполнения подобных “расчетов” можно использовать даже мобильный телефон, но данный пример — это иллюстрация концепции.) Налог с оборота и общая сумма выглядят как два столбца результата, а таблица DUAL служит “псевдотаблицей”, к которой можно “привязать” два расчета.

Для выполнения расчетов, в которых задействованы столбцы с данными типа NUMERIC, необходимо задать задействованные столбцы и связать их подходящими математическими операторами. Например, вы можете написать команду SELECT, определяющую общий объем комиссионных агентства, который распределяется между покупающим риэлтором, продающим риэлтором и брокером (брокерами). Одним из столбцов таблицы Listings, требуемых для выполнения данного расчета, является AskingPrice — в нем записана цена, которую желает получить продавец за свою собственность. Вы хотите отобразить на экране комиссионные, которые бы получились при ставке комиссионных 6 процентов. Помимо этого, вы хотите знать чистую сумму продажи, которая в данном случае составлена из запрашиваемой цены минус комиссионные. Ниже приведен код, отображающий на экране столбец AskingPrice и два рассчитанных значения. Обратите внимание на то, что круглые скобки в данном примере не нужны (первым выполняется умножение). Мы использовали их для улучшения визуального восприятия.

```
SELECT AskingPrice, AskingPrice*0.06, AskingPrice-(AskingPrice*0.06)
FROM Listings;
```

Рассмотрим для примера несколько расчетов. Для подготовки к описанным ниже действиям запустите SQL\*Plus либо iSQL\*Plus и войдите в Oracle.

Для того чтобы включить в запрос SQL расчеты, выполните следующие действия.

- Чтобы перечислить запрашиваемую цену, размер комиссионных и чистую стоимость выбранной недвижимости, введите приведенное ниже выражение. Чтобы результат запроса был более управляемым, мы ограничиваем возвращаемую информацию строками, соответствующими дорогой недвижимости.

```
CLEAR SCREEN
SELECT AskingPrice, AskingPrice*0.05, AskingPrice*0.06,
       AskingPrice-(AskingPrice*0.05), AskingPrice-(AskingPrice*0.06)
  FROM Listings
 WHERE AskingPrice > 350000
 ORDER BY AskingPrice DESC;
```

Oracle возвращает восемь строк с рассчитанными значениями комиссионных и чистой стоимости (второй и третий столбец соответственно).

- Таблица Properties содержит столбец YearBuilt с числовым четырехзначным значением — годом постройки дома. Рассчитайте и выведите на экран “возраст” десяти старейших домов в таблице Properties (предполагается, что текущий год — 2006).

```
COLUMN Address FORMAT A20
COLUMN City FORMAT A7
SELECT Address, City, YearBuilt, 2006-YearBuilt
  FROM (SELECT Address, City, YearBuilt, 2006-YearBuilt
         FROM Properties ORDER BY 4 DESC)
 WHERE ROWNUM < 11;
```

- Можете не закрывать SQL\*Plus (или iSQL\*Plus). Если вы используете iSQL\*Plus, по прошествии определенного времени бездействия этот интерфейс может выгрузить вас принудительно. Если это произойдет, запустите iSQL\*Plus еще раз и войдите в Oracle.

В результате вы должны увидеть на экране то же, что показано на рис. 5.12. Благодаря форматированию столбцов (ограничена ширина столбцов Address и City) выполнение данного задания в SQL\*Plus дает немного более симпатичный результат.

## Расчет с использованием дат

Расчет с помощью столбцов типа date является типичным для большинства приложений баз данных. Главный бухгалтер может, например, для отчета изучать информацию за 30, 60 и 90 прошедших дней. Риэлтору могут потребоваться имена агентов, чья лицензия на работу с недвижимостью истекает в течение месяца. В обоих приведенных

```

SQL> SELECT AskingPrice, AskingPrice*0.06, AskingPrice-(AskingPrice*0.06)
  2  FROM Listings
  3 WHERE AskingPrice > 350000
  4 ORDER BY AskingPrice DESC;

ASKINGPRICE ASKINGPRICE*0.06 ASKINGPRICE-(ASKINGPRICE*0.06)
----- ----- -----
725000    43500    681500
560000    33600    526400
506000    30360    475640
500000    30000    470000
495000    29790    465300
469950    28197    441753
400000    24000    376000
380000    22800    357200

8 rows selected.

SQL> COLUMN Address FORMAT A28
SQL> COLUMN City FORMAT A7
SQL> SELECT Address, City, YearBuilt, 2006-YearBuilt
  2  FROM (SELECT Address, City, YearBuilt, 2006-YearBuilt
  3   FROM Properties ORDER BY 4 DESC)
  4 WHERE ROWNUM < 11;

ADDRESS      CITY      YEARBUILT 2006-YEARBUILT
-----      -----      -----      -----
309 Davenport Rd Arcata      1899      107
4087 Hartman Ln Eureka      1914      92
3327 18th St   Eureka      1918      88
1625 Bayview St Arcata      1919      87
3565 G Eun     Eureka      1929      77
3205 Freeze Ave Eureka      1931      75
513 I St      Arcata      1931      75
58 Lundblade  Orick       1933      73
1819 Campton Rd Eureka      1938      68
2042 Haeger Ave Arcata      1940      66

10 rows selected.

SQL>

```

Рассчитанные столбцы

Десять самых старых домов в базе данных

**Рис. 5.12.** Написание выражений в командах SELECT

примерах в расчетах явно фигурирует текущая дата. Напомним, что текущую дату и время возвращает псевдостолбец SYSDATE. Чтобы отобразить на экране текущую дату, необходимо ввести следующую команду:

```
SELECT SYSDATE FROM DUAL;
```

Если в запросе вам необходимо извлечь не только текущую дату, но и другие столбцы, имя таблицы DUAL можно заменить именем таблицы, к которой направлен запрос. Предположим, например, что вы хотите отобразить на экране столбец LicenseExpire и число дней до прекращения действия лицензий всех агентов, отсортировав значения по уменьшению времени. Для решения данной задачи применяется следующее выражение SELECT:

```
SELECT LicenseExpire, LicenseExpire-SYSDATE
  FROM Agents
 ORDER BY LicenseExpire-SYSDATE DESC;
```

Точно так же вы можете рассчитать число месяцев или лет до прекращения действия всех лицензий, разделив полученные результаты на 30 или 365 соответственно. Например, приведенная ниже команда отображает на экране приблизительное число лет до момента прекращения действия лицензии.

```
SELECT LicenseExpire-SYSDATE
  FROM Agents
 ORDER BY 1 DESC;
```

Значение “1” перед кодовым словом DESC указывает Oracle выполнять сортировку по первой позиции в списке выбора. Расчет по приведенной формуле *приблизительный*, поскольку в каждом году (в среднем) больше 365 дней. Более точным является значение 365.24, поскольку каждый четвертый год является високосным, а последний год столетия является високосным, если нацело делится на 400. Чтобы получить даты, относящиеся к прошлому (будущему), от текущей даты вы можете отнимать (прибавлять) необходимые значения. Разумеется, вы можете найти разность двух дат и найти число дней, прошедших между ними. Складывать, умножать и делить даты друг на друга не следует, поскольку подобные расчеты не имеют смысла. (Что, например, может означать текущая дата в днях, деленная на 2?) Для работы с датами Oracle предлагает несколько функций, которые мы рассмотрим позже в данной главе. Ниже приведено несколько примеров допустимых математических операций с датами (предполагается, что HireDate и BirthDate имеют тип данных date). Последний пример списка рассчитывает возраст в днях сотрудника, находя разницу между текущей датой и датой рождения.

```
SYSDATE + 27  
SYSDATE - HireDate  
HireDate - BirthDate  
InvoiceDate - 60  
SYSDATE - BirthDate
```

## Псевдонимы столбцов

Как видно на рис. 5.12 и из многих других выражений SELECT, приводившихся ранее, при выводе результатов на экран Oracle представляет заголовки прописными буквами. На рис. 5.12, например, в списке SELECT указано имя Address, но при отображении на экране база данных использовала вариант ADDRESS. При наличии выражений в списке SELECT Oracle удаляет все пробелы и использует все выражение как заголовок столбца. На рис. 5.12 обратите внимание на столбец с заголовком 2006-YEARBUILT, в котором отображается “возраст” домов. Вообще, вы не обязаны отображать генерированные Oracle имена столбцов и можете задать собственные, использовав для этого *псевдоимена столбцов* (column alias). Псевдоимена имеют несколько достоинств: псевдоимя столбца (или просто *псевдоним*) служит более значимым заголовком, чем сложные арифметические выражения; псевдоним упрощает обращение к столбцу при использовании оператора ORDER BY; псевдоним предлагает контроль над использованием в заголовках прописных и строчных букв.

Чтобы определить псевдоним, его необходимо указать после имени соответствующего столбца. Псевдоним состоит из последовательности букв, цифр и символов подчеркивания (без пробелов!). Максимальный размер псевдонима – 30 символов, перед псевдонимом указывается необязательное (в стандарте ANSI – *обязательное*)

ключевое слово. Если вы хотите сохранить регистр или использовать в псевдониме пробелы, то псевдоним необходимо взять в двойные кавычки. Пример определения псевдонима для столбца приведен ниже.

```
SELECT AskingPrice-0.06*AskingPrice AS NET_PRICE,
       AskingPrice*0.06 "Commission Rate"
  FROM Listings;
```

Обратите внимание на то, что во втором столбце мы пропустили ключевое слово AS. Хотя мы рекомендуем вам придерживаться стандарта ANSI и использовать это слово, вы можете этого и не делать. Мы считаем, что AS немного облегчает восприятие выражения SELECT, в котором определяется псевдоним.

## Конкатенация столбцов

Периодически два столбца требуется объединить в один. Объединение двух последовательностей символов, при котором второй набор добавляется в конец первого, называется *конкатенацией*. Иногда с помощью конкатенации можно улучшить восприятие результатов, отображаемых на экране. В сфере баз данных конкатенация реализуется как добавление строки символов к столбцу, объединение двух столбцов или двух строк символов. Оператор конкатенации записывается как две вертикальные черты (||) и означает, что столбец или строка, расположенные справа от оператора, будут добавлены в конец строки или столбца, расположенных слева. (На клавиатуре символ вертикальной черты расположен справа, над символом косой черты.) Например, таблица Agents содержит столбцы с именем и фамилией. Часто информацию из этих двух столбцов удобно объединять (на конвертах, в письмах электронной почты и т.п.). Ниже показано, как это сделать с помощью конкатенации, или “сложения строк”.

```
SELECT FirstName || LastName
  FROM Agents;
```

Если вы попробуете выполнить приведенное выражение, то столкнетесь с небольшой проблемой: имя и фамилия идут вместе без пробела (например, “AlecBaldwin”). Для решения подобных проблем при формировании строковых выражений вы можете включать в операцию конкатенации литеральные символы. Чтобы исправить обнаруженную ошибку, вставьте пустой символ между двумя столбцами, т.е. выполните конкатенацию трех строк. Далее мы используем оператор ORDER BY, сортирующий выход по фамилиям, а затем по именам (при совпадении фамилий). Кстати, конкатенация строк является еще одной областью, в которой оказываются весьма полезными псевдонимы.

```
SELECT FirstName||' '||LastName AS "Name"
  FROM Agents
 ORDER BY LastName, FirstName;
```

The screenshot shows the Oracle SQL\*Plus interface with three separate SELECT statements displayed in the command window. Each statement includes a calculated column 'Age when hired'.

```

Name          Age when hired
-----        -----
Edwin Townsend      52
Patricia Lewis     47
Sindisive Weber    46
Lee Reed           44
Ramanathan Rowe   44
Cornelis Dann      42
David Gagnon       40
Barbara Herring    39
Danial Silverburg 39
Belinda Chong      38
Clair Robinson    37

Name          Age when hired
-----        -----
Lora Allee        37
Cecilia Romero   37
Jessica Taylor    37
Elizabeth Dahlen 36
Crystal Fernandez 32
James Kellogg     32
Jackson Flamenbaum 32
Tim St-Onge        29
Christine Williams 28
Stanislaw Soltwedel 27
Bruce Voss         26

Name          Age when hired
-----        -----
Kai Marcoux       26
Tobias Carling    25
Nancy Piperova    25
Essi Okindo       22
Ricki Selby        22
Tim Schutz         21
Heather Sheibani   20

```

29 rows selected.

SQL>

**Рис. 5.13.** Использование псевдонимов и расчетов даты

**Совет.** Не путайте в выражениях SQL апостроф ('') и двойную кавычку (""). С помощью апострофов обособляются лiteralные строки, а с помощью двойных кавычек — псевдонимы столбцов. Данные символы не являются взаимозаменяемыми.

Ну что ж, пришло время поэкспериментировать с расчетами даты, псевдонимами и конкатенацией столбцов. Запустите SQL\*Plus или iSQL\*Plus и войдите в Oracle.

Для того чтобы поэкспериментировать с расчетами даты, псевдонимами и конкатенацией в выражениях SELECT, выполните следующие действия.

1. Введите, а затем выполните следующую команду, чтобы перечислить агентов по убыванию возраста:

```

CLEAR SCREEN
COLUMN "Name" FORMAT A20
COLUMN "Age when hired" FORMAT 99
SELECT FirstName||' '|LastName AS "Name",
       (HireDate-BirthDate)/365.24 AS "Age when hired"
  FROM Agents
 ORDER BY "Age when hired" DESC, LastName, FirstName;

```

Имена агентов выводятся на экран по убыванию возраста (рис. 5.13). Возраст округляется до ближайшего года (это следует из спецификации форматирования столбца — “99”).

2. Рассчитаем приблизительный возраст агентов на текущий момент. Ваши результаты будут соответствовать текущей дате.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT LastName||' is'||TO_CHAR((SYSDATE-BirthDate)/365.24)
  2  ||' years old today.' AS "How old are selected agents?"
  3  FROM Agents
  4  WHERE BirthDate > '31-DEC-1969'
  5  ORDER BY Birthdate;

How old are selected agents?
-----
Voss is 36.4160146030311880164082173669672300709 years old today.
Marcoux is 35.7233193889253945655216052762871293152 years old today.
Okindio is 32.4980428584249017388870636050572530695 years old today.
Carling is 38.8717149644373866802954526895474512953 years old today.
Piperova is 28.19949949434922547122744253398870866562 years old today.
Selby is 26.318544446218352620099154285158808737 years old today.
Sheibani is 23.581218852291948645907105221528614366 years old today.
Schutz is 23.0380335494773603962082770537485245622 years old today.

8 rows selected.

SQL>

```

Рис. 5.14. Запись информации в одну строку с помощью конкатенации

```

COLUMN FirstName FORMAT A15
COLUMN LastName FORMAT A15
SELECT FirstName, LastName, (SYSDATE-BirthDate)/365.24 AS Age
FROM Agents
ORDER BY LastName, FirstName;

```

Обратите внимание на то, что псевдоним Age записывается прописными буквами, поскольку мы не взяли его в кавычки (следовательно, не задали регистр).

3. В предыдущем выражении SELECT попробуйте изменить псевдоним Age на Age Today (без двойных кавычек). Что произошло? Oracle генерирует сообщение об ошибке.
4. Введите и выполните следующую команду:

```

SELECT LastName||' is'||TO_CHAR((SYSDATE-BirthDate)/365.24)
      ||' years old today.' AS "How old are selected agents?"
  FROM Agents
 WHERE BirthDate > '31-DEC-1969'
 ORDER BY Birthdate;

```

Oracle выдаст длинные значения — возраст агентов в годах (рис. 5.14). Так получилось из-за того, что функция TO\_CHAR преобразовала дату и время в строку символов, а текущее время при переводе в количество лет дало очень много знаков после запятой.

5. Выходите из SQL\*Plus или iSQL\*Plus.

## Введение в функции SQL

Oracle предлагает множество встроенных функций. Функция может принимать в качестве входного параметра нуль, одно или несколько значений, а в ответ выдает

выходной параметр. Функции Oracle можно разделить на две группы: односторонние и агрегирующие. Односторонняя функция воздействует на одну строку таблицы и возвращает одну строку для каждой входной строки. Агрегирующая функция одновременно действует на несколько строк таблицы и возвращает в качестве результата одну строку. В качестве примера односторонней функции приведем функцию UPPER, которая принимает строку символов и возвращает ее значение прописными буквами; на каждую строку, извлеченную запросом SELECT, функция возвращает одну модифицированную строку. Агрегирующие функции используются для расчета резюмирующей (обобщенной) информации. Например, с помощью функции AVG можно рассчитать средние продажи по региону, а с помощью функции SUM — общую сумму продаж риэлтора. Подробное рассмотрение данной темы мы начнем с односторонних функций.

## Использование в запросах односторонних функций

Односторонние функции возвращают одну строку результата для каждой строки, извлеченной оператором SELECT, причем их можно использовать не только в списках выбора. Помимо этого, односторонние функции могут входить в операторы WHERE и ORDER BY. Все односторонние функции Oracle можно разделить на следующие категории.

- *Символьные функции.* Позволяют манипулировать строками символов; после обработки возвращают также строку символов.
- *Числовые функции.* Позволяют выполнять расчеты; после обработки возвращают числовые результаты.
- *Функции преобразования.* Позволяют преобразовывать значение из одного типа данных в другой.
- *Функции даты.* Позволяют манипулировать датой и временем.
- *Функции регулярных выражений.* Позволяют выполнять поиск данных с использованием регулярных выражений.

## Использование символьных функций

Символьная функция принимает в качестве входного аргумента набор символов, обрабатывает эти символы и выдает результат. Примером функции из этой категории служит LOWER(), переводящая все символы строки в нижний регистр и выдающая в качестве результата получающуюся последовательность строчных букв. Другая символьная функция, INITCAP(), преобразует все первые буквы слов в прописные. Следует отметить, что использование символьных функций в команде SELECT никак не влияет на задействованные столбцы. Вместо этого из базы данных извлекается столбец (или выражение, основанное на столбце), по которому генерируется результат. Например, если вам требуется отобразить на экране имена агентов и гарантировать,

чтобы все имена и фамилии начинались с прописной буквы, то следует использовать команду

```
SELECT INITCAP(FirstName), INITCAP.LastName
FROM Agents;
```

В таком случае, независимо от того, как (в каком регистре) данные вводились в таблицу, имена и фамилии на экране будут отображены привычным образом. Наиболее распространенные односторонние символьные функции SQL представлены в табл. 5.2.

Чтобы использовать одностороннюю символьную функцию в списке SELECT, операторе WHERE или ORDER BY, запишите имя функции, открывающую скобку, аргумент (обычно это имя столбца) и закрывающую скобку. Ниже показано, как вы можете поэкспериментировать с односторонними символьными функциями в SQL\*Plus.

Для того чтобы использовать выбранные односторонние символьные функции, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle, очистите экран (CLEAR SCREEN) и введите следующий код, чтобы просмотреть некоторые функции, рассмотренные в главе:

```
SELECT UPPER('oracle') "Upper", LOWER('DATABASE') AS "Lower",
       INITCAP('for whom the bell tools') AS "Initcap",
       LENGTH('for whom the bell tools') AS "Length"
  FROM DUAL;
```

2. Введите и выполните нижеприведенную команду, чтобы перечислить клиентов, имеющих самые длинные адреса. Чтобы собрать информацию по адресам и их длинам и отсортировать их по убыванию длины, используйте внутреннее представление (inline view). После этого попросите Oracle извлечь только десять верхних строк.

```
SELECT Address, LENGTH(Address) AS "Addr. Size"
  FROM (SELECT Address, LENGTH(Address)
        FROM Customers
       ORDER BY 2 DESC)
 WHERE ROWNUM < 11;
```

3. Введите и выполните следующую команду, чтобы отобразить на экране имена клиентов (в правильном регистре), чьи фамилии заканчиваются на “ling”. Выведите на экран инициал, точку и фамилию.

```
SELECT SUBSTR(FirstName,1,1) || '.' || LastName AS "Name"
  FROM Customers
 WHERE UPPER(LastName) LIKE '%LING';
```

Результаты, возвращаемые Oracle в ответ на приведенные запросы, показаны на рис. 5.15.

4. Введите и выполните следующую команду, чтобы отобразить на экране имена агентов с заполнением точками до конца столбца и телефонными номерами, выровненными по правому краю. Чтобы сделать страницу длиннее, используйте

**ТАБЛИЦА 5.2.** Некоторые односторонние символьные функции

Функция	Описание
INSTR(<str1>, <str2>)	Ищет в строке <str1> положение, в котором расположена строка <str2>
LENGTH(<str>)	Возвращает число символов в <str>
LOWER(<str>)	Возвращает строку <str>, представленную строчными буквами
UPPER(<str>)	Возвращает строку <str>, представленную прописными буквами
INITCAP(<str>)	Превращает первые буквы всех слов строки <str> в прописные
NVL(<str>, <value>)	Возвращает <value>, если строка <str> имеет значение NULL; в противном случае возвращается значение <str>
NVL2(<str>, <value1>, <value2>)	Возвращает <value1>, если строка <str> не равна NULL; в противном случае возвращается значение <value2>
REPLACE(<str1>, <str2>, <str3>)	Ищет в строке <str1> строку <str2>. Если находит, заменяет строку <str2> строкой <str3>
LPAD(<str>, <width>, [<pad string>])	Заполняет строку <str> пробелами слева направо, доводя ее размер до <width> символов. При желании можно задать символ заполнения <pad string>
RPAD(<str>, <width>, [<pad string>])	Дополняет строку <str> пробелами справа, доводя ее размер до <width> символов. При желании можно задать символ заполнения <pad string>
SUBSTR(<str>, <start> [, <length>])	Возвращает фрагмент (подстроку) строки <str>, начинаящийся с позиции <start>. При желании можно задать длину возвращаемой строки <length>
TRIM(<str>)	Убирает пробелы в левой и правой части строки <str>
LTRIM(<str>)	Убирает пробелы в левой части строки <str>
RTRIM(<str>)	Убирает пробелы в правой части строки <str>

команду SQL\*Plus PAGESIZE. (Параметр PAGESIZE изменяется после оператора SELECT.)

```
SET PAGESIZE 40
SELECT
    RPAD(FirstName||' '||LastName,40,'.') AS "Agent",
    LPAD(WorkPhone,18) AS "Work Number"
FROM Agents
```

The screenshot shows the Oracle SQL\*Plus interface with several SQL statements demonstrating various string functions:

```

SQL> SELECT UPPER('oracle') "Upper", LOWER('DATABASE') AS "Lower",
2   INITCAP('for whom the bell tolls') AS "Initcap",
3   LENGTH('for whom the bell tolls') AS "Length"
4  FROM DUAL;

Upper Lower    Initcap          Length
ORACLE database For Whom The Bell Tools      23

SQL> SELECT Address, LENGTH(Address) AS "Addr. Size"
2  FROM (SELECT Address, LENGTH(Address)
3    FROM Customers
4   ORDER BY 2 DESC)
5  WHERE ROWNUM < 11;

ADDRESS                  Addr. Size
Washington Dulles International            31
5704 West Las Positas Boulevard          31
3801 John F. Kennedy Boulevard           38
2525 East El Segundo Boulevard          38
460 Point San Bruno Boulevard           29
1458 Fashion Island Boulevard           29
1100 North Washington Street            28
2600 West Magnolia Boulevard             28
691 South Milpitas Boulevard            28
11812 San Vicente Boulevard             27

10 rows selected.

SQL> SELECT SUBSTR(FirstName,1,1) || ' ' || LastName AS "Name"
2  FROM Customers
3  WHERE UPPER(LastName) LIKE '%LING';

Name
A. Gibeling
M. Pilling
R. Colling
M. Hotaling

SQL>

```

**Рис. 5.15.** Использование нескольких односторонних символьных функций

```

ORDER BY LastName, FirstName;
SET PAGESIZE 14

```

Использование односторонних символьных функций UPPER, LPAD (заполнение слева) и RPAD (заполнение справа) показано на рис. 5.16.

## Использование числовых функций

Oracle предлагает несколько односторонних числовых функций, возможности которых существенно превышают те, которыми располагают обычные математические операторы. Односторонние числовые функции Oracle позволяют округлять значения до различного числа десятичных знаков, вычислять остаток, усекать число до ближайшего целого числа и возводить число в степень. Некоторые наиболее распространенные числовые функции перечислены в табл. 5.3. Полный список встроенных функций (включая односторонние числовые) представлен в документации по Oracle.

Односторонние числовые функции применяются для выполнения расчетов. Данный класс функций принимает в качестве аргумента число, введенное в таблицу, или выражение, которое дает число. Далее функция предопределенным образом модифицирует входное выражение и дает числовой результат. Отметим, что не все числовые функции выполняют расчеты с участием входных аргументов. Иногда они

The screenshot shows the Oracle SQL\*Plus interface. The command line displays:

```
SQL> SET PAGESIZE 40
SQL> SELECT
  2    RPAD(UPPER(LastName)||', '||FirstName,50,'.') AS "Agent",
  3    LPAD(WorkPhone,17) AS "Work Number"
  4   FROM Agents
  5  ORDER BY LastName, FirstName;
```

The output shows a list of agents with their names and work numbers. The names are right-padded with commas and first names, and the work numbers are left-padded with zeros.

Agent	Work Number
ALLEE, Lora.....	(707) 555-9990
CARLING, Tobias.....	(707) 555-1983
CHONG, Belinda.....	(707) 555-0962
DAHLEN, Elizabeth.....	(707) 555-7218
DANN, Cornelis.....	(707) 555-9652
FERNANDEZ, Crystal.....	(707) 555-1652
FLAMENBAUM, Jackson.....	(707) 555-5551
GAGNON, David.....	(707) 555-8668
HERRING, Barbara.....	(707) 555-3355
KELLOGG, James.....	(707) 555-2075
LEWIS, Patricia.....	(707) 555-8690
MARCOUX, Kai.....	(707) 555-8361
OKINODO, Essi.....	(707) 555-0719
PIPERDUE, Mancy.....	(707) 555-0492
REED, Lee.....	(707) 555-7213
ROBINSON, Clair.....	(707) 555-1950
RUMERO, Cecilia.....	(707) 555-2430
ROWE, Ramanathan.....	(707) 555-9839
SCHUTZ, Tim.....	(707) 555-6005
SELBY, Ricki.....	(707) 555-3933
SHEIBANI, Heather.....	(707) 555-5635
SILVERBURG, Daniel.....	(707) 555-5865
SOLTWEDEL, Stanislav.....	(707) 555-0089
ST-ONGE, Tim.....	(707) 555-4697
TAYLOR, Jessica.....	(707) 555-4529
TOWNSEND, Edwin.....	(707) 555-9485
VOSS, Bruce.....	(707) 555-8389
WEBER, Sindisive.....	(707) 555-0616
WILLIAMS, Christine.....	(707) 555-5525

29 rows selected.

```
SQL> SET PAGESIZE 14
SQL>
```

Рис. 5.16. Использование символьных функций LPAD и RPAD

возвращают некоторые характеристики входного числа. Например, функция SIGN() возвращает значение  $-1$ , если входное число отрицательное; значение  $1$  — если оно положительное, и  $0$  — если оно равно нулю. Далее запустите SQL\*Plus и войдите в Oracle. После этого введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран, подготовив его к следующим этапам.

Для того чтобы использовать выбранные однострочные числовые функции, выполните следующие действия.

1. Ведите и выполните следующую команду:

```
SELECT ABS(-234), CEIL(23.112), MOD(21,6),
       ROUND(567.93537,2), SQRT(5280)
  FROM DUAL;
```

2. Ведите и выполните нижеприведенную команду, чтобы увидеть разницу между функциями TRUNC и ROUND. Результаты выполнения этой команды показаны на рис. 5.17. Разумеется, в вашем случае значения, возвращаемые Oracle, зависят от текущей даты.

```
COLUMN LastName FORMAT A15
SELECT LastName,
       TRUNC((SYSDATE-BirthDate)/365.25) AS "Age",
       ROUND((SYSDATE-BirthDate)/365.25,2) AS "Rounded-2",
```

**ТАБЛИЦА 5.3.** Однострочные числовые функции

<i>Функция</i>	<i>Описание</i>	<i>Примеры</i>
ABS(<n>)	Возвращает абсолютное значение <n>	ABS(5) = 5 ABS(-58) = 58
CEIL(<n>)	Возвращает наименьшее целое число, превышающее или равное <n>	CEIL(6.8) = 7 CEIL(-34.5) = -34
FLOOR(<n>)	Возвращает наибольшее целое число, меньшее или равное <n>	FLOOR(78.9) = 78 FLOOR(-3.1) = -4
MOD(<n>, <m>)	Возвращает остаток от деления <n> на <m>	MOD(7, 3) = 1 MOD(29, 6) = 5
POWER(<n>, <k>)	Возвращает <n> в степени <k>	POWER(2, 3) = 8 POWER(5, 2) = 25
ROUND(<n> [ , <m> ] )	Возвращает значение <n>, округленное до <m> десятичных знаков. Если число <m> пропущено, <n> округляется до ближайшего целого числа	ROUND(7.467, 2) = 7.47 ROUND(96.87, -1) = 100
SIGN(<n>)	Возвращает -1, если <n> отрицательное, 0 — если равно нулю, и 1 — если <n> положительное	SIGN(-23) = -1 SIGN(456) = 1
SQRT(<n>)	Возвращает корень квадратный из <n>	SQRT(81) = 9 SQRT(7) = 2.645751
TRUNC(<n> [ , <k> ] )	Возвращает значение <n>, усеченное до <k> десятичных знаков. Второй аргумент необязательный	TRUNC(56.999) = 56 TRUNC(789.559, 1) = 789.5

```

ROUND((SYSDATE-BirthDate)/365.25, 3) AS "Rounded-3",
(SYSDATE-BirthDate)/365.25 AS "Full"
FROM Agents WHERE BirthDate < '01-JAN-61'
ORDER BY 5;

```

3. SQL\*Plus можете не закрывать.

**Использование функций даты**

Oracle предоставляет несколько встроенных однострочных функций даты, которые действуют на значения типа date. Несколько функций, используемых для работы с датами, показаны в табл. 5.4. Эта таблица также содержит две функции преобразования, одна из которых (TO\_DATE) переводит входной аргумент (строку) в тип данных date, а вторая (TO\_CHAR) выполняет обратное преобразование.

The screenshot shows the Oracle SQL\*Plus interface. The command line displays several SQL queries demonstrating various numeric functions:

```

SQL> SELECT ABS(-234), CEIL(23.112), MOD(21,6),
2      ROUND(567.99537,2), SQRT(5280)
3   FROM DUAL;

ABS(-234) CEIL(23.112) MOD(21,6) ROUND(567.99537,2) SQRT(5280)
----- ----- ----- -----
234          24          3           567.94    72.6636085

SQL> COLUMN LastName FORMAT A15
SQL> SELECT LastName,
2      TRUNC((SYSDATE-BirthDate)/365.25) AS "Age",
3      ROUND((SYSDATE-BirthDate)/365.25,2) AS "Rounded-2",
4      ROUND((SYSDATE-BirthDate)/365.25,3) AS "Rounded-3",
5      (SYSDATE-BirthDate)/365.25          AS "Full"
6   FROM Agents
7  WHERE BirthDate < '01-JAN-61'
8  ORDER BY 5;

LASTNAME          Age Rounded-2 Rounded-3      Full
----- ----- ----- -----
Weber            45     45.7   45.696 45.6963852
Dann             47     47.13   47.131 47.131019
Alleé            47     47.31   47.306 47.3062415
Lewis            47     47.4   47.397 47.3965986
Silverburg       47     47.64   47.64 47.6402593
Gagnon           47     47.82   47.821 47.8209574
Chong            48     48.87   48.867 48.8668164
Reed              50     50.44   50.438 50.4383428
Townsend          51     51.58   51.582 51.5827644
Rowe              52     52.86   52.859 52.8586029

10 rows selected.

SQL>

```

**Рис. 5.17.** Эксперименты с выбранными числовыми функциями

Лучший способ ознакомиться с функциями даты и связанными с ними функциями преобразования — это поупражняться в их использовании.

Для того чтобы выполнить запросы, содержащие даты и функции преобразования дат, выполните следующие действия.

1. Используя SQL\*Plus, введите и выполните команду CLEAR SCREEN.
2. Ведите и выполните следующую команду:

```

SELECT
  EXTRACT(YEAR FROM HireDate) AS "Hire Year",
  EXTRACT(MONTH FROM HireDate) AS "Hire Month",
  EXTRACT(DAY FROM HireDate) AS "Hire Day",
  HireDate, ADD_MONTHS(HireDate,5) AS "+5 Months"
FROM Agents
WHERE HireDate > '14-JUN-02'
ORDER BY 1;

```

3. Используя функцию преобразования, отобразите на экране текущее время, набрав и выполнив следующие выражения SQL\*Plus и SQL (разумеется, полученный вами результат будет зависеть от времени выполнения запроса):

```

COLUMN "Quarter" FORMAT A7
COLUMN "1 hour later" FORMAT A12
COLUMN "1 Day later" FORMAT A11
SELECT
  TO_CHAR(SYSDATE, 'MM/DD/YYYY HH:MM:SS AM') AS "Year и time",
  TO_CHAR(SYSDATE, 'YYYY') AS "Year",
  TO_CHAR(SYSDATE, 'Q') AS "Quarter",
  TO_CHAR(SYSDATE+1/24, 'HH24:MM:SS') AS "1 hour later",

```

**ТАБЛИЦА 5.4.** Функции даты и функции, связанные с преобразованием в тип данных date

Функция	Описание	Пример
ADD_MONTHS(<date>, <no. of months>)	Добавляет заданное число месяцев <no. of months> к дате <date>. (Вычитает, если значение <no. of months> меньше нуля.)	ADD_MONTHS('10-ОCT-06', 3) возвращает значение 10-JAN-07
EXTRACT(<ymd> FROM <date>)	Вычитает целочисленное значение, являющее годом, месяцем или днем даты	EXTRACT(YEAR FROM '10-ОCT-85') возвращает значение 1985
LAST_DAY(<date>)	Возвращает последний день месяца, к которому относится указанная дата	LAST_DAY(SYSDATE) возвращает последний день текущего месяца
MONTHS_BETWEEN(<date1>, <date2>)	Возвращает число месяцев между двумя датами. Если дата <date1> относится к более позднему моменту времени, результат будет положительный	MONTHS_BETWEEN(SYSDATE, HireDate) возвращает число месяцев, прошедших между текущей датой и полем HireDate
NEW_TIME(<date>, <cur-time-zone>, <new-time-zone>)	Возвращает дату и время в другом часовом поясе	NEW_TIME(SYSDATE, 'PST', 'EST')
NEXT_DAY(<date>, <string>)	Возвращает дату первого дня, относящегося к указанному дню недели (<string>), после заданного момента времени <date>	NEXT_DAY('14-JUN-2006', 'Tuesday') возвращает дату '20-JUN-06'
SYSDATE	Возвращает текущую дату (функция не имеет аргумента)	'07-ОCT-06' (если это текущая дата)
TO_CHAR(<date>, <format>)	Преобразование даты/времени в строку, формат которой задается выражением <format>	TO_CHAR(SYSDATE, 'MM/DD/YYYY HH24:MM:SS')
TO_DATE(<str>, <format>)	Преобразует строку <str> в дату, используя <format> для интерпретации строки	TO_DATE('10/30/2006', 'MM/DD/YYYY')

```
TO_CHAR(SYSDATE+1, 'MM/DD/YYYY') AS "1 Day later"
FROM DUAL;
```

**Рис. 5.18.** Использование функций даты и функций преобразования

Типичные результаты выполнения данного запроса показаны на рис. 5.18 (ваши результаты будут другими из-за иной даты/времени выполнения запроса).

4. Определите, сколько лет каждый сотрудник работает в Redwood Realty. Прежде всего введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы максимизировать доступное пространство на экране. После этого введите и выполните следующую команду:

```
SET PAGESIZE 40
COLUMN "Unrounded Hire Age" FORMAT A20
SELECT TO_CHAR(BirthDate,'MM/DD/YYYY') AS "Birth date",
       TO_CHAR(HireDate,'MM/DD/YYYY') AS "Hire date",
       TRUNC(MONTHS_BETWEEN(HireDate, BirthDate)/12,0) AS "Hire Age",
       TO_CHAR(MONTHS_BETWEEN(HireDate, BirthDate)/12, '99.99')
          AS "Unrounded Hire Age"
FROM Agents ORDER BY 3 DESC;
```

Полученные результаты, отсортированные по убыванию возраста сотрудников на момент их найма в Redwood Realty, показаны на рис. 5.19.

5. Введите и выполните нижеприведенную команду, чтобы рассчитать время, в течение которого были активизированы все предложения по продаже. Выразите это время (конечная дата минус начальная дата) в днях, полных неделях и полных месяцах

```

SQL> SET PAGESIZE 40
SQL> COLUMN "Unrounded Hire Age" FORMAT A20
SQL> SELECT TO_CHAR(BirthDate,'MM/DD/YYYY') AS "Birth date",
2      TO_CHAR(HireDate,'MM/DD/YYYY') AS "Hire date",
3      TRUNC(MONTHS_BETWEEN(HireDate, BirthDate)/12,0) AS "Hire Age",
4      TO_CHAR(MONTHS_BETWEEN(HireDate, BirthDate)/12, '99.99') AS "Unrounded Hire Age"
5  FROM Agents ORDER BY 3 DESC;

```

Birth date	Hire date	Hire Age	Unrounded Hire Age
02/01/1955	09/12/2006	51	51.61
08/10/1959	05/30/2006	47	47.14
12/21/1968	08/25/2006	45	45.68
03/25/1956	09/18/2000	44	44.48
10/23/1953	09/05/1997	43	43.87
07/16/1954	09/02/2001	42	42.13
11/06/1956	01/17/1999	38	38.69
01/11/1959	09/21/1997	38	38.69
02/04/1965	12/07/2003	38	38.84
10/21/1963	11/01/1999	37	37.93
10/20/1957	08/30/1995	37	37.86
05/13/1959	07/14/1996	37	37.17
07/01/1969	10/16/2006	37	37.29
09/26/1961	06/29/1998	36	36.76
10/03/1963	05/23/2005	35	35.64
11/02/1964	02/05/2002	32	32.26
11/29/1964	05/30/1997	32	32.50
05/03/1968	02/08/2000	31	31.76
05/19/1968	07/13/1997	29	29.15
08/16/1967	01/08/1996	28	28.40
04/04/1969	05/04/1996	27	27.98
04/03/1973	06/05/1996	26	26.17
12/12/1973	10/03/1996	25	25.81
10/19/1975	12/19/2000	25	25.17
06/21/1978	12/24/2002	24	24.51
03/04/1974	12/20/1995	21	21.79
05/08/1980	01/05/2002	21	21.66
03/03/1983	07/18/2003	20	20.37
08/21/1983	06/21/2004	20	20.83

29 rows selected.

SQL>

Рис. 5.19. Расчет возраста с использованием функций TRUNC и MONTHS\_BETWEEN

```

CLEAR SCREEN
SELECT DISTINCT
    (EndListDate-BeginListDate) AS "Market Time: Days",
    TRUNC((EndListDate-BeginListDate)/7) AS "Weeks",
    FLOOR(MONTHS_BETWEEN(EndListDate,BeginListDate)) AS "Months"
FROM LISTINGS
ORDER BY 1 DESC;

```

В ответ Oracle вычислит и отобразит на экране несколько значений, представляющих время нахождения недвижимости на рынке продажи.

6. Введите следующий код, чтобы отобразить на экране первый день месяца LicenseExpire, дату LicenseExpire и последний день месяца, в который заканчивается срок действия лицензии. Сделайте это только для тех агентов, которые являются брокерами и срок действия лицензии которых заканчивается до 2002 года.

```

SELECT ADD_MONTHS(LAST_DAY(LicenseExpire),-1)+1 AS "Beginning",
LicenseExpire AS "Actual",
LAST_DAY(LicenseExpire) AS "Ending"
FROM Agents
WHERE UPPER(TITLE) = 'BROKER'
AND EXTRACT(YEAR FROM LicenseExpire) < 2002;

```

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT DISTINCT
  2   (EndListDate-BeginListDate) AS "Market Time: Days",
  3   TRUNC((EndListDate-BeginListDate)/7) AS "Weeks",
  4   FLOOR(MONTHS_BETWEEN(EndListDate,BeginListDate)) AS "Months"
  5 FROM LISTINGS
  6 ORDER BY 1 DESC;

Market Time: Days      Weeks      Months
-----  -----  -----
 184        26       6
 183        26       6
 182        26       6
 181        25       6
 153        21       5
 151        21       5
 123        17       4
 122        17       4
 121        17       4
 120        17       4
  92         13       3
  91         13       3
  90         12       3
  62          8       2
  61          8       2
  31          4       1
  30          4       1

17 rows selected.

SQL> SELECT ADD_MONTHS(LAST_DAY(LicenseExpire),-1)+1 AS "Beginning",
  2   LicenseExpire AS "Actual",
  3   LAST_DAY(LicenseExpire) AS "Ending"
  4  FROM Agents
  5 WHERE UPPER>Title) = 'BROKER' AND EXTRACT(YEAR FROM LicenseExpire) < 2002;

Beginning Actual      Ending
-----  -----  -----
01-MAY-99 01-MAY-99 31-MAY-99
01-DEC-98 01-DEC-98 31-DEC-98

```

**Рис. 5.20.** Расчет значений прошедшего времени и дат выдачи/окончания срока действия лицензий

Результаты, полученные при выполнении пп. 5 и 6, показаны на рис. 5.20.

**Совет.** Возможно, разрешение вашего монитора не совпадает с тем, который использовался при получении рисунков, однако результаты у вас должны получиться точно такие же. Если необходимо, прокрутите окно SQL\*Plus вверх-вниз, чтобы просмотреть все результаты.

7. Введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из SQL\*Plus.

Просмотрите полученные результаты и представленные рисунки. Обратите внимание на то, что при вычитании одной даты из другой полученное количество дней является целочисленной величиной. Более того, степень детализации представляемых результатов (даты и времени) зависит от форматирования, заданного в функции `TO_CHAR`. При желании вы можете отобразить на экране только дату или время или же и дату, и время.

## Использование функций преобразования

Oracle предлагает больше тридцати односторонних функций преобразования, позволяющих переводить значение из одного типа данных в другой. Одной из самых популярных функций преобразований является `TO_CHAR`, которая превращает дату или время в строку символов. Не менее популярными функциями являются `TO_DATE` и `TO_NUMBER`. Кроме того, существуют функции преобразования из двоичного формата в формат с плавающей запятой, из многобайтовых в однобайтовые символы и из числовых или символьных данных в формат CLOB (Character Large Object). Ниже мы рассмотрим три популярные функции преобразования, `TO_CHAR`, `TO_DATE` и `TO_NUMBER`, и попрактикуемся в их использовании. Если вам интересны подробности использования других функций преобразования, обратитесь к документации Oracle или любому из множества доступных сетевых источников.

### Функция `TO_CHAR`

Однострочная функция `TO_CHAR` применяется для преобразования даты или времени в строку символов и имеет следующий синтаксис:

```
TO_CHAR(<date-number> [, <format-string>])
```

Параметр `<date-number>` — это дата/время или числовое значение, столбец или выражение, за которым после необязательного параметра следует `<format-string>`. Параметр `<format-string>`, формально называемый *моделью формата*, задает точный формат выходной строки символов. Например, чтобы преобразовать столбец `HireDate` таблицы `Agents` в выходную строку, имеющую другой формат, необходимо следующим образом использовать функцию `TO_CHAR` в списке `SELECT`:

```
SELECT FirstName, LastName, (HireDate, 'Month, DD, YYYY')
  FROM Agents;
```

В результате вы получите имя и фамилию, за которыми будет следовать дата найма сотрудника, например “October 03, 1996” или “August 30, 1995”. Кроме того, вы можете пропустить строку формата, и Oracle будет использовать формат, принятый по умолчанию, — `DD-MON-YY` (например, 03-ОКТ-96).

Функция `TO_CHAR` часто используется для форматирования числовых результатов в форму, более удобную для восприятия. Например, с помощью приведенной ниже функции `TO_CHAR` вы можете преобразовать и отобразить на экране значения `BidPrice`, расположив перед ними символ доллара, используя для отделения миллионов и тысяч запятую и представляя значения с точностью до двух знаков после десятичной точки.

```
SELECT TO_CHAR(BidPrice, '$99,999,999.99')
  FROM CustAgentList;
```

Указанная модель формата, `'$99,999,999.99'`, означает, что перед значением ставится знак \$, а каждый символ 9 означает один разряд. Точка указывает положение десятичного разделителя, а следующие за ней две девятки обозначают, что в результате необходимо указывать два знака после запятой (даже если это 00). Су-

**ТАБЛИЦА 5.5.** Элементы, применяющиеся для задания формата чисел

Элемент	Пример	Описание
9	9999	Возвращает значение с заданным количеством цифр. Если число положительное — оно начинается с пробела, если отрицательное — со знака “минус”. Число девяток соответствует максимальному количеству цифр, отображаемых на экране
, (запятая)	9,999	Возвращает запятую в заданной позиции. Запятая не может быть первым элементом или появляться после десятичного разделителя
. (точка)	99.99	Возвращает точку и помечает начало дробной части числа
0	0999	Число начинается с нулей
	9990	Число завершается нулями
\$	\$999,999	Число начинается со знака доллара
B	B999	Если целая часть числа равна нулю, вместо нее подставляются пробелы
M <small>I</small>	9999M <small>I</small>	Если значение отрицательное, знак “минус” ставится в конце. Если число положительное — в конце ставится пробел
R <small>N</small>	R <small>N</small>	Возвращает число прописными римскими цифрами
	r <small>n</small>	Возвращает число строчными римскими цифрами
S	S9999	Для отрицательных значений <i>в начале</i> числа ставится знак “минус”, для положительных — знак “плюс”
	или 9999S	Для отрицательных значений <i>в конце</i> числа ставится знак “минус”, для положительных — знак “плюс”
E <small>EEE</small>	9.99E <small>EEE</small>	Возвращает значение в научной форме записи (например, 1.78E+03)

ществует целый набор символов модели формата, которые можно использовать для форматирования дат или чисел. Некоторые из важнейших символов перечислены в табл. 5.5. Их можно использовать в различных комбинациях при задании формата, добиваясь желаемого отформатированного выхода.

Чтобы лучше понять, для чего нужны различные элементы модели формата, вам стоит поэкспериментировать с несколькими командами `SELECT`.

Чтобы использовать функцию `TO_CHAR` для форматирования столбцов, содержащих даты и числовые значения, необходимо выполнить следующие действия.

1. Запустите `SQL*Plus`, войдите в `Oracle`, введите `CLEAR SCREEN` и нажмите клавишу `<Enter>`.
2. Введите и выполните следующую команду, чтобы отобразить на экране отформатированное значение `BidPrice` для ряда самых дорогих домов:

```
SET PAGESIZE 20
SELECT TO_CHAR(BidPrice, '$99,999,999.00') AS "Bid Price"
FROM CustAgentList WHERE BidPrice > 360000;
```

3. Введите и выполните следующую команду, чтобы просмотреть, как представляются числа со знаком и римские числа. Набирая оператор WHERE, будьте внимательны, иначе Oracle вернет слишком много (или слишком мало) строк. В частности, обратите внимание на то, что в строке в кавычках, следующей после слова LIKE, пробелов нет.

```
SELECT TO_CHAR(Longitude, 'S9999.99') AS "S9999.99",
       TO_CHAR(Longitude, '000999.99MI') AS "000999.99MI",
       TO_CHAR(Latitude, 'S9999.9') AS "S9999.9",
       TO_CHAR(Latitude, 'RN') AS "Roman Latitude"
  FROM Properties
 WHERE UPPER(Address) LIKE '%RIVER%';
```

Результаты выполнения пп. 2 и 3 должны выглядеть так, как показано на рис. 5.21.

4. Введите и выполните следующую команду, задающую модели формата, и посмотрите, что дают другие элементы модели:

```
CLEAR SCREEN
SELECT TO_CHAR(345.678,'S99,999.99') AS "S99,999.99",
       TO_CHAR(0.00678,'9.999EEEE') AS "9.999EEEE",
       TO_CHAR(5280, '09.99') AS "09.99",
       TO_CHAR(1256, 'RN') AS "RN: 1,256",
       TO_CHAR(1234.5678, '$999,999.00') AS "$999,999.00"
  FROM DUAL;
```

5. Обратите внимание на то, что третий столбец содержит ряд знаков #. Это означает, что строка формата слишком короткая, чтобы вместить значение. Чтобы решить эту проблему, исправьте третью строку следующим образом:

```
TO_CHAR(5280, '09999.99') AS "09999.99",
```

6. После этого выполните модифицированный запрос. В результате вы должны получить такое же изображение, как на рис. 5.22.

7. Введите exit, нажмите клавишу <Enter>, чтобы выйти из Oracle, и закройте SQL\*Plus.

### **Функция TO\_DATE**

Функция TO\_DATE преобразует строку в дату и время. Данная функция весьма полезна, когда требуется вставить дату в столбец. Вы задаете дату как строку, указывая необязательную строку формата. Затем функция преобразует строку во внутренний формат даты/времени согласно предоставленной строке формата. Если формат не указывается, строка даты должна иметь один из принятых по умолчанию форматов – DD-MON-YY или DD-MON-YYYY (в одинарных кавычках). Синтаксис данной функции очень похож на синтаксис функции TO\_CHAR.

The screenshot shows the Oracle SQL\*Plus interface with the following session history:

```

SQL> SET PAGESIZE 20
SQL> SELECT TO_CHAR(BidPrice, '$99,999,999.00') AS "Bid Price"
  2  FROM CustAgentList
  3  WHERE BidPrice > 360000;
Bid Price
-----
$369,836.00
$485,196.00
$399,745.00
$482,402.00
$563,091.00
$688,638.00
$377,155.00
$417,423.00
$361,195.00
$382,746.00

10 rows selected.

SQL> SELECT TO_CHAR(Longitude, 'S9999.99') AS "S9999.99",
  2  TO_CHAR(Longitude, '000999.99MI') AS "000999.99MI",
  3  TO_CHAR(Latitude, 'S9999.9') AS "S9999.9",
  4  TO_CHAR(Latitude, 'RN') AS "Roman Latitude"
  5  FROM Properties
  6  WHERE UPPER(Address) LIKE '%RIER%';

S9999.99 000999.99M S9999.9 Roman Latitude
-----+-----+-----+
-124.10 000124.-10- +40.9      XLI
-124.11 000124.-11- +40.9      XLI
-124.12 000124.-17- +40.7      XLI
-124.20 000124.-20- +40.7      XLI
-124.10 000124.-10- +40.9      XLI
-124.17 000124.-17- +40.7      XLI
-124.11 000124.-17- +40.7      XLI
-124.09 000124.-09- +41.0      XLI
-124.09 000124.-09- +41.0      XLI
-124.17 000124.-17- +40.6      XLI
-124.17 000124.-17- +40.7      XLI
-124.10 000124.-10- +41.0      XLI
-124.14 000124.-14- +40.7      XLI

12 rows selected.

SQL>

```

**Рис. 5.21.** Использование символов задания числовых форматов

`TO_DATE(<date-string> [,<format-string>])`

В данном выражении `<date-string>` — это строка, которую необходимо преобразовать в дату, а `<format-string>` — необязательная модель формата, указывающая Oracle, как интерпретировать строку даты. Ниже приведено несколько примеров функции `TO_DATE`.

```

TO_DATE('January 10, 2006', 'Month DD, YYYY')
TO_DATE('10/17/2004 10:27:48', 'MM/DD/YYYY HH24:MI:SS')
TO_DATE('7.14.03', 'MM.DD.YY')
TO_DATE('2006/9/23','YYYY/MM/DD')
TO_DATE('10-OCT-2006')

```

В первой строке модель формата указывает роль каждого элемента строки даты. Из нее следует, что дата начинается с полного названия месяца, затем идет двухзначное представление дня месяца (DD), запятая и четырехзначное представление года (YYYY). Другие строки формата точно так же соответствуют *форме* строк даты, находящихся перед ними. Исключением является только пятый пример. В последней строке даты он имеет форму, принятую по умолчанию, поэтому Oracle интерпретирует ее правильно без помощи строки формата. Несколько важных элементов модели формата даты и их значения перечислены в табл. 5.6. В табл. 5.7 приведено несколько важных элементов модели формата времени, а также их значения и примеры использования.

```

SQL> SELECT TO_CHAR(345.678, '$99,999.99') AS "S99,999.99",
2   TO_CHAR(0.00678, '9.999EEEE') AS "9.999EEEE",
3   TO_CHAR(5280, '99.99') AS "99.99",
4   TO_CHAR(1256, 'RN') AS "RN: 1,256",
5   TO_CHAR(1234.5678, '$999,999.00') AS "$999,999.00"
6  FROM DUAL;

S99,999.99 9.999EEEE 99.99 RN: 1,256      $999,999.00
+345.68    6.780E-03 #####      MCCLUI      $1,234.57
1 row selected.

SQL> SELECT TO_CHAR(345.678, '$99,999.99') AS "S99,999.99",
2   TO_CHAR(0.00678, '9.999EEEE') AS "9.999EEEE",
3   TO_CHAR(5280, '9999.99') AS "9999.99",
4   TO_CHAR(1256, 'RN') AS "RN: 1,256",
5   TO_CHAR(1234.5678, '$999,999.00') AS "$999,999.00"
6  FROM DUAL;

S99,999.99 9.999EEEE 9999.99 RN: 1,256      $999,999.00
+345.68    6.780E-03 05280.00      MCCLUI      $1,234.57
1 row selected.

SQL>

```

Рис. 5.22. Дополнительные символы задания формата

ТАБЛИЦА 5.6. Элементы моделей формата, относящихся к дате

Понятие	Параметр	Описание	Пример
Столетие	CC	Двухзначное представление века	20, 21
Квартал	Q	Однозначное представление квартала 1, 2, 3 или 4 года	
Год	YYYY, YYY, YY, Y, RR	Год, представленный четырьмя, тремя, двумя или одной цифрой. RR — округление до ближайшего года	2006, 006, 06, 6
Месяц	MONTH, Month, MON, Mon	Месяц, представленный полным или JANUARY, сокращенным именем. Использование прописных/строчных соответственно букв — как в шаблоне	January, JAN или Jan
	MM	Двухзначное представление месяца	01, 02, ..., 12
Неделя	WW, W	Двухзначное представление недели года, однозначное представление недели месяца	01–52; 1–5
День	DDD, DD, D	Трехзначное представления дня года, двухзначное представление дня месяца, однозначное представление дня недели	
	DAY, Day, DY, Dy	День недели: полное название, сокращенное название, прописными SUN, Sun буквами, прописная и строчные буквы соответственно	SUNDAY, Sunday,

**ТАБЛИЦА 5.7.** Элементы моделей формата, относящихся ко времени

Понятие	Параметр	Описание	Пример
Час	HH24, HH	Двухзначное представление часа в 24- или 12-часовом формате	23, 11
Минута	MI	Двухзначное представление минуты	Диапазон: 0–59
Секунда	SS	Двухзначное представление секунды. Диапазон: 0–59	
Разделители – / ; , : .		Символы, используемые для разделения значения даты и времени (дефис, косая черта, точка с запятой, запятая, двоеточие, точка)	DD-MON-YYYY; HH:MM:SS; YYYY/MM/DD
Суффиксы времени	AM, A.M., PM, P.M.	Время до (AM, A.M.) или после полудня (PM, P.M.)	12:45 P.M.
	AD, BC, A.D., B.C.	Год до нашей эры (BC, B.C.) или нашей эры (AD, A.D.)	1452 B.C.
	TH	Задает суффикс для порядковых чисел	1ST, 2ND, 3RD или 15TH
	SP	Число записывается словами	DDSP дает FIFTEEN (если день равен 15)
	TZR	Часовой пояс	PST, EST

Скорее всего, первое время вы будете использовать только часть описанных элементов. Позже, работая с более сложными приложениями баз данных, вы, возможно, вернетесь к данным таблицам, чтобы освежить их в памяти. Далее мы ознакомимся с некоторыми из представленных элементов модели формата, использовав их в следующем упражнении.

Для того чтобы поэкспериментировать с функцией TO\_DATE и некоторыми элементами модели формата, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle и очистите экран.
2. Введите и выполните следующую команду:

```
SELECT TO_DATE('January 10, 2006','MONTH DD, YYYY') AS "Date-1",
       TO_DATE('10/17/1999 14:23:39', 'MM/DD/YYYY HH24:MI:SS')
         AS "Date-2"
  FROM DUAL;
```

Обратите внимание на то, что результат отображается в “родном” формате Oracle:DD-MON-YY. Хотя из результатов видно, что значения были введены правильно, информация о времени отсутствует.

**Совет.** Как обычно, когда требуется отобразить на экране выражения, которые не зависят ни от одного столбца таблицы (выражения, состоящие из функций, математических операторов, строк символов и других литералов), используется таблица DUAL.

3. Модифицируйте приведенное в п. 2 выражение SELECT, используя каждую функцию TO\_DATE в роли первого аргумента функции TO\_CHAR. После этого добавьте выражение модели формата, выводящее на экран более подробную информацию о дате/времени. В результате у вас должна получиться следующая команда:

```
SELECT TO_CHAR(TO_DATE('January 10, 2006','MONTH DD, YYYY'),
               'MM/DD/YYYY HH:MI:SS AM') AS "Date-1",
       TO_CHAR(TO_DATE('10/17/1999 14:23:39', 'MM/DD/YYYY HH24:MI:SS'),
               'Month DD, YYYY HH:MI:SS AM')
               AS "Date-2"
  FROM DUAL;
```

Результаты выполнения двух приведенных команд представлены на рис. 5.23.

4. Введите и выполните приведенную ниже команду, чтобы отобразить на экране даты из базы данных Redwood Realty в выбранном вами формате.

```
CLEAR SCREEN
SELECT TO_DATE('01/01/'||YearBuilt, 'MM/DD/YYYY') AS "Computed",
       YearBuilt AS "Data Col".
  FROM Properties
 WHERE YearBuilt > 1994 AND UPPER(City) = 'MCKINLEYVILLE';
```

Результат выполнения данной команды показан на рис. 5.24. Обратите внимание на то, что при получении даты к строке “01/01” с помощью конкатенации присоединяется четырехзначный год.

5. Введите exit, нажмите клавишу <Enter>, чтобы выйти из Oracle, и выйдите из SQL\*Plus.

Функции TO\_DATE и TO\_CHAR используются не только для *отображения* значений на экране. Также часто они применяются для вставки дат и символьных данных в таблицы базы. Форма этих команд в выражении INSERT идентична форме приведенных выше примеров, только на этот раз функция TO\_CHAR и ее аргументы являются частью списка VALUES.

### Функция TO\_NUMBER

Функция TO\_NUMBER имеет следующий синтаксис:

```
TO_NUMBER(<str> [,<format-string>])
```

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT TO_DATE('January 10, 2006','MONTH DD, YYYY') AS "Date-1",
2      TO_DATE('10/17/1999 14:23:39', 'MM/DD/YYYY HH24:MI:SS') AS "Date-2"
3  FROM DUAL;
Date-1      Date-2
10-JAN-06 17-OCT-99

SQL> SELECT TO_CHAR(TO_DATE('January 10, 2006','MONTH DD, YYYY'),
2      'MM/DD/YYYY HH:MI:SS AM') AS "Date-1",
3      TO_CHAR(TO_DATE('10/17/1999 14:23:39', 'MM/DD/YYYY HH24:MI:SS'),
4      'Month DD, YYYY HH:MI:SS AM') AS "Date-2"
5  FROM DUAL;
Date-1          Date-2
01/10/2006 12:00:00 AM October    17, 1999 02:23:39 PM
SQL>

```

**Рис. 5.23.** Примеры функции TO\_DATE

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT TO_DATE('01/01/'||YearBuilt, 'MM/DD/YYYY') AS "Computed",
2      YearBuilt AS "Data Col."
3  FROM Properties
4 WHERE YearBuilt > 1994 AND UPPER(City) = 'MCKINLEYVILLE';

Computed      Data Col.
01-JAN-95      1995
01-JAN-96      1996
01-JAN-95      1995
01-JAN-96      1996
01-JAN-95      1995
SQL>

```

**Рис. 5.24.** Создание дат из констант и столбца с числами

В данном выражении `<str>` — это входной аргумент (выражение, имя столбца таблицы и т.д.), который преобразуется в число. Подобно функциям TO\_CHAR и TO\_DATE, в TO\_NUMBER можно указать необязательную модель формата, сообщающую, как интерпретировать первый аргумент. Стока формата модели может содержать те же элементы, что перечислены в табл. 5.5. Ниже для примера приведена команда, которая построчно преобразует столбец StreetNumber в число, добавляя к нему 400 и отображая результат на экране.

```
SELECT TO_NUMBER(StreetNumber) + 400
FROM DUAL;
```

## Использование специальных функций

Oracle предлагает несколько функций, которые выполняют специализированные преобразования данных и называются *специальными*. В данной книге рассматриваютсѧ функции DECODE, NVL, NVL2, TRANSLATE и CASE, относящиеся к этому классу. Подробное обсуждение выражения CASE приводилось в главе 4 (раздел “Введение в структуру CASE”), и здесь мы его дублировать не будем.

### Функция DECODE

Одной из наиболее интересных встроенных функций Oracle является DECODE, которая подставляет значения, используя заданное условие, и позволяет реализовывать логику IF-THEN-ELSE в SQL, не требуя PL/SQL. Синтаксис команды DECODE выглядит так:

```
DECODE(<value>, <search-value1>, <result1>,
       [<search-value2>, <result2>,...] <default-result>)
```

Значение <value> сравнивается с <search-value1>. Если <value> и <search-value1> равны, то Oracle возвращает в качестве ответа <result1>. Если значение не равно первому <search-value1> и при этом существует другая (необязательная) пара <search-value>/<result>, то <value> сравнивается со следующим значением <search-value>. Таким образом проверки продолжаются до тех пор, пока не будет найдено соответствие или не будут перебраны все значения <search-value>. Если соответствие так и не было получено, возвращается результат <default-result>, если он указан. Если необязательный аргумент <default-result> отсутствует, возвращается значение NULL. Ниже приведено несколько примеров функции DECODE.

```
SELECT DECODE(YearInSchool,1,'Freshman',2,'Sophomore',3,'Junior',
               4,'Senior')
  FROM SchoolData;
SELECT InventoryID, ItemName,
  DECODE(InventoryStatus,
         0,'Backordered',
         1,'Available',
         2,'Near reorder point',
         'Unknown')
  FROM Inventory;
```

В первой команде DECODE на основе значения столбца YearInSchool (1, 2, 3 или 4) на экране отображается название Freshman (“первокурсник”), Sophomore (“второкурсник”), Junior (“третекурсник”) или Senior (“старшекурсник”) соответственно. Таким образом, мы получаем более понятную информацию, не модифицируя при этом исходные значения столбца. Во втором примере вместо числовых кодов, представляющих состояние товара на складе, подставляются значимые слова. Если в столбце InventoryStatus хранится значение NULL или значение, отличное от 0, 1 или 2, то функция DECODE возвращает значение по умолчанию — Unknown.

### Функции NVL и NVL2

Функция NVL преобразует значение NULL в не NULL значение. Функция NVL2 действует приблизительно так же, только может превращать свой аргумент в не NULL значение или другое значение, если аргумент не равен NULL. Синтаксис команды NVL выглядит следующим образом:

```
NVL(<input>, <value>)
```

В данном выражении параметр `<input>` обычно является именем столбца таблицы, а `<value>` — это значение, подставляемое в том случае, если экземпляр значения столбца равен `NULL`. Обратите внимание на то, что тип данных `<value>` должен согласовываться с типом данных входного параметра. Если с этим возникают проблемы — преобразуйте значение `<input>` в другой тип данных, используя функцию преобразования, совместимую с `<value>`. Например, ниже приведено выражение, при выполнении которого генерируется ошибка Oracle “ORA-01722: invalid number”.

```
SELECT NVL(BidPrice, 'Null BidPrice') FROM CustAgentList;
```

Чтобы исправить данную ошибку, необходимо с помощью функции `TO_CHAR` преобразовать `BidPrice` в строку символов, чтобы это значение стало совместимым с символьным типом данных значения `Null BidPrice` функции `NVL`.

```
SELECT NVL(TO_CHAR(BidPrice), 'Null BidPrice') FROM CustAgentList;
```

Функция `NVL` оказывается весьма удобной еще в одной ситуации — когда требуется задействовать значение в числовых расчетах, но исходное значения столбца может быть равно `NULL`. В подобной ситуации следует помнить, что при добавлении к полю со значением `NULL` какого-либо значения результат все равно будет равен `NULL`. Следовательно, функцию `NVL` можно использовать, например, для нахождения общей суммы выплаты продавцу, которая складывается из стандартной и поощрительной сумм (причем размер премии может быть нулевым — соответствующее поле содержит значение `NULL`). Если решать задачу “в лоб” и использовать приведенную ниже команду `SELECT`, мы, независимо от значения в столбце `BasePay` (“зарплата”), получим на экране `NULL`, если значение в столбце `Incentive` (“премия”) равно `NULL`.

```
SELECT BasePay+Incentive FROM Employees;
```

Чтобы решить данную проблему, необходимо использовать функцию `NVL`, которая, находя в столбце `Employees` значение `NULL`, подставит вместо него 0. Таким образом, мы сможем нормально сложить два значения, не получая в результате `NULL`. Следовательно, требуемая команда `SELECT` выглядит так:

```
SELECT BasePay+NVL(Incentive, 0) AS Compensation FROM Employees;
```

При использовании функции `NVL2` Oracle отображает одно из двух значений, заданных в функции, а не столбец таблицы. Синтаксис данной команды выглядит следующим образом:

```
NVL2(<input>, <value-if-not-null>, <value-if-null>)
```

При выполнении данной команды второй аргумент возвращается, если значение `<input>` не равно `NULL`, а третий аргумент — если значение `<input>` равно `NULL`. В отличие от функции `NVL` тип данных второго и третьего аргументов *не обязательно* должен согласовываться со значением `<input>`.

### Функция `TRANSLATE`

Функция `TRANSLATE` находит в строке символы из заданного набора и заменяет их символами из другой строки. При этом замена выполняется по одному символу за раз. Синтаксис функции `TRANSLATE` приведен ниже.

```
TRANSLATE (<string1>,<string-chars-to-replace>,<replacement-chars>)
```

В данном выражении `<string1>` — это строка, символы которой Oracle изучает и выполняет операцию замены. Oracle использует второй аргумент, чтобы определить, какие символы исходной строки необходимо заменить. Символы, предназначенные для замены, находятся в третьем аргументе. Число символов во втором и третьем аргументах должно быть одинаковым. Как правило, команда `TRANSLATE` используется для кодирования сообщения с использованием (очень небезопасной) посимвольной подстановки. Другой вариант использования команды — в операторе `WHERE` для проверки *формата* значений столбца (но не его *содержимого*). Например, приведенная ниже команда гарантирует, что для всех сотрудников значения `TaxID`, соответствующие номерам социального обеспечения, будут иметь правильную форму (`XXX-XX-XXXX`, независимо от реальных значений). Приведенная функция `TRANSLATE` позволяет выделить все числа `TaxID`, не соответствующие допустимому формату.

```
SELECT FirstName, LastName, TaxID AS "Bad Format"
FROM Agents
WHERE TRANSLATE(TaxID,'0123456789','XXXXXXXXXX') <> 'XXX-XX-XXXX' ;
```

В приведенном выше операторе `WHERE` Oracle изучает поле `TaxID` и подставляет букву X вместо любой цифры (во втором аргументе перечислено десять цифр, а в третьем приведен список из десяти букв “X”, подставляемых вместо этих цифр). Таким образом выводимые на экран числа `TaxID` (но не значения в базе данных) преобразуются в общую форму, которую вы можете сравнить с эталонной строкой “`XXX-XX-XXXX`”. Любое выявленное отличие выводится на экран. Если все поля `TaxID` соответствуют формату, запрос не возвращает ни одной строки.

Чтобы понять действие описанных функций, с ними следует поэкспериментировать. Для начала запустите SQL\*Plus, войдите в Oracle и очистите экран. После этого выполните указанные ниже действия со специальными функциями.

1. Введите и выполните нижеприведенную команду, чтобы отобразить на экране уникальные значения `SaleStatusID` в таблице `Listings`. С помощью команды `DECODE` коды переводятся в понятные термины.

```
SELECT DISTINCT DECODE(SaleStatusID,
101,'For Sale',
102,'Pending',
103,'Sold',
'UNKNOWN') AS "Status"
FROM Listings;
```

2. Введите и выполните следующее выражение SQL, чтобы получить на экране `Licensed` или `Something Else` для всех возможных значений поля `LicenseStatusID`:

```
SELECT DISTINCT DECODE(LicenseStatusID,
    1001, 'Licensed', 'Something Else') AS "License Status"
FROM Agents;
```

Oracle вернет две строки, указывающие, что в выбранной таблице существуют коды, соответствующие Licensed, а также другие коды. Не забудьте использовать оператор DISTINCT, чтобы сократить число строк, возвращаемых Oracle.

3. Ведите и выполните нижеприведенное выражение SQL, чтобы перечислить выбранную информацию о клиентах (мобильный, рабочий и домашний телефоны). Для искусственного ограничения числа возвращаемых строк применяется оператор WHERE.

```
SELECT
    NVL(HomePhone, 'unknown') AS "Home Phone",
    NVL(CellPhone, 'unknown') AS "Cell Phone",
    NVL(WorkPhone, 'unknown') AS "Work Phone"
FROM Customers WHERE CustomerID < 25030;
```

Результаты выполнения указанных действий показаны на рис. 5.25. У вас должно получиться то же самое.

4. Обновите столбец в таблице Agents, чтобы с помощью SELECT можно было определить его неправильную форму. Ведите и выполните следующие команды:

```
CLEAR SCREEN
COMMIT;
UPDATE Agents
SET WorkPhone = TO_CHAR(SYSDATE, 'HH:MI:SS')
WHERE Gender = 'F';
```

5. Теперь используйте выражение TRANSLATE, чтобы выделить все номера телефонов WorkPhone, не имеющих стандартной формы “(ddd) ddd-dddd” (где “d” — любая цифра). Выполните следующую команду:

```
SELECT LastName, WorkPhone FROM Agents
WHERE TRANSLATE(WorkPhone, '9876543210', 'aaaaaaaaaa') <>
      '(aaa) aaa-aaa';
```

На рис. 5.26 показаны номера телефонов, не соответствующие стандартному шаблону.

6. Отмените изменения базы данных, которые мы использовали для иллюстрации команды TRANSLATE, выполнив следующее выражение:

```
ROLLBACK;
```

7. Ведите exit, нажмите клавишу <Enter>, чтобы выйти из Oracle, и закройте SQL\*Plus.

The screenshot shows the Oracle SQL\*Plus interface with the following content:

```

SQL> SELECT DISTINCT DECODE(SaleStatusID,
2      101,'For Sale',
3      102,'Pending',
4      103,'Sold',
5      'UNKNOWN') AS "Status"
6  FROM Listings;

Status
-----
For Sale
Pending
Sold

SQL> SELECT DISTINCT DECODE(LicenseStatusID,
2      1001,'Licensed',
3      'Something Else') AS "License Status"
4  FROM Agents;

License Status
-----
Licensed
Something Else

SQL> SELECT
2      NVL(HomePhone,'unknown') AS "Home Phone",
3      NVL(CellPhone,'unknown') AS "Cell Phone",
4      NVL(WorkPhone,'unknown') AS "Work Phone"
5  FROM Customers WHERE CustomerID < 25030;

Home Phone          Cell Phone          Work Phone
(707) 555-4027    (707) 555-7090    unknown
(707) 555-1747    (707) 555-0738    (707) 555-2401
(707) 555-7223    (707) 555-5161    unknown
(707) 555-8746    (707) 555-6708    (707) 555-5148
(707) 555-8137    (707) 555-4895    unknown
(707) 555-2636    (707) 555-9427    (707) 555-7202
(707) 555-0476    (707) 555-2853    unknown

7 rows selected.

SQL>

```

Рис. 5.25. Использование специальных функций

## Использование агрегирующих функций

Все функции, которые вы изучили до данного момента, относятся к односторонним. Они возвращают одну строку для каждой обработанной строки. В данном разделе вы узнаете об агрегирующих функциях. *Агрегирующие функции* действуют на группу связанных строк и возвращают одну строку для каждой обработанной группы. Функции называются “агрегирующими”, поскольку они формируют результат из набора связанных элементов. Кроме того, их часто называют *групповыми*. Можно привести множество примеров использования агрегирующих функций. Возможно, вам требуется общая сумма продаж домов, расположенных в определенном районе, или необходимо рассчитать общую сумму комиссионных, полученных отдельно риэлторами-мужчинами и риэлторами-женщинами. Кроме того, вы можете вывести на экран минимальную и максимальную цены продаваемого дома в г. Арката.

Агрегирующие функции преимущественно возвращают числовые результаты и обычно используются для расчета определенной статистики. Ниже мы подробно разберем наиболее распространенные агрегирующие, или многострочные, функции AVG, COUNT, MAX, MIN и SUM (табл. 5.8). Кроме того, “близким родственником” данных функций является оператор GROUP BY выражения SELECT, который группирует строки

The screenshot shows a session in Oracle SQL\*Plus. The user has run a COMMIT command, followed by an UPDATE statement that updates 14 rows in the Agents table. Then, a SELECT statement is run to show the results of a WHERE clause using the TRANSLATE function, which filters out rows where the WorkPhone number starts with '(aaa) aaa-aaa'. Finally, a ROLLBACK command is issued to undo the changes.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> COMMIT;
Commit complete.

SQL> UPDATE Agents
  2 SET WorkPhone = TO_CHAR(SYSDATE, 'HH:MI:SS')
  3 WHERE Gender = 'F';
14 rows updated.

SQL> SELECT LastName, WorkPhone
  2 FROM Agents
  3 WHERE TRANSLATE(WorkPhone, '9876543210', 'aaaaaaaaaa') <> '(aaa) aaa-aaa';
LASTNAME          WORKPHONE
-----            -----
Dahlen           12:21:55
Sheibani         12:21:55
Romero           12:21:55
Robinson         12:21:55
Piperova        12:21:55
Reed              12:21:55
Chong             12:21:55
Selby             12:21:55
Taylor            12:21:55
Herring           12:21:55
Fernandez        12:21:55

LASTNAME          WORKPHONE
-----            -----
Williams          12:21:55
Allee             12:21:55
Lewis              12:21:55

14 rows selected.

SQL> ROLLBACK;
Rollback complete.

SQL>

```

**Рис. 5.26.** Функция TRANSLATE находит данные, имеющие неправильный формат

**ТАБЛИЦА 5.8.** Выбранные многострочные функции

Функция	Описание
AVG (<n>)	Возвращает среднее значение <n>
COUNT (*)	Возвращает число строк, извлеченных запросом
MAX (<n>)	Возвращает максимальное значение <n>
MIN (<n>)	Возвращает минимальное значение <n>
SUM (<n>)	Возвращает сумму <n>

по одной или нескольким общим характеристикам. Существует еще и необязательный оператор HAVING, фильтрующий возвращаемые группы. Подробно операторы GROUP BY и HAVING будут рассмотрены ниже.

Общий синтаксис агрегирующей функции выглядит следующим образом:

Aggregate-function-name( [DISTINCT | ALL] expression )

Необязательная команда ALL в общем-то не нужна, поскольку она задает поведение агрегирующей функции по умолчанию и указывает Oracle задействовать в обработке все значения (исключая NULL). Оператор DISTINCT задает использование в функции только различных ненулевых значений.

Существует несколько моментов, о которых следует помнить при работе с агрегирующими функциями.

- Агрегирующие функции игнорируют значения `NULL`, поскольку их использование недопустимо при расчете любой значимой статистики.
- Функции `COUNT`, `MAX` и `MIN` можно использовать не только с числами, но и со строками и с данными, относящимися к дате/времени.
- Ключевое слово `DISTINCT` исключает из расчетов дублирующиеся значения.

## Функции `AVG`, `MAX` и `MIN`

Функция `AVG` рассчитывает среднее числовых значений в заданном столбце. Если вы хотите собрать какую-то статистику и узнать минимальную, среднюю и максимальную запрашиваемую цену для всей представленной в базе недвижимости, выполните следующий запрос:

```
SELECT MIN(AskingPrice), AVG(AskingPrice), MAX(AskingPrice)
  FROM Listings;
```

Вы можете получить более подробную информацию, добавив в запрос оператор `WHERE`, который ограничит возвращаемые (а следовательно, обрабатываемые) строки. Допустим, например, что вам требуется та же статистика, что и ранее, но для домов, для которых дата снятия с продажи принадлежит к первому кварталу 2007. Ниже показано, как выбрать из базы данных требуемое подмножество строк и выполнить с ними необходимые вычисления.

```
SELECT MIN(AskingPrice), AVG(AskingPrice), MAX(AskingPrice)
  FROM Listings
 WHERE EndListDate BETWEEN '01-JAN-07' AND '31-MAR-07';
```

В качестве аргументов агрегирующих функций вы можете использовать выражения. Например, чтобы рассчитать среднюю прибыль гипотетической таблицы `Products`, необходимо выполнить команду `SELECT`, усредняющую разность между значениями в столбцах `Price` и `Cost` каждой строки, отформатировав результат с помощью функции `TO_CHAR`:

```
SELECT TO_CHAR(AVG(Price-Cost), '$999.99') AS "Average Profit"
  FROM Products;
```

Помещая функцию `AVG` в функцию `TO_CHAR`, мы указываем Oracle рассчитать среднее по всем строкам таблицы `Products`, значения в столбцах `Price` и `Cost` которых не равны `NULL`, а затем отформатировать результаты с использованием модели формата функции `TO_CHAR`.

Как вы, возможно, уже догадались, функция `MIN` для всех задействованных строк вычисляет наименьшее значение в заданном столбце или рассчитанном выражении. Подобным образом, `MAX` выводит на экран максимальное значение столбца или выражения для всех указанных строк. Функции `MIN` и `MAX` также могут обрабатывать

нечисловые данные. В частности, вы можете рассчитать наибольшую или наименьшую дату, время или строку символов. Считается, что самый поздний момент времени является наибольшим, соответственно самый ранний — наименьшим. Для строк символов буква А “меньше” буквы В и т.д. Применяя функцию MAX к строке символов, мы получим “самый большой” заголовок. Например, приведенная ниже команда выводит на экран “самый большой” (в указанном смысле) адрес из всех домов, перечисленных в таблице Properties.

```
SELECT MAX(Address) FROM Properties;
```

В качестве ответа данный запрос выдает “991 Bayside Rd”, поскольку числа считаются даже “больше”, чем буквы. Подобным образом приведенная ниже команда находит пользователя с “наименьшим” именем — Aaron.

```
SELECT MIN(LastName) FROM Customers;
```

Если вы попытаетесь найти среднее строки символов, Oracle генерирует сообщение об ошибке: “ORA-01722: invalid number”.

Необязательная фраза DISTINCT гарантирует сбор и использование в расчетах только уникальных значений. Предположим, например, что вам требуется узнать средний метраж (в квадратных футах) домов, перечисленных в таблице Properties. Решить данную задачу можно двумя способами:

```
SELECT AVG(SqFt) FROM Properties; (1)
```

или

```
SELECT AVG(DISTINCT SqFt) FROM Properties; (2)
```

Результат первого запроса — 1784,57 (округленно), а результат второго — 1874,41. Во втором случае дома, имеющие одинаковое значение SqFt, представлялись одной записью, после чего вычислялось среднее значение. Чтобы понять сказанное, предположим, что у нас есть набор строк, имеющих в столбце Bedroom значения 2, 3, 2, 5, 3 и 3. Применение функции AVG к этим значениям дает  $(2+3+2+5+3+3)/6$ , или 3,00. Функция же AVG (DISTINCT) дает  $(2+3+5)/3$ , или 1,67.

## Введение в функции SUM и COUNT

Функция SUM находит сумму значений, записанных в столбце, или выражений, основанных на значениях столбца. Суммирование полей с датами, временем или символами бессмысленно, поэтому Oracle выдаст сообщение об ошибке, если вы попытаетесь применить функцию SUM к чему-то отличному от числовых столбцов или выражений. Опция DISTINCT указывает Oracle удалить дублирующиеся числовые значения перед их суммированием. Значение ALL (принятое по умолчанию, следовательно, необязательное) указывает Oracle просуммировать все ненулевые значения столбца. Как и в любой запрос, вы можете включить в команду SUM оператор WHERE, позволяющий контролировать отбор строк для суммирования. Если требуется, вы можете просуммировать общую сумму предполагаемых комиссионных, которую бы получили все

риэлторы при ставке 3% от BidPrice. Ниже соответствующий расчет выполняется для продаж, выполненных в течение ноября 2006 г.

```
SELECT SUM(BidPrice * 0.03)
FROM CustAgentList
WHERE ContactDate BETWEEN '01-NOV-06' AND '30-NOV-06';
```

Приведенный запрос собирает значения BidPrice между указанными датами, последовательно умножает их на 0,03 и суммирует произведения. Если вы запустите этот запрос в базе данных Redwood Realty, то получите результат, равный 322069,77 (не отформатировано). Выполняя тот же запрос *без* оператора WHERE, вы получите гораздо большую величину – 2119914,81 (не отформатировано).

Функция COUNT немного отличается от других агрегирующих функций. Она может использовать любой тип данных – числовой, символьный или дату/время. Всего существует три разновидности этой функции.

- COUNT(\*) подсчитывает все строки указанной таблицы, включая те, что имеют в различных полях значения NULL.
- COUNT(<expression>) подсчитывает число строк, которые в указанных столбцах имеют значения, не равные NULL.
- COUNT(DISTINCT <expression>) подсчитывает число различных, не равных NULL, значений в столбце или выражении.

Синтаксис функции COUNT выглядит следующим образом:

```
COUNT(* | [ | DISTINCT | ALL] <expression>)
```

Если вам требуется узнать, сколько всего записей в таблице, выполните следующую команду:

```
COUNT(*)
```

Если вам нужно подсчитать число разных записей в столбце, используйте такую функцию COUNT:

```
COUNT(DISTINCT <column-name>)
```

Наконец, если вы просто хотите подсчитать все значения в столбце, включая дублирующиеся значения, но не считая значения NULL, выполните следующую команду:

```
COUNT(<column-name>)
```

Когда вам потребуется подсчитать число строк для определенного подмножества строк, включите в команду оператор WHERE, чтобы ограничить строки, которые извлекает Oracle перед подсчетом строк, значений или уникальных значений. Предположим, например, что вам требуется узнать, сколько строк находится в таблице Properties базы данных Redwood Realty. Чтобы вывести на экран только одну строку с указанием счетчика, выполните следующую команду:

```
SELECT COUNT(*) FROM Properties;
```

Кроме того, для подсчета числа возвращаемых строк вы можете использовать в скобках вместо \* любую константу (например, 1 или 7 или ‘wow’):

```
SELECT COUNT(7) FROM Properties;
```

В любом случае результат будет одним и тем же — 2000. Для каждой возвращаемой строки Oracle подсчитывает число звездочек, или число единиц, или число семерок и т.д., т.е. в таблице *Properties* представлено 2 000 домов. С другой стороны, вашему брокеру может понадобиться узнать, сколько клиентов (*Customers*) живет в г. Орик (*Orick*). Допустим, он хочет отправить всем им письма, но желает предварительно оценить стоимость подобного шага. Итак, ниже приведена команда, позволяющая подсчитать число жителей г. Орика в таблице *Customers*.

```
SELECT COUNT(*)  
FROM Properties  
WHERE City = 'Orick';
```

Используя ключевое слово *DISTINCT* с функцией *COUNT*, будьте очень осторожны. Если вы поместите *DISTINCT* сразу после слова *SELECT*, подобный запрос подсчитает число строк в таблице. С другой стороны, если поместить слово *DISTINCT* после *COUNT* в скобках, Oracle подсчитает число уникальных значений столбца — совершенно другую величину. Что это означает? Чтобы ответить на этот вопрос, выполним два запроса и обсудим разницу между ними. Запустите SQL\*Plus, войдите в Oracle и очистите экран.

Чтобы использовать функцию *COUNT*, необходимо выполнить следующие действия.

1. Введите и выполните следующую команду:

```
SELECT DISTINCT COUNT(City) AS "Unique Cities"  
FROM Customers;
```

Какой ответ вы получили? Данное число, 2500, — это количество строк в таблице *Customers*, а не число различных городов, содержащихся в ней.

2. Модифицируйте запрос и еще раз выполните его.

```
SELECT COUNT(DISTINCT City) AS "Unique Cities"  
FROM Customers;
```

Какой ответ вы получили теперь? (Если вы не уверены насчет использования прописных/строчных букв в базе данных, подставьте вместо *DISTINCT City* выражение *DISTINCT UPPER(City)*.) Oracle возвращает ответ 9. Следовательно, в столбце *City* таблицы *Customers* фигурирует 9 различных городов.

3. Узнаем теперь, для скольких клиентов, перечисленных в таблице *Customers*, пуст столбец *WorkPhone*? Чтобы ответить на данный вопрос, введите и выполните следующую команду:

```
SELECT COUNT(*)  
FROM Customers  
WHERE WorkPhone IS NULL;
```

The screenshot shows a window titled "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area contains the following SQL queries and their results:

```

SQL> SELECT DISTINCT COUNT(City) AS "Unique Cities"
  2  FROM Customers;
Unique Cities
-----
2500

SQL> SELECT COUNT(DISTINCT City) AS "Unique Cities"
  2  FROM Customers;
Unique Cities
-----
9

SQL> SELECT COUNT(*)
  2  FROM Customers
  3  WHERE WorkPhone IS NULL;
COUNT(*)
-----
1991

SQL> SELECT COUNT(*) FROM Properties WHERE LOWER(city) = 'arcata';
COUNT(*)
-----
419

SQL> SELECT COUNT(*) FROM Properties
  2  WHERE LOWER(City) LIKE 'mck%';
COUNT(*)
-----
255

SQL>

```

**Рис. 5.27.** Примеры использования функции COUNT

Oracle возвращает ответ 1991. Следовательно, из 2 500 перечисленных клиентов для 1 991 не указаны рабочие телефоны.

4. Сколько домов из г. Арката, Калифорния, перечислено в таблице Properties? Введите и выполните следующую команду:

```
SELECT COUNT(*) FROM Properties WHERE LOWER(City) = 'arcata';
```

5. Сколько домов из г. Маккинливиль (McKinleyville) перечислено в таблице Properties? Чтобы найти ответ на этот вопрос, введите и выполните приведенную ниже команду. (Поскольку вы можете не знать, как правильно пишется название данного города, — используйте групповые символы. Не переусердствуйте с сокращением, иначе критерий “захватит” и другие города и подсчет будет неверным.)

```
SELECT COUNT(*) FROM Properties
WHERE LOWER(City) LIKE 'mck%';
```

Результаты приведенных действий показаны на рис. 5.27.

6. Из SQL\*Plus не выходите, поскольку чуть позже мы проиллюстрируем в данном интерфейсе использование функции COUNT в другой роли.

Помните, в главе 4 мы представляли структуру CASE? В данной главе мы используем CASE для подсчета числа домов на продажу в различных ценовых категориях.

Для выполнения частотного анализа в диапазоне цен мы используем функцию COUNT и выражение CASE. Предположим, что вам требуется узнать число домов в различных диапазонах частот. Для решения этой задачи можно ввести серию запросов типа

```
SELECT COUNT(1)
  FROM Listings
 WHERE AskingPrice BETWEEN 100001 и 200000;
```

Oracle возвращает значение 455, поскольку именно столько домов принадлежит ценовому диапазону от 100 001 до 200 000 долларов включительно. Тем не менее, чтобы получить полную картину, потребуется несколько выражений SELECT, каждое со своим диапазоном цен. Разумеется, вы можете использовать подстановочные переменные и вводить границы диапазона при запуске команды, но такое решение все равно предполагает запуск нескольких запросов. Возникает вопрос: можно ли как-то запустить один запрос, который выдаст требуемое нам распределение? Да, можно. Вы можете использовать выражения CASE и COUNT, чтобы разбить значения в столбце AskingPrice на группы, а затем подсчитать число элементов в каждой группе. Фраза GROUP BY, которая будет рассмотрена в следующем разделе, вам не поможет, поскольку группы будут слишком маленькими, —различие в цене составляет всего несколько долларов. Для разбивки на группы используем значения, кратные 100 000 долларов.

Решение данной задачи представлено ниже: вы создадите структуру CASE/WHEN/ELSE в каждой функции COUNT. Выражение CASE дает значение 1, когда AskingPrice попадает в указанный диапазон, в противном случае генерируется значение NULL. При выполнении общего запроса каждая функция COUNT подсчитывает значения, попавшие в ее диапазон, и выдает частотное распределение. Чтобы лучше понять сказанное, желательно написать и выполнить этот запрос.

Итак, от вас требуется построить по базе данных Redwood Realty частотное распределение значений AskingPrice по шести диапазонам: от 0 до 600 000 долларов с шагом 100 000. В последнюю (седьмую) категорию попадают все дома, цена на которые превышает 600 000 долларов.

Чтобы создать с помощью выражений COUNT и CASE запрос относительно частотного распределения, необходимо выполнить следующие действия.

1. В окне SQL\*Plus введите CLEAR SCREEN, нажмите клавишу <Enter>, а затем введите и выполните следующую команду:

```
SELECT
  COUNT(CASE WHEN AskingPrice BETWEEN 0 AND 100000
            THEN 1 ELSE NULL END) "<= 100000"
  FROM Listings;
```

Oracle отобразит на экране значение 6 под заголовком “<= 100000”.

2. Модифицируйте выражение SELECT из п. 1 на указанную ниже полную версию. Внимательно проверьте, чтобы диапазоны частот не перекрывались, а затем вы-

The screenshot shows the Oracle SQL\*Plus interface with the following command and output:

```

SQL> SELECT
2   COUNT(CASE WHEN AskingPrice BETWEEN 0 AND 100000
3      THEN 1 ELSE NULL END) "<= 100000"
4  FROM Listings;
<= 100000
-----
6

SQL> SELECT
2   COUNT(CASE WHEN AskingPrice BETWEEN 0 AND 100000
3      THEN 1 ELSE NULL END) "<= 100000",
4   COUNT(CASE WHEN AskingPrice BETWEEN 100001 AND 200000
5      THEN 1 ELSE NULL END) "<= 200000",
6   COUNT(CASE WHEN AskingPrice BETWEEN 200001 AND 300000
7      THEN 1 ELSE NULL END) "<= 300000",
8   COUNT(CASE WHEN AskingPrice BETWEEN 300001 AND 400000
9      THEN 1 ELSE NULL END) "<= 400000",
10  COUNT(CASE WHEN AskingPrice BETWEEN 400001 AND 500000
11     THEN 1 ELSE NULL END) "<= 500000",
12  COUNT(CASE WHEN AskingPrice BETWEEN 500001 AND 600000
13     THEN 1 ELSE NULL END) "<= 600000",
14  COUNT(CASE WHEN AskingPrice > 600000
15     THEN 1 ELSE NULL END) "> 700000"
16 FROM Listings;

```

<= 100000	<= 200000	<= 300000	<= 400000	<= 500000	<= 600000	> 700000
6	455	29	6	3	2	1

SQL>

**Рис. 5.28.** Создание частотного распределения

полните эту команду. (Для создания выражения SELECT используйте программу Блокнот, затем скопируйте готовый код в SQL\*Plus. Таким образом, вам будет легче исправить ошибки, если они возникнут.) В итоге вы должны получить результат, показанный на рис. 5.28.

```

SELECT
  COUNT(CASE WHEN AskingPrice BETWEEN 0 AND 100000
             THEN 1 ELSE NULL END) "<= 100000",
  COUNT(CASE WHEN AskingPrice BETWEEN 100001 AND 200000
             THEN 1 ELSE NULL END) "<= 200000",
  COUNT(CASE WHEN AskingPrice BETWEEN 200001 AND 300000
             THEN 1 ELSE NULL END) "<= 300000",
  COUNT(CASE WHEN AskingPrice BETWEEN 300001 AND 400000
             THEN 1 ELSE NULL END) "<= 400000",
  COUNT(CASE WHEN AskingPrice BETWEEN 400001 AND 500000
             THEN 1 ELSE NULL END) "<= 500000",
  COUNT(CASE WHEN AskingPrice BETWEEN 500001 AND 600000
             THEN 1 ELSE NULL END) "<= 600000",
  COUNT(CASE WHEN AskingPrice > 600000
             THEN 1 ELSE NULL END) "> 700000"
FROM Listings;

```

3. В SQL\*Plus введите `exit`, нажмите клавишу `<Enter>`, чтобы выйти из Oracle, и закройте клиентскую программу SQL\*Plus.

## Группировка результатов

Задействовав необязательную фразу GROUP BY в выражении SELECT, вы сможете генерировать суммарную статистику, используя агрегирующие функции на основе

группировки строк. В предыдущем разделе вы создали групповую функцию, генерирующую результаты на основе *всех* выбранных строк. С помощью фразы GROUP BY вы сможете определять группы, обладающие общими характеристиками, а затем генерировать статистику для каждой группы. В результате Oracle будет возвращать столько строк, сколько групп вы определите.

Чтобы понять, насколько полезна фраза GROUP BY, рассмотрим, как вы можете собрать статистику о недвижимости. Предположим, что вам требуется узнать средний метраж домов в каждом городе из представленных в таблице Properties, а также количество этих домов. Итак, прежде всего необходимо запустить запрос, возвращающий все уникальные имена городов:

```
SELECT DISTINCT City FROM Properties ORDER BY City;
```

Oracle возвращает упорядоченный список из девяти городов, начинающийся с г. Арката (Arcata). Далее вам требуется запустить девять запросов, подобных приведенному ниже, чтобы собрать статистику по двум пунктам (по одному для каждого отдельного города).

```
SELECT COUNT(*), AVG(SqFt), 'Arcata' AS "City"  
FROM Properties WHERE City = 'Arcata';
```

Да, запуск девяти запросов — это очень трудоемкий процесс. Более того, даже если вы запишете все девять запросов как файл сценария и когда-либо запустите его снова, всегда будет существовать возможность, что в таблице Properties появится строка с новым названием города. Наверняка должен быть какой-то лучший способ. И он действительно есть — это добавление фразы GROUP BY в выражение SELECT.

Чтобы получить многострочный результат, отображающий собранные значения для различных групп, необходимо добавить одно или несколько имен столбцов в список SELECT. Добавленные столбцы определяют группы и сообщают Oracle группировать суммарные результаты, вначале сортируя строки в группах по столбцу группы, а затем генерируя суммарные результаты для каждой группы. Ниже приведена одна команда SELECT, которая дает ту же статистику, что и девять приведенных ранее выражений SELECT.

```
SELECT City, COUNT(*), AVG(SqFt)  
FROM Properties  
GROUP BY City;
```

Столбец City в списке SELECT является столбцом группировки. Oracle группирует строки по значению в столбце City, а затем подсчитывает общее число и среднее значение для каждой из десяти групп, в результате чего выдается девятистрочный результат.

Используя фразу GROUP BY, следует помнить о нескольких ограничениях команды SELECT. Вам необходимо задать только один столбец или комбинацию столбцов, которые будут основой для группировки, константу или агрегирующую функцию в списке SELECT. Более того, столбцы группировки, которые вы задаете в списке

SELECT, необходимо включить в список GROUP BY. Если вы используете несколько столбцов, их имена необходимо разделить запятыми. Например, в приведенном ниже выражении SELECT группы формируются на основе значений City и Bedrooms (любых комбинаций), поскольку эти два элемента списка SELECT не являются агрегирующими функциями.

```
SELECT City, Bedrooms, COUNT(*), AVG(SqFt)
  FROM Properties
 GROUP BY City, Bedrooms;
```

Следовательно, оператор GROUP BY должен содержать имена обоих столбцов. Порядок, в котором вы перечисляете столбцы в GROUP BY, определяет порядок сортировки групп: первый столбец GROUP BY является первым параметром сортировки, а группы, которые получаются после сортировки по этому параметру, сортируются внутренне уже по следующему параметру. Разумеется, вы можете использовать оператор GROUP BY, не упоминая никаких агрегирующих функций в списке SELECT. Например, ниже показано, как перечислить все уникальные комбинации числа комнат и санузлов в таблице Properties.

```
SELECT Bedrooms, Bathrooms FROM Properties
 GROUP BY Bedrooms, Bathrooms;
```

Если вы запустите этот запрос, Oracle выдаст 18 строк, начинающихся с характеристики “2 комнаты/1 санузел” и заканчивающихся “6 комнат/4 санузла”.

## Фильтрация групп с помощью оператора HAVING

Необязательный оператор HAVING выражения SELECT ограничивает (фильтрует) группы строк, возвращаемых запросом. Оператор HAVING располагается после оператора GROUP BY и указывается только тогда, когда в выражении есть оператор GROUP BY. Хотя и WHERE, и HAVING применяются для фильтрации строк, оператор WHERE действует на отдельную строку — *до того*, как Oracle сформирует группы, и *до того*, как группы будут профильтрованы с помощью оператора HAVING. Задавая оператор WHERE, вы исключаете выбранные строки из рассмотрения любой операцией группировки. Оператор HAVING, наоборот, начинает действовать только после того, как группы сформированы, определяя, какие группы будут включены в результат. В каком-то смысле HAVING — это оператор WHERE для групп. Синтаксис оператора HAVING и его относительное положение в операторе SELECT выглядят следующим образом:

```
SELECT ...
  FROM ...
 WHERE ...
 GROUP BY ...
 HAVING ...
 ORDER BY ...;
```

Оператор HAVING не обязан содержать агрегирующие функции из списка SELECT. Например, вы можете сформировать группы по названиям городов и подсчи-

тать число элементов в каждой группе; однако оператор HAVING может фильтровать группы по другому совокупному значению. Сказанное иллюстрирует следующая команда SELECT:

```
SELECT City, COUNT(*)
FROM Properties
GROUP BY City
HAVING AVG(SqFt) > 1800;
```

В результате на экране отображаются три названия городов и подсчитанное число строк в каждой группе, но только для городов, средний метраж недвижимости в которых превышает 1800 квадратных футов (соответствующую статистику рассчитывает Oracle и использует для фильтрации групп). Ну что ж, теперь пришло время потренироваться в использовании операторов GROUP BY и HAVING. Для подготовки к последующим действиям запустите SQL\*Plus.

Для того чтобы сгруппировать и отфильтровать совокупную информацию по столбцам, выполните следующие действия.

1. Очистите экран SQL\*Plus, а затем введите и выполните следующую команду, чтобы подсчитать по городам число значений в столбце Properties и рассортировать их по убыванию счетчика:

```
SELECT City, COUNT(*)
FROM Properties
GROUP BY City
ORDER BY 2 DESC;
```

2. Модифицируйте предыдущую команду, добавив оператор WHERE, удаляющий из рассмотрения дома, в которых имеется меньше 5 комнат.

```
SELECT City, COUNT(*)
FROM Properties
WHERE Bedrooms >= 5
GROUP BY City
ORDER BY 2 DESC;
```

Результат выполнения данной команды показан на рис. 5.29.

3. Модифицируйте запрос, удалив оператор WHERE и добавив оператор HAVING, чтобы отобразить на экране только те города, в которых проживает более 300 клиентов. Удалите оператор ORDER BY. Введите и выполните следующие команды:

```
CLEAR SCREEN
SELECT City, COUNT(*)
FROM Properties
GROUP BY City
HAVING COUNT(*) > 300;
```

4. Рассчитайте несколько совокупных значений из таблицы Properties и сгруппируйте их по городам. Для удобства отображения значений на экране используйте указанное форматирование.

```

SQL> SELECT CITY, COUNT(*)
  2  FROM Properties
  3  GROUP BY City
  4  ORDER BY 2 DESC;

CITY                  COUNT(*)
-----                -----
Eureka                   889
Arcata                    419
Fortuna                   301
McKinleyville              255
Trinidad                     40
Blue Lake                      37
Westhaven                     26
Loleta                        24
Orick                          9

9 rows selected.

SQL> SELECT CITY, COUNT(*)
  2  FROM Properties
  3  WHERE Bedrooms >= 5
  4  GROUP BY City
  5  ORDER BY 2 DESC;

CITY                  COUNT(*)
-----                -----
Eureka                   29
Arcata                    12
Fortuna                   10
McKinleyville                 8
Blue Lake                      2
Loleta                        1
Trinidad                     1

7 rows selected.

SQL>

```

**Рис. 5.29.** Использование оператора GROUP BY с оператором WHERE

```

COLUMN AVG(SqFt) FORMAT 9999
COLUMN AVG(YearBuilt) FORMAT 9999
SELECT City, MIN(SqFT), AVG(SqFt), MAX(SqFT), AVG(YearBuilt)
FROM Properties
GROUP BY CITY
ORDER BY City DESC;

```

Результаты выполнения пп. 3 и 4 показаны на рис. 5.30. Вы должны получить тоже самое.

5. Введите CLEAR COLUMNS и нажмите клавишу <Enter>, чтобы удалить со столбцов наложенное форматирование.
6. Если вы решили передохнуть — сейчас для этого наступает удобный момент. Введите exit и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus.

## Форматирование выхода SQL\*Plus и создание простых отчетов

В главе 5 вы использовали iSQL\*Plus (немного) и SQL\*Plus (много) для создания запросов, отображающих на экране информацию, выбранную из таблицы. Несколько раз вы использовали псевдонимы столбцов, выполняли команду CLEAR COLUMN

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT City, COUNT(*)
  2  FROM Properties
  3  GROUP BY City
  4  HAVING COUNT(*) > 300;
CITY          COUNT(*)
Arcata           419
Eureka            889
Fortuna           301

SQL> COLUMN AVG(SqFt)      FORMAT 9999
SQL> COLUMN AVG(YearBuilt) FORMAT 9999
SQL> SELECT City, MIN(SqFT), AVG(SqFt), MAX(SqFT), AVG(YearBuilt)
  2  FROM Properties
  3  GROUP BY CITY
  4  ORDER BY City DESC;

CITY          MIN(SQFT)    AVG(SQFT)    MAX(SQFT)    AVG(YEARBUILT)
Westhaven       1130        1910        4200        1976
Trinidad        960         1707        2727        1975
Orick           1158        1715        2700        1971
McKinleyville   959         1738        5600        1976
Luled           905         1715        2705        1978
Fortuna          940         1778        4498        1978
Eureka            840         1793        7000        1976
Blue Lake        1020        1815        4828        1975
Arcata           825         1801        6900        1975

9 rows selected.
SQL>

```

**Рис. 5.30.** Два запроса SELECT с оператором GROUP BY

и даже команды COLUMN. Однако при этом вы не использовали большинство команд SQL\*Plus (в противоположность командам, или выражениям, SQL) для создания отчетов, отформатированных удобным для восприятия образом. Фактически SQL\*Plus содержит большой набор команд, позволяющих задавать заголовки и подписи отчетов, подавлять вывод дублирующейся информации, форматировать столбцы, содержащие строки символов и числа, а также выполнять различные расчеты. Все вместе это часто называют возможностями *форматирования отчетов*. Хотя число операторов форматирования отчетов достаточно велико, мы рассмотрим только важнейшие из них (табл. 5.9). Отметим также, что хотя SQL\*Plus предлагает огромное количество команд, которые помогают создавать простые отчеты, когда дело доходит до создания более сложных отчетов, ничто не сравнится с Oracle Reports (подробно об этом вы узнаете в главе 10).

Хотя ниже вы узнаете, как использовать несколько команд SQL\*Plus, перечисленных в табл. 5.9, интерактивно, запуская отдельные команды в командной строке SQL\*Plus, отчеты выглядят лучше, если создать полный файл сценария, запустить его и записать результаты в буферный файл. Таким образом, отдельные запросы SQL\*Plus не будут отображаться в отчете — выводятся только результаты. Поэкспериментировав с командами форматирования SQL\*Plus, вы создадите отчет, набрав сначала все команды форматирования страниц, строк и столбцов. После этого кода будет идти запрос, предоставляющий полезную нагрузку — строки из одной или нескольких таблиц. Итак, прежде всего мы рассмотрим настройку нескольких критических переменных среды SQL\*Plus.

**ТАБЛИЦА 5.9.** Избранные команды SQL\*Plus

<b>Команда</b>	<b>Описание</b>
@ путь	Запускает сценарий SQL (то же, что и START)
ACCEPT	Считывает вводимую строчную константу и записывает ее в данной подстановочной переменной
BREAK	Задает, где и как изменится форматирование
BTITLE	Размещает и форматирует заголовок внизу каждой страницы
CLEAR	Очищает экран SQL*Plus, буфер, заголовки и т.д.
COLUMN	Изменяет характеристики отображения столбца на экране
COMPUTE	Рассчитывает и отображает суммы
EXIT [n]	Фиксирует конец сеанса и завершает работу (n — код ошибки)
LINESIZE	Задает максимальное число символов в одной строке выхода
PAGESIZE	Устанавливает максимальное число строк на странице экрана. В том числе считаются строка заголовка, строка-подчеркивание и пустая строка в конце страницы
PROMPT	Отправляет на экран заданное сообщение или пустую строку
SET	Отображает на экране или изменяет настройки SQL*Plus
SHOW	Перечисляет значения системных переменных
SPOOL имя_файла	Записывает результаты запроса в файл
SPOOL OFF	Отключает запись результатов в файл
SPOOL OUT	Отправляет результаты на принтер
TTITLE	Определяет заголовок страницы
UNDERLINE	Задает символ, используемый в качестве символа подчеркивания. По умолчанию это дефис (-)

**Настройка переменных среды SQL\*Plus**

SQL\*Plus имеет несколько переменных среды, которым можно присваивать различные значения. *Переменная среды*, также называемая *системной переменной SQL\*Plus*, является внутренним значением, с помощью которого SQL\*Plus определяет, как различные элементы выглядят при отображении на экране. Одной из переменных среды, например, является запрос командной строки SQL\*Plus. Присвоив переменной сре-

ды SQLPROMPT строку символов, отличную от принятой по умолчанию, вы можете заменить запрос командной строки с “SQL>” любым удобным для вас значением. Например, чтобы в качестве запроса командной строки отображалось слово “Oracle”, необходимо использовать следующую команду:

```
SET SQLPROMPT Oracle>
```

Из других важных переменных среды можно назвать UNDERLINE, определяющую символ, с помощью которого на экране заголовок столбца отделяется от значений столбца (обычно это дефис). Кроме того, важны описанные в табл. 5.9 переменные LINESIZE и PAGESIZE. Разумеется, любую переменную среды SQL\*Plus вы можете записать прописными или строчными буквами (плюс любая их комбинация), однако мы будем представлять их только прописными, чтобы отличать от имен объектов базы данных. Чтобы перечислить все переменные среды SQL\*Plus, запустите SQL\*Plus и введите следующую команду:

```
SHOW ALL
```

Команда SHOW ALL выводит на экран алфавитный список имен переменных и их текущих значений. Например, значение по умолчанию переменной LINESIZE равно 80 (максимум 80 символов на строку); обычное значение PAGESIZE равно 14.

Как правило, для создания отчета результат выполнения запроса направляется в буферный файл. Следовательно, чтобы это стало возможным, значение переменной среды SPOOL изменяется на ON. Кроме этого, обычно изменяется и значение переменной среды FEEDBACK — на OFF, чтобы генерируемый отчет не засорялся сообщениями, подобными “29 rows selected”. В данном разделе мы подробно расскажем о настройке и использовании различных переменных среды, включая BTITLE, COLUMN, TTITLE и т.д. Пожалуй, главное, что вы должны помнить о переменных среды, — это то, что значения по умолчанию или значения, заданные вами явно, сохраняются на время текущего сеанса SQL\*Plus. Как только вы завершаете сеанс, SQL\*Plus возвращается к значениям переменных среды по умолчанию (эти значения хранятся в специальном файле). Таким образом, когда вы запускаете в одной строке несколько сценариев, генерирующих отчеты, помните, что каждый сценарий независимо от других должен заново задать значения переменных среды. Другими словами, каждый файл сценария для генерации отчета должен начинаться с серии команд SQL\*Plus, устанавливающих требуемые значения переменных среды.

## Форматирование столбцов

Команда SQL\*Plus COLUMN устанавливает заголовок столбца и формат данных, появляющихся под ним. Если вы забудете синтаксис и допустимые настройки, введите HELP COLUMN в командной строке SQL\*Plus, чтобы отобразить на экране список ключевых слов, которые можно использовать с командой COLUMN. Общий ее синтаксис выглядит так:

```
COLUMN [column-name|column-alias] [options]
```

Аргумент <column-name> или <column-alias> определяет столбец или псевдоним из запроса SELECT, к которому применяется данная настройка COLUMN. Если выражение SELECT предоставляет псевдоним для столбца или выражения в списке SELECT, то вы должны использовать этот псевдоним, а не имя столбца. Аргумент <options> может иметь следующие значения: CLEAR (стирает настройки для указанного столбца); FORMAT (предоставляет явное форматирование символов или чисел); а также ON или OFF (включают или отключают настройки столбца). Кроме того, допускается использование опций WRAPPED, WORD WRAPPED, JUSTIFY и NULL. Например, приведенная ниже команда COLUMN задает форматирование для столбца FirstName таблицы Agents.

```
COLUMN LastName FORMAT A15 HEADING "Last Name"
```

Согласно этой команде выражение SELECT всегда будет отображать значения LastName в столбце шириной 15 символов и заголовком Last Name. Ниже показано, как можно задать формат для столбца AskingPrice таблицы Listings.

```
COLUMN AskingPrice FORMAT $9,999,999 HEADING "Asking|Price"
```

В приведенной команде COLUMN задается внешний вид значений столбца AskingPrice: первым символом идет знак доллара, разряды разделяются запятыми, а заголовок Asking Price пишется в две строки. Если вам требуется временно отключить форматирование, заданное для столбца AskingPrice, введите следующую команду:

```
COLUMN AskingPrice OFF
```

Чтобы вновь включить форматирование, введите команду

```
COLUMN AskingPrice ON
```

Если вы желаете изучить на экране форматирование, заданное для столбца, введите следующую команду:

```
COLUMN <column-name>
```

Чтобы совсем удалить форматирование столбца, потребуется такая команда:

```
COLUMN <column-name> CLEAR
```

Ну что ж, давайте попробуем ввести несколько команд форматирования. Прежде всего (как обычно) запустите SQL\*Plus и войдите в Oracle.

Для того чтобы установить формат отображения столбцов на экране, выполните следующие действия.

1. Откройте программу Блокнот, сохраните новый файл как <имя>Ch05Report1.sql, а затем введите приведенные ниже команды SQL\*Plus. Скопируйте их в SQL\*Plus и нажмите клавишу <Enter> после последней строки, чтобы выполнить их.

```
CLEAR SCREEN
SET FEEDBACK OFF
CLEAR COLUMNS
SET LINESIZE 60
```

```
SET PAGESIZE 40
SET UNDERLINE =
COLUMN Address FORMAT A25 HEADING "Address" TRUNCATED
COLUMN City FORMAT A10 HEADING "City"
COLUMN SqFt FORMAT 9,999 HEADING "Square|Feet"
COLUMN YearBuilt FORMAT 9999 HEADING "Year|Built"
```

2. В шести первых строках задайте переменные среды SQL\*Plus, очищающие экран, удаляющие форматирование столбца (возможно, установленное ранее), задающие размер строки, длину страницы и новый символ подчеркивания, а также подавляющие вывод на экран информации о числе строк, выбранных с помощью команд SELECT.
3. Отобразите на экране выбранные столбцы из таблицы Properties — столбцы, форматы которых были определены в п. 1. В текстовом файле, открытом в Блокноте, введите следующий код, скопируйте его SQL\*Plus и выполните:

```
SELECT Address, City, Sqft, YearBuilt
FROM Properties
WHERE City = 'Loleta'
ORDER BY 2,4 DESC;
```

4. Возможно, вам придется прокрутить окно SQL\*Plus, чтобы увидеть все строки, возвращенные в ответ на запрос, однако вы должны получить такой же результат, как показан на рис. 5.31.

5. Выведите на экран настройки столбца Address, выполнив следующую команду:

```
COLUMN Address
```

SQL\*Plus отобразит строки характеристик форматирования столбца Address.

6. Верните всем переменным среды значения по умолчанию (просто, чтобы попрактиковаться) и удалите форматирование столбцов, набрав в Блокноте следующие строки, а затем скопировав их в SQL\*Plus и выполнив:

```
CLEAR COLUMNS
SET LINESIZE 80
SET PAGESIZE 14
SET UNDERLINE '-'
SET FEEDBACK ON
```

**Совет.** Обратите внимание на апострофы вокруг дефиса в четвертой строке. В SQL\*Plus дефис также является символом продолжения, поэтому его необходимо взять в одинарные кавычки, чтобы SQL\*Plus правильно интерпретировал его значение.

7. Введите exit и нажмите клавишу <Enter>, чтобы завершить сеанс SQL\*Plus и выйти из Oracle.
8. Переключитесь на Блокнот, выберите File (Файл), а затем Save (Сохранить), чтобы сохранить последние модификации. Закройте программу Блокнот.

The screenshot shows the Oracle SQL\*Plus interface. In the top-left corner, there's a toolbar with icons for File, Edit, Search, Options, and Help. Below the toolbar, the command line displays several SQL commands:

```

SQL>SET FEEDBACK OFF
SQL>CLEAR COLUMNS
column cleared
SQL>SET LINESIZE 60
SQL>SET PAGESIZE 40
SQL>SET UNDERLINE =
SQL>COLUMN Address FORMAT A25 HEADING "Address" TRUNCATED
SQL>COLUMN City FORMAT A10 HEADING "City"
SQL>COLUMN Sqft FORMAT 9,999 HEADING "Square|Feet"
SQL>COLUMN YearBuilt FORMAT 9999 HEADING "Year|Built"
SQL>SELECT Address, City, Sqft, YearBuilt
  2  FROM Properties
  3 WHERE City = 'Loleta'
  4 ORDER BY 2,4 DESC;
  
```

Below the commands, the output is displayed in a tabular format:

Address	City	Square Feet	Year Built
676 Scenic Dr	Loleta	1,349	1991
552 Pershing	Loleta	2,060	1991
632 Pershing	Loleta	2,552	1989
794 Eel River Rd	Loleta	1,908	1987
229 Church	Loleta	1,764	1987
3840 Tompkins Hill Rd	Loleta	1,912	1985
500 Echo Ln	Loleta	2,785	1985
460 Singley Hill Rd	Loleta	965	1983
281 Table Bluff Rd	Loleta	1,857	1982
51 Loleta Dr	Loleta	1,328	1981
6121 Tompkins Hill Rd	Loleta	1,877	1981
166 Hay Rake Ln	Loleta	1,695	1981
51 Clough Rd	Loleta	1,440	1978
48 Table Bluff Rd	Loleta	1,831	1978
356 Loleta Dr	Loleta	1,714	1977
4363 Tompkins Hill Rd	Loleta	1,695	1977
296 Spring	Loleta	1,640	1976
12 Table Bluff Rd	Loleta	1,640	1975
1107 Singley Hill Rd	Loleta	1,420	1975
321 Hookton Cemetery Rd	Loleta	1,450	1971
3146 Copenhagen Rd	Loleta	1,918	1968
700 Singley Hill Rd	Loleta	1,360	1966
119 Bay View Dr	Loleta	1,941	1957
840 Perrott Ave	Loleta	1,138	1955

At the bottom left of the window, there are scroll bars and a status bar.

Рис. 5.31. Форматирование столбцов

Поскольку вы вышли из SQL\*Plus, данный интерфейс обновит все переменные среды SQL\*Plus, вернув им значения по умолчанию, которые и будут использованы в следующий раз, когда вы запустите SQL\*Plus. Действия, описанные в п. 4, — это просто напоминание вам, что всем измененным переменным хорошо бы вернуть значения по умолчанию. Таким образом, последующие выражения SQL, нацеленные на работу с другими таблицами, не будут отформатированы новым способом, если вы явно этого не укажете.

## Настройка размеров, заголовка и примечания страницы и отчета

Определив столбцы вашего нового отчета (файл Ch05Report1.sql), вы можете задать для него заголовок и примечание. Как указывалось в табл. 5.9, команды SQL\*Plus TTITLE и BTITLE задают строки, которые появляются соответственно вверху и внизу экрана. Данные команды имеют следующий вид:

```
TTITLE | BTITLE [options [text|variable]...] | [ON|OFF]
```

Чтобы временно отключить любую заданную опцию TTITLE (или BTITLE), воспользуйтесь следующей командой:

```
TTITLE OFF
```

Для того чтобы вновь активизировать настройки, введите `TTITLE ON`. Кроме того, вы можете использовать опции `LEFT`, `CENTER` или `RIGHT`, выравнивающие текстовую информацию по левому краю, по центру и по правому краю соответственно. Опция `FORMAT` применяется к отображаемым данным заданную модель формата. В качестве элементов этой модели применяются те же составляющие, что и для выражения `COLUMN`. Опция `SKIP n` задает число строк, которые нужно пропустить после заголовка перед данными. Текст команды `BTITLE` или `TTITLE` может быть литеральным или представлять собой переменную `SQL*Plus`. В частности, в любой заголовок можно вставить переменные `SQL*Plus`: `SQL.LNO`, `SQL.PNO` и `SQL.USER`. `SQL.LNO` содержит номер текущей строки отчета; `SQL.PNO` — номер текущей страницы; `SQL.USER` — имя пользователя, запустившего отчет. Ниже приведен пример выражения `TTITLE` с несколькими опциями и текстом.

```
TTITLE LEFT SQL.USER CENTER 'Properties Available' -
        RIGHT 'Confidential' SKIP 3
```

Дефис, которым заканчивается первая строка, — это символ продолжения `SQL*Plus` (пробел, за которым следует дефис). Он нужен для того, чтобы длинные выражения `TTITLE` продолжались на второй строке. Без него `SQL*Plus` интерпретирует вторую строку как другую команду. Что задает приведенная команда? В ней указано, что имя пользователя выравнивается по левому краю, а за ним следует центрированный литеральный заголовок. По правому краю той же строки выравнивается строка `Confidential`. Наконец, действие выражения `SKIP 3` аналогично троекратному нажатию клавиши `<Enter>`. Первая команда `SKIP` указывает продолжить печать на следующей строке. После этого `SQL*Plus` вставляет две пустые строки под заголовком. Чтобы отключить настройки `TTITLE` (или `BTITLE`), необходимо выполнить следующую команду:

```
TTITLE OFF
```

Чтобы обновить верхний или нижний колонтитул до значения по умолчанию, после `BTITLE` или `TTITLE` можно задать пустую строку (''):

```
TTITLE ''
```

Ниже приведено выражение `BTITLE`, которое помещает внизу каждой страницы (определенной переменной `PAGESIZE`) центрированный номер текущей страницы.

```
BTITLE CENTER 'Page ' FORMAT 999 SQL.PNO
```

Когда число строк достаточно для достижения значения `PAGESIZE`, `SQL*Plus` переносит заголовок `BTITLE` на следующую страницу согласно спецификации (при этом требуется вставить новый заголовок `TTITLE`). В предыдущем примере `BTITLE` задает, что слово “`Page`”, после которого следует пробел и трехзначный номер страницы, центрируется в нижней строке каждой страницы. `SQL*Plus` определяет центр строки на основе значения переменной `INESIZE`. Если значение `INESIZE` равно 60, то номер страницы центрируется по позиции 30.

Ниже мы модифицируем файл Ch05Report1.sql, включив в него три изученных элемента. Запустите SQL\*Plus, войдите в Oracle и откройте в Блокноте текстовый файл <имя>Ch05Report1.sql. После этого модифицируйте его описанным ниже образом.

Для того чтобы модифицировать файл сценария, генерирующий отчет, выполните следующие действия.

1. В Блокноте добавьте под выражением SET FEEDBACK OFF три команды:

```
TTITLE LEFT 'Properties' CENTER 'Redwood Realty' -
RIGHT 'Confidential' SKIP 2
BTITLE CENTER 'Page ' FORMAT 99 SQL.PNO
BREAK ON City SKIP PAGE
```

**Совет.** Команда BREAK сообщает SQL\*Plus, что следует перейти на следующую страницу (“разбавить” вывод нижним и верхним колонтитулами) при изменении значения в столбце City. Разумеется, чтобы данная команда работала должным образом, вам необходимо отсортировать выход по значениям столбца, фигурирующего в команде BREAK (в данном случае – City).

2. Следующим образом отредактируйте строку, содержащую переменную PAGESIZE:  
PAGESIZE 22
3. Модифицируйте оператор WHERE запроса SELECT, включив в него г. Орик (Orick), как показано ниже.

```
WHERE City IN ('Loleta', 'Orick')
```

4. Удалите все строки, следующие после запроса SELECT, начиная с “Column Address” и заканчивая “SET FEEDBACK ON”. В итоге модифицированный файл сценария должен выглядеть так, как показано на рис. 5.32.
5. Запишите модифицированный файл сценария под именем <имя>Ch05Report2.sql.
6. Скопируйте модифицированный файл сценария целиком в SQL\*Plus, нажмите клавишу <Enter>, чтобы выполнить его, и изучите результаты. Две последние страницы отчета показаны на рис. 5.33. Обратите внимание на то, что название города не повторяется, а информация по г. Орику начинается с новой страницы.
7. Введите exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и закрыть SQL\*Plus. Данные действия также возвращают переменным среды их значения по умолчанию, переопределевшиеся в ходе выполнения последних упражнений.

## Создание HTML-отчетов с помощью SQL\*Plus

SQL\*Plus позволяет распространять отчеты через Web, предлагая специальную команду, выдающую код с HTML-форматированием. Чтобы создать отчет в формате Web, необходимо отправить его в буферный файл, а затем отобразить в Web-браузере

```

MyNameCh05Report2.sql - Notepad
File Edit Format View Help
CLEAR SCREEN
SET FEEDBACK OFF
TTITLE LEFT 'Properties' CENTER 'Redwood Realty' -
RIGHT 'Confidential' SKIP 2
BTITLE SKIP 2 CENTER 'Page ' FORMAT 99 SQL.PNO
BREAK ON City SKIP PAGE
CLEAR COLUMNS
SET LINESIZE 60
SET PAGESIZE 22
SET UNDERLINE =
COLUMN Address FORMAT A25 HEADING "Address" TRUNCATED
COLUMN City FORMAT A10 HEADING "City"
COLUMN SqFt FORMAT 9,999 HEADING "Square|Feet"
COLUMN YearBuilt FORMAT 9999 HEADING "Year|Built"
SELECT Address, City, Sqft, YearBuilt
FROM Properties
WHERE City IN ('Loleta', 'Orick')
ORDER BY 2,4 DESC;

```

**Рис. 5.32.** Модифицированный файл сценария, генерирующий отчет

или загрузить на Web-сайт. HTML-отчеты лучше всего генерировать, запуская файл сценария. Если вы создаете отчет интерактивно, то SQL\*Plus применяет к результату выполнения сценария команду `SELECT`, и вы получаете отчет. Команда SQL\*Plus, выполняющая HTML-форматирование, выглядит следующим образом:

```
SET MARKUP HTML ON SPOOL ON
```

Данную команду необходимо внедрить в файл сценария или выполнить интерактивно. Команда `MARKUP` просто задает, что выход SQL\*Plus должен быть HTML-кодированным. Фрагмент `SPOOL ON` команды указывает SQL\*Plus записать в буферный файл HTML-дескрипторы `<html>`, `<head>` и `<body>` перед записью данных, генерируемых запросом. Команда `MARKUP` в действительности не пишет в буферный файл HTML-код — за это отвечает команда `SPOOL <filename>`.

Закрыв буферный файл с помощью команды `SQL*Plus SPOOL OFF`, вы сообщаете `SQL*Plus` приостановить генерацию HTML-кода, выполнив следующую команду:

```
SET MARKUP HTML OFF
```

Если вы забудете отключить генерацию HTML-кода, то увидите на экране вместо обычного выхода `SQL*Plus` ряд HTML-команд для работы с таблицами (`<td>`, `<tr>` и т.д.). Чтобы сделать Web-отчеты еще более интересными, вы можете ввести в файл собственный HTML-код, однако делать это не обязательно. Команда `SQL*Plus MARKUP` выдает результаты в виде достаточно привлекательных Web-страниц.

Приступая к самостоятельной генерации отчетов, помните о нескольких вещах. Во-первых, Web-страницы представляют собой бесконечно длинные окна, поэтому вам следует присвоить переменной `SQL*Plus PAGESIZE` значение, большее предполагаемого числа выходных строк. Заголовки, которые появляются в середине Web-страницы, совсем не привлекательны. Во-вторых, переменной `FEEDBACK` следует присвоить значение `OFF`, поскольку в Web-отчете вам совсем не нужны сообщения,

The screenshot shows the Oracle SQL\*Plus interface with three distinct sections of output:

- Page 1:** Displays a table titled "Properties" with columns "Address", "City", "Square Feet", and "Year Built". The data includes rows for various addresses like "48 Table Bluff Rd" and "356 Loleta Dr".
- Page 2:** Displays a table titled "Properties" with columns "Address", "City", "Square Feet", and "Year Built". The data includes rows for various addresses like "300 Robinson Rd" and "3321 Old State Hwy".
- Page 3:** Shows the SQL prompt "SQL>" followed by a blank line.

Рис. 5.33. Модифицированные результаты отчета

подобные “354 rows selected”. Наконец, не забудьте вернуть всем переменным SQL\*Plus исходные значения. Таким образом, продолжая интерактивно использовать SQL\*Plus, вы не получите неприятных сюрпризов.

Далее вы напишете небольшой файл сценария SQL\*Plus, запишете его, а затем выполните, чтобы получить Web-страницу. Данный файл сценария будет перечислять клиентов (*Customers*) из любого города, выбранного пользователем. Несколько столбцов из таблицы *Customers* будут заполнять созданный вами файл в формате Web — *CustomerReport.html*. Наконец, вы запустите Web-браузер, чтобы посмотреть генерированную Web-страницу.

Итак, чтобы создать и записать файл сценария, генерирующий HTML-отчет, выполните следующие действия.

1. Откройте программу Блокнот и сразу же запишите пустой файл как <имя>Ch05WebReport.sql в любой удобной для вас папке, подставив вместо <имя> свое имя.
2. В Блокноте введите нижеприведенный код, чтобы отобразить на экране доступные имена городов, задать значения нескольких переменных среды и запросить у

пользователя имя города, которое будет использоваться в качестве критерия отбора в Web-отчете. (Не забудьте вставить пробел после двоеточия в строке ACCEPT.)

```
SET FEEDBACK OFF
CLEAR SCREEN
SELECT DISTINCT City "City Names" FROM Customers ORDER BY City;
PROMPT
ACCEPT cityname PROMPT 'List Customer rows for city: '
```

3. Ниже данной команды добавьте следующие строки:

```
SET TERMOUT OFFSET VERIFY OFF
SET PAGESIZE 50000
SET MARKUP HTML ON SPOOL ON
SPOOL C:\CustomerReport.html
```

4. Сразу после последней строки запишите выражение SELECT, которое извлечет затребованные строки, отфильтровав их с использованием подстановочной переменной “&cityname”:

```
SELECT FirstName, LastName, Address, City,
       HomePhone, CellPhone, WorkPhone
  FROM Customers
 WHERE UPPER(City) = UPPER('&cityname')
 ORDER BY 2,1;
```

5. Сразу после последней строки введите следующий код, чтобы отключить вывод в буферный файл и вернуть переменным среды SQL\*Plus значения по умолчанию:

```
SPOOL OFF
REM Обновить переменные SQL*Plus
SET MARKUP HTML OFF
SET PAGESIZE 14
SET VERIFY ON
SET FEEDBACK ON
SET TERMOUT ON
```

6. Запишите завершенный файл сценария, выбрав в строке меню File (Файл), а затем Save (Сохранить).

7. Выберите File (Файл), а затем Exit (Выход), чтобы выйти из программы Блокнот.

Теперь вам нужно запустить файл сценария и загрузить Web-страницу, чтобы изучить результаты.

Для того чтобы запустить файл сценария в SQL\*Plus, а затем просмотреть результаты в Web-браузере, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle, а затем выполните созданный выше файл сценария, набрав и выполнив приведенную ниже команду. Не забудьте подставить полный путь к вашему файлу сценария вместо <путь> и свое имя вместо <имя>.

```
START <путь><имя>Ch05WebReport
```

The screenshot shows the Oracle SQL\*Plus interface. The title bar reads "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following text:

```

City Names
-----
Arcata
Blue Lake
Eureka
Fortuna
Loleta
McKinleyville
Orick
Trinidad
Westhaven

List Customer rows for city: orick
SQL>

```

**Рис. 5.34.** Названия городов и запрос, отображаемый файлом сценария

2. Введите `orick` и нажмите клавишу <Enter>, чтобы ответить на запрос. В результате вы должны увидеть то же, что показано на рис. 5.34: список городов, запрос и введенное название города.
3. Введите `exit` и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus.
4. Используя свой любимый Web-браузер, найдите Web-страницу `CustomerReport.html` и откройте ее, чтобы просмотреть в Web-формате список клиентов, проживающих в г. Орике. Верхняя часть этой страницы показана на рис. 5.35.
5. После того как вы закончите изучать отчет, закройте Web-браузер. Если хотите, после этого можете удалить файл `CustomerReport.html`.

## Резюме

Выражение SQL SELECT запрашивает таблицы и возвращает строки. Вам необходимо задать таблицу (таблицы), из которой извлекаются строки, и необязательные операторы. Например, критерий поиска в необязательном операторе `WHERE` определяет, какие строки будут представлены в результате. Оператор `GROUP BY` формирует группу из извлекаемых строк; оператор `HAVING` фильтрует группы, а `ORDER BY` сортирует строки перед передачей их пользователю. Чтобы получить уникальные строки, используйте перед списком отбора ключевое слово `DISTINCT`.

Условия `WHERE` состоят из операторов сравнения, имен столбцов, выражений и литералов, комбинация которых формирует логическое выражение. Когда значение логического выражения равно `true`, строка таблицы возвращается; в противном случае — нет. SQL предлагает несколько операторов SQL, включая `BETWEEN ... AND`, `IN`, `IS NULL` и `LIKE`, позволяющих выполнять поиск в диапазонах и дискретных списках значений, определять, равно ли значение `NONE`, или использовать групповые символы соответственно.

FIRSTNAME	LASTNAME	ADDRESS	CITY	HOMEPHONE	CELLPHONE	WORKPHONE
Joan	Anyon	3321 Old State Hwy	Oriick	(707) 555-4455	(707) 555-6932	
Lon	Battles	Benson Road	Oriick	(707) 555-3220	(707) 555-9052	
Marcia	Blom	100 Gando Drive	Oriick	(707) 555-6847	(707) 555-8366	
Rick	Boteilho	525 North Brand Boulevard	Oriick	(707) 555-2173	(707) 555-0334	(707) 555-5068
Rich	Brewington	555 River Oaks Parkway	Oriick	(707) 555-6159	(707) 555-4739	(707) 555-3300
Sally	Brotzman	4900 West 78Th Street	Oriick	(707) 555-6332	(707) 555-2216	
Patrick	Cole	1700 Lincoln Street	Oriick	(707) 555-3113	(707) 555-7970	
Mark	Dvorak	105 Reynolds Drive	Oriick	(707) 555-8108	(707) 555-5025	(707) 555-2540
Mike	East	550 California Street	Oriick	(707) 555-9627	(707) 555-6653	
Margie	Edgley	225 High Ridge Road	Oriick	(707) 555-0461	(707) 555-1045	
Philip	Etter	2027 Harpers Way	Oriick	(707) 555-4770	(707) 555-1234	
Teresa	Farah	3200 San Fernando Road	Oriick	(707) 555-9475	(707) 555-9155	
William	Gass	700 East Middlefield Road	Oriick	(707) 555-9335	(707) 555-1588	
Michael	Gibson	380 Middlesex Avenue	Oriick	(707) 555-0487	(707) 555-9351	(707) 555-7500
Mike	Haithcoat	10201 Torre Avenue	Oriick	(707) 555-6379	(707) 555-3834	
Virginia	Hunt	16071 Mojave Drive	Oriick	(707) 555-1183	(707) 555-4965	
Mohamud	Hutchinson	2600 West Magnolia Boulevard	Oriick	(707) 555-4277	(707) 555-4899	
Randy	Johnson	4101 Jerome Avenue	Oriick	(707) 555-7869	(707) 555-8615	
Mark	Kasky	300 Nyala Farms	Oriick	(707) 555-6632	(707) 555-6791	
Ed	Kieman	1100 H Street Northwest	Oriick	(707) 555-2726	(707) 555-5097	
Michael	Krause	Brandywine Corporate Cent	Oriick	(707) 555-6939	(707) 555-9687	
Vicki	Lafferty	One Rollins Plaza	Oriick	(707) 555-7411	(707) 555-6655	(707) 555-6137
Marilyn	Lee	65 Locust Avenue	Oriick	(707) 555-3730	(707) 555-2691	
Jeff	Leung	1050 Caribbean Way	Oriick	(707) 555-4483	(707) 555-4344	
Lynn	Lewis	50 Locust Avenue	Oriick	(707) 555-7769	(707) 555-3994	(707) 555-2901
Teresa	Manning	4343 Von Karman Avenue	Oriick	(707) 555-0806	(707) 555-6731	(707) 555-2888
Roger	McCallister	866 West Peachtree Street	Oriick	(707) 555-8388	(707) 555-4970	
John	Mclean	50 Lundblade	Oriick	(707) 555-0622	(707) 555-9925	
Paul	Medovic	275 South Hillview Drive	Oriick	(707) 555-3107	(707) 555-3744	
April	Miller	810 North Belcher Road	Oriick	(707) 555-1067	(707) 555-7359	

Рис. 5.35. Web-страница, сгенерированная файлом сценария

Необязательный оператор ORDER BY, являющийся последней фразой выражения SELECT, сортирует строки по одному или нескольким значениям или выражениям, разделенным запятыми (в убывающем или возрастающем порядке для каждого значения или выражения). Оператор ORDER BY позволяет обращаться к столбцам или выражениям, отсутствующим в списке отбора. При использовании совместно с внутренним представлением оператор ORDER BY позволяет получить данные типа “*n* верхних” — упорядоченного перечня *n* верхних или нижних позиций списка.

SQL поддерживает полный набор математических операторов для выполнения расчетов в списке отбора или вспомогательных выражениях оператора WHERE. В их число входят сложение, вычитание, умножение и деление. Функции SQL предлагают множество встроенных программ, позволяющих выполнять расчеты с символьными, числовыми значениями и датами. Функции преобразования SQL переводят данные из одного типа в другой. Специальные функции, подобные NVL, помогают, например, определить, содержит ли столбец значение NULL, и если да, то вернуть вместо него другое значение.

Агрегирующие функции позволяют рассчитывать резюмирующую информацию по группам строк, включая возможность суммирования и подсчета числа

значений в сгруппированных строках, а также расчет среднего, минимального или максимального значения.

Наконец, SQL\*Plus поддерживает генерацию HTML-кода, подходящего для просмотра в Web-браузере. Генерация HTML-кода инициируется с помощью команды MARKUP HTML ON SPOOL ON. Затем, присвоив нескольким переменным среды SQL\*Plus соответствующие значения и использовав команду SPOOL, вы получаете удобный для восприятия документ в HTML-формате, который можно открывать с помощью браузера. Таким образом, в результате запроса получается отформатированная таблица с равномерными столбцами и стандартными цветами фона и текста.

## Основные термины

- Агрегирующая функция
- ASC / DESC
- Звездочка (\*)
- Оператор BETWEEN
- Символьная функция
- Псевдоним столбца
- Конкатенация
- Функция преобразования
- Функция даты
- Переменная среды
- Модель формата
- Функция
- Оператор GROUP BY
- Групповая функция
- Оператор HAVING
- Оператор IN
- Внутреннее представление
- Оператор LIKE
- Числовая функция
- Приоритет операторов
- Оператор ORDER BY
- Функция с регулярными выражениями
- Команды форматирования отчетов
- Условие поиска

- Список отбора
- Однострочная функция
- Список таблиц
- Оператор WHERE
- Групповой символ

## Повторение пройденного материала

### Истина или ложь?

1. Выражение SELECT должно содержать оператор FROM. Все остальные операторы являются необязательными.
2. Если вам нужны строки таблицы Agent, соответствующие агентам, рожденным в определенном году, вам удобно использовать оператор IN в команде WHERE, чтобы ограничить число возвращаемых строк.
3. Если вам требуется отсортировать строки на основе выражения, вы должны использовать псевдоним столбца. Другими словами, в операторе ORDER BY вы не можете использовать выражения.
4. Агрегирующие функции, подобные AVG, игнорируют в своих расчетах значения NULL. Например, среднее пяти значений, одно из которых равно NULL, будет равно сумме четырех значимых величин, деленной на четыре.
5. По крайней мере один из элементов списка выбора должен представлять собой имя столбца или не агрегирующую функцию, поскольку вы не можете записать выражение SELECT, список отбора которого состоит исключительно из агрегирующих функций.

### Заполнить пропущенное

1. Оператор \_\_\_\_\_ позволяет выполнять фильтрацию строк для выражений SELECT перед формированием групп.
2. Символ % в выражении, подобно LIKE '%month', называется \_\_\_\_\_ символом.
3. Чтобы определить порядок вычисления подвыражений в выражении, применяется концепция \_\_\_\_\_.
4. Чтобы сгенерировать список из десяти самых дорогих домов, после ключевого слова FROM необходимо поставить \_\_\_\_\_ \_\_\_\_\_ (два слова).
5. Чтобы рассчитать чей-то возраст на основе столбца с указанной датой рождения, можно с помощью функции \_\_\_\_\_ рассчитать число дней, прошедших с этого момента, и уже по ним ответить на поставленный вопрос.

## Варианты ответов

1. Если в списке отбора указано City, MAX(BidPrice), какую еще фразу должно содержать выражение SELECT?
  - a) ORDER BY.
  - б) DISTINCT.
  - в) GROUP BY.
  - г) HAVING.
2. Какое ключевое слово используется для отображения на экране уникальных названий городов в выражении SELECT City FROM Properties;?
  - a) GROUP BY.
  - б) UNIQUE.
  - в) NVL.
  - г) DISTINCT.
3. Как называется объединение двух строк символов в списке отбора, подобное SELECT FirstName || LastName FROM Agents;?
  - а) Агрегация.
  - б) Конкатенация.
  - в) Преобразование.
  - г) Форматирование.
4. Что вы используете в функции TO\_DATE, чтобы указать точную структуру даты?
  - а) Строку синтаксиса.
  - б) Модель формата.
  - в) Константу даты.
  - г) Системную переменную.
5. Значение какой переменной среды SQL\*Plus необходимо переопределить, чтобы при выполнении выражения SELECT подавить вывод на экран сообщений, подобных “15 rows selected”?
  - а) SQLPROMPT.
  - б) PAGESIZE.
  - в) LINESIZE.
  - г) FEEDBACK.

## Упражнения

### 1. Readwood Realty

Роберт Стирлинг попросил вас организовать запрос к нескольким таблицам базы данных Redwood Realty и сгенерировать несколько отчетов. Ему требуется возможность запуска отчетов в тот момент, когда потребуется информация о делах Redwood Realty. Поэтому он просит вас создать файлы сценариев SQL, которые он сможет запускать с помощью команды START <scriptfilename>. Некоторые запросы будут давать файлы данных, другие — выводить результаты на экран, а один отчет будет представлять собой HTML-документ, который Стирлинг откроет в своем браузере. Вы поговорили с Робертом, и он конкретизировал свои требования.

1. Для начала вам необходимо запустить SQL\*Plus и войти в Oracle, используя свое имя пользователя и пароль. Далее запустите Блокнот (или любой другой текстовый процессор), чтобы вы могли разрабатывать и записывать код в файлах.
2. Рассчитайте минимальное, среднее и максимальное значения AskingPrice в таблице Listings и подсчитайте число записей в ней. Сгруппируйте результаты по AgentID. Введите следующий код в Блокноте и запишите полученный текстовый файл как <имя>Ch05Prob1Script1.sql. Затем скопируйте этот текстовый файл в SQL\*Plus и выполните его. Вместо <Your full name here> введите свои имя и фамилию.

```
REM <Ваше имя полностью>
CLEAR COLUMNS
SET PAGESIZE 35
SET FEEDBACK OFF
CLEAR SCREEN
COLUMN Stat1 FORMAT $9,999,999 HEADING
'Minimum'
COLUMN Stat2 FORMAT $9,999,999 HEADING
'Average'
COLUMN Stat3 FORMAT $9,999,999 HEADING
'Maximum'
COLUMN Stat4 FORMAT 999 HEADING 'Count'
SELECT ListingAgentID "Agent ID",
       MIN(AskingPrice) AS "Stat1",
       AVG(AskingPrice) AS "Stat2",
       MAX(AskingPrice) AS "Stat3",
       COUNT(AskingPrice) AS "Stat4"
  FROM Listings
 GROUP BY ListingAgentID
 HAVING COUNT(AskingPrice) > 18
 ORDER BY "Stat4" DESC;
SET FEEDBACK ON
```

3. Переключитесь на программу Блокнот, введите нижеприведенный код, скопируйте его в SQL\*Plus и выполните. Не забудьте подставить путь к буферному файлу вместо <путь> (шестая строка) и свое имя вместо <имя> (там же).

```

CLEAR COLUMNS
SET PAGESIZE 35
CLEAR SCREEN
COLUMN "Name" FORMAT A25 HEADING
  'Agent''s Name'
COLUMN "Age" FORMAT A10 HEADING
  'Age at|Hire Date'
SPOOL C:\<путь>\<имя>Ch05Prob1Part1.txt
SHOW USER
SELECT LastName||', '|| FirstName AS "Name",
       LPAD(TO_CHAR((HireDate-BirthDate)/365.25,
      '99'),7) AS "Age"
FROM Agents
WHERE (HireDate-BirthDate)/365.25 > 30.00
ORDER BY 2 DESC;
SPOOL OFF
SET PAGESIZE 14

```

- Найдите буферный файл <имя>Ch05Prob1Part1.txt и распечатайте его.
- Еще раз запишите файл, открытый в Блокноте. Откройте новый (пустой) текстовый файл. Введите нижеприведенный код. В восьмой строке подставьте вместо <путь> подходящий путь к папке, в которой необходимо сохранить HTML-файл. Запишите текстовый файл как <имя>Ch05Prob1Part2.sql, подставив вместо <имя> свое имя. Программу Блокнот не закрывайте.

```

SET TERMOUT OFF
SET FEEDBACK OFF
SET VERIFY OFF
SET ECHO OFF
TTITLE OFF
CLEAR COLUMNS
SET PAGESIZE 50000
SET MARKUP HTML ON SPOOL ON
SPOOL C:\<путь>\Ch05Customers.html
SELECT FirstName, LastName, Address, City
FROM Customers
WHERE UPPER(City) = 'BLUE LAKE'
ORDER BY 2,1;
SPOOL OFF
SET MARKUP HTML OFF
SET PAGESIZE 14
SET ECHO ON
SET VERIFY ON
SET FEEDBACK ON
SET TERMOUT ON

```

- Переключитесь на SQL\*Plus и введите следующий код, подставив вместо <путь> путь к файлу сценария SQL, а вместо <имя> — свое имя, и нажмите клавишу <Enter>, чтобы выполнить файл сценария:

```

START
C:\<путь>\<имя>Ch05Prob1Part2.sql

```

7. Выходите из SQL\*Plus (введите `exit` и нажмите клавишу <Enter>), откройте Web-страницу (`Ch05Customers.html`), сгенерированную в п. 5, напечатайте Web-страницу и закройте браузер. Не забудьте написать на Web-странице свое имя и фамилию, чтобы в случае необходимости предоставить ее Роберту.
8. Запишите текстовый файл в Блокноте, напечатайте его и закройте Блокнот.

## 2. Coffee Merchant

Администратор базы данных попросил, чтобы вы создали два отчета по таблицам Coffee Merchant. Для начала вы решаете попрактиковаться на отчете, рассчитывающем плотность населения каждого штата США из таблицы `States`. Для АБД вы создадите отчет, в котором перечислены сорта кофе, имеющиеся на складе, и общее количество всех товаров на складе (включая чай). Наконец, вы напишете файл сценария, который может запустить АБД, чтобы отобразить каталог на экране в виде Web-страницы. Перед началом найдите файл `BuildCoffee.sql` в папке `CoffeeMerchant`. Данный сценарий создает с нуля базу данных Coffee Merchant. Запомните или запишите путь к нему.

1. Запустите SQL\*Plus, войдите в Oracle, введите в командной строке SQL следующий код, подставив вместо <путь> реальный путь:

```
START <путь>\BuildCoffee
```

2. Откройте Блокнот и введите приведенные ниже выражения. Скопируйте их и выполните в SQL\*Plus. Запишите открытый в Блокноте текстовый файл в папке по своему усмотрению под именем <имя>`Ch05Problem2Part1.sql`. Вместо <имя> подставьте свое имя (то же самое в приведенном коде). Вместо <путь> в третьей строке подставьте полный путь к файлу (включая букву, обозначающую диск).

```
REM Расчет плотности населения во всех штатах
CLEAR SCREEN
SPOOL <путь>\<имя>Ch05Problem2Report1.txt
SHOW USER
SET PAGESIZE 55
CLEAR COLUMNS
COLUMN StateID FORMAT A03
    HEADING 'ID'
COLUMN StateName FORMAT A14
    HEADING 'State'
COLUMN Population FORMAT 99,999,999
    HEADING 'Population'
COLUMN LandArea FORMAT 999,999
    HEADING 'Land Area'
COLUMN Density FORMAT 99,999
    HEADING 'Density'
SELECT StateID, StateName, Population,
    LandArea, Population/LandArea AS Density
FROM States
WHERE UPPER(StateID) <> 'DC'
```

```
ORDER BY Density DESC;
SPOOL OFF
```

3. Введите в Блокноте нижеприведенный код, скопируйте его в SQL\*Plus и выполните. В третьей строке подставьте нужные значения вместо <путь> и <имя>. Приведенный код отображает на экране информацию по 15 штатам с самыми длинными названиями. Обратите внимание на форматирование.

```
REM Отображение на экране 15 штатов с самыми длинными названиями
CLEAR SCREEN
SPOOL <путь>\<имя>Ch05Problem2Report2.txt
SHOW USER
SELECT StateName, "Length"
FROM (SELECT StateName, LENGTH(StateName)
AS "Length"
      FROM STATES
      WHERE UPPER(StateID) <> 'DC'
      ORDER BY "Length" DESC)
WHERE ROWNUM < 16;
SPOOL OFF
CLEAR COLUMNS
SET PAGESIZE 14
```

4. Запишите файл, открытый в программе Блокнот, чтобы сохранить результаты своей работы. Введите в Блокноте приведенный ниже код и скопируйте его в SQL\*Plus. Данный сценарий отображает на экране все продукты, имеющиеся на складе, — вначале кофе, затем чай. С помощью выражений Column выполняется привлекательное форматирование столбцов. Как и ранее, внесите необходимые изменения в третью строку, подставив путь и имя буферного файла.

```
REM Перечень сортов кофе на складе и их общего количества
CLEAR SCREEN
SPOOL <путь>\<имя>Ch05Problem2Report3.txt
SHOW USER
COLUMN Name FORMAT A33
      HEADING 'Coffee Name'
COLUMN OnHand FORMAT 99,999
      HEADING 'In Stock'
COLUMN Price FORMAT $999.99
      HEADING 'Price'
COLUMN TotValue FORMAT $999,999
      HEADING 'Total|Value'
SELECT Name, OnHand, Price, Price*OnHand AS
TotValue
FROM Inventory
WHERE UPPER(ItemType)='C' AND OnHand > 0
ORDER BY 4 DESC;
SPOOL OFF
```

5. Снова запишите текстовый файл, открытый в Блокноте, и создайте новый. Запишите его под именем <имя>Ch05Problem2Part2.sql. Введите в Блокноте приведенный ниже код. В семнадцатой строке измените значения <путь> и <имя>. Набрав весь код, запишите файл и закройте программу Блокнот.

```
SET TERMOUT OFF
SET FEEDBACK OFF
SET ECHO OFF
CLEAR COLUMNS
SET PAGESIZE 234
SET MARKUP HTML ON SPOOL ON
SPOOL <путь>\<имя>Ch05Problem2Part2WebPage.html
SELECT Name, Price, OnHand, Description,
CASE WHEN ItemType='C' THEN 'Coffee'
ELSE 'Tea' END AS "Type"
FROM Inventory
ORDER BY "Type", Name;
SPOOL OFF
SET MARKUP HTML OFF
SET PAGESIZE 14
SET ECHO ON
SET FEEDBACK ON
SET TERMOUT ON
```

6. Переключитесь на SQL\*Plus и выполните созданный файл сценария. Наберите следующий код, подставив вместо <path> путь, а вместо <имя> — свое имя:

```
START <путь>\<имя>Ch05Problem2Part2.sql
```

7. Загрузите Web-страницу, созданную в п. 6, в браузер и напечатайте ее.  
8. Напечатайте все остальные буферные файлы и закройте все программы, включая SQL\*Plus.

### 3. Rowing Ventures

Используя базу данных RowingVentures, создайте отчеты, содержащие информацию о командах, выставленных на заезды с участием команд из восьми гребцов и команд из четырех гребцов. Прежде всего вам необходимо заполнить базу данных, отыскав в папке RowingVentures файл BuildRowing.sql (на компакт-диске в папке, соответствующей главе 5). Запишите названия таблиц Rowing Ventures, отображенные в SQL\*Plus при запуске файла сценария. Если потребуется, выполните команду DESCRIBE, чтобы выяснить, какие столбцы содержит каждая таблица. Создайте описанные ниже отчеты, направив результаты запросов в буферный файл. Напечатайте файл сценария и буферные файлы.

- Создайте отчет <имя>Ch05Problem3Report1.txt, в котором будут перечислены столбцы Nation, OrganizationName, Address, City и State (в указанном порядке) из таблицы Organization. Используйте команду BREAK, чтобы убрать дублирующиеся значения в столбце и отсортировать результаты по этому столбцу. Отформатируйте все столбцы, используя команду COLUMN, чтобы гарантировать, что заголовки столбцов начинаются с прописной буквы (например, “Organization ID” или “Address”) и что ни один из заголовков не длиннее 15 символов.

Максимальную длину страницы SQL\*Plus задайте равной 25 строкам. Подавите вывод сообщений, подобных “10 rows selected”. Добавьте в сценарий выражение TTITLE, включающее заголовок страницы: слева — ваше имя, в центре — “Organizations by Nation”, справа — номер страницы. После команды SPOOL OFF восстановите исходные значения всех переменных среды.

- Создайте отчет <имя>Ch05Problem3Report2.txt, в котором перечисляется содержимое таблицы Person. С помощью конкатенации объедините в одном столбце FirstName и LastName. Разработайте приемлемое форматирование столбцов с заголовками, которые начинаются с прописной буквы. Сделайте так, чтобы ширина столбцов была как можно меньше, но чтобы все значения или метки столбцов помещались полностью в одну строку. (Найдите максимальный размер значений с помощью выражения MAX(LENGTH(<columnname>)) — это и будет ширина столбца.) Задайте заголовок (TTITLE) со своим именем (слева) и названием таблицы (по центру). Отсортируйте выход по возрастанию даты рождения. Отобразите только информацию о людях, родившихся после 1963 года.

#### 4. Broadcloth Clothing

Прежде всего вы должны заполнить базу данных, отыскав в папке Broadcloth файл BuildClothing.sql. Запишите названия таблиц Broadcloth Clothing, отображаемых в SQL\*Plus при запуске файла сценария. Если потребуется, выполните выражение DESCRIBE, чтобы выяснить названия столбцов таблиц. Создайте на основе таблицы Contact Web-страницу. Напечатайте файл сценария, который создает Web-страницу и две последние ее страницы. Затем запустите специальный запрос, чтобы рассчитать определенную статистику. Напечатайте полученный файл сценария и буферный файл. Ниже кратко описаны требуемые отчеты и их результаты.

- Выведите на экран все столбцы таблицы ProductionBatch, отобразив только строки, в поле StartTime которых записано значение, относящееся к августу 2006 года. Отсортируйте возвращаемые строки по убыванию стоимости производства.
- Напишите специальный запрос, отображающий минимальную, среднюю и максимальную цены поставки, а также число записей, относящихся к различным фабрикам, рассортованным по идентификационному номеру фабрики и по возрастанию цены поставки. Отобразите только группы, средняя цена поставки которых меньше 200. Все требуемая информация находится в таблице Shipment. (Если потребуется, выполните команду DESCRIBE Shipment.) Результаты запроса направьте в файл, который вы сможете напечатать после завершения работы.

# ГЛАВА 6

## Создание запросов и представлений на основе нескольких таблиц

**В этой главе...**

- Создание запросов, объединяющих несколько таблиц
- Использование псевдонимов таблиц в запросах к нескольким таблицам
- Создание естественных объединений, внутренних объединений, самообъединений и внешних объединений
- Использование операторов UNION, MINUS и INTERSECT, выполняющих действия над множествами
- Создание скалярных, многострочных и многостолбцовых подзапросов
- Написание коррелированных подзапросов
- Создание однотабличных и сложных представлений
- Обновление, вставка и удаление данных из представлений
- Перечисление определений представлений и удаление представлений

## Создание и использование представлений на основе нескольких таблиц

Одной из самых сильных сторон SQL является возможность связывать данные, записанные в отдельных таблицах. Это позволяет хранить в одной таблице информацию об агентах по недвижимости, в другой — данные о домах на продажу, а в третьей — имена и адреса владельцев. Используя правильно построенный запрос, вы можете объединять всю эту информацию. Запросы, рассмотренные в главе 5, включали работу только с одной таблицей. Из данной главы вы узнаете, как писать запросы, обращающиеся к нескольким таблицам и объединяющие их. *Объединением (join)* называется временная связь, создаваемая между двумя таблицами. Часто для формирования данной связи строки из одной таблицы соотносятся со строками другой на основе общих столбцов, имеющих одинаковое значение. Обычно система баз данных объединяет таблицы путем согласования внешнего ключа одной таблицы с первичным ключом другой, хотя допускаются и другие условия объединения (в том числе и неравенства). Условие, на основе которого Oracle определяет, удовлетворяют ли два значения критерию объединения, называется *условием объединения*. Условие объединения является частью оператора `FROM` или (в более старых системах) входит в оператор `WHERE`. Проверка того, удовлетворяют ли строки из двух таблиц условию объединения, выполняется для каждой пары строк и дает нуль, одну или несколько согласующихся пар. Наиболее распространенным типом условия объединения является поиск в двух таблицах соответствующих значений. Данное условие называется *объединением по эквивалентности (equijoin)*, поскольку система баз данных пытается найти равные (эквивалентные) значения в заданных столбцах обеих таблиц.

Существуют и другие способы объединения таблиц, при которых значения не обязательно должны совпадать. Эти техники объединения не по эквивалентности подробно рассматриваются в данной главе. На рис. 6.1 показан гипотетический пример значений первичного и внешнего ключей из двух таблиц. Столбцы, сканируемые Oracle на предмет поиска соответствий, — `ID` в таблице `Employees` и `Emp#` в таблице `Sales`. Обратите внимание на то, что таблица `Sales` сортируется по значениям `Emp#`, что делает более удобной демонстрацию объединения. На практике же столбец `Emp#` — столбец внешнего ключа, ссылающегося на таблицу `Employees`, — не имеет какого-либо специального упорядочения.

Объединение нескольких (двух или большего числа) таблиц требуется из-за того, что базы данных и таблицы, которые в них содержатся, были до определенной степени нормализованы. Это означает, что различная, но родственная информация хранится в отдельных таблицах. Например, база данных Redwood Realty содержит восемь таблиц с информацией. Одна из этих таблиц содержит информацию об агентах

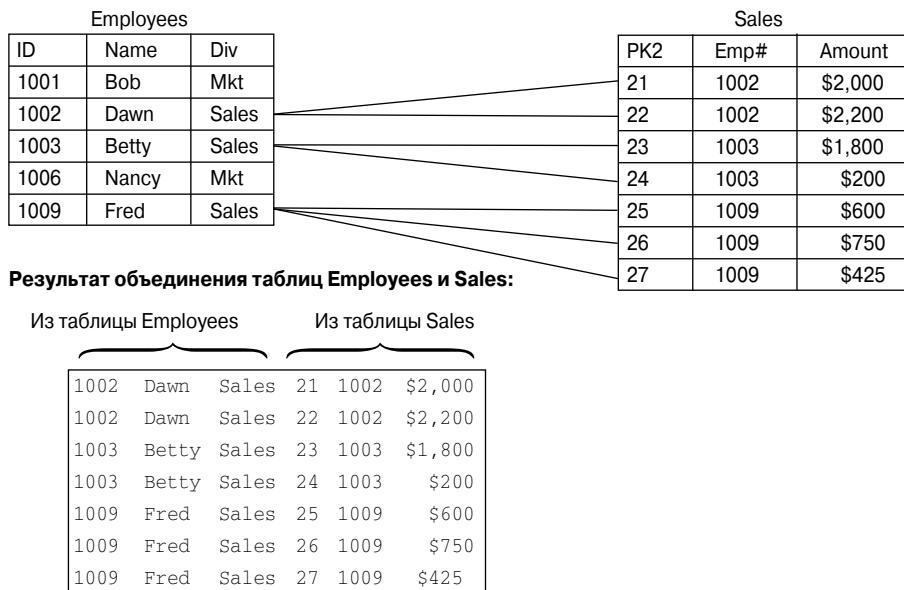


Рис. 6.1. Объединение двух таблиц

по недвижимости. Другая — информацию о продаваемой недвижимости. Объединяя две указанные таблицы по общему столбцу (идентификационному номеру агента), можно определить, с какими домами работает каждый агент. В чистом виде эта информация недоступна ни в одной таблице, однако объединяя различные факты, можно вывести дополнительную информацию. Подобным образом можно написать запрос, объединяющий таблицы *Customers* с таблицей *CustAgentList*, а затем — таблицу *CustAgentList* с *Listings*. Объединяя пары, образованные тремя указанными таблицами, вы можете извлечь информацию о клиентах, имеющих дома на продажу, и дату выставления дома на продажу. Ранее, когда в главе 5 мы использовали запросы, действующие на одну таблицу, эта информация была недоступна. Собирая информацию из разных, но связанных таблиц, вы можете получить из реляционных баз данных очень полезные данные.

## Объединение таблиц, имеющих совпадающие значения столбцов

Оператор *FROM* выражения *SELECT*, объединяющий несколько таблиц, задает условие объединения. *Внутреннее объединение* (иногда называемое простым) представляет собой объединение нескольких таблиц, которое возвращает только строки, удовлетворяющие условию объединения. Одним из самых популярных условий объединения, определение которого приводилось выше, является объединение по эквивалентности.

Напомним, что это — внутреннее объединение, при котором значения из столбца одной таблицы должны *совпадать* со значениями в столбце другой таблицы. Имена столбцов могут отличаться, хотя, разумеется, удобнее, чтобы они были своеобразной подсказкой по объединению таблиц. Частным случаем объединения по эквивалентности является *естественное объединение* (natural join), при котором имена столбцов в объединяемых таблицах совпадают. Естественное объединение требует, чтобы все значения в одноименных столбцах совпадали. Довольно часто в роли таких одноименных столбцов выступают столбцы первичного ключа и соответствующего ему внешнего ключа (хотя это и не обязательно).

## Использование внутреннего объединения

Гибкая форма объединения, определенная в стандартизованной ANSI<sup>1</sup> версии языка SQL99, предполагает задание в операторе FROM названий таблиц и столбцов, значения которых удовлетворяют условию объединения. (В версиях Oracle, предшествовавших 8, условия могли задаваться только в операторе WHERE вместе с критерием фильтрации строк.) Объединение, предложенное ANSI, имеет несколько преимуществ по сравнению со старыми традиционными методами объединения таблиц.

- Случайное создание декартова произведения (каждая строка одной таблицы объединяется с каждой строкой другой) менее вероятно, поскольку вы должны явно задать критерий объединения.
- Синтаксис объединения ANSI SQL99 понятнее.
- Использование стандарта гарантирует, что другие базы данных, совместимые со стандартом ANSI, будут правильно выполнять запросы.
- Критерии объединения задаются отдельно от критериев фильтрации строк.
- При использовании необязательного оператора USING приходится набирать меньше кода.

Синтаксис внутреннего объединения выглядит так, как показано ниже. Слово INNER является необязательным. Если его пропустить, будет предполагаться, что используется внутреннее объединение. (Показан только оператор FROM, однако выражение SELECT может также содержать и другие необязательные операторы — WHERE, ORDER BY и т.д.)

```
SELECT <column-list>
FROM <table1> [<alias1>] [INNER] JOIN <table2> [<alias2>]
  ON <table1>|<alias1>. <join-column1> =
     <table2>|<alias2>. <join-column2>;
```

---

<sup>1</sup>American National Standards Institute — Национальный институт по стандартизации США.

Выражение FROM задает пару таблиц с необязательными псевдонимами таблиц (описываются ниже) и фразой ON, которая задает имена столбцов из каждой таблицы. Оператор “равно” между именами столбцов в фразе ON указывает, что значения двух столбцов должны совпадать, чтобы команда объединила две строки и отобразила столбцы, упомянутые в списке выбора. Если значения не совпадают, строки не объединяются и, следовательно, не попадают в список столбцов. Предположим, например, что требуется отобразить на экране имена и фамилии заказчиков, а также улицу и город, в котором расположена их недвижимость. Соответствующий запрос извлекает информацию из двух таблиц базы данных: таблица Customers содержит информацию о клиентах, а таблица Properties — улицу и город, в котором расположена недвижимость. Для объединения связанных строк из двух таблиц Oracle использует столбец OwnerID таблицы Properties и соответствующий ему столбец CustomerID таблицы Customers. Идентификатор владельца (значение OwnerID) каждого дома находится в столбце CustomerID таблицы Customers. Чтобы согласовать владельцев недвижимости (Customers) с их домами, база данных должна просто найти соответствующие пары значений CustomerID/OwnerID. Ниже показано, как выглядит требуемое выражение SELECT, объединяющее информацию из двух указанных таблиц.

```
SELECT FirstName, LastName, Properties.Address, Properties.City  
FROM Customers INNER JOIN Properties  
ON Customers.CustomerID=Properties.OwnerID;
```

Обратите внимание на то, что третий и четвертый элементы в списке отбора содержат имя таблицы, точку и имя столбца. Подобное *уточненное имя* (qualified name) задает и таблицу, и столбец, который в ней содержится. В данном случае использование уточненного имени двух столбцов необходимо, поскольку столбцы Address и City имеются в обеих таблицах. Если не уточнить имя таблицы, Oracle укажет, что ссылка на столбец неоднозначная — т.е. неизвестно, какой столбец City имеется в виду. Далее мы поэкспериментируем с объединением по эквивалентности, использовав выражения SELECT с условиями объединения таблиц.

В предыдущем примере выражения SELECT использован синтаксис объединения ANSI/ISO SQL99. (В версиях Oracle, предшествовавших 9i, такая форма была недопустимой.) Мы настоятельно рекомендуем, чтобы вы использовали именно этот синтаксис. Тем не менее вам могут встретиться выражения SELECT, в которых использован старый метод объединения таблиц. Согласно этому методу (мы проиллюстрируем его на одном примере и больше нигде в книге использовать не будем) условие объединения помещается в оператор WHERE. При этом возникает проблема смешивания условия объединения с критериями фильтрации в операторе WHERE — визуально очень сложно отличить условия объединения от критериев фильтрации строк. Итак, ниже приведен единственный в книге пример использования старого синтаксиса.

```
SELECT FirstName, LastName, Properties.Address, Properties.City
FROM Customers, Properties
WHERE Customers.CustomerID=Properties.OwnerID;
```

Прежде чем вы перейдете к практическому исследованию запросов, относящихся к нескольким таблицам, вам необходимо инициализировать базу данных Redwood Realty. Даже если вы уже делали это в главе 5, важно, чтобы вы запустили сценарий инициализации именно для главы 6. Этот сценарий вводит новые таблицы, которые были недоступны в предыдущих главах, и безопасным образом удаляет устаревшие версии базы данных, оставшиеся из предыдущих глав. Никакие таблицы, не относящиеся к базе данных Redwood Realty, сценарий не затронет.

Для того чтобы инициализировать базу данных Redwood Realty для главы 6, выполните следующие действия.

1. Найдите папку **RedwoodRealty**, а в ней — файл сценария **BuildRedwood.sql**, который нужно запустить, чтобы инициализировать базу данных (тот же файл сценария вы запускали в начале главы 5). Запишите путь к этому файлу, поскольку он понадобится вам на следующем этапе.
2. Запустите SQL\*Plus и войдите в Oracle.
3. Вместо **<path>** подставьте путь, записанный на этапе 1, а затем введите и выполните следующую команду:

```
START <path>\BuildRedwood
```

Oracle будет отображать на экране имена заполняемых таблиц. Кроме того, будет показано число строк в каждой таблице.

Выражение **SELECT**, которое вы должны написать, содержит *столбцы для отображения* (*display columns*), которые записываются после **SELECT**, но перед **FROM**, и *столбцы для поиска* (*search columns*), которые указываются в выражении **WHERE**. Столбцы первичного и внешнего ключей, которые Oracle использует для объединения двух таблиц, называются *столбцами для объединения* (*join columns*).

Для того чтобы объединить две таблицы и извлечь столбцы из обеих, выполните следующие действия.

1. После запуска SQL\*Plus запустите программу Блокнот, чтобы вы могли легко записывать и редактировать выражения SQL.
2. Введите нижеприведенный код, чтобы очистить экран и задать форматирование столбцов. Как обычно, скопируйте готовые выражения из Блокнота в SQL\*Plus, а затем выполните их.

```
CLEAR SCREEN
COLUMN FirstName FORMAT A12
COLUMN LastName FORMAT A12
COLUMN City FORMAT A15
COLUMN Address FORMAT A25
```

The screenshot shows the Oracle SQL\*Plus interface with the following command and its output:

```

SQL> COLUMN FirstName FORMAT A12
SQL> COLUMN LastName FORMAT A12
SQL> COLUMN City FORMAT A15
SQL> COLUMN Address FORMAT A25
SQL> SELECT FirstName, LastName, Properties.Address, Properties.City
2  FROM Customers JOIN Properties
3  ON Customers.CustomerID=Properties.OwnerID
4  WHERE Properties.City = 'Orick';

```

Below the command, a note states: "Объединение таблиц на основе пар \"первичный-внешний ключ\"".

FIRSTNAME	LASTNAME	ADDRESS	CITY
Ranakrishnan	Vadi	120779 Hwy 101	Orick
Stevan	Power	696 Old State Hwy	Orick
John	McLean	50 Lundblade	Orick
Albert	Reynolds	265 Idlewood Ln	Orick
Huong	Truong	124388 US Highway 101 N	Orick
Israel	Quijada	300 Robinson Rd	Orick
Edwin	Pezda	24 Hufford Rd	Orick
Joan	Anyon	3321 Old State Hwy	Orick
Darci	Tapia	124244 US Highway 101 N	Orick

9 rows selected.

SQL>

Рис. 6.2. Объединение таблиц Customers и Properties

3. Введите следующий код в программе Блокнот, скопируйте его в SQL\*Plus и выполните, чтобы объединить две связанные таблицы и отобразить на экране выбранные столбцы для недвижимости, расположенной в г. Орик (Orick):

```

SELECT FirstName, LastName, Properties.Address, Properties.City
FROM Customers JOIN Properties
    ON Customers.CustomerID=Properties.OwnerID
WHERE Properties.City = 'Orick';

```

Oracle объединит две таблицы по парам совпадающих значений в столбцах CustomerID/OwnerID (рис. 6.2).

4. Переключитесь на Блокнот и запишите сценарий целиком как <имя>Ch06Query01 .sql, подставив вместо <имя> свое имя.
5. Модифицируйте выражение SELECT, дважды в списке отбора удалив уточнение “Properties” и следующую за ним точку (в других местах этого делать не нужно). В результате вы должны получить приведенный ниже запрос. Выполните его. Что произошло?

```

SELECT FirstName, LastName, Address, City
FROM Customers JOIN Properties
    ON Customers.CustomerID=Properties.OwnerID
WHERE Properties.City = 'Orick';

```

6. SQL\*Plus не закрывайте.

При выполнении п. 5 Oracle сообщает, что ссылка на столбец — неоднозначная, возвращая сообщение об ошибке: “ORA-00918: column ambiguously defined”.

Существует альтернативный и немного более простой способ объединения таблиц, используемый, когда объединяемые столбцы двух таблиц имеют одинаковые имена.

Синтаксис альтернативного оператора FROM выглядит следующим образом:

```
SELECT <column-list>
  FROM <table1> [<alias1>] [INNER] JOIN <table2> [<alias2>]
    USING (<join-column>);
```

Не забудьте взять имя столбца в скобки, поскольку в противном случае Oracle сгенерирует сообщение о синтаксической ошибке.

Для того чтобы объединить две таблицы с одинаковыми именами столбцов первичного и внешнего ключей, выполните следующие действия.

1. Переключитесь на программу Блокнот и наберите следующий код под уже существующим текстом:

```
CLEAR SCREEN
SET PAGESIZE 24
SELECT FirstName, LastName, StatusText
  FROM Agents JOIN LicenseStatus
    USING (LicenseStatusID)
   WHERE GENDER = 'M'
  ORDER BY 2,1;
```

2. Скопируйте набранный на этапе 1 код в SQL\*Plus и выполните его. Обратите внимание на то, что форматирование, заданное с помощью COLUMN, по-прежнему применяется к столбцам FirstName и LastName (так будет до тех пор, пока вы не закроете SQL\*Plus). Результат ваших действий должен соответствовать показанному на рис. 6.3.
3. SQL\*Plus не закрывайте.

## Использование естественного объединения

Естественное объединение является близким “родственником” внутреннего. Разница между ними заключается в том, что естественное объединение требует, чтобы значения объединяемых столбцов были одинаковыми, помимо этого, должны совпадать и названия столбцов. При естественном объединении одинаковые значения ищутся во всех одноименных столбцах двух таблиц. Следовательно, если таблица А содержит два столбца, имена которых соответствуют именам столбцов в таблице В, тогда для выполнения объединения необходимо, чтобы значения совпадали в обеих парах столбцов. Использование естественного объединения означает, что вы должны быть аккуратными, указывая Oracle сопоставить *все* пары значений в одноименных столбцах. (Внутреннее объединение ищет соответствия значений столбцов точно так же, как и естественное объединение, но объединяемые столбцы при этом могут иметь разные имена.) И естественное, и внутреннее объединение соединяют столбцы, имеющие одинаковый тип данных. Естественное объединение имеет несколько более простую структуру, поскольку не требует оператора ON. Его синтаксис выглядит следующим образом:

```

SQL> SET PAGESIZE 24
SQL> SELECT FirstName, LastName, StatusText
  2  FROM Agents JOIN LicenseStatus
  3    USING (LicenseStatusID) ← Задание в обеих таблицах
  4  WHERE GENDER = 'M'           связующих столбцов
  5  ORDER BY 2,1;

FIRSTNAME      LASTNAME      STATUSTEXT
-----          -----          -----
Tobias         Carling       Licensed NBA
Cornelis        Dann          Licensed
Jackson        Flamenbaum   Licensed
David          Gagnon        Licensed
James          Keilogg       Licensed
Kai             Marcus        Licensed
Essi            Okindo       Licensed
Ramanathan     Rowe          Licensed
Tim             Schutz        Licensed
Danial         Silverburg   Licensed
Stanislaw      Soitwedel   Licensed
Tim             St-Onge       Licensed
Edwin          Townsend     Licensed
Bruce          Voss          Licensed
Sindisiwe      Weber        Military Service

15 rows selected.

SQL>

```

**Рис. 6.3.** Объединение таблиц по столбцам, имеющим похожие имена

```

SELECT <column-list>
  FROM <table1> [<alias1>] NATURAL JOIN <table2> [<alias2>];

```

Некоторые таблицы Redwood Realty содержат столбцы первичных и внешних ключей с одинаковыми именами. Например, в таблицах *CustAgentList* и *Customers* имеются столбцы с именем *CustomerID*. Следовательно, две указанные таблицы связаны общим столбцом. Подобным образом, таблицы *Agents* и *LicenseStatus* имеют столбец *LicenseStatusID*, а таблицы *Properties* и *Listings* — столбец *PropertyID*. Ниже мы немного поэкспериментируем с естественным объединением. В частности, вы свяжете таблицы *Properties* и *Listings*, чтобы отобразить на экране их родственные столбцы. Вы перечислите столбцы адреса, города, числа комнат и метража из таблицы *Properties*, а также цену (*AskingPrice*) из таблицы *Listings*. Строки двух указанных таблиц будут объединяться на основе совпадающих значений в столбце *PropertyID*, причем это будет единственный столбец, имеющий одинаковые имена в обеих таблицах. Таким образом, природным выбором является естественное объединение.

Для того чтобы объединить две таблицы с помощью естественного объединения, выполните следующие действия.

1. Откройте программу Блокнот и введите нижеприведенный код. Затем скопируйте его в SQL\*Plus и выполните. (Не забудьте нажать клавишу *<Enter>* после последней строки.) Заданное форматирование облегчит восприятие результатов запроса.

```

CLEAR SCREEN
SET PAGESIZE 20

```

```

SQL> SET PAGESIZE 20
SQL> COLUMN Address FORMAT A25 HEADING 'Address'
SQL> COLUMN City FORMAT A15 HEADING 'City'
SQL> COLUMN Bedrooms FORMAT 99 HEADING 'Br'
SQL> COLUMN Bathrooms FORMAT 99 HEADING 'Ba'
SQL> COLUMN SqFt FORMAT 9,999 HEADING 'Square|Feet'
SQL> COLUMN AskingPrice FORMAT $9,999,999 HEADING 'Asking|Price'
SQL> BREAK ON CITY SKIP 1
SQL> SELECT Address, City, Bedrooms, Bathrooms, SqFt, AskingPrice
  2 FROM Properties NATURAL JOIN Listings
  3 WHERE SqFt >= 2000 AND Bathrooms > 3 AND Bedrooms > 3
  4 ORDER BY City, AskingPrice;

Address          City      Br   Ba  Square    Asking
-----          -----  ---  ---  -----  -----
8130 W End Rd    Arcata    5   4  5,150  $560,000
5990 Stover Rd   Blue Lake  4   4  4,828  $725,000
2312 Myrtle Ave Eureka     5   4  2,845  $180,000
6739 Myrtle Ave Eureka     6   4  3,548  $219,000
3209 W St        Eureka     4   4  3,200  $325,000
1845 Quaker St   Eureka     4   4  3,249  $335,000
1591 Kings Row   Fortuna    4   4  4,490  $495,000
3340 Dows Prairie Rd McKinleyville  4   4  2,580  $235,000
1138 Perini Rd   McKinleyville 4   4  3,000  $380,000

9 rows selected.
SQL>

```

**Рис. 6.4.** Использование естественного объединения

```

COLUMN Address FORMAT A25 HEADING 'Address'
COLUMN City FORMAT A15 HEADING 'City'
COLUMN Bedrooms FORMAT 99 HEADING 'Br'
COLUMN Bathrooms FORMAT 99 HEADING 'Ba'
COLUMN SqFt FORMAT 9,999 HEADING 'Square|Feet'
COLUMN AskingPrice FORMAT $9,999,999 HEADING 'Asking|Price'
BREAK ON CITY SKIP 1

```

2. Введите нижеприведенное выражение SELECT, чтобы объединить две таблицы и вернуть их общие строки. С помощью критерия WHERE вы сильно ограничиваете число возвращаемых строк. Если хотите, можете ослабить критерий или вообще его убрать, чтобы в ответ на запрос получить несколько сотен строк.

```

SELECT Address, City, Bedrooms, Bathrooms, SqFt, AskingPrice
  FROM Properties NATURAL JOIN Listings
 WHERE SqFt >= 2000 AND Bathrooms > 3 AND Bedrooms > 3
 ORDER BY City, AskingPrice;

```

Oracle возвращает девять пар строк из объединенных таблиц (рис. 6.4).

3. SQL\*Plus не закрывайте.

## Объединение трех и большего числа таблиц

Ваши возможности не ограничиваются объединением всего лишь двух таблиц. Число таблиц, которые Oracle позволяет объединять, не ограничено. Чтобы объединить три или большее количество таблиц, Oracle вначале объединяет две, используя условия

объединения, заданные для двух таблиц. Затем база данных соединяет результат со следующие таблицей, основываясь на условиях объединения, касающихся столбцов объединенной таблицы и новой таблицы, которая к ней присоединяется. Данный процесс продолжается до тех пор, пока не будут объединены все таблицы. (Порядок объединения определяет оптимизатор запросов Oracle, основываясь на условиях объединения, индексах таблиц и доступной статистике таблиц. Подробно оптимизация описана в главе 13.)

Например, чтобы объединить таблицы Agents, Listing и Properties, необходимо задать имена таблиц и условия объединения для каждой пары таблиц (последовательно), указанных в операторе FROM. Приведенное ниже выражение SELECT объединяет три таблицы на основе значений, расположенных в двух парах столбцов:

```
SELECT FirstName, LastName, Address, City,
       Bedrooms, Bathrooms, AskingPrice
  FROM Agents INNER JOIN Listings
    ON AgentID = ListingAgentID INNER JOIN Properties
    ON Listings.PropertyID = Properties.PropertyID;
```

Разберем приведенный фрагмент кода. Чтобы объединить две таблицы, попарно указываются имена таблиц и столбцы для объединения. Например, приведенная ниже фраза сообщает Oracle, что необходимо объединить столбцы таблиц Agents и Listings, согласовав значения в столбцах AgentID и ListingAgentID.

```
FROM Agents JOIN Listings
  ON AgentID = ListingAgentID
```

В третьей и четвертой строках оператора FROM на основе совпадающих значений столбцов объединяются таблицы Listings и Properties:

```
      JOIN Properties
  ON Listings.PropertyID = Properties.PropertyID;
```

В данном фрагменте кода столбцы для объединения указываются через имена соответствующих таблиц. Например, Listings.PropertyID задает столбец PropertyID в таблице Listings. Подобным образом Properties.PropertyID обозначает столбец PropertyID в таблице Properties. Чтобы избежать неоднозначности, задавая элементы списка отбора, следует всегда использовать уточненные имена, если в объединяемых таблицах имеются столбцы с одинаковыми именами. Кроме того, чтобы сократить набираемый код, с таблицей можно соотнести короткое имя, называемое ее псевдонимом. Затем вы можете (и должны!) использовать псевдоним, чтобы уточнять все имена столбцов в списке отбора.

Альтернативным способом записи оператора FROM, объединяющего две таблицы, является размещение рядом всех выражений ON. Вы можете использовать тот вариант, который вам удобнее, поскольку результат от этого не изменится. Не забудьте, что выражения ON необходимо вкладывать в порядке, обратном порядку перечисления имен объединяемых таблиц (см. ниже).

```
SELECT FirstName, LastName, Address, City,
       Bedrooms, Bathrooms, AskingPrice
```

```
FROM Agents JOIN Listings JOIN Properties
  ON Listings.PropertyID = Properties.PropertyID
  ON AgentID = ListingAgentID;
```

Поскольку столбцы LastName и FirstName имеются только в таблице Agents (из трех таблиц, фигурирующих в команде), в списке отбора можно задавать неуточненные имена. Использование псевдонимов таблиц мы проиллюстрируем ниже.

Для того чтобы объединить три таблицы, используя их псевдонимы, выполните следующие действия.

1. Запустив SQL\*Plus, введите нижеприведенный код в Блокноте, скопируйте его в SQL\*Plus и нажмите клавишу <Enter>, чтобы очистить экран, убрать разрывы страниц и определить дополнительное форматирование столбцов. (Не забудьте набрать две одинарные кавычки между словом Agent и буквой s в команде COLUMN, что даст в заголовке столбца одинарную кавычку.)

```
CLEAR SCREEN
CLEAR BREAKS
COLUMN Agent FORMAT A25 HEADING 'Agent''s|Name'
```

2. Введите нижеприведенный код, скопируйте его в SQL\*Plus и выполните. Oracle свяжет три указанные таблицы и выведет на экран четыре столбца с информацией, отобранной из всех трех таблиц. В результате вы должны получить то же, что показано на рис. 6.5.

```
SELECT ag.FirstName||' '| |ag.LastName AS Agent,
       pr.Address, pr.City, li.AskingPrice
  FROM Agents ag INNER JOIN Listings li
    ON ag.AgentID = li.ListingAgentID
       INNER JOIN Properties pr
      ON li.PropertyID = pr.PropertyID
 WHERE UPPER(pr.City) IN ('ORICK', 'BLUE LAKE', 'LOLETA')
 ORDER BY pr.City, li.AskingPrice;
```

3. Введите CLEAR COLUMNS и нажмите клавишу <Enter>, чтобы удалить форматирование, наложенное на столбцы.
4. Переключитесь на программу Блокнот и запишите текстовый файл, чтобы сохранить результаты своей работы.
5. Закройте программу Блокнот и SQL\*Plus. (Закрывая SQL\*Plus, вы отменяете все переопределенные настройки среды — размер страницы, заголовки и т.д., — возвращая их к значениям по умолчанию.)

## Другие типы объединений и условий объединения

Все условия объединения, рассмотренные до этого момента, относятся к объединениям по эквивалентности (или аналогичным им естественным объединениям). Во всех случаях Oracle проверяет равенство (оператор =) значений двух столбцов, фор-

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> CLEAR BREAKS
breaks cleared
SQL> COLUMN Agent FORMAT A25 HEADING 'Agent''s|Name'
SQL> SELECT ag.FirstName||' '||ag.LastName AS Agent,
2      pr.Address, pr.City, li.AskingPrice
3  FROM Agents ag INNER JOIN Listings li
4    ON ag.AgentID = li.ListingAgentID
5          INNER JOIN Properties pr
6    ON li.PropertyID = pr.PropertyID
7 WHERE UPPER(pr.City) IN ('ORICK','BLUE LAKE','LOLETA')
8 ORDER BY pr.City, li.AskingPrice;

Agent's
Name      Address            City      Asking
          Price
James Kellogg   540 Broad     Blue Lake $112,000
Lee Reed       2115 Chezum Rd  Blue Lake $139,500
Tobias Carling 830 Railroad Ave Blue Lake $145,000
Stanislaw Soltwedel 833 Blue Lake Bl Blue Lake $159,446
Lee Reed       741 2nd Ave     Blue Lake $198,463
Ramanathan Rowe 5998 Stover Rd  Blue Lake $725,000
Bruce Voss     3146 Copenhagen Rd Loleta   $104,000
Heather Sheibani 356 Loleta Dr  Loleta   $111,950
Barbara Herring 296 Spring    Loleta   $114,000
Lora Allee      229 Church    Loleta   $145,000
Sindisive Weber 552 Pershing  Loleta   $154,000
Edwin Townsend 119 Bay View Dr Loleta   $162,500
Lora Allee      602 Pershing  Loleta   $282,500
David Gagnon    124380 US Highway 101 N Orick   $129,950
Cecilia Romero 120779 Hwy 101   Orick   $178,266
Lee Reed       124244 US Highway 101 N Orick   $184,950

16 rows selected.

SQL>

```

**Рис. 6.5.** Объединение трех таблиц и фильтрация строк

мирующих условие объединения. Существуют и другие условия объединения, когда между именами столбцов в условии объединения находится не оператор равенства. Вообще, в условиях допускается использованием операторов BETWEEN, >, < и др. Соответствующие примеры представлены ниже в этой главе. Кроме того, существуют три типа объединений, одно из которых мы уже рассмотрели: внутреннее, внешнее и самообъединение. Несколько примеров внутренних объединений приводилось выше. *Внешнее объединение* возвращает строку не только тогда, когда столбцы удовлетворяют условиям объединения, но и тогда, когда один из объединяемых столбцов содержит значение NULL. Внешнее объединение является расширением концепции внутреннего объединения — внутреннее объединение плюс все остальные строки из другой таблицы. *Самообъединение* возвращает строки из одной таблицы, объединяя таблицу саму с собой и рассчитывая условие объединения. Далее мы приступаем к подробному рассмотрению запросов с внешним объединением.

## Использование запросов с внешним объединением

Напомним, что запрос с внешним объединением извлекает строки из двух таблиц только тогда, когда во всех объединяемых таблицах существуют значения в указанных столбцах (причем, если речь идет об объединении по эквивалентности, эти значения должны совпадать). Если один из указанных для объединения столбцов таблицы имеет значение NULL, в ответ на запрос строка возвращаться не будет. Пред-

Salespersons			Sales		
ID	Name	Div	PK2	Emp#	Amount
1001	Bob	Mkt	21	1002	\$2,000
1002	Dawn	Sales	22	1002	\$2,200
1003	Betty	Sales	23	1003	\$1,800
1006	Nancy	Mkt	24	1003	\$200
1009	Fred	Sales			

### Результат объединения таблиц Salespersons и Sales:

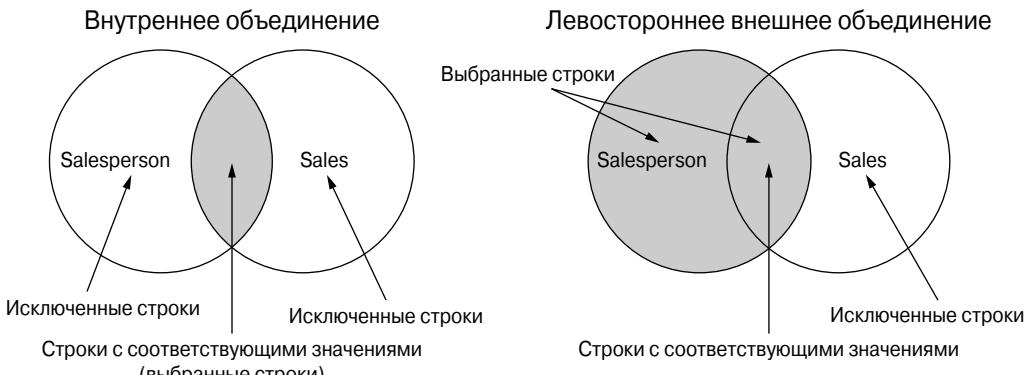
Из таблицы Salespersons      Из таблицы Sales

1002	Dawn	Sales	21	1002	\$2,000
1002	Dawn	Sales	22	1002	\$2,200
1003	Betty	Sales	23	1003	\$1,800
1003	Betty	Sales	24	1003	\$200
1001	Bob	Mkt	null	null	null
1009	Nancy	Mkt	null	null	null
1009	Fred	Sales	null	null	null

Рис. 6.6. Иллюстрация внешнего объединения

ставьте, например, что мы объединяем таблицу *Salesperson*, содержащую информацию об именах продавцов, и таблицу *Sales*, в которой записываются все продажи и идентификаторы (*ID*) продавцов, оформивших их. Объединив таблицу *Salesperson* с таблицей *Sales*, мы получим строки, в которых записаны имена продавцов и совершенные ими сделки. При этом не совсем понятно, что будет с продавцами, не заключившими ни одной сделки. Чтобы отобразить на экране подобную скрытую информацию, необходимо использовать внешнее объединение, которое вернет те же строки, что и внутреннее, плюс дополнительные строки с именами продавцов и значениями *NULL* в графе продаж. Подобным образом, запрос с внутренним объединением не позволяет получить информацию о товарах, не продававшихся в течение указанного месяца, — на экран можно вывести только данные о совершившихся продажах. Чтобы получить искомую информацию, необходимо использовать запрос с внешним объединением. Простой пример объединения информации о продажах и продавцах показан на рис. 6.6. Используя внешнее объединение, можно увидеть, что три продавца не завершили сделки за обозначенный период (факт, который нельзя получить на основании только внутреннего объединения). На рис. 6.7 сказанное иллюстрируется с помощью диаграмм Венна.

Существуют три типа внешних объединений: левостороннее внешнее объединение, правостороннее внешнее объединение и полное внешнее объединение. Во всех случаях синтаксис команды *SELECT* такой же, как для внутреннего объединения:



**Рис. 6.7.** Диаграммы Венна, иллюстрирующие внутреннее объединение и левостороннее внешнее объединение

```
SELECT <column-list>
FROM <table1> [<alias1>]
{LEFT|RIGHT|FULL} [OUTER] JOIN <table2> [<alias2>]
ON {<table1>|<alias1>}.<join-column1> =
{<table2>|<alias2>}.<join-column2>;
```

Слово *OUTER* использовать необязательно, хотя и удобно, так как оно напоминает, что запрос является внешним объединением. *Левостороннее внешнее объединение* задает принцип обработки таблицы, указанной в левой части оператора *FROM*. Оно означает, что если в левой таблице останутся строки, которым не было найдено соответствие, то они должны сопоставляться с записью *NULL* и возвращаться в составе результата. Подобным образом *правостороннее внешнее объединение* задает, как должна обрабатываться правая часть выражения *RIGHT OUTER JOIN* в операторе *FROM*. Если в правой таблице останутся строки, которым не было найдено соответствие, они должны сопоставляться с записью *NULL* и возвращаться в составе результата. *Полное внешнее объединение* означает, что *Oracle* вернет не только строки, которым было найдено соответствие, но и все строки, не имеющие аналогов в других таблицах (из числа подлежащих объединению).

**Совет.** Разумеется, запрос с левосторонним внешним объединением даст те же самые результаты, что и запрос с правосторонним внешним объединением, в котором порядок таблиц в операторе *FROM* изменен на обратный.

Возможно, вам будет интересно узнать, какие агенты Redwood Realty не имеют никаких записей в таблице *Listings*. Все низкопродуктивные агенты, особенно те, у которых нет домов на продажу, должны пересмотреть свое отношение к работе! Выявить подобных агентов помогает запрос с внешним объединением. В настоящее время все двадцать девять агентов Redwood Realty интенсивно работают, и с ними сопоставлены продаваемые дома в таблице *Listings*. Следовательно, компания

решает нанять трех новых сотрудников, поместив соответствующие записи в таблицу Agents. Поскольку сотрудники только приступают к работе, в таблице Listings не будет записей, соответствующих внешнему ключу ListingAgentID, равному первичным ключам AgentID трех новых агентов. В следующем упражнении вы добавите в таблицу Agents трех новых агентов, организуете стандартный запрос с внутренним объединением таблиц Agents и Listings, а затем выполните запрос с внешним объединением тех же двух таблиц.

Для того чтобы поэкспериментировать с право- или левосторонним внешним объединением, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle и запустите программу Блокнот. Затем введите (вначале в Блокноте, а затем скопируйте в SQL\*Plus) и выполните следующую команду, вставляющую новые строки в таблицу Agents. Будьте очень внимательны при вводе трех указанных выражений INSERT.

```
CLEAR SCREEN
REM Вставляются строки с информацией о новых агентах.
REM Соответствующих записей в CustAgentList нет
INSERT INTO Agents(AgentID,FirstName,LastName,LicenseStatusID)
    VALUES(23456,'Robert','Sellsmore',1001);
INSERT INTO Agents(AgentID,FirstName,LastName,LicenseStatusID)
    VALUES(23471,'Susan','Swarthmore',1001);
INSERT INTO Agents(AgentID,FirstName,LastName,LicenseStatusID)
    VALUES(23498,'George','Nagy',1002);
```

2. Введите COMMIT; и нажмите клавишу <Enter>, чтобы зафиксировать внесенные изменения.
3. Введите указанный код в Блокноте, а затем выполните его в SQL\*Plus:

```
SELECT FirstName, LastName
FROM Agents a LEFT OUTER JOIN CustAgentList c
    ON a.AgentID = c.AgentID
WHERE c.ListingID IS NULL
ORDER BY LastName, FirstName;
```

Oracle выведет на экран информацию о трех агентах, не имеющих соответствующих строк в таблице Listings. В результате вы должны получить то же, что показано на рис. 6.8.

4. Запишите левостороннее внешнее объединение, чтобы другим способом просмотреть на “пропущенные строки”. Введите и выполните следующие команды SQL\*Plus и модифицированное выражение SELECT:

```
SET PAGESIZE 50
CLEAR SCREEN
SELECT FirstName, LastName, COUNT(ListingID)
FROM Agents a LEFT OUTER JOIN CustAgentList c
    ON a.AgentID = c.AgentID
GROUP BY FirstName, LastName
ORDER BY LastName, FirstName;
```

The screenshot shows a window titled "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main area contains the following SQL session:

```

SQL> REM Insert new agent rows. They have no entries in CustAgentList.
SQL> INSERT INTO Agents(AgentID,FirstName,LastName,LicenseStatusID)
  2  VALUES(23456,'Robert','Sellsmore',1001);

1 row created.

SQL> INSERT INTO Agents(AgentID,FirstName,LastName,LicenseStatusID)
  2  VALUES(23471,'Susan','Swarthmore',1001);

1 row created.

SQL> INSERT INTO Agents(AgentID,FirstName,LastName,LicenseStatusID)
  2  VALUES(23498,'George','Nagy',1002);

1 row created.

SQL> COMMIT;

Commit complete.

SQL> SELECT FirstName, LastName
  2  FROM Agents a LEFT OUTER JOIN CustAgentList c
  3    ON a.AgentID = c.AgentID
  4  WHERE c.ListingID IS NULL
  5  ORDER BY LastName, FirstName;

FIRSTNAME          LASTNAME
-----            -----
George             Nagy
Robert            Sellsmore
Susan              Swarthmore

```

**Рис. 6.8.** Использование внешнего левостороннего объединения

Из полученных результатов (рис. 6.9) видно, что для трех агентов функция COUNT возвращает значение 0. Это означает, что данным строкам нет соответствия в таблице Listings.

5. Введите и выполните следующую команду, чтобы обновить значение переменной среды PAGESIZE:

```
SET PAGESIZE 14
```

6. SQL\*Plus не закрывайте.

## Использование запросов с самообъединением

Напомним, что самообъединение — это внутреннее или внешнее объединение двух столбцов, расположенных в одной таблице. Довольно часто для иллюстрации самообъединения применяется таблица сотрудников, в которой указаны как сами сотрудники, так и их руководители. В подобной таблице имеется столбец (допустим, ManagerID), в котором обозначено, кто является непосредственным начальником сотрудника. Уникальным идентификатором каждого сотрудника является первичный ключ, допустим, представленный в столбце EmployeeID. Чтобы объединить столбцы для получения требуемой информации, таблицу необходимо дважды указать в операторе FROM выражения SELECT. Чтобы в подобном случае различать идентификатор менеджера и соответствующий идентификатор сотрудника, вы должны использовать

The screenshot shows the Oracle SQL\*Plus interface with a query window. The query is:

```
SQL> SELECT FirstName, LastName, COUNT(ListingID)
  2  FROM Agents a LEFT OUTER JOIN CustAgentList c
  3  ON a.AgentID = c.AgentID
  4  GROUP BY FirstName, LastName
  5  ORDER BY LastName, FirstName;
```

The output is a table with three columns: FIRSTNAME, LASTNAME, and COUNT(ListingID). The last four rows of the table have a value of 0 in the COUNT column. Three arrows point from these rows to a note on the right side of the screen.

FIRSTNAME	LASTNAME	COUNT(ListingID)
Lora	Allee	47
Tobias	Carling	25
Belinda	Chong	31
Elizabeth	Dahlen	28
Cornelis	Dane	45
Crystal	Fernandez	34
Jackson	Flamenbaum	34
David	Gagnon	44
Barbara	Herring	29
James	Kellogg	35
Patricia	Lewis	24
Kai	Marcoux	24
George	Nagy	0
Essi	Okindo	28
Nancy	Piperova	48
Lee	Reed	53
Clair	Robinson	38
Cecilia	Romero	42
Ramanathan	Rove	41
Tim	Schutz	38
Ricki	Selby	33
Robert	Sellsmore	0
Heather	Sheibani	37
Daniel	Silverburg	42
Stanislaw	Soltwedel	34
Tim	St-Onge	36
Susan	Swarthmore	0
Jessica	Taylor	34
Edwin	Townsend	25
Bruce	Voss	35
Sindisiwe	Weber	35
Christine	Williams	43

32 rows selected.

SQL>

Значение 0 означает отсутствие соответствий в таблице Listings

Рис. 6.9. Использование функции COUNT для вывода на экран строк, не имеющих соответствий в другой таблице

псевдоним таблицы. Например, в приведенном ниже гипотетическом самообъединении перечислены сотрудники и их начальники, причем вся информация извлекается из одной таблицы Employees.

```
SELECT w.FirstName, w.LastName, s.FirstName, s.LastName
  FROM Employees w INNER JOIN Employees s
    ON w.ManagerID = s.EmployeeID;
```

Для каждой записи таблицы условие объединения по эквивалентности соотносит значение в столбце ManagerID (w.ManagerID) с идентификатором сотрудника EmployeeID (s.EmployeeID). Полное имя сотрудника формируется левой парой значений FirstName и LastName, а правая пара значений FirstName/LastName является именем начальника этого сотрудника. В таблицах Redwood Realty подобная иерархическая структура отсутствует, поэтому для демонстрации самообъединения мы создадим и заполним данными отдельную таблицу, которая *не является* частью базы данных Redwood Realty.

Для того чтобы создать таблицу Employee и заполнить ее данными, выполните следующие действия.

1. Если необходимо, запустите SQL\*Plus и войдите в Oracle.
2. Используя Windows Explorer или другую программу, найдите в папке, соответствующей главе 6, файл Ch06EmpSelfJoin.sql и запишите путь к нему.
3. Введите и выполните следующую команду, подставив реальный путь к файлу сценария вместо <path> (включая букву диска):

```
START <path>\Ch06EmpSelfJoin
```

Файл сценария отобразит сообщение, указывающее, что создание и заполнение таблицы EmpSelfJoin прошло успешно.

Теперь вы можете поэкспериментировать с запросом, содержащим самообъединение. Для начала вам потребуется вывести на экран содержимое столбцов.

Для того чтобы отобразить на экране содержимое столбцов и произвести самообъединение, выполните следующие действия.

1. Введите и выполните в SQL\*Plus следующую команду:

```
CLEAR SCREEN  
SELECT * FROM EmpSelfJoin;
```

2. Введите и выполните

```
DESCRIBE EmpSelfJoin
```

Получаемая в результате 21 строка таблицы EmpSelfJoin, а также имена и тип данных ее столбцов показаны на рис. 6.10.

Рассмотрим рис. 6.10. Столбец EmployeeID содержит уникальные идентификационные номера всех сотрудников. Крайний правый столбец (BossID) записи содержит идентификационный номер начальника этого сотрудника. Рассмотрим, например, строку с информацией о Кенте Конраде (Kent Conrad), сотруднике с номером EmployeeID, равным 12765. В столбце BossID этой строки содержится значение 10497, представляющее собой внешний ключ, указывающий на столбец той же самой таблицы. В частности, указанное значение 10497 соответствует EmployeeID Джона Уорнера (John Warner). Следовательно, Джон Уорнер является начальником Кента Конрада. Рассмотрим далее нижнюю строку, соответствующую Биллу Фристу (Bill Frist). Идентификационный номер его начальника (BossID) равен 12765. Такому значению EmployeeID соответствует запись Кента Конрада, следовательно, Кент является начальником Билла.

Подумаем теперь, как можно отразить данную связь (таблицу, содержащую столбец, ссылающийся на другой столбец той же таблицы) в виде, удобном для восприятия? Чтобы решить эту задачу, можно написать запрос с самообъединением, который может быть запросом с внутренним или внешним объединением. В частности, в запросе можно *дважды* записать имя таблицы и различать столбцы на основе двух

The screenshot shows the Oracle SQL\*Plus interface with the following command and its results:

```
SQL> SELECT * FROM EmpSelfJoin;
```

EMPLOYEEID	FIRSTNAME	LASTNAME	HIREDATE	BIRTHDATE	G	BOSSID
10497	John	Warner	05-SEP-97	23-OCT-53	M	
12765	Kent	Conrad	12-SEP-06	01-FEB-55	M	10497
12963	Bobby	Durbin	18-SEP-08	25-MAR-56	F	10497
13555	Susan	McCain	30-AUG-95	20-OCT-57	F	10497
15233	Mike	DeWine	17-JAN-99	06-NOV-58	M	10497
12301	Thomas	Daschle	21-SEP-97	11-JAN-59	M	12963
15521	Celia	Pryor	30-MAY-06	10-APR-59	F	12963
14601	Hitch	McConnell	25-AUG-06	21-DEC-60	M	12963
13771	Felicity	Dorgan	29-JUN-98	26-SEP-61	F	12963
11775	Patty	Murray	01-NOV-99	21-OCT-62	F	12963
14883	Connie	Burns	30-MAY-97	29-NOV-64	F	13555

EMPLOYEEID	FIRSTNAME	LASTNAME	HIREDATE	BIRTHDATE	G	BOSSID
14599	Clair	Baucus	07-DEC-03	04-FEB-65	F	13555
14117	Christopher	Dodd	13-JUL-97	19-MAY-68	M	13555
13353	John	Sununu	04-MAY-97	04-APR-69	M	13555
12499	Barbara	Fitzgerald	16-OCT-06	01-JUL-69	F	15233
10429	Jessica	Chafee	23-MAY-05	03-OCT-69	F	15233
10913	Ted	Stevens	05-JUN-96	03-APR-70	M	15233
10041	Bob	Graham	03-OCT-96	12-DEC-70	M	15233
12875	John	Edwards	28-DEC-05	04-MAR-74	M	12765
10849	Lora	Gregg	18-JUL-03	03-MAR-83	F	12765
15349	Bill	Frist	21-JUN-04	21-AUG-83	M	12765

21 rows selected.

```
SQL> DESCRIBE EmpSelfJoin
```

Name	Null?	Type
EMPLOYEEID	NOT NULL	NUMBER(38)
FIRSTNAME		NVARCHAR2(15)
LASTNAME		NVARCHAR2(15)
HIREDATE		DATE
BIRTHDATE		DATE
GENDER		NVARCHAR2(1)
BOSSID		NUMBER(38)

```
SQL>
```

Рис. 6.10. Отображение на экране таблицы EmpSelfJoin

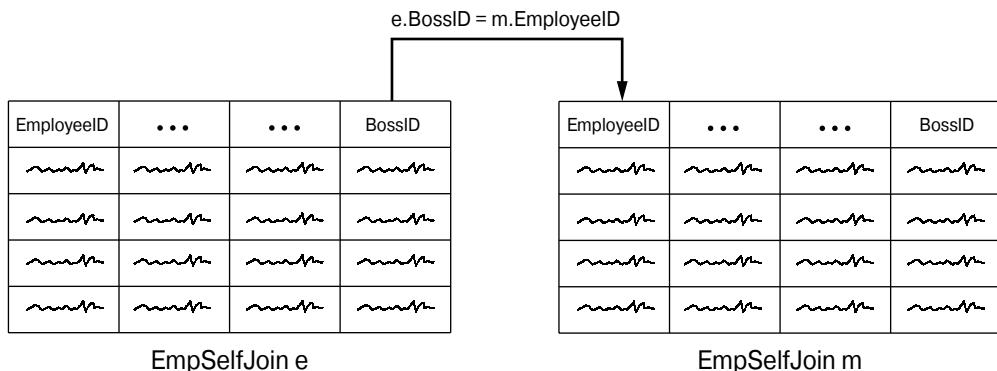
заданных псевдонимов таблицы. Считайте, что вместо одной таблицы у нас есть две, расположенные рядом. Связи между ними устанавливаются точно так же, как между двумя различными таблицами, — вы сообщаете Oracle, какой столбец одной таблицы соотносится с указанным столбцом другой. На рис. 6.11 таблица EmpSelfJoin показана как две отдельные таблицы, каждой из которых присвоен свой псевдоним. Псевдоним “e” обозначает таблицу EmpSelfJoin с точки зрения сотрудника (employee), а “m” — ту же таблицу EmpSelfJoin с точки зрения менеджера (manager).

Ключами к объединению таблицы с самой собой являются: использование псевдонимов и применение объединения по эквивалентности с правильно уточненными названиями столбцов. Чуть позже вы создадите выражение SQL для объединения таблицы EmpSelfJoin с самой собой путем согласования пар значений EmployeeID/BossID.

Для того чтобы объединить таблицу с самой собой, выполните следующие действия.

1. Очистите экран, набрав и выполнив следующую команду:

```
CLEAR SCREEN
```



**Рис. 6.11.** Иллюстрация таблицы, объединяемой с самой собой посредством псевдонима

2. Введите приведенный ниже код в программе Блокнот, чтобы облегчить его редактирование. Затем скопируйте его в SQL\*Plus и выполните. Если потребуется, исправьте код в Блокноте, а затем скопируйте и выполните его повторно.

```
SELECT m.FirstName || ' ' || m.LastName AS Manager,
       e.FirstName || ' ' || e.LastName AS Employee
  FROM EmpSelfJoin e INNER JOIN EmpSelfJoin m
    ON e.BossID = m.EmployeeID
 ORDER BY m.LastName, e.LastName;
```

Oracle отображает на экране двадцать строк; в первом столбце указывается имя менеджера, а во втором — сотрудника, отчитывающегося перед этим менеджером (рис. 6.12). Стоп! В списке, показанном на рис. 6.10, фигурировал 21 сотрудник. Где 21-й сотрудник? Изучая записи сотрудников, представленные на рис. 6.10, видим, что Джон Уорнер (John Warner) имеет значение BossID, равное NULL, поскольку он — генеральный директор Readwood Realty, который ни перед кем не отчитывается. Поскольку при внутреннем объединении строки, не имеющие соответствия в другой (в нашем случае — в той же) таблице, в ответе на запрос не представляются, Джон Уорнер не указывается в столбце сотрудников — он фигурирует только как начальник. Чтобы включить его в перечень вместе с другими сотрудниками, будем считать, что он подотчетен совету директоров. Используем функцию NVL и заменим внутреннее объединение внешним.

Для того чтобы использовать внешнее объединение для отображения на экране всех сотрудников и их начальников, выполните следующие действия.

1. Выполните команду CLEAR SCREEN, чтобы очистить экран.
2. Модифицируйте введенное ранее выражение SELECT до указанного ниже вида и снова выполните запрос.

```

SQL> SELECT m.FirstName||' '||m.LastName AS Manager,
2       e.FirstName||' '||e.LastName AS Employee
3   FROM EmpSelfJoin e INNER JOIN EmpSelfJoin m
4     ON e.BossID = m.EmployeeID
5   ORDER BY m.LastName, e.LastName;

MANAGER          EMPLOYEE
-----          -----
Kent Conrad      John Edwards
Kent Conrad      Bill Frist
Kent Conrad      Lora Gregg
Mike DeWine       Jessica Chafee
Mike DeWine       Barbara Fitzgerald
Mike DeWine       Bob Graham
Mike DeWine       Ted Stevens
Mike DeWine       Thomas Daschle
Bobby Durbin     Felicity Dorgan
Bobby Durbin     Mitch McConnell
Bobby Durbin     Patty Murray

MANAGER          EMPLOYEE
-----          -----
Bobby Durbin     Celia Pryor
Susan McCain     Clair Baucus
Susan McCain     Connie Burns
Susan McCain     Christopher Dodd
Susan McCain     John Sununu
John Warner      Kent Conrad
John Warner      Mike DeWine
John Warner      Bobby Durbin
John Warner      Susan McCain

20 rows selected.

SQL>

```

Рис. 6.12. Перечисление сотрудников и их руководителей с помощью самообъединения

```

SELECT
    NVL2(m.LastName, m.FirstName||' '||m.LastName,
          'Board of Directors') AS Manager,
    e.FirstName||' '||e.LastName AS Employee
  FROM EmpSelfJoin e LEFT OUTER JOIN EmpSelfJoin m
    ON e.BossID = m.EmployeeID
  ORDER BY m.LastName, e.LastName;

```

3. Теперь можете удалить таблицу EmpSelfJoin, поскольку она вам больше не нужна. Введите и выполните следующую команду:

```
DROP TABLE EmpSelfJoin CASCADE CONSTRAINTS PURGE;
```

Oracle ответит сообщением “Table dropped”.

Результат данного запроса показан на рис. 6.13. Полный перечень из двадцати одного начальника/подчиненного включает и генерального директора Джона Уорнера. Обратите внимание на то, что в столбце Manager записи John Warner появилась запись “Board of Directors” (“совет директоров”), поскольку именно такое значение возвращает функция NVL2, если фамилия непосредственного начальника не указана, т.е. равна NULL (а это именно такой случай, поскольку эта фамилия находится по значению BossID, которое для записи John Warner не указано). В противном случае NVL2 возвращает фамилию и имя начальника, которые объединены с помощью конкатенации.

The screenshot shows the Oracle SQL\*Plus interface with a query window. The query is:

```
SQL> SELECT
  2   NVL(m.LastName,m.FirstName||' '||m.LastName,'Board of Directors')
  3   AS Manager,
  4   e.FirstName||' '||e.LastName AS Employee
  5  FROM EmpSelfJoin e LEFT OUTER JOIN EmpSelfJoin m
  6  ON e.BossID = m.EmployeeID
  7 ORDER BY m.LastName, e.LastName;
```

The results are displayed in two sections: MANAGER and EMPLOYEE.

MANAGER	EMPLOYEE
Kent Conrad	John Edwards
Kent Conrad	Bill Frist
Kent Conrad	Lora Gregg
Mike DeVine	Jessica Chafee
Mike DeVine	Barbara Fitzgerald
Mike DeVine	Bob Graham
Mike DeVine	Ted Stevens
Bobby Durbin	Thomas Daschle
Bobby Durbin	Felicity Dorgan
Bobby Durbin	Mitch McConnell
Bobby Durbin	Patty Murray
MANAGER	EMPLOYEE
Bobby Durbin	Celia Pryor
Susan McCain	Clair Baucus
Susan McCain	Connie Burns
Susan McCain	Christopher Dodd
Susan McCain	John Sununu
John Warner	Kent Conrad
John Warner	Mike DeVine
John Warner	Bobby Durbin
John Warner	Susan McCain
Board of Directors	John Warner

A note on the right side of the results states: "Если значение BossID равно NULL, появляется запись "Board of Directors"" (If the value of BossID is NULL, the record "Board of Directors" appears).

21 rows selected.

Рис. 6.13. Руководители и их подчиненные, перечисленные на экране с помощью запроса с внешним объединением

## Использование запросов с полным объединением

Стандарт SQL предлагает оператор FULL OUTER JOIN, используя который запрос выдает не только согласованные и объединенные строки двух таблиц, но и атрибуты обеих таблиц, аналоги которым в другой таблице не найдены (в таком случае “одиноким” строкам ставятся в соответствие строки со значениями NULL второй таблицы). При грамотной разработке базы данных такие запросы обычно не требуются.

## Использование запросов с объединением не по эквивалентности

При использовании объединения по эквивалентности значения в двух сопоставляемых столбцах должны совпадать. Тем не менее существуют ситуации, когда полезнее оказывается *объединение не по эквивалентности* (non-equality join, или non-equi-join) — запрос, использующий условие объединения, которое задается не оператором равенства. В качестве операторов объединения не по эквивалентности применяются BETWEEN, <, <=, >, >=, <> и =. На самом деле объединение не по эквивалентности полезно тогда, когда вы объединяете две таблицы, в которых атрибут из одной таблицы попадает в заданный диапазон значений, которые находятся во второй таблице. В качестве примера приведем следующий запрос, который находит сотрудников, занятых между двумя датами, которые берутся из другой таблицы:

```
SELECT e.LastName e.HireDate
FROM Employees e JOIN CompanyHistory h
ON e.HireDate BETWEEN h.BeginDate AND h.EndDate;
```

В предыдущем примере вы хотели перечислить фамилии сотрудников и даты их найма, причем отобрать только сотрудников, нанятых в течение определенного промежутка времени, начальный и конечный моменты которого — не предопределенные постоянные значения, а величины, извлекаемые из другой таблицы. Другой распространенный пример — использование объединения не по эквивалентности для поиска в одной таблице цены погрузки объекта на основе его веса, указанного в другой таблице. Объединение не по эквивалентности позволяет “согласовать” любые значения, входящие в заданный диапазон.

Менеджер Redwood Realty желает подсчитать число домов в категориях, определяемых их ценами (*AskingPrice*). Подобного рода статистика, называемая *распределением*, демонстрирует число домов в каждой ценовой категории. Ранее с помощью структуры CASE и функции COUNT вы написала запрос, дававший примерно ту же информацию. В данном случае менеджер желает, чтобы категории время от времени можно было изменять. Категории задаются таблицей, содержащей первичный ключ и столбец с указанием нижней и верхней цены. Эту таблицу, имеющую всего три столбца, в любой момент может модифицировать менеджер, указав другую нижнюю/верхнюю цену или задав более мелкое/крупное разбиение на диапазоны. С помощью объединения не по эквивалентности вы можете сравнить значение *AskingPrice* в таблице *Listings* со значениями нижней/верхней цены в дополнительной таблице. Подсчет элементов в категории означает использование функции COUNT для каждой пары значений и суммирование полученных чисел. Итак, первое, что вы должны сделать, — это создать таблицу, которая будет хранить нижнюю и верхнюю цену диапазона. Один из вариантов ее содержимого, используемого для расчета частотного распределения, приведен в табл. 6.1. Таблица имеет первичные ключи, так что менеджер может добавлять, удалять или обновлять таблицу, если условия на собираемую статистику потребуется изменить.

Мы начнем с создания таблицы *PriceCat*, которая хранит в базе данных эквивалент табл. 6.1. Завершив эту таблицу, вы можете писать запрос, с помощью объединения не по эквивалентности генерирующий распределение цен.

Для того чтобы создать таблицу распределения ценовых категорий и отобразить на экране ее содержимое, выполните следующие действия.

1. Найдите файл сценария Ch06PriceRange.sql в файлах главы 6 и запишите путь к нему.
2. Если необходимо, запустите SQL\*Plus. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить “рабочую поверхность” SQL\*Plus. Затем введите и выполните следующую команду, подставив вместо <path> реальный путь к файлу сценария (включая букву диска).

**ТАБЛИЦА 6.1.** Таблица распределения запрашиваемых цен

CategoryID	LowLimit	HighLimit
1000	\$0	\$50,000
1010	\$50,000	\$100,000
1020	\$100,000	\$150,000
1030	\$150,000	\$200,000
1040	\$200,000	\$250,000
1050	\$250,000	\$300,000
1060	\$300,000	\$350,000
1070	\$350,000	\$400,000
1080	\$400,000	\$2,000,000

```
START <path>\Ch06PriceRange
```

Oracle создаст и заполнит таблицу PriceCat. Появится сообщение “PriceCat table created and populated”.

- Чтобы отобразить на экране содержимое таблицы, введите и выполните следующую команду:

```
SELECT * FROM PriceCat ORDER BY LowLimit;
```

- Откройте программу Блокнот и введите нижеприведенный код. Затем скопируйте его в SQL\*Plus. Таким образом, если вдруг потребуется, вы сможете легко исправить ошибки в Блокноте и повторно скопировать код в SQL\*Plus для выполнения.

```
SELECT c.LowLimit, c.HighLimit, COUNT(AskingPrice) AS "Frequency"
FROM Listings s JOIN PriceCat c
    ON s.AskingPrice >= c.LowLimit AND
        s.AskingPrice < c.HighLimit
GROUP BY c.LowLimit, c.HighLimit
ORDER BY c.LowLimit;
```

Oracle отобразит на экране частотное распределение цен (рис. 6.14). Обратите внимание на использование неравенства в фразе ON (операторы сравнения  $\geq$  и  $<$ ).

- Вы можете немного улучшить восприятие информации, включив в рассмотрение другие категории, в которые не попадает ни один дом. Напомним, что участие в объединении строки со значением NULL (объединение по эквивалентности или не по эквивалентности) автоматически выводит эту строку из рассмотрения. Поэтому мы изменим выражение SELECT, заменив оператор внутреннего объединения JOIN оператором правостороннего внешнего объединения RIGHT OUTER JOIN. Выполните приведенное ниже модифицированное выражение SELECT.

```
SELECT c.LowLimit, c.HighLimit, COUNT(AskingPrice) AS "Frequency"
FROM Listings s RIGHT OUTER JOIN PriceCat c
    ON s.AskingPrice >= c.LowLimit AND
```

**ТАБЛИЦА 6.2.** Операторы SQL действий над множествами

<i>Оператор действия над множествами</i>	<i>Описание</i>
UNION	Возвращает все уникальные строки, извлеченные запросами
UNION ALL	Возвращает все строки, извлеченные запросами, включая все дублирующиеся строки
INTERSECT	Возвращает строки, извлеченные обоими запросами
MINUS	Возвращает строки, которые оставались после удаления (вычитания) строк, извлеченных вторым запросом, из строк, извлеченных первым запросом

```
s.AskingPrice < c.HighLimit
GROUP BY c.LowLimit, c.HighLimit
ORDER BY c.LowLimit;
```

Отличия файлов сценариев и результатов, получаемых при их выполнении, показаны на рис. 6.15. Обратите внимание на то, что в две категории не попало ни одного дома, поэтому соответствующая частота равна нулю.

6. Все, больше таблица PriceCat вам не нужна. Чтобы удалить ее, введите и выполните следующую команду:

```
DROP TABLE PriceCat CASCADE CONSTRAINTS PURGE;
```

7. Введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из SQL\*Plus. Запишите открытые в Блокноте файлы сценария и закройте эту программу.

## Операторы действий над множествами

Иногда бывает нужно объединить результаты, полученные от нескольких выражений SELECT, в единое целое. Для реализации этой возможности в SQL предусмотрены *операторы действий над множествами*. Результат каждого выражения SELECT считается множеством, и вы можете объединять эти множества, используя операторы UNION, UNION ALL, MINUS и INTERSECT. Краткая сводка по этим операторам приведена в табл. 6.2.

Выражения SQL, содержащие операторы действия над множествами, называются *сложными запросами* (compound query), а каждый отдельный запрос именуется *составляющим* (component query). При работе со сложными запросами следует учитывать несколько указанных ниже правил и ограничений.

Oracle SQL\*Plus

```

SQL> START C:\CH06\PriceRange
PriceCat table created and populated.
SQL> SELECT *
  2  FROM PriceCat
  3 ORDER BY LowLimit;

CATEGORYID  LOWLIMIT  HIGHLIMIT
-----  -----  -----
1000          0      50000
1010      50000     100000
1020     100000     150000
1030     150000     200000
1040     200000     250000
1050     250000     300000
1060     300000     350000
1070     350000     400000
1080     400000     2000000

9 rows selected.

SQL> SELECT c.LowLimit, c.HighLimit, COUNT(AskingPrice) AS "Frequency"
  2  FROM Listings s JOIN PriceCat c
  3  ON s.AskingPrice >= c.LowLimit AND
  4      s.AskingPrice <  c.HighLimit
  5 GROUP BY c.LowLimit, c.HighLimit
  6 ORDER BY c.LowLimit;

LOWLIMIT  HIGHLIMIT  Frequency
-----  -----  -----
100000      150000       318
150000      200000        149
200000      250000        23
250000      300000         8
300000      350000         4
350000      400000         1
400000      2000000        7

7 rows selected.

SQL>

```

Условие "не-объединение по эквивалентности"

Рис. 6.14. Создание таблицы PriceCat и запуск запроса с объединением не по эквивалентности

- Тип данных каждого столбца второго запроса должен быть идентичным типу данных соответствующего столбца первого запроса.
- Оба составляющие запроса должны возвращать одинаковое число столбцов.
- Первое выражение SELECT определяет имена столбцов множества результатов.
- Если в составляющем запросе вы желаете использовать оператор ORDER BY, его необходимо поместить в конец всего выражения.

Иногда составляющие запросы называются *вертикальными объединениями* (vertical join), поскольку извлекаемые строки формируются на основе столбцов, а не строк. Рассмотрим теперь синтаксис операции над множествами:

```
<component query>
{UNION | UNION ALL | INTERSECT | MINUS}
<component query>
```

Используя приведенную структуру, вы выбираете один из операторов UNION, UNION ALL, INTERSECT или MINUS. Сложные запросы, состоящие из трех выражений SELECT, содержат два оператора действий над множествами; состоящие из четырех выражений SELECTS – три оператора и т.д.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT c.LowLimit, c.HighLimit, COUNT(AskingPrice) AS "Frequency"
  2  FROM Listings s JOIN PriceCat c
  3  ON s.AskingPrice >= c.LowLimit AND
  4      s.AskingPrice < c.HighLimit
  5 GROUP BY c.LowLimit, c.HighLimit
  6 ORDER BY c.LowLimit;
  7
    LOWLIMIT HIGHLIMIT Frequency
    -----
  100000  150000       310
  150000  200000        149
  200000  250000        23
  250000  300000         8
  300000  350000         4
  350000  400000         1
  400000  2000000        7
  7 rows selected.

SQL> SELECT c.LowLimit, c.HighLimit, COUNT(AskingPrice) AS "Frequency"
  2  FROM Listings s RIGHT OUTER JOIN PriceCat c
  3  ON s.AskingPrice >= c.LowLimit AND
  4      s.AskingPrice < c.HighLimit
  5 GROUP BY c.LowLimit, c.HighLimit
  6 ORDER BY c.LowLimit;
  9
    LOWLIMIT HIGHLIMIT Frequency
    -----
     0      50000       0
  50000  100000       0
  100000  150000       310
  150000  200000        149
  200000  250000        23
  250000  300000         8
  300000  350000         4
  350000  400000         1
  400000  2000000        7
  9 rows selected.

SQL>

```

**Рис. 6.15.** Сравнение запроса с объединением не по эквивалентности и правостороннего внешнего запроса с объединением не по эквивалентности

## Использование операторов UNION и UNION ALL

Напомним, что в главе 3 рассказывалось, как просмотреть информацию о принадлежащих вам таблицах. В главе 4 есть раздел, в котором описано, как перечислить обработчики событий, созданные в вашей схеме. Иногда бывает полезно просмотреть все объекты, которые принадлежат вам в вашей схеме. Для такого типа интеграции информации (объединения строк, извлекаемых различными запросами SELECT, и придания им удобной структуры) прекрасно подходит оператор UNION. Ниже приведен сложный запрос, извлекающий информацию о ваших таблицах, обработчиках событий и последовательностях.

```

SELECT Table_Name AS "Object", 'Table' AS "Type"
FROM User_Tables
UNION
SELECT Sequence_Name "Object", 'Sequence' AS "Type"
FROM User_Sequences
ORDER BY 2,1;

```

Первый составляющий запрос извлекает имена таблицы и константу “Table” (чтобы идентифицировать их при окончательном выводе информации на экран). Второй составляющий запрос извлекает имена последовательности и константу “Sequence” для идентификации этих строк. Затем Oracle сортирует получающееся объединение

строк, возвращаемых двумя составляющими запросами. Посмотрим, как это выглядит на практике. Правда, перед этим вам придется создать последовательности и обработчики событий, информацию о которых мы потом извлечем с помощью запросов.

Чтобы в своей схеме создать обработчики событий и последовательности, необходимо выполнить следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle и введите CLEAR SCREEN, чтобы очистить экран.
2. Найдите сценарий Ch06CreateObjects.sql и выполните его, набрав приведенную ниже команду. Вместо <path> подставьте реальный путь к файлу (включая имя диска) и нажмите клавишу <Enter>.

```
START <path>\Ch06CreateObjects
```

Oracle создаст две последовательности и два обработчика событий.

Теперь вы готовы создать сложный запрос, используя оператор UNION для перечисления на экране всех принадлежащих вам объектов. Необходимая последовательность действий описана ниже. Возможно, вы захотите сохранить набранный код как файл сценария, поэтому используйте Блокнот и запишите окончательный вариант команд.

Для того чтобы использовать оператор UNION для объединения результатов, поступивших в ответ на несколько запросов, выполните следующие действия.

1. Введите следующий код в программе Блокнот, скопируйте его в SQL\*Plus и выполните.

```
CLEAR SCREEN
SELECT Table_Name AS "Object", 'Table' AS "Type"
FROM User_Tables
WHERE Table_Name NOT LIKE 'BIN%'
      UNION
SELECT Sequence_Name "Object", 'Sequence' AS "Type"
FROM User_Sequences
      UNION
SELECT Trigger_Name "Object", 'Trigger' AS "Type"
FROM User_Triggers
ORDER BY 2,1;
```

Oracle вернет три типа строк: отвечающие последовательностям, таблицам и обработчикам событий (рис. 6.15). С помощью псевдонимов столбцы, возвращающиеся тремя составляющими запросами, получили одинаковые имена. Кроме того, с помощью оператора WHERE были отфильтрованы имена таблиц Oracle, начинающиеся с “BIN”.

The screenshot shows the Oracle SQL\*Plus interface with the following command and output:

```

SQL> SELECT Table_Name AS "Object", 'Table' AS "Type"
  2  FROM User_Tables
  3  WHERE Table_Name NOT LIKE 'BIN%'
  4  UNION
  5  SELECT Sequence_Name "Object", 'Sequence' AS "Type"
  6  FROM User_Sequences
  7  UNION
  8  SELECT Trigger_Name "Object", 'Trigger' AS "Type"
  9  FROM User_Triggers
 10 ORDER BY 2,1;

Object          Type
-----          -----
ACENTID_SEQ     Sequence
CUSTOMER_SEQ    Sequence
AGENTS          Table
CONTACTREASON   Table
CUSTAGENTLIST   Table
CUSTOMERS        Table
LICENSESTATUS   Table
LISTINGS         Table
PROPERTIES       Table
SALESTATUS       Table
AGENTS_BI_TRG   Trigger

Object          Type
-----          -----
CUSTOMERS_BI_TRG Trigger

12 rows selected.

SQL>

```

**Рис. 6.16.** Использование оператора UNION в сложном запросе

**Совет.** Разумеется, таблицы, обработчики событий и последовательности, представленные в вашем списке, могут отличаться от приведенных на рис. 6.16, поскольку вы могли создать дополнительные объекты, кроме 12, показанных на рисунке.

2. Введите следующий код, чтобы удалить последовательности и обработчики событий, созданные Oracle на этапе 1:

```

DROP SEQUENCE AgentID_seq;
DROP SEQUENCE Customer_seq;
DROP TRIGGER Agents_bi_trg;
DROP TRIGGER Customers_bi_trg;

```

Oracle удалит последовательности и обработчики событий.

Оператор UNION ALL подобен оператору UNION, только он не удаляет дублирующиеся строки. Иногда UNION ALL полезен для просмотра *всех* строк из нескольких выражений SELECT (с учетом дубликатов).

## Использование оператора MINUS

Результатом действия оператора MINUS являются строки, оставшиеся после вычитания строк, возвращенных вторым запросом, из строк, возвращенных первым. Чтобы понять данную концепцию, рассмотрим очень простую пару однотабличных таблиц. Предположим, что таблица Table1 содержит значения A, C, D, F, G, H и Z, а таблица

Table2 – значения A, B, C, H и W. Приведенное ниже выражение SQL вернет значения D, F, G и Z – те записи Table1, которых нет в Table2. “Вычитаемыми” (удаляемыми из Table1) значениями являются A, C и H – записи, которые присутствуют в обеих таблицах.

```
SELECT *
FROM Table1
MINUS
SELECT *
FROM Table2;
```

Когда же оператор MINUS может потребоваться в базе данных Redwood Realty? Его можно использовать, например, для “вычитания” строк, соответствующих продаваемой недвижимости в Орике, из строк, соответствующих клиентам, живущим в Орике. В результате вы получите список жителей Орика, которые не продают свое жилье. По сути, запрос выглядит как “Адреса клиентов, живущих в Орике” минус “Адреса домов в Орике, выставленных на продажу”. Запрос, выдающий на экран требуемую информацию, выглядит следующим образом (при желании вы можете поэкспериментировать с ним самостоятельно):

```
SELECT Address, City
FROM Customers
WHERE City = 'Orick'
MINUS
SELECT Address, City
FROM Properties
WHERE City = 'Orick';
```

Если вы выполните данный запрос, Oracle выведет на экран 53 строки. Такой результат выглядит разумным, поскольку таблица *Customers* базы данных содержит записи о 62 жителях Орика, а число домов, выставленных на продажу, равно 9 ( $62 - 9 = 53$ ). Тем не менее в общем случае такое совпадение не обязательно, так как некоторые владельцы домов, выставленных на продажу, могут жить в других городах или штатах.

**Совет.** Существует небольшая уловка, позволяющая гарантировать идентичность двух таблиц: использование операторов MINUS и UNION ALL. Предположим, что *GoodTable* и *TestTable* – две таблицы, которые должны иметь одинаковое содержимое (в качестве эталона используется таблица *GoodTable*). Ниже приведен запрос, который не возвращает ни одной строки, если таблицы имеют идентичное содержимое. Если таблицы отличаются, на экран выводятся не согласующиеся строки.

```
(SELECT * FROM GoodTable MINUS SELECT * FROM TestTable)
UNION ALL
(SELECT * FROM TestTable MINUS SELECT * FROM GoodTable);
```

(Круглые скобки являются обязательными, поскольку они задают, что два составляющих запроса должны вычислять перед выполнением оператора UNION ALL.)

## Использование оператора INTERSECT

Оператор `INTERSECT` возвращает строки, извлекаемые обоими запросами, т.е. строки, которые имеются и в одной, и в другой таблице. Предположим, что нам требуется сравнить таблицу `Customers`, содержащую имена и адреса всех людей, контактировавших с Redwood Realty, и таблицу `Properties`, содержащую адреса продаваемой недвижимости. В частности, вам требуется узнать, какие клиенты (таблица `Customers`) живут в домах, выставленных на продажу (таблица `Properties`). Чтобы ограничить результаты до нормального размера, мы рассмотрим только тех клиентов, которые живут в Орике. Одно из возможных решений поставленной задачи заключается в использовании оператора `INTERSECT`. Ниже приведен запрос, возвращающий адреса и города домов, фигурирующих и в таблице `Customers`, и в таблице `Properties`, — т.е. пересечение данных таблиц. Если вы запустите данный запрос в SQL\*Plus, то получите десять записей, удовлетворяющих поставленным условиям.

```
SELECT Address, City
FROM Customers
WHERE City = 'Orick'
INTERSECT
SELECT Address, City
FROM Properties
WHERE City = 'Orick';
```

## Использование подзапросов

*Подзапрос* — это запрос, который содержится в другом выражении SQL (часто это еще один запрос). Он представляет собой завершенное выражение `SELECT`, вложенное в другое выражение SQL. Рассмотрим пример подзапроса. Предположим, что нам требуется перечислить агентов (`Agents`), нанятых компанией Redwood Realty после найма Тобиаса Карлинга (`Tobias Carling`). Прочитав только предыдущие главы, вы бы могли решить эту задачу в два этапа: (1) организовать запрос, возвращающий дату найма (`HireDate`) Тобиаса Карлинга из таблицы `Agents`; а затем (2) выполнить запрос, перечисляющий всех агентов, значение `HireDate` которых больше найденного на первом этапе. Подзапрос можно использовать в DML<sup>2</sup>-выражениях. Представьте, что вам требуется удалить (уволить) всех агентов, нанятых перед Тобиасом Карлингом. Данную задачу можно решить с помощью одной команды, использовав в операторе `WHERE` выражения `DELETE` подзапрос для получения даты найма Карлинга.

В данной главе описывается, как с помощью подзапроса извлекать требуемые результаты “в одно действие”. Подзапросы могут принадлежать к одному из двух основных типов: однострочные или многострочные подзапросы. *Однострочные подзапросы* передают внешнему выражению SQL нуль или одну строку. *Многостроч-*

---

<sup>2</sup>Data Manipulation Language — язык манипулирования данными.

ные запросы передают внешнему выражению SQL две или больше строк. В свою очередь, из двух указанных категорий выделяют еще *многостолбцовые* запросы, возвращающие внешнему выражению SQL несколько столбцов, и *связанные* (correlated) подзапросы, ссылающиеся на один или несколько столбцов внешнего выражения.

## Однострочные подзапросы

Простейшим подзапросом является однострочный и одностолбцовый, именуемый также *скалярным*, поскольку он возвращает одно значение. Скалярный подзапрос можно поместить в любом месте выражения SQL, где допускается применение строк: например, его можно использовать как одно из имен списка в операторе SELECT; в операторе FROM; в операторе WHERE или HAVING. Например, в приведенном ниже скалярном подзапросе перечислены агенты, возраст которых больше среднего возраста всех агентов.

```
SELECT FirstName, LastName
  FROM Agents
 WHERE MONTHS_BETWEEN(SYSDATE, BirthDate) >
    (SELECT AVG(MONTHS_BETWEEN(SYSDATE, BirthDate))
      FROM Agents);
```

Первым выполняется подзапрос (выражение SELECT в круглых скобках после знака “больше”). Он возвращает средний возраст (в месяцах) всех агентов. Далее это число ставится вместо подзапроса и выполняется внешний запрос. Оператор WHERE отбирает строки, в которых возраст агента превышает средний возраст всех агентов. Результатом выполнения запроса, подобного данному, будет список приблизительно из половины агентов.

Для того чтобы написать и выполнить скалярный подзапрос, выполните следующие действия.

- Если требуется, запустите SQL\*Plus и войдите в Oracle. Запустите Блокнот, чтобы использовать его в качестве текстового редактора. Введите в Блокноте следующий код, скопируйте его в SQL\*Plus и выполните, чтобы перечислить недвижимость, которая по крайней мере на 18 лет “младше” среднего “возраста” домов Аркаты:

```
CLEAR SCREEN
SELECT Address, YearBuilt
  FROM Properties
 WHERE City = 'Arcata' AND
       YearBuilt > (SELECT AVG(YearBuilt)
                      FROM Properties
                     WHERE City = 'Arcata')+18
 ORDER BY Address;
```

- Ведите следующий код, чтобы перечислить агентов по недвижимости, имеющих то же звание, что и Джессика Тейлор (Jessica Taylor). (Функция UPPER используется для того, чтобы мы не задумывались о прописных/строчных буквах записей.)

```

SQL> SELECT Address, YearBuilt
  2  FROM Properties
  3 WHERE City = 'Arcata' AND
  4   YearBuilt > (SELECT AVG(YearBuilt)
  5   FROM Properties
  6   WHERE City = 'Arcata')+18
  7 ORDER BY Address;

ADDRESS          YEARBUILT
-----          -----
1205 E California Ave      1996
123 Barley Rd             1996
1781 Buttermilk Ln        1996
2095 Appaloosa Ln         1994
3300 Rieber Ln            1994

SQL> COLUMN FirstName FORMAT A10
SQL> COLUMN LastName FORMAT A10
SQL> COLUMN Title FORMAT A11
SQL> SELECT FirstName, LastName, Title
  2  FROM Agents
  3 WHERE UPPER>Title) =
  4   (SELECT UPPER>Title)
  5   FROM Agents
  6 WHERE LOWER(FirstName) = 'jessica'
  7   AND LOWER(LastName) = 'taylor';

FIRSTNAME    LASTNAME    TITLE
-----        -----    -----
Essi          Okindo     Broker
Ricki         Selby      Broker
Jessica       Taylor     Broker
David          Gagnon    Broker
Lora          Allee      Broker
Patricia      Lewis     Broker

6 rows selected.

SQL>

```

Вывести на экран информацию по домам, которые на 18 лет "молодже" среднего "возраста" недвижимости Арката

Отобразить агентов, имеющих то же звание, что и Джессика Тейлор

Рис. 6.17. Использование подзапросов в операторах WHERE

```

COLUMN FirstName FORMAT A10
COLUMN LastName FORMAT A10
COLUMN Title FORMAT A11
SELECT FirstName, LastName, Title
FROM Agents
WHERE UPPER>Title) =
  (SELECT UPPER>Title)
  FROM Agents
  WHERE LOWER(FirstName) = 'jessica'
    AND LOWER(LastName) = 'taylor';

```

Результаты выполнения описанных действий представлены на рис. 6.17.

## Многострочные подзапросы

К многострочным относятся подзапросы, которые *могут* возвращать внешнему запросу больше одной строки. Обычно многострочные запросы появляются в операторах WHERE и HAVING. Чтобы правильно обработать многострочный подзапрос, внешний (или родительский) запрос использует многострочный оператор IN, ANY или ALL. Напомним, что оператор IN пытается сопоставить столбец со списком значений, помещенным в круглые скобки. Оператор ANY сопоставляет одно значение с *любым*

значением списка; оператор ALL сравнивает одно значение со всеми значениями списка. Помните, что вы обязаны использовать многострочный оператор с подзапросом, который может вернуть несколько строк. Операторы сравнения, которые вы использовали до этого момента, в том числе  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ ,  $=$  и  $\neq$ , приведут к ошибке, если подзапрос вернет несколько строк. Например, приведенная ниже команда, скорее всего, приведет к генерации сообщения об ошибке, поскольку подзапрос может вернуть несколько дат рождения агентов-мужчин.

```
SELECT LastName
  FROM Agents
 WHERE Birthdate > (SELECT Birthdate FROM Agents
                      WHERE Gender = 'M');
```

Получаем сообщение об ошибке ORA-01427: *single-row subquery returns more than one row* (“однострочный запрос возвращает более одной строки”), из которого следует, что оператор “больше” ( $>$ ) может сравнивать одно значение только с одним другим, но не со списком значений.

### **Использование оператора IN с многострочным запросом**

Оператор IN полезен, когда необходимо проверить, совпадает ли значение со списком значений другой таблицы (который генерируется подзапросом). Кроме того, вы можете использовать оператор NOT IN, чтобы найти значение, отсутствующее в списке значений. Например, приведенный ниже запрос содержит два подзапроса, определяющих, какие детали, пришедшие от некоторого поставщика, были произведены в г. Энчино (Encino).

```
SELECT * FROM Parts
 WHERE PartNumber IN (SELECT PartNumber FROM Shipments
                       WHERE SupplierNumber IN
                           (SELECT SupplierNumber FROM Suppliers
                            WHERE UPPER(City) = 'ENCINO'));
```

Самый внутренний запрос возвращает значения SupplierNumber (идентификационный номер поставщика), в столбце City имеющие значение Encino. Данные значения сравниваются с SupplierNumber и используются для отбора значений PartNumber из таблицы Shipments. Далее значения PartNumber из таблицы Parts сравниваются со списком значений PartNumber из таблицы Shipments. Далее на экране отображаются все столбцы таблицы Parts и строки, в которых значения PartNumber соответствуют поставкам от производителя, чей завод расположен в Энчино.

Вернемся теперь к базе данных Readwood Realty. Чтобы перечислить фамилии агентов, не имеющих в работе недвижимости на продажу, необходимо выполнить следующие действия.

1. Откройте Блокнот, чтобы вы могли записать все строки сценария, введенного в ходе описанных ниже этапов.
2. Введите сценарий в Блокноте, скопируйте его в SQL\*Plus и выполните. Выражение SELECT определяет, с какими агентами в настоящее время не связаны клиенты, покупающие или продающие дома.

```
CLEAR SCREEN
SELECT DISTINCT FirstName, LastName
FROM Agents
WHERE AgentID NOT IN (SELECT AgentID FROM CustAgentList);
```

Подзапрос возвращает список идентификаторов агентов из таблицы CustAgentList, которые соответствуют агентам, работающим с клиентами. Далее со значениями из этого списка сравниваются значения AgentID из таблицы Agents. В качестве результата запроса выводятся имена агентов, отсутствующие в списке.

3. Перечислите имена и фамилии агентов, имеющих клиентов в Орике (Orick). Приведенный код, по сути, является альтернативой объединения трех таблиц.

```
SELECT DISTINCT FirstName, LastName
FROM Agents
WHERE AgentID IN (SELECT AgentID FROM CustAgentList
                   WHERE CustomerID IN (SELECT CustomerID
                                         FROM Customers
                                         WHERE City = 'Orick'));
```

4. Введите и выполните модифицированную версию предыдущего запроса, чтобы отобразить на экране фамилии агентов по продаже, т.е. агентов, представляющих интересы продавцов. (Помните: если вы хотите, чтобы к сценарию можно было вернуться позже, наберите его в Блокноте и сохраните. После этого скопируйте необходимый фрагмент кода в SQL\*Plus и выполните его.)

```
SELECT DISTINCT a.FirstName, a.LastName
FROM Agents a JOIN CustAgentList c USING (AgentID)
WHERE c.ContactReason = 'Sell'
      AND c.CustomerID IN (SELECT CustomerID FROM CUSTOMERS
                                WHERE UPPER(City) = 'ORICK');
```

Результат выполнения приведенных команд представлен на рис. 6.18. Обратите внимание на то, что только два агента по недвижимости имеют клиентов в Орике, продающих свои дома. Впрочем, количество таких клиентов может быть большим.

### **Использование операторов ANY и ALL**

Операторы ANY и ALL можно использовать для сравнения одной величины со списком значений. Операторы ANY или ALL можно комбинировать со всеми операторами сравнения (>, >=, <, <=, = или <>). Поскольку результатом подзапроса является не отдельное значение, а список значений, перед командой ANY или ALL вашего запроса должен стоять один из операторов сравнения. Для примера в табл. 6.3 приведены значения операторов ANY и ALL, используемых с некоторыми операторами сравнения.

Прежде чем мы приступим к изучению операторов ANY и ALL, необходимо установить новую таблицу Redwood Realty —AgentsHR. Она содержит более подробную информацию об агентах, включая число материально зависимых лиц, базовый оклад, семейное положение, номер социального обеспечения, расовую принадлежность, количество недель отпуска и команду по продажам, к которой они относятся в Redwood

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT DISTINCT FirstName, LastName
  2  FROM Agents
  3 WHERE AgentID NOT IN (SELECT AgentID FROM CustAgentList);

FIRSTNAME          LASTNAME
-----            -----
George             Nagy
Robert            Sellsmore
Susan              Swarthmore

SQL> SELECT DISTINCT FirstName, LastName
  2  FROM Agents
  3 WHERE AgentID IN (SELECT AgentID FROM CustAgentList
  4   WHERE CustomerID IN (SELECT CustomerID
  5     FROM Customers
  6   WHERE City = 'Orick'));

FIRSTNAME          LASTNAME
-----            -----
Cecilia            Romero
Cornelis           Dann
Crystal            Fernandez
Danial             Silverburg
David              Gagnon
Lee                Reed
Sindisiwe          Weber
Tim                Schutz

8 rows selected.

SQL> SELECT DISTINCT a.FirstName, a.LastName
  2  FROM Agents a JOIN CustAgentList c USING (AgentID)
  3 WHERE c.ContactReason = 'Sell'
  4   AND c.CustomerID IN (SELECT CustomerID FROM CUSTOMERS
  5   WHERE UPPER(City) = 'ORICK');

FIRSTNAME          LASTNAME
-----            -----
Cecilia            Romero
David              Gagnon
Lee                Reed

SQL>

```

**Рис. 6.18.** Примеры использования оператора IN и подзапросов

**ТАБЛИЦА 6.3.** Описание операторов ANY И ALL

<i>Оператор</i>	<i>Значение</i>
=ANY	Равно любому значению, возвращаемому подзапросом. То же самое, что и оператор IN
<ANY	Меньше наибольшего значения, возвращаемого подзапросом
>ANY	Больше наименьшего значения, возвращаемого подзапросом
>ALL	Больше наибольшего значения, возвращаемого подзапросом
<ALL	Меньше наименьшего значения, возвращаемого подзапросом

Realty. Всего в компании имеется пять команд, организованных по территориальному принципу, — в одну команду входят сотрудники, лучше всего знакомые с конкретным регионом. Итак, прежде всего вам нужно найти файл сценария Ch06AgentsHR.sql. Запишите путь к нему или скопируйте его в удобную папку и запомните новый путь к нему.

Для того чтобы добавить новую таблицу в базу данных Redwood Realty, выполните следующие действия.

The screenshot shows the Oracle SQL\*Plus interface with the following content:

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> DESCRIBE AgentsHR
Name          Null?    Type
-----        -----
AGENTID      NOT NULL NUMBER(38)
DEPENDENTS   NUMBER(38)
BASESALARY   NUMBER(38)
MARITAL      NCHAR(2)(1)
SSN          NCHAR(11)
ETHNICITY    NUMBER(38)
VACATION     NUMBER(38)
TEAM         NUMBER(38)

SQL> SELECT * FROM AgentsHR;

```

AGENTID	DEPENDENTS	BASESALARY	M	SSN	ETHNICITY	VACATION	TEAM
10041	5	3051	M	193-82-2099	3	4	14
10235	1	2642	S	299-39-6678	1	2	11
10429	4	2985	S	113-60-1549	1	4	13
10497	1	4377	M	306-16-5360	4	2	14
10849	4	2585	D	313-36-4389	2	4	14
10913	2	3493	D	486-11-3733	4	4	13
11775	4	2216	M	817-28-2698	1	4	14
12211	2	3520	D	327-25-8628	1	2	14
12301	2	3637	D	152-64-9996	1	2	14
12499	1	3413	W	675-31-4312	2	3	13
12715	1	2066	M	789-98-2311	1	3	11

AGENTID	DEPENDENTS	BASESALARY	M	SSN	ETHNICITY	VACATION	TEAM
12765	2	2360	W	667-78-9822	2	2	11
12875	2	2343	M	861-49-4737	1	3	14
12963	2	4433	S	944-32-6394	4	3	14
13353	4	4064	S	687-13-4756	2	3	14
13555	2	3420	M	928-51-8656	1	4	10
13649	1	4171	D	679-68-1774	3	2	14
13771	5	3838	M	312-74-1536	1	2	10
14117	1	2567	M	934-97-5821	2	3	14
14447	3	3723	V	393-17-5266	5	3	12
14599	4	4263	M	251-32-2456	5	3	11
14681	3	2135	W	487-39-2872	1	2	12

AGENTID	DEPENDENTS	BASESALARY	M	SSN	ETHNICITY	VACATION	TEAM
12765	2	2360	W	667-78-9822	2	2	11
12875	2	2343	M	861-49-4737	1	3	14
12963	2	4433	S	944-32-6394	4	3	14
13353	4	4064	S	687-13-4756	2	3	14
13555	2	3420	M	928-51-8656	1	4	10
13649	1	4171	D	679-68-1774	3	2	14
13771	5	3838	M	312-74-1536	1	2	10
14117	1	2567	M	934-97-5821	2	3	14
14447	3	3723	V	393-17-5266	5	3	12
14599	4	4263	M	251-32-2456	5	3	11
14681	3	2135	W	487-39-2872	1	2	12

(дополнительные  
строки не видны)

Рис. 6.19. Описание и содержимое таблицы AgentsHR

- В SQL\*Plus введите следующий код, подставив вместо <path> реальный путь к файлу сценария Ch06AgentsHR.sql.

```
START <path>\Ch06AgentsHR
```

Oracle создаст таблицу и заполнит ее дополнительной информацией об агентах.

- Ведите следующий код, чтобы отобразить на экране структуру и содержимое новой таблицы.

```
CLEAR SCREEN
DESCRIBE AgentsHR
SELECT * FROM AgentsHR;
```

В результате вы должны увидеть то же, что показано на рис. 6.19.

Новая таблица содержит немного новой числовой информации. На ее примере мы изучим использование операторов ANY и ALL для выполнения сравнительного анализа окладов и различных средних значений. Для начала вы создадите запрос с подзапросом, определяющий, какие агенты имеют большую зарплату, чем самый высокооплачиваемый агент команды 10. Очевидно, что при этом из рассмотрения исключают всех агентов десятой команды, поскольку мы ищем людей, зарабатывающих

больше самого высокооплачиваемого агента этой группы. Разумеется, для решения поставленной задачи лучше всего подходит оператор ALL.

Для того чтобы использовать в подзапросе операторы ANY и ALL, выполните следующие действия.

1. Используя Блокнот в качестве текстового редактора, введите следующий код, а затем скопируйте его в SQL\*Plus и выполните:

```
CLEAR SCREEN
COLUMN BaseSalary FORMAT $9,999
SELECT AgentID, BaseSalary, Team
FROM AgentsHR
WHERE BaseSalary >ALL (SELECT BaseSalary
                           FROM AgentsHR
                           WHERE Team = 10);
```

Oracle вернет пять строк, указывающих, что только 5 из 32 агентов компании зарабатывают больше любого агента десятой команды.

2. Немного модифицируйте запрос, чтобы перечислить агентов (по крайней мере, их номера), *не принадлежащих к команде 10*, но зарабатывающих больше *какого-нибудь* агента команды 10. Отредактируйте предыдущую команду SELECT, заменив >ALL на >ANY, добавив в оператор WHERE условие AND Team <> 10 и снова запустив запрос. Модифицированное выражение SELECT должно выглядеть следующим образом:

```
SELECT AgentID, BaseSalary, Team
FROM AgentsHR
WHERE BaseSalary >ANY (SELECT BaseSalary modified line
                           FROM AgentsHR
                           WHERE Team = 10)
AND Team <> 10; modified line
```

Oracle отобразит на экране одиннадцать строк, указывающих агентов, не принадлежащих команде 10, но зарабатывающих больше самого низкооплачиваемого агента этой команды (рис. 6.20).

3. SQL\*Plus и Блокнот не закрывайте.

С точки зрения практики было бы интересно посмотреть на максимальную зарплату команды 10, чтобы можно было легко проверять результаты запросов, подобных выполненным ранее. Давайте объединим таблицу Agents с AgentsHR, чтобы отобразить на экране не идентификационные номера, а имена агентов. Кроме того, используем в списке отбора внешнего запроса подзапрос SELECT, чтобы рассчитать и отобразить на экране максимальную зарплату десятой команды. Наконец, вы выведете на экран оклад всех сотрудников, зарабатывающих больше самого высокооплачиваемого агента десятой команды. Данный запрос подобен тому, что делалось в предыдущем примере, только на этот раз на экран выводятся имена агентов и максимальный оклад.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> COLUMN BaseSalary FORMAT $9,999
SQL> SELECT AgentID, BaseSalary, Team
2  FROM AgentsHR
3  WHERE BaseSalary >ALL (SELECT BaseSalary
4                            FROM AgentsHR
5                            WHERE Team = 10);
6
AGENTID BASESALARY      TEAM
-----
10497    $4,377          14
12963    $4,433          14
13353    $4,864          14
13649    $4,171          14
14599    $4,263          11
12963    $4,433          14
10497    $4,377          14
14599    $4,263          11
13649    $4,171          14
13353    $4,864          14
14447    $3,723          12
15233    $3,664          12
12301    $3,637          14
15349    $3,638          11
12211    $3,520          14
10913    $3,493          13

11 rows selected.

SQL>

```

Рис. 6.20. Использование в запросе операторов ALL и ANY

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> COLUMN Agent      FORMAT A22      HEADING 'Agent''s Name'
SQL> COLUMN BaseSalary FORMAT $9,999 HEADING 'BaseSalary'
SQL> COLUMN TeamMax     FORMAT $9,999 HEADING 'Team 10|Max. Salary'
SQL> SELECT FirstName || ' ' || LastName AS Agent, BaseSalary,
2        (SELECT MAX(BaseSalary)
3         FROM AgentsHR
4         WHERE Team = 10) AS TeamMax, Team
5  FROM Agents INNER JOIN AgentsHR USING (AgentID)
6  WHERE BaseSalary >ALL (SELECT BaseSalary
7                        FROM AgentsHR
8                        WHERE Team = 10);

Agent's Name      Base      Team 10
Agent's Name      Salary    Max. Salary   TEAM
-----           -----    -----
Ramanathan Rowe   $4,377   $3,838       14
Lee Reed          $4,433   $3,838       14
Stanislaw Soltwedel $4,864   $3,838       14
Ricki Selby        $4,171   $3,838       14
Barbara Herring    $4,263   $3,838       11

SQL>

```

Рис. 6.21. Использование двух подзапросов и оператора ALL

Чтобы отобразить на экране имена сотрудников, зарабатывающих больше самого высокооплачиваемого агента десятой команды, выполните следующие действия.

1. Введите следующий код, чтобы очистить экран и отформатировать столбцы, которые вы собираетесь отобразить на экране с помощью команды SELECT:

```
CLEAR SCREEN
COLUMN Agent FORMAT A22 HEADING 'Agent''s Name'
COLUMN BaseSalary FORMAT $9,999 HEADING 'Base|Salary'
COLUMN TeamMax FORMAT $9,999 HEADING 'Team 10|Max. Salary'
```

2. Ведите следующий запрос с *двумя* подзапросами. Не забудьте использовать Блокнот, чтобы при необходимости вы могли легко отредактировать запрос. Аккуратно введите код, скопируйте его в SQL\*Plus и выполните.

```
SELECT FirstName || ' ' || LastName AS Agent, BaseSalary,
       (SELECT MAX(BaseSalary)
        FROM AgentsHR
        WHERE Team = 10) AS TeamMax, Team
  FROM Agents INNER JOIN AgentsHR USING (AgentID)
 WHERE BaseSalary >ALL (SELECT BaseSalary
                        FROM AgentsHR
                        WHERE Team = 10);
```

Oracle вернет те же строки, что и при выполнения п. 1 предыдущего примера, только на этот раз из-за объединения двух таблиц и дополнительного подзапроса, возвращающего максимальную зарплату, вам будет доступно немного больше информации. На экране вы должны увидеть то же, что изображено на рис. 6.21.

3. Ведите CLEAR COLUMNS, чтобы удалить все форматирование столбцов.
4. Ведите exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и SQL\*Plus.
5. Запишите все, что вы набирали в Блокноте, и закройте эту программу.

## **Коррелированные подзапросы**

Коррелированные подзапросы довольно сильно отличаются от подзапросов, рассмотренных до этого момента. Поскольку коррелированный подзапрос ссылается на один или несколько столбцов внешнего запроса, Oracle выполняет внутренний запрос один раз для каждой строки внешнего запроса. Коррелированный подзапрос удобен в тех случаях, когда вам требуется ответить на вопрос, для каждой строки внешнего запроса зависящий от значения столбца внешнего запроса. Следует, однако, помнить, что хотя коррелированные подзапросы являются очень мощным инструментом, они также могут быть очень дорогими. Рассмотрим, например, приведенный ниже подзапрос, который перечисляет идентификаторы агентов, соответствующие сотрудникам, имеющим отпуска, большие среднего значения для соответствующей команды.

```
SELECT *
  FROM AgentsHr outer
 WHERE outer.Vacation >
       (SELECT AVG(Vacation)
        FROM AgentsHR inner
        WHERE outer.Team = inner.team)
 ORDER BY Team;
```

Разберем приведенный код. Первым делом Oracle отбирает строку из внешнего запроса (`SELECT * . . .`). Затем база данных определяет значение коррелированных столбцов (`outer.Team` и `inner.Team`). После этого Oracle для каждой строки внешнего запроса выполняет внутренний запрос, передает результаты внутреннего запроса внешнему и выполняет внешний запрос. Затем Oracle берет следующую строку внешнего запроса и повторяет описанную процедуру. Рассмотрите запрос внимательно, и вы поймете, почему таблицы с большим числом строк, обрабатываемые с помощью коррелированного запроса, могут быть очень дорогими. Рассмотрим, например, “стоимость” обработки, выраженную в числе извлекаемых и изучаемых строк. Если таблица содержит 1 000 строк, выполнение внутреннего запроса означает обработку 1 000 строк для *каждой* строки внешнего запроса. Поскольку таблица, используемая во внешнем запросе, также содержит 1 000 строк, общая “стоимость” равна  $1\,000 \times 1\,000 = 1\,000\,000$  строк! Даже самой быстрой базе данных потребуется значительное время, чтобы обработать такой запрос. Таким образом, коррелированные подзапросы мы рекомендуем использовать очень осторожно.

### Использование операторов сравнения

В качестве первого эксперимента с коррелированными подзапросами ответим на интересный вопрос: “Для каких домов, выставленных на продажу, средняя запрашиваемая цена больше среднего значения для этого города?” Для решения этой задачи мы используем коррелированный подзапрос и выявим дома, считающиеся дорогими в своем районе. Любой реальный сотрудник агентства недвижимости скажет вам, что подобная постановка вопроса является слишком грубой и в действительности необходимо учитывать множество факторов. Тем не менее ответ на этот вопрос стоит иметь в своем арсенале. Запустите SQL\*Plus и откройте Блокнот, подготовившись к описываемым ниже действиям. Запрос, который вы собираетесь написать, объединяет таблицы `Listings` и `Properties`, чтобы столбцы `AskingPrice` и `City` были доступны одновременно. Подзапрос также объединяет две таблицы, чтобы для каждого города можно было определить среднюю запрашиваемую цену. Поскольку в ответ на запрос вы можете получить большое число строк, внешний запрос ограничивает анализ тремя небольшими городами: Лолета (`Loleta`), Орик (`Orick`) и Тринидад (`Trinidad`). Если убрать это ограничение, запрос будет работать по-прежнему, однако вернет в ответ 194 строки.

Для того чтобы написать коррелированный подзапрос, выполните следующие действия.

1. Введите приведенный ниже код в Блокноте, а затем скопируйте его в SQL\*Plus и выполните. В данном фрагменте мы задаем форматирование столбцов.

```
CLEAR SCREEN
COLUMN Address FORMAT A30
COLUMN City FORMAT A16
COLUMN AskingPrice FORMAT $9,999,999
```

The screenshot shows the Oracle SQL\*Plus interface. The command window contains the following SQL code:

```

SQL> COLUMN Address FORMAT A30
SQL> COLUMN City FORMAT A16
SQL> COLUMN AskingPrice FORMAT $9,999,999
SQL> SELECT PropertyID, p1.Address, p1.City, l1.AskingPrice
  2  FROM Properties p1 INNER JOIN Listings l1 USING(PropertyID)
  3 WHERE l1.AskingPrice >
  4       (SELECT AVG(AskingPrice)
  5        FROM Properties p2 INNER JOIN Listings l2
  6        USING(PropertyID)
  7        WHERE p1.City = p2.City)
  8   AND p1.City IN ('Loleta', 'Orick', 'Trinidad')
  9 ORDER BY 3 ASC,4 DESC;

```

The output window displays the results of the query:

PROPERTYID	ADDRESS	CITY	ASKINGPRICE
1932	632 Pershing	Loleta	\$282,500
1679	119 Bay View Dr	Loleta	\$162,500
1574	552 Pershing	Loleta	\$154,000
1886	1242&#44; US Highway 101 N	Orick	\$184,950
298	120779 Hwy 101	Orick	\$170,266
292	281 Parker Crk Dr	Trinidad	\$165,504
1683	2078 Patricks Point Dr	Trinidad	\$155,000
193	839 Stagecoach Rd	Trinidad	\$144,853
1398	870 Patricks Point Dr	Trinidad	\$140,000
1347	1180 Park Dr	Trinidad	\$138,500
88	89 Berry Rd	Trinidad	\$132,542

11 rows selected.

SQL>

Рис. 6.22. Коррелированный подзапрос, находящий запрашиваемую цену, превышающую средний уровень

2. Введите следующий код и выполните его в SQL\*Plus, чтобы отобразить на экране дома, цена которых выше среднего уровня по сравнению с другими домами того же города:

```

SELECT PropertyID, p1.Address, p1.City, l1.AskingPrice
FROM Properties p1 INNER JOIN Listings l1 USING(PropertyID)
WHERE l1.AskingPrice >
      (SELECT AVG(AskingPrice)
       FROM Properties p2 INNER JOIN Listings l2
       USING(PropertyID)
       WHERE p1.City = p2.City)
      AND p1.City IN ('Loleta', 'Orick', 'Trinidad')
      ORDER BY 3 ASC,4 DESC;

```

На рис. 6.22 показано одиннадцать домов для трех выбранных городов, стоимость которых выше средней цены недвижимости этих городов.

3. Чтобы убрать форматирование столбцов, введите и выполните команду

```
CLEAR COLUMNS
```

Коррелированные строки представлены в операторе WHERE внизу выражения SELECT:

```
WHERE p1.City = p2.City
```

В данном фрагменте кода название города из внешнего запроса сравнивается с названием города, извлекаемым подзапросом. Если они отличаются, то значение AskingPrice внутреннего запроса не рассматривается в текущем расчете средней цены. Внутренний запрос продолжает перебирать строки объединенных таблиц Properties

ties и Listings, пока не рассмотрит их все. Затем извлекается новая строка внешнего запроса и внутренний запрос начинает перебирать строки сверху вниз.

Приведенный ранее в этой главе запрос можно немного модифицировать, чтобы получить более конкретную информацию по ставкам агентов и перечислить тех сотрудников, зарплата которых выше среднего для их команд уровня. Такой запрос отличается от приведенного ранее, поскольку зарплата сравнивается не со средним значением по фирме, а со средним значением по команде. Для решения задачи в такой постановке прекрасно подходит коррелированный подзапрос, цена которого в данном случае невелика, так как рассматривается всего 29 агентов.

Для того чтобы с помощью коррелированного подзапроса определить, кто зарабатывает больше средней зарплаты своей команды, выполните следующие действия.

1. Введите следующий код в Блокноте, скопируйте его в SQL\*Plus и выполните:

```
CLEAR SCREEN
SET PAGESIZE 30
COLUMN BaseSalary FORMAT $9,999 HEADING 'Base|Salary'
BREAK ON Team SKIP 1
```

2. Введите и выполните следующую команду:

```
SELECT Poss.Team, Poss.AgentID, Poss.BaseSalary
FROM AgentsHR Poss
WHERE BaseSalary >
  (SELECT AVG(BaseSalary)
   FROM AgentsHR Aver
   WHERE Poss.Team = Aver.Team)
ORDER BY 1,2;
```

Oracle отобразит на экране шестнадцать строк, в которых представлены агенты, имеющие зарплату выше среднего для их команд уровня. Полученные данные должны соответствовать показанным на рис. 6.23.

3. Введите и выполните следующую команду, чтобы вернуть к настройкам по умолчанию переопределенные разрывы страниц, форматирование столбцов и размер страницы:

```
CLEAR BREAKS
CLEAR COLUMNS
SET PAGESIZE 14
```

4. Запишите открытый в Блокноте текстовый файл, если вы хотите сохранить результаты своей работы. Если желаете передохнуть, можете выйти из SQL\*Plus.

## Использование оператора EXISTS

Оператор `EXISTS` используется, когда требуется определить наличие условия в подзапросе. Если условие существует, оператор `EXISTS` возвращает значение `true`, в противном случае — значение `false`. Хотя `EXISTS` можно использовать и не с коррелированными подзапросами, внимательное изучение этого оператора показывает, что

The screenshot shows the Oracle SQL\*Plus interface. The command window contains the following SQL code:

```

SQL> SET PAGESIZE 30
SQL> COLUMN BaseSalary FORMAT $9,999 HEADING 'Base|Salary'
SQL> BREAK ON Team SKIP 1
SQL> SELECT Poss.Team, Poss.AgentID, Poss.BaseSalary
  2 FROM AgentsHR Poss
  3 WHERE BaseSalary >
  4   (SELECT AVG(BaseSalary)
  5    FROM AgentsHR Aver
  6   WHERE Poss.Team = Aver.Team)
  7 ORDER BY 1,2;

```

The output displays a table with three columns: TEAM, AGENTID, and Salary. The data is grouped by TEAM. For each team, it lists all agent IDs and their salaries, followed by a single row showing the average salary for that team.

TEAM	AGENTID	Base Salary
10	13771	\$3,838
	14883	\$3,833
11	14599	\$4,263
	15861	\$3,380
	15349	\$3,630
12	14447	\$3,723
	15233	\$3,664
13	18429	\$2,985
	18913	\$3,493
	12499	\$3,413
14	18497	\$4,377
	12211	\$3,520
	12301	\$3,637
	12963	\$4,433
	13353	\$4,064
	13649	\$4,171

16 rows selected.

SQL>

**Рис. 6.23.** Коррелированный подзапрос, используемый для отображения на экране зарплат, превышающих средний уровень

его целесообразно использовать только с коррелированными подзапросами. Другими словами, оператор EXISTS интересен только тогда, когда результат подзапроса зависит от внешнего запроса (т.е. мы имеем коррелированный подзапрос). Логическим антонимом данного оператора является NOT EXISTS, который используется, когда требуется узнать, что подзапрос не нашел соответствующей строки. Поскольку оператор EXISTS проверяет сам факт наличия или отсутствия строки, столбцы в списке отбора подзапроса значения не имеют. Поэтому распространенной практикой является использование в списке отбора подзапроса констант '1' или 'X' либо ключевого слова NULL. Синтаксис типичного запроса EXISTS и дополняющего его подзапроса представлен ниже.

```

SELECT <outer select list>
FROM <outer table>
WHERE EXISTS
  (SELECT 'X'
   FROM <inner table> inner
   WHERE <outer table.columnname> = <inner table.column name>);

```

Коррелированные подзапросы можно использовать вместе с условием EXISTS и определить, что кто-то с невыполненным заказом заказал новый товар, сопоставив заказы пользователей с таблицей, представляющей состояния заказов. Подобным образом оператор EXISTS, объединенный с коррелированным подзапросом, позво-

ляет эффективно определить, курируется ли клиент агентом по недвижимости или сопоставлен ли с агентом какой-либо продаваемый дом, представленный в таблице *Listings*. Для ответа на последний вопрос можно использовать приведенную ниже команду с коррелированным подзапросом, включающую две таблицы Redwood Realty — *Agents* и *Listings* (обратите внимание, что они не объединены, как можно было бы предположить).

```
SELECT LastName, FirstName, 'No Listings' AS "Situation"
FROM Agents
WHERE NOT EXISTS (SELECT 'X'
                   FROM Listings
                   WHERE AgentID = ListingAgentID);
```

Обратите внимание на то, что список отбора подзапроса содержит константу 'X', поскольку *что* именно возвращается подзапросом в подзапрос EXISTS (или NOT EXISTS), не имеет значения. Все, что вам нужно знать, — это сам факт, имеется ли строка, которая удовлетворяет критерию подзапроса. В данном случае оператор NOT EXISTS ставит вопрос: “Существуют ли агенты по недвижимости, идентификаторы *AgentID* которых отсутствуют в таблице *Listings*?” и возвращает ответ “да”. На самом деле таких агентов трое (проверьте самостоятельно).

Для того чтобы сформулировать коррелированный подзапрос, используя оператор EXISTS, выполните следующие действия.

- Если требуется, запустите SQL\*Plus и войдите в Oracle. Введите и выполните команду CLEAR SCREEN, чтобы очистить экран SQL\*Plus.
- Ведите приведенное ниже выражение SELECT и выполните его. Данная команда подсчитает уникальных клиентов, с которыми не сопоставлен ни один сотрудник агентства.

```
SELECT COUNT(*) AS "Customers w/o agents"
FROM Customers outer
WHERE NOT EXISTS
  (SELECT 'X'
   FROM CustAgentList inner
   WHERE outer.CustomerID = inner.CustomerID);
```

Запрос вернет значение 1745, означающее, что для 1 745 клиентов, представленных в таблице *Customers*, не указаны идентификаторы агентов, перечисленных в таблице *Agents*.

- Напишите другой запрос, подсчитывающий число клиентов, не имеющих выставленных на продажу домов (таблица *Properties*). Введите и выполните следующую команду. (Хотя вы можете просмотреть имена всех клиентов, список будет длинным и его придется прокручивать.)

```
SELECT COUNT(LastName) AS "Customers w/o properties"
FROM Customers
WHERE NOT EXISTS
  (SELECT 123
```

```
FROM Properties
WHERE CustomerID = OwnerID);
```

Не пугайтесь значения 123, указанного в списке отбора подзапроса. Мы указали его лишь для того, чтобы еще раз подчеркнуть, что при использовании условия EXISTS список отбора подзапроса не имеет значения.

4. Перечислите агентов, состояние лицензий которых не равно “Licensed”. Не используйте для этого объединение таблиц — только коррелированные подзапросы и оператор EXISTS. Введите и выполните следующую команду:

```
SELECT FirstName, LastName, AgentID
FROM Agents a
WHERE EXISTS (SELECT 'X'
               FROM LicenseStatus b
               WHERE a.LicenseStatusID = b.LicenseStatusID
                     AND UPPER(StatusText) <> 'LICENSED')
ORDER BY LastName, FirstName;
```

В результате у вас должно получиться то же, что показано на рис. 6.24.

5. Если в качестве текстового редактора вы использовали Блокнот, запишите набранный код и закройте Блокнот.
6. Введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из Oracle и закрыть SQL\*Plus.

## Использование подзапросов в DML-выражениях

Подзапросы могут использоваться не только в выражениях SELECT, их можно применять и в DML<sup>3</sup>-выражениях (включая INSERT, DELETE и UPDATE). Например, приведенная ниже команда DELETE удаляет записи об агентах из таблицы Agents, если им не соответствуют записи в таблице Listings. (Не выполняйте эту команду, поскольку вы удалите информацию о трех новых сотрудниках, которые еще не приступили к работе!)

```
DELETE
FROM Agents
WHERE NOT EXISTS
  (SELECT 'X' FROM Listings
   WHERE AgentID = ListingAgentID);
```

В данном случае подзапрос просто определяет, отображается ли идентификационный номер агента AgentID, поставляемый внешним запросом, в любом из полей столбца ListingAgentID таблицы Listings. Если нет, то данный сотрудник не имеет соотнесенного с ним дома на продажу. Запустив данную команду, вы найдете и удалите трех агентов (Susan Swarthmore, George Nagy и Robert Sellsmore) из

---

<sup>3</sup>Data Manipulation Language — язык манипулирования данными.

The screenshot shows an Oracle SQL\*Plus session with three distinct queries:

- Подсчет клиентов без агентов**: A query to count customers who do not have agents assigned.
- Подсчет клиентов без недвижимости**: A query to count customers who do not own properties.
- Перечень агентов, статус лицензии которых отличается от "Licensed"**: A query to list agents whose license status is not 'LICENSED'.

```

SQL> SELECT COUNT(*) AS "Customers w/o agents"
2  FROM Customers outer
3 WHERE NOT EXISTS
4   (SELECT 'X'
5    FROM CustAgentList inner
6    WHERE outer.CustomerID = inner.CustomerID);

Customers w/o agents
-----
1745

SQL> SELECT COUNT(LastName) AS "Customers w/o properties"
2  FROM Customers
3 WHERE NOT EXISTS
4   (SELECT 123
5    FROM Properties
6    WHERE CustomerID = OwnerID);

Customers w/o properties
-----
580

SQL> SELECT FirstName, LastName, AgentID
2  FROM Agents a
3 WHERE EXISTS (SELECT 'X'
4   FROM LicenseStatus b
5   WHERE a.LicenseStatusID = b.LicenseStatusID
6   AND UPPER(StatusText) <> 'LICENSED')
7 ORDER BY LastName, FirstName;

FIRSTNAME      LASTNAME      AGENTID
-----          -----        -----
Tobias          Carling       10235
George          Nagy          23498
Sindisiwe      Weber         14601

```

Рис. 6.24. Использование оператора EXISTS в нескольких коррелированных подзапросах

таблицы Agents. В приложении, связанном с продажей каких-либо товаров, с помощью похожей команды можно вставить наименование товара в список вещей, которые необходимо заказывать, так как их запас ограничен.

Коррелированный подзапрос можно использовать для обновления таблицы. Например, приведенные ниже команды ALTER TABLE и SELECT добавляют столбец в таблицу AgentsHR, а затем заполняют ее соответствующими значениями дат рождения агентов, которые находятся в столбце BirthDate таблицы Agents. Обратите внимание на то, что коррелированный подзапрос применяется для того, чтобы связать информацию о каждом агенте с правильной датой рождения внешнего запроса.

```

REM В таблицу AgentsHR добавляется столбец, в котором
REM будет храниться значение BirthDate:
ALTER TABLE AgentsHR
  ADD (BirthDate DATE);
REM Дата рождения каждого агента вставляется в правильную строку
UPDATE AgentsHR hr
SET BirthDate = (SELECT BirthDate
                 FROM Agents ag
                 WHERE hr.AgentID = ag.AgentID);

```

## Создание и использование представлений

*Представление* — это предопределенный запрос к одной или нескольким таблицам, называемым *базовыми*. Представление может состоять из подмножества базовой таблицы или быть комбинацией произвольного числа таблиц, объединенных таким же образом, как вы объединяете базовые таблицы. Представление не хранится как физическая таблица. Вместо этого Oracle в своих системных таблицах хранит информацию, позволяющую создать представление — его *определение*. При использовании представления Oracle активизирует определение и выполняет команду SELECT или DML-выражение, опираясь на определение представления. Например, информацию из представления или виртуальной таблицы вы извлекаете точно так же, как из любой другой таблицы базы данных: имя представления указывается в операторе FROM выражения SELECT точно так же, как имя любой другой базовой таблицы. С точки зрения пользователя, представления и таблицы неразличимы. Хотя вы всегда можете использовать представление для извлечения информации из базовых таблиц, вы не всегда можете применять к нему различные DML-выражения, подобные INSERT, UPDATE или DELETE. (Подробнее об этом вы узнаете ниже в этой главе.) Вы уже использовали представления в предыдущих главах — в частности, представления словаря данных. Например, при отображении на экране информации о таблицах вы можете запросить представление словаря данных user\_tables. Подобным образом, представление словаря данных user\_sequences содержит информацию о создаваемых пользователем последовательностях, а представление словаря данных user\_triggers — определение всех обработчиков событий пользователя.

Далее в этой главе вы узнаете, как определять представления и организовывать запросы к ним. Помимо этого, вы используете выражения DML применительно к представлениям и узнаете, как представления помогают защищать важные таблицы от нежелательных модификаций или даже нежелательного просмотра. Кроме того, будет рассказано, как с помощью представлений обеспечить необходимую безопасность выбранных таблиц и упростить организацию запросов.

### Зачем нужны представления?

Представления создаются путем задания выражений SELECT, описывающих, какие строки, столбцы и таблицы используются для создания представления. Как и в случае с другими объектами, вы можете предоставлять другим пользователям доступ к своим представлениям. Ниже перечислен ряд преимуществ, которые дают представления.

- *Безопасность*. Представления могут ограничить доступ пользователя к строкам и столбцам базовых таблиц, скрывая информацию, которую пользователь не должен видеть.

- *Упрощенная организация запросов.* Представления облегчают написание запросов. Если для извлечения информации требуется сложное выражение `SELECT`, в котором фигурирует множество таблиц, вы можете организовать запрос как представление, позволив пользователям обращаться к информации посредством более простого представления.
- *Конфиденциальность.* Представления позволяют скрыть имена базовых таблиц.
- *Независимость от данных.* Если интенсивно используемая таблица реструктуризируется в несколько таблиц, изменение структуры можно скрыть за представлением, объединяющим несколько таблиц.

Представления позволяют определить конкретную информацию, извлекаемую из базовых таблиц, для которых определено представление. Ограничивающая возможности различных классов пользователей базы данных просмотром конкретных представлений таблицы или набора связанных таблиц, вы обеспечиваете определенную разновидность безопасности, именуемую *разграничительным контролем доступа* (*discretionary access control*). Безопасность такого рода означает, что различным пользователям разрешено видеть различные строки и столбцы одной и той же базовой таблицы. Начальнику отдела кадров, например, требуется определенная секретная информация о сотрудниках — расовая принадлежность, дата рождения, число материально зависимых лиц, информация о медицинском страховании и т.д. Данная информация извлекается из тех же таблиц, из которых менеджер берет идентификационные номера сотрудников, их адреса и информацию о практических навыках. Аккуратно спроектировав представления, вы можете разграничить информацию, доступную различным группам пользователей, не предоставляя всем им полный доступ к базовым таблицам, содержащим всю информацию о сотрудниках.

Представления действуют как фильтры, позволяя одной информации проходить через фильтр к пользователям и блокируя передачу другой информации. *Представление подмножества столбцов* (*column subset view*), например, скрывает определенные столбцы от указанных пользователей, но открывает их для других сотрудников. Данная возможность реальна благодаря удалению столбцов из определения таблицы. *Представление подмножества строк* (*row subset view*) запрещает выбранным группам пользователей доступ к строкам, удовлетворяющим определенным критериям. Эта возможность реализуется посредством использования в выражении `SELECT` определения представления оператора `WHERE`, отфильтровывающего строки, которые требуется скрыть.

Существуют два фундаментальных типа представлений: сложные и простые. *Сложное представление* содержит подзапрос, извлекающий данные из нескольких таблиц, группирующий строки с использованием оператора `GROUP BY` или `DISTINCT` либо содержащий вызов функции. *Простое представление* извлекает строки из одной базовой таблицы. Подробное исследование данной темы мы начнем с простых представлений.

## Определение и организация запросов к однотабличным представлениям

Чтобы создать (определить) представление, необходимо использовать выражение CREATE VIEW. Упрощенный синтаксис этой команды выглядит следующим образом:

```
CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW <view-name>
[(<column-name>, . . . , <column-name>)]
AS <subquery>
[WITH CHECK OPTION [CONSTRAINT <constraint-name>]]
[WITH READ ONLY];
```

В данном выражении необязательная фраза OR REPLACE указывает на то, что в случае, если в схеме пользователя существует представление с указанным именем, база данных Oracle должна заменить его новым. Модифицировать представление невозможно, поэтому вам необходимо заново создавать представление, заменяя им старый вариант. Опция FORCE позволяет создавать представление на основе еще не существующей базовой таблицы. NOFORCE (значение по умолчанию) означает, что представление нельзя создать, если фигурирующая в нем таблица на текущий момент не существует. Аргумент "<view-name>" представляет собой имя создаваемого представления, а "<column-name>" — имя столбца представления, соответствующего выражению или столбцу подзапроса. Параметр <subquery> — это подзапрос, извлекающий данные из базовых таблиц. Опция WITH CHECK OPTION задает, что в представление вставляются, обновляются или удаляются только строки, удовлетворяющие критерию, указанному в операторе WHERE подзапроса. Если пропустить команду WITH CHECK OPTION, Oracle обработает строки, не определяя, будут ли они доступны посредством представления. Опция READ ONLY задает, что представление позволяет только считывать строки (обновление, удаление или вставка строк не разрешается). Аргумент "<constraint-name>" задает имя условия WITH CHECK OPTION.

Простое представление основано на *одной* базовой таблице. В приведенных ниже примерах для таблицы Agents создается представление подмножества строк, которое будет отображать на экране только строки, соответствующие агентам-мужчинам. Более того, оператор WITH READ ONLY не даст никому случайно (или намеренно) изменить значения в столбцах таблицы Agents.

```
CREATE OR REPLACE VIEW MaleEmployees
AS SELECT *
  FROM Agents
 WHERE Gender = 'M'
 WITH READ ONLY;
```

Представление называется MaleEmployees. Оно возвращает те же столбцы, которые содержатся в базовой таблице Agents. Для отображения на экране имен и типов данных столбцов представления можно использовать то же выражение SQL\*Plus DESCRIBE, что и для базовой таблицы.

Представления ограничивают доступные столбцы, не упоминая их в подзапросе, определяющем представление. Например, приведенное ниже выражение CREATE VIEW исключает из рассмотрения практически все столбцы таблицы Agents.

```
CREATE OR REPLACE VIEW Employees
AS SELECT AgentID, FirstName, LastName, Title
      FROM Agents;
```

С помощью данного представления можно изучать только столбцы AgentID, FirstName, LastName и Title.

После того как вы создадите однотабличное представление, организовать запрос к нему будет делом одной минуты. Просто в тех местах, где вы бы указывали имя таблицы, вам нужно использовать имя представления. Выражение SELECT, несущее запрос к представлению, может иметь список отбора, возвращающий подмножество столбцов представления, и другие необязательные операторы выражения SELECT, включая WHERE, GROUP BY, HAVING и ORDER BY. Ниже для примера приведено выражение SELECT, запрашивающее представление Employees, определение которого представлено выше.

```
SELECT LastName || ', ' || FirstName, Title
  FROM Employees
 WHERE Title = 'Broker'
 ORDER BY 1;
```

Данный запрос использует представление Employees для извлечения полей имени и фамилии (соединяемых с помощью конкатенации), а также значений поля Title (которое равно “Broker”). Перед отображением строк представления на экране Oracle сортирует их.

Довольно разговоров! Пришло время определить представление, а может быть, даже несколько, а затем организовать запрос к ним. Для начала запустите SQL\*Plus и войдите в Oracle. Если хотите, можете запустить текстовый редактор, что позволит записать окончательный вариант выражений SQL.

Для того чтобы создать и запросить однотабличное представление, выполните следующие действия.

1. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
2. Введите следующий код, скопируйте его в SQL\*Plus и выполните, чтобы создать Brokers — представление подмножества строк, отображающее все столбцы для агентов, имеющих звание (Title) “Broker”.

```
CREATE OR REPLACE VIEW Brokers
AS SELECT *
      FROM Agents
     WHERE UPPER>Title) = 'BROKER'
      WITH CHECK OPTION;
```

Oracle сообщит, что представление создано.

The screenshot shows the Oracle SQL\*Plus interface with the following session history:

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> CREATE OR REPLACE VIEW Brokers
  2  AS SELECT *
  3    FROM Agents
  4   WHERE UPPER>Title) = 'BROKER'
  5 WITH CHECK OPTION;
View created.

SQL> CREATE OR REPLACE VIEW NonBrokers
  2  AS SELECT AgentID, LastName, FirstName, BirthDate
  3        Gender, HireDate, Title, TaxID
  4    FROM Agents
  5   WHERE UPPER>Title) <> 'BROKER'
  6 WITH CHECK OPTION;
View created.

SQL> DESCRIBE NonBrokers
Name          Null?    Type
-----        -----
AGENTID      NOT NULL NUMBER(38)
LASTNAME     NVARCHAR2(30)
FIRSTNAME    NVARCHAR2(30)
GENDER       DATE
HIREDATE     DATE
TITLE        NVARCHAR2(20)
TAXID        NVARCHAR2(20)
SQL>

```

**Рис. 6.25.** Создание и описание представлений

3. Создайте представление подмножества столбцов для агентов, имеющих звания, отличные от “Broker”. Введите и выполните следующую команду, чтобы создать представление NonBrokers:

```

CREATE OR REPLACE VIEW NonBrokers
AS SELECT AgentID, LastName, FirstName, BirthDate
       Gender, HireDate, Title, TaxID
  FROM Agents
 WHERE UPPER>Title) <> 'BROKER'
WITH CHECK OPTION;

```

Oracle сообщит, что представление создано.

4. Введите следующий код, чтобы отобразить на экране информацию о представлении NonBrokers:

```
DESCRIBE NonBrokers
```

Oracle выведет на экран имена и типы данных столбцов представления NonBrokers. В результате вы должны получить то же, что показано на рис. 6.25.

5. Введите и выполните следующую команду, чтобы очистить экран, задать формат столбцов и организовать запрос к созданным представлениям:

```

CLEAR SCREEN
COLUMN FirstName FORMAT A15
COLUMN LastName FORMAT A15
COLUMN Title FORMAT A12
SELECT FirstName, LastName, Title
  FROM NonBrokers
 ORDER BY LastName, FirstName;

```

Полученные результаты должны соответствовать представленным на рис. 6.26.

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The command line displays:

```
SQL> COLUMN FirstName FORMAT A15
SQL> COLUMN LastName FORMAT A15
SQL> COLUMN Title FORMAT A12
SQL> SELECT FirstName, LastName, Title
  2  FROM NonBrokers
  3  ORDER BY LastName, FirstName;
```

Three result sets are displayed as tables:

FIRSTNAME	LASTNAME	TITLE
Tobias	Carling	Salesperson
Belinda	Chong	Salesperson
Elizabeth	Dahlen	Salesperson
Cornelis	Dann	Salesperson
Crystal	Fernandez	Salesperson
Jackson	Flamenbaum	Salesperson
Barbara	Herring	Salesperson
James	Kellogg	Salesperson
Kai	Marcoux	Salesperson
Nancy	Piperova	Salesperson
Lee	Reed	Salesperson

FIRSTNAME	LASTNAME	TITLE
Clair	Robinson	Salesperson
Cecilia	Romero	Salesperson
Ramanathan	Rove	Salesperson
Tim	Schutz	Salesperson
Heather	Sheibani	Salesperson
Daniel	Silverburg	Salesperson
Stanislaw	Soltwedel	Salesperson
Tim	St-Onge	Salesperson
Edwin	Townsend	Salesperson
Bruce	Voss	Salesperson
Sindisiwe	Weber	Salesperson

FIRSTNAME	LASTNAME	TITLE
Christine	Williams	Salesperson

23 rows selected.

SQL>

Рис. 6.26. Вывод на экран строк с использованием представления NonBrokers

При выполнении последнего пункта Oracle отобразит 23 строки таблицы Non-Brokers, соответствующих строкам базовой таблицы Agents. Ни одна строка в столбце Title не содержит значения “Broker”, поскольку представление отфильтровало подобные строки. Если вы присвоили переменной PAGESIZE значение, отличающееся от 14, число заголовков в вашем случае может отличаться от представленного на рисунке.

Вы можете модифицировать имена столбцов, определенных в представлении, дав новые названия между словом VIEW и именем представления. Чтобы поэкспериментировать с этой возможностью, мы создадим еще одно представление.

Для того чтобы создать представление, определяющее альтернативные имена столбцов, выполните следующие действия.

1. Введите следующий код в SQL\*Plus (или в Блокноте, а затем скопируйте в SQL\*Plus):

```
CLEAR SCREEN
CREATE OR REPLACE VIEW
  Cust (CustNo, LName, FName, Addr, City, St, Zip)
AS SELECT CustomerID, LastName, FirstName,
          Address, City, State, Zipcode
     FROM Customers;
```

The screenshot shows the Oracle SQL\*Plus interface. The command line displays:

```
SQL> CREATE OR REPLACE VIEW
  2   Cust (CustNo, LName, FName, Addr, City, St, Zip)
  3   AS SELECT CustomerID, LastName, FirstName,
  4             Address, City, State, Zipcode
  5   FROM Customers;
```

Below the command, the message "View created." is shown. Then, the command:

```
SQL> DESCRIBE Cust
```

is run, and the output shows the columns and their types:

Name	Null?	Type
CUSTNO	NOT NULL	NUMBER(38)
LNAME	NOT NULL	NVARCHAR2(30)
FNAME	NOT NULL	NVARCHAR2(30)
ADDR		NVARCHAR2(40)
CITY		NVARCHAR2(30)
ST		NVARCHAR2(20)
ZIP		NVARCHAR2(20)

**Рис. 6.27.** Определение представления с альтернативными именами столбцов

2. Введите следующий код и нажмите клавишу <Enter>, чтобы отобразить на экране информацию о представлении:

```
DESCRIBE Cust
```

Результаты выполнения данной команды показаны на рис. 6.27. У вас должно получиться то же самое.

3. Введите `exit` и нажмите клавишу <Enter>, чтобы выйти из SQL\*Plus и Oracle.

## Модификация данных с использованием однотабличного представления

К представлениям можно применять DML-выражения, обновляя, вставляя или удаляя данные базовых таблиц, на основе которых создано представление. При этом, правда, существует несколько ограничений. Если вы определили представление с опцией `WITH READ ONLY`, Oracle не позволит никакой операции DML подействовать на представление. Если вы желаете задействовать представление в DML-операциях, вы должно повторно создать его без фразы `WITH READ OPTION`. Подобным образом, операции обновления и вставки, примененные к представлению, которое было определено с опцией `WITH CHECK OPTION`, дадут ошибку, если предполагаемая модификация приведет к тому, что получить измененную информацию через представление станет невозможно. Например, если вы попытаетесь вставить новую строку, используя представление `Brokers`, а в столбце `Title` новой строки будет располагаться значение “Salesperson”, Oracle сгенерирует сообщение об ошибке “ORA-01402: view WITH CHECK OPTION where-clause violation” (“ORA-01402: нарушение условия where в опции WITH CHECK OPTION представления”). Если вы создаете простые представления без опции проверки, то сможете обновлять, вставлять и удалять строки, которые не извлекаются представлением.

Существует несколько правил применения DML-выражений к простым и сложным представлениям. Если представление не содержит опции WITH READ ONLY, DML-выражения будут выполняться правильно на простом представлении при выполнении условия: модификация не должна нарушать правила домена или целостности на уровне ссылок базовой таблицы. В частности, все модификации должны соглашаться с ограничениями таблицы, в том числе (1) PRIMARY KEY; (2) NOT NULL; (3) UNIQUE; (4) FOREIGN KEY и (5) WITH CHECK OPTION. Чтобы лучше разобраться в применении DML-операций к представлениям, рассмотрим их на примере.

Для того чтобы вставить, обновить и удалить строки простого представления, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle и запустите Блокнот, если вы желаете использовать его в качестве текстового редактора. Введите CLEAR SCREEN, чтобы очистить экран SQL\*Plus.
2. Введите и выполните следующую команду, чтобы вставить строку в представление *Brokers*:

```
INSERT INTO Brokers (AgentID, LastName, FirstName, Title)
VALUES (87654, 'Foxworthy', 'Robert', 'Salesperson');
```

Oracle отклонит попытку выполнения команды INSERT, поскольку представление *Brokers* допускает только вставку строк, в столбце *Title* которых отображается значение *Broker*.

3. Введите и выполните следующую команду, чтобы, используя представление *Cust*, вставить строку с записью нового клиента:

```
INSERT INTO Cust (CustNo, FName, LName, Addr, City)
VALUES (76543, 'Allison', 'Honeycutt', '123 Main', 'Loleta');
```

4. В предыдущей команде INSERT мы забыли указать почтовый индекс (столбец *ZIP*). Чтобы исправить эту ошибку, введите и выполните приведенную ниже команду UPDATE. (Не забудьте записать оператор WHERE точно так, как показано ниже, поскольку в противном случае изменения могут затронуть большое число столбцов.)

```
UPDATE Cust
SET St = 'CA', Zip = '95551'
WHERE CustNo = 76543;
```

5. Чтобы зафиксировать вставку и обновление, выполните следующую команду:

```
COMMIT;
```

6. Выведите новую строку на экран, набрав и выполнив следующую команду:

```
SELECT CustNo, FName, LName, City, St, Zip
FROM Cust
WHERE CustNo = 76543;
```

The screenshot shows a session in Oracle SQL\*Plus. The user is attempting to insert data into a view named 'Brokers'. The first two INSERT statements succeed, but the third fails with an ORA-01402 error because it includes a WHERE clause with a CHECK OPTION, which is not allowed for views.

```

SQL> INSERT INTO Brokers (AgentID, LastName, FirstName, Title)
  2  VALUES (87654, 'Foxworthy', 'Robert', 'Salesperson');
INSERT INTO Brokers (AgentID, LastName, FirstName, Title)
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation

SQL> INSERT INTO Cust (CustNo, FName, LName, Addr, City)
  2  VALUES (76543,'Allison', 'Honeycutt', '123 Main','Loleta');

1 row created.

SQL> UPDATE Cust
  2  SET St = 'CA', Zip = '95551'
  3 WHERE CustNo = 76543;

1 row updated.

SQL> COMMIT;

Commit complete.

SQL> SELECT CustNo, FName, LName, City, St, Zip
  2  FROM Cust
  3 WHERE CustNo = 76543;

```

CUSTNO	FNAME	LNAME	CITY	ST	ZIP
76543	Allison	Honeycutt	Loleta	CA	95551

```

SQL> DELETE FROM Cust
  2 WHERE CustNo = 76543;

1 row deleted.

SQL>

```

Рис. 6.28. Действие DML-выражений на представление

7. Удалите эту строку с помощью команды

```
DELETE FROM Cust
WHERE CustNo = 76543;
```

Результаты выполнения приведенных команд показаны на рис. 6.28.

## Создание сложных представлений

Для создания сложного представления применяется та же команда CREATE VIEW, что и для простого однотабличного представления. Напомним, что сложное представление содержит подзапрос, извлекающий строки из нескольких базовых таблиц, группирующий строки с использованием оператора GROUP BY или DISTINCT либо содержащий одну или несколько функций либо выражений. При выполнении любого из предыдущих условий представление классифицируется как сложное. Пожалуй, основным отличием сложных представлений от простых является то, что сложные не допускают применения DML-операций.

## Однотабличное сложное представление

Чтобы понять, что же такое сложные представления, лучше всего начать с однотабличного представления и выяснить, что делает его сложным и какие проблемы возникают при попытках применения к нему DML-операций. Ниже вы создадите представление, основанное на таблице Agents, а затем примените к нему пару DML-выражений. Как обычно, для выполнения выражений используйте SQL\*Plus.

Чтобы создать сложное однотабличное представление и подействовать на него DML-выражениями, необходимо выполнить следующие действия.

1. Введите и выполните в SQL\*Plus следующую команду, чтобы создать новое представление, извлекающее и отображающее на экране четыре столбца таблицы Agents, создавая новый столбец для указания возраста каждого агента:

```
CREATE OR REPLACE VIEW AgentView AS
  SELECT AgentID, FirstName, LastName,
         BirthDate, (SYSDATE-BirthDate)/365.25 AS Age
    FROM Agents;
```

2. Посредством созданного представления модифицируйте дату рождения агента, набрав и выполнив следующую команду:

```
UPDATE AgentView
  SET BirthDate = TO_DATE('10/17/1962', 'MM/DD/YYYY')
 WHERE Agentid = 10041;
```

Oracle обновит строку, вставив в нее новую дату рождения.

3. С помощью представления модифицируйте возраст агента, набрав и выполнив следующую команду:

```
UPDATE AgentView
  SET AGE = 50 WHERE Agentid = 10041;
```

Oracle отклонит попытку обновления и выдаст сообщение об ошибке

```
ORA-01733: virtual column not allowed here
```

4. Отмените все незафиксированные действия, выполнив следующую команду:

```
ROLLBACK;
```

Описанные команды и результаты их выполнения показаны на рис. 6.29.

Выполняя команду ROLLBACK, мы отменяем обновление, произведенное на этапе 2. Тем не менее она не влияет на само представление, поскольку CREATE VIEW относится к DDL-выражениям. Из приведенного примера видим, что Oracle отклоняет все попытки изменить виртуальный (рассчитываемый) столбец через представление. Таким образом, некоторые однотабличные представления, по сути, являются сложными. Следовательно, однотабличные сложные представления, содержащие выражения, не являются на 100% обновляемыми. Такое положение дел нельзя назвать ни хорошим, ни плохим, вы просто должны об этом знать.

The screenshot shows a Windows-style window titled "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", "Help". The main area contains the following SQL session:

```

SQL> CREATE OR REPLACE VIEW AgentView AS
  2   SELECT AgentID, FirstName, LastName,
  3         Birthdate, (SYSDATE-BirthDate)/365.25 AS Age
  4   FROM Agents;
View created.

SQL> UPDATE AgentView
  2   SET BirthDate = TO_DATE('10/17/1962', 'MM/DD/YYYY')
  3 WHERE Agentid = 10041;
1 row updated.

SQL> UPDATE AgentView
  2   SET AGE = 50
  3 WHERE Agentid = 10041;
SET AGE = 50
*
ERROR at line 2:
ORA-01733: virtual column not allowed here

SQL> ROLLBACK;
Rollback complete.

SQL>

```

Рис. 6.29. Невозможность обновления сложного однотабличного представления

## Многотабличные сложные представления

Большую часть времени вам придется работать с представлениями, состоящими из нескольких таблиц. Довольно часто многотабличные представления создаются просто для того, чтобы упростить выражения, объединяющие большое число таблиц. Другой причиной создания представлений является защита исходной таблицы от несанкционированного просмотра секретных столбцов или нежелательной модификации. Лучший способ овладеть соответствующими навыками — это создать пару представлений, включающих несколько таблиц. Сделав это, вы поймете, что представления облегчают организацию запросов к базе данных. Впрочем, они же и создают несколько проблем с модификацией данных.

Итак, прежде всего создадим очень полезное представление, позволяющее отобразить на экране объекты (включая их имена), созданные в данной главе. Напомним, что Oracle хранит информацию о ваших объектах (таблицах, последовательностях, представлениях и т.д.) в представлениях словаря данных (`user_tables`, `user_sequences`, `user_views` и т.д.). Зная это, мы можем создать представление `MyOwnObjects`, отображающее многие из этих объектов.

Для того чтобы создать многотабличное представление, выполните следующие действия.

1. Используя SQL\*Plus, введите `CLEAR SCREEN` и нажмите клавишу `<Enter>`.
2. Введите и выполните следующую команду, чтобы создать представление `MyOwnObjects`:

```
CREATE OR REPLACE VIEW MyOwnObjects (ObjType, ObjName) AS
  SELECT 'Table', Table_Name FROM User_Tables
```

The screenshot shows the Oracle SQL\*Plus interface with the following content:

```

File Edit Search Options Help
SQL> CREATE OR REPLACE VIEW MyOwnObjects (ObjType, ObjName) AS
2   SELECT 'Table', Table_Name FROM User_Tables
3   WHERE Table_Name NOT LIKE 'BIN%'
4   UNION
5   SELECT 'View', View_Name FROM User_VIEWS
6   UNION
7   SELECT 'Sequence', Sequence_Name FROM User_Sequences
8   UNION
9   SELECT 'Trigger', Trigger_Name FROM User_Triggers;

```

View created.

```

SQL> SELECT ObjType, ObjName
2   FROM MyOwnObjects
3   ORDER BY 1,2;

```

OBJTYPE	OBJNAME
Table	AGENTS
Table	AGENTSHR
Table	CONTACTREASON
Table	CUSTAGENTLIST
Table	CUSTOMERS
Table	LICENSESTATUS
Table	LISTINGS
Table	PROPERTIES
Table	SALESTATUS
View	AGENTVIEW
View	BROKERS

OBJTYPE	OBJNAME
View	CUST
View	MYOWNOBJECTS
View	NONBROKERS

14 rows selected.

SQL>

A brace on the right side of the screenshot groups the first part of the code (the view creation) and is labeled "Создание представления, содержащего несколько таблиц". Another brace groups the results of the second part of the code (the select statement) and is labeled "Запрос представления, содержащего несколько таблиц, и результаты этого запроса".

Рис. 6.30. Создание представления, состоящего из нескольких таблиц, и организация запроса к нему

```

WHERE Table_Name NOT LIKE 'BIN%'
UNION
SELECT 'View', View_Name FROM User_VIEWS
UNION
SELECT 'Sequence', Sequence_Name FROM User_Sequences
UNION
SELECT 'Trigger', Trigger_Name FROM User_Triggers;

```

Oracle создаст представление, которое после соответствующего запроса отобразит на экране несколько ваших объектов.

- Чтобы создать запрос относительно своих объектов, введите и выполните следующую команду:

```

SELECT ObjType, ObjName
FROM MyOwnObjects
ORDER BY 1,2;

```

Oracle отобразит девять таблиц и пять представлений, включая представление, которое, собственно, и формирует данный список MyOwnView (рис. 6.30).

Иногда представления, состоящие из нескольких таблиц, удобны в случаях, когда запросы информации слишком сложны, громоздки или еще каким-то образом неудобны. Представления позволяют сэкономить время, если пользователи часто об-

рашаются с запросами, касающимися нескольких таблиц. Ниже приведен пример многотабличного представления, которое упрощает сравнительно сложное выражение SELECT. Вы создадите представление *ForSale*, объединяющее пять таблиц с целью извлечения информации о продавцах, домах, выставленных на продажу, и агентах продавцов. Используемые для этого таблицы — *Agents*, *CustAgentList*, *Listings*, *Properties* и *Customers*. Соответствующее выражение CREATE VIEW — достаточно длинное, поэтому набирайте его внимательно. Как обычно, выравнивание кода вы можете использовать по своему усмотрению, однако мы считаем, что отступы с правого края строки облегчают восприятие кода. Кроме того, рекомендуется использовать текстовый редактор, поскольку это облегчит внесение исправлений и отладку сценария. Как обычно, мы предлагаем воспользоваться программой Блокнот. Если необходимо, запустите SQL\*Plus и войдите в Oracle.

Для того чтобы создать многотабличное представление, упрощающее сложный запрос с операцией объединения, выполните следующие действия.

1. Введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран SQL\*Plus.
2. Введите и выполните следующую команду, чтобы создать представление, объединяющее пять таблиц:

```
CREATE OR REPLACE VIEW
  ForSale(AgID, AgentFirst, AgentLast, Bid, Ask,
          Addr, City, Br, Ba, Ft,
          SellerFirst, SellerLast) AS
SELECT
  a.AgentID, a.FirstName, a.LastName, b.BidPrice, c.AskingPrice,
  d.Address, d.City, d.Bedrooms, d.Bathrooms, d.SqFt,
  e.FirstName, e.LastName
FROM Agents a
  INNER JOIN CustAgentList b ON a.AgentID = b.AgentID
  INNER JOIN Listings c ON b.ListingID = c.ListingID
  INNER JOIN Properties d ON c.PropertyID = d.PropertyID
  INNER JOIN Customers e ON d.OwnerID = e.CustomerID
WHERE b.ContactReason = 'Sell';
```

Oracle сообщит, что представление было создано. Если вместо этого вы получите сообщение об ошибке, проверьте внимательно сценарий, исправьте его и выполните еще раз.

3. Получив представление *ForSale*, объединяющее пять таблиц, вы можете организовать запрос к нему, используя определенные в представлении имена столбцов (*AgID*, *AgentFirst*, *Br* и т.д.). Введите и выполните следующие команды SQL\*Plus, чтобы отформатировать столбцы для более удобного восприятия:

```
COLUMN Asking FORMAT $9,999,999
COLUMN Addr FORMAT A25
COLUMN City FORMAT A10
COLUMN SellerLast FORMAT A14
```

```

SQL> CREATE OR REPLACE VIEW
2   ForSale(AgID, AgentFirst, AgentLast, Bid, Ask,
3          Addr, City, Br, Ba, Ft,
4          SellerFirst, SellerLast) AS
5   SELECT
6     a.AgentID, a.FirstName, a.LastName, b.BidPrice, c.AskingPrice,
7     d.Address, d.City, d.Bedrooms, d.Bathrooms, d.Sqft,
8     e.FirstName, e.LastName
9   FROM Agents a
10  INNER JOIN CustAgentList b ON a.AgentID = b.AgentID
11  INNER JOIN Listings c ON b.ListingID = c.ListingID
12  INNER JOIN Properties d ON c.PropertyID = d.PropertyID
13  INNER JOIN Customers e ON d.OwnerID = e.CustomerID
14 WHERE b.ContactReason = 'Sell';

View created.

SQL> COLUMN Asking      FORMAT $9,999,999
SQL> COLUMN Addr        FORMAT A25
SQL> COLUMN City         FORMAT A10
SQL> COLUMN SellerLast  FORMAT A14
SQL> COLUMN Br           FORMAT 99
SQL> COLUMN Ba           FORMAT 99
SQL> COLUMN Ft           FORMAT 9,999
SQL> SELECT SellerLast, Ask AS "asking", Addr, City, Br, Ba, Ft
2   FROM ForSale
3   WHERE Ask > 400000
4   ORDER BY Ask DESC;

SELLERLAST      Asking ADDR          CITY       BR    BA    FT
-----          ----- -----
Dorsch          $725,000 5990 Stover Rd  Blue Lake  4    4   4,828
Siemion         $560,000 8130 W End Rd  Arcata    5    4   5,150
Bivens          $506,000 605 Del Norte St Eureka    4    3   3,861
Frumer          $500,000 1199 Trojan St  Arcata    4    3   4,300
Sriboonlert     $495,000 1591 Kings Row Fortuna   4    4   4,490
Johnson         $469,950 2190 Hannah Ct  Fortuna   4    3   3,650

```

Создание представления, состоящего из пяти таблиц

Форматирование столбцов

Отображение строк, используемых представлением ForSale

Рис. 6.31. Создание представления, объединяющего пять таблиц

```

COLUMN Br FORMAT 99
COLUMN Ba FORMAT 99
COLUMN Ft FORMAT 9,999

```

4. Напишите запрос для отбора определенных столбцов представления ForSale, набрав и выполнив в SQL\*Plus следующую команду:

```

SELECT SellerLast, Ask AS "Asking", Addr, City, Br, Ba, Ft
FROM ForSale
WHERE Ask > 400000
ORDER BY Ask DESC;

```

Oracle возвратит шесть строк, удовлетворяющих критерию. Результаты выполнения описанных действий показаны на рис. 6.31.

5. В SQL\*Plus введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из Oracle и закрыть SQL\*Plus. Запишите открытый в Блокноте файл сценария и закройте Блокнот.

## Создание синонимов для упрощения доступа к таблице

Если у вас есть право доступа к таблицам или иным объектам, принадлежащим другому пользователю, вы можете обращаться с этими таблицами и объектами так, как определил их владелец, предоставивший вам права доступа. Например, если пользователь `CoffeeMerchant` предоставил вам право применять команду `SELECT` к таблице `Merchandise`, вы можете использовать эту команду для извлечения строк из таблицы, уточнив имя таблицы путем указания имени схемы — `CoffeeMerchant`. Например, приведенная ниже команда извлекает все строки и столбцы таблицы, принадлежащей другой схеме (разумеется, при условии предоставления необходимых прав).

```
SELECT *  
FROM CoffeeMerchant.Merchandise;
```

Если вы обладаете правом доступа к нескольким таблицам другой схемы, то постоянное уточнение имен этих таблиц может потребовать чересчур много времени (не говоря уже о возможности случайной ошибки при наборе). Чтобы сократить объем набираемого кода, можно использовать очень полезную возможность: сопоставление синонима с каждым уточненным именем таблицы, к которой вы имеете доступ. После этого вы можете использовать этот синоним вместо имени таблицы. Чтобы создать личный синоним, необходимо использовать команду `CREATE SYNONYM` (разумеется, при наличии привилегии `CREATE SYNONYM`). Упрощенный синтаксис этой команды выглядит следующим образом:

```
CREATE SYNONYM <synonym-name> FOR <qualified-name>;
```

Предположим, например, что у вас есть привилегии `CREATE SYNONYM` и `SELECT` применительно к упомянутой выше таблице. В таком случае вы можете создать синоним `MyCoffee`, с его помощью упростив обращение к реальной таблице:

```
CREATE SYNONYM MyCoffee FOR CoffeeMerchant.Merchandise;
```

Определив синоним, вы можете вводить команды `SELECT`, используя уже его, а не полное название таблицы. Например, приведенная ниже команда извлекает строки из таблицы `Merchandise`, владельцем которой является схема `CoffeeMerchant`. Подробнее о предоставлении прав доступа к таблицам других пользователей рассказываеться в главе 12.

```
SELECT * FROM MyCoffee;
```

## Перечисление определений представлений

Информацию о создаваемых вами объектах Oracle хранит в представлениях словаря данных. В частности, информация о созданных вами представлениях хранится (как вы наверняка уже догадались) в представлении словаря данных `user_views`. Представление `all_views` содержит все доступные вам представления — не только те, владельцем которых вы являетесь, но и те, которые принадлежат другим поль-

**ТАБЛИЦА 6.4.** Описание некоторых столбцов представления user\_views

Имя столбца	Тип данных	Описание
view_name	VARCHAR2 (30)	Имя представления
text_length	NUMBER	Число символов в подзапросе представления
text	LONG	Текст подзапроса, который создает представление

зователям (схемам), предоставившим вам необходимые права доступа. Разумеется, вы можете выполнить команду DESCRIBE, указав после нее имя представления, чтобы просмотреть имена и типы данных столбцов этого представления. Тем не менее user\_views содержит гораздо больше информации. Для примера в табл. 6.4 приведены некоторые столбцы этого представления.

Далее мы рассмотрим определения ваших представлений.

Для того чтобы просмотреть определение одного или нескольких принадлежащих вам представлений, выполните следующие действия.

1. Запустите SQL\*Plus, войдите в Oracle и очистите экран.
2. Введите и выполните команды форматирования столбцов и страниц, чтобы позже удобно отобразить на экране информацию о представлении:

```
SET PAGESIZE 50
COLUMN view_name FORMAT A15
COLUMN text_length FORMAT 99,999
COLUMN text FORMAT A40 WRAPPED
```

3. Введите и выполните следующую команду, чтобы отобразить на экране информацию о принадлежащих вам представлениях:

```
SELECT view_name, text_length, text
FROM user_views
ORDER BY view_name;
```

Oracle выведет на экран шесть строк с информацией о представлениях (рис. 6.32).

4. Введите и выполните следующую команду, чтобы изменить форматирование столбцов и размер страницы:

```
CLEAR COLUMNS
SET PAGESIZE 14
```

Если вы желаете перечислить *все* представления, к которым имеете доступ, то измените запрос, приведенный в п. 3, до представленной ниже формы. *Предупреждение:* запустив этот запрос, вы получите сотни представлений, поскольку Oracle допускает пользователя к множеству представлений, а не только к тем, которые созданы в схеме этого пользователя.

```
SELECT view_name, text_length, text
FROM all_views
ORDER BY view_name;
```

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SET PAGESIZE 50
SQL> COLUMN view_name FORMAT A15
SQL> COLUMN text_length FORMAT 99,999
SQL> COLUMN text FORMAT A40 WRAPPED
SQL> SELECT view_name, text_length, text
  2  FROM user_views
  3  ORDER BY view_name;

VIEW_NAME      TEXT_LENGTH TEXT
-----          -----
AGENTVIEW        104 SELECT AgentID, FirstName, LastName,
                           BirthDate, (SYSDATE-BirthDate)/
                           365
RBNKRS           249 SELECT "AGENTID","FIRSTNAME","LASTNAME",
                           "HIREDATE","BIRTHDATE","GENDER","WORKPHO
CUST              97 SELECT CustomerID, LastName, FirstName,
                           Address, City, State, Zipcode
FORSALE          432 SELECT
                           a.AgentID, a.FirstName, a.LastName,
                           b.BidPrice, c.AskingPrice,
                           d.
MYOWNOBJECTS     261 SELECT 'Table', Table_Name FROM User_Tab
les
                           WHERE Table_Name NOT LIKE 'BIN%'
NONBROKERS       154 SELECT AgentID, LastName, FirstName, Bir
                           thdate
                           Gender, HireDate, Title

6 rows selected.

SQL>

```

Рис. 6.32. Отображение на экране информации о представлении

По нашим подсчетам любому пользователю доступно более 1 500 представлений. Поэтому избегайте просмотра представления `all_views`, не ограниченного достаточно жестким условием `WHERE`.

## Удаление представлений

Для удаления представления применяется следующая команда:

```
DROP VIEW <view-name>
```

Само по себе удаление представлений ничем не отличается от удаления других объектов, например, последовательностей или таблиц. В завершение данной главы мы “уберем за собой”, удалив все созданные представления. Помните, что *до* удаления всех представлений вы можете обратиться с запросом к `MyOwnObjects`, чтобы посмотреть, какие и представления принадлежат вам. Чтобы вывести список представлений на экран, потребуется следующий запрос:

```
SELECT *
  FROM MyOwnObjects
 WHERE LOWER(ObjType) = 'view';
```

The screenshot shows the Oracle SQL\*Plus interface with the title bar "Oracle SQL\*Plus". The menu bar includes "File", "Edit", "Search", "Options", and "Help". The main window displays the following SQL session:

```

SQL> SELECT *
  2  FROM MyOwnObjects
  3  WHERE LOWER(ObjType) = 'view';

OBJTYPE  OBJNAME
-----
View     AGENTVIEW
View     BROKERS
View     CUST
View     FORSALE
View     MYOWNOBJECTS
View     NONBROKERS

6 rows selected.

SQL> DROP VIEW AgentView;
View dropped.

SQL> DROP VIEW Brokers;
View dropped.

SQL> DROP VIEW Cust;
View dropped.

SQL> DROP VIEW ForSale;
View dropped.

SQL> DROP VIEW MyOwnObjects;
View dropped.

SQL> DROP VIEW NonBrokers;
View dropped.

SQL>

```

**Рис. 6.33.** Перечисление и удаление созданных представлений

Чтобы отобразить на экране, а затем удалить все представления, созданные вами в этой главе, необходимо выполнить следующие действия.

1. В SQL\*Plus введите и выполните следующую команду:

```

CLEAR SCREEN
SELECT *
FROM MyOwnObjects
WHERE LOWER(ObjType) = 'view';

```

Oracle отобразит на экране шесть представлений, созданных в данной главе.

2. Удалите представления, набрав и выполнив следующую команду:

```

DROP VIEW AgentView;
DROP VIEW Brokers;
DROP VIEW Cust;
DROP VIEW ForSale;
DROP VIEW MyOwnObjects;
DROP VIEW NonBrokers;

```

Oracle удалит все созданные вами представления (рис. 6.33).

3. Введите `exit` и нажмите клавишу `<Enter>`, чтобы выйти из Oracle и закрыть SQL\*Plus.

## Резюме

Выражения SELECT позволяют попарно объединять по общим столбцам неограниченное число таблиц. При этом вы указываете имена таблиц в операторе FROM и используете оператор ON, чтобы обозначить столбец или столбцы таблицы, используемые для объединения. Если два столбца объединяемых таблиц должны совпадать, следует использовать объединение по эквивалентности (оператор =). Для упрощения выражений, объединяющих большое число таблиц, можно использовать псевдонимы таблиц. Чтобы присвоить таблице псевдоним, необходимо указать его после имени таблицы в операторе FROM. Операции внутреннего объединения возвращают строки двух таблиц, имеющие соответствующие столбцы. Внешние объединения делятся на левосторонние, правосторонние и полные. Внешнее объединение позволяет включать в результат объединения строки таблиц, потенциально соответствующие друг другу, с учетом значений NULL. Лево- или правостороннее внешнее объединение позволяет выяснить, какие строки одной таблицы не имеют аналогов во второй (с помощью внутреннего объединения это сделать невозможно).

Операторы действий над множествами UNION, UNION ALL, MINUS и INTERSECT применяются для объединения результатов нескольких выражений SELECT. UNION и UNION ALL объединяют все строки, возвращаемые запросами. UNION ALL включает в результат дублирующиеся строки, а UNION — только уникальные. MINUS удаляет из результата строки, возвращаемые одним выражением SELECT, которые имеют аналоги в другом запросе. INTERSECT исключает из результатов двух выражений SELECT строки, присутствующие только в одном ответе.

Подзапросами называются запросы внутри запросов, которые могут возвращать одну строку и один столбец, несколько строк и один столбец, а также несколько строк и несколько столбцов. Подзапрос выполняется перед внешним запросом; исключением являются только коррелированные подзапросы. При использовании коррелированных подзапросов внешний запрос выполняется для одной строки, после этого подзапрос выполняется для всех строк, используя информацию от внешнего запроса. Подзапросы могут входить в список отбора запроса, оператор FROM запроса (внешнее представление), оператор WHERE выражения SELECT (наиболее распространенный вариант) или некоторые DML-выражения, подобные CREATE TABLE. Дополнительное пространство для маневра предлагают ключевые слова ANY и ALL в операторе WHERE, которые можно объединять с операторами отношения и определять, больше или меньше указанное значение какого-нибудь или всех значений, возвращаемых подзапросом.

Представления имеют уникальные имена и предопределенные выражения SELECT, которые обращаются к базовым таблицам и извлекают подмножества данных. Определения представлений хранятся и активизируются всякий раз, когда имя представления используется вместо имени таблицы. Представления обеспечивают целостность данных и необходимую безопасность, а также упрощают многие сложные запросы.

Чтобы упростить обращение к таблицам других схем (к которым вы имеете право доступа), вы можете создавать личные синонимы — псевдонимы существующих объектов.

## Основные термины

- Представление подмножества столбцов
- Сложное представление
- Составляющий запрос
- Составной запрос
- Коррелированный подзапрос
- Разграничительный контроль доступа
- Столбцы для отображения
- Распределение
- Объединение по эквивалентности
- Оператор EXISTS
- Полное внешнее объединение
- Внутреннее объединение
- Объединение
- Столбцы для объединения
- Условие объединения
- Левостороннее внешнее объединение
- Многостолбцовый подзапрос
- Многострочный подзапрос
- Естественное объединение
- Объединение не по эквивалентности
- Внешнее объединение
- Уточненное имя
- Правостороннее внешнее объединение
- Представление подмножества строк
- Скалярный подзапрос
- Столбцы поиска
- Самообъединение
- Оператор действия над множествами
- Простое представление

- Однострочный подзапрос
- Подзапрос
- Псевдоним таблицы
- Вертикальное объединение
- Представление

## Повторение пройденного материала

### Истина или ложь?

1. При объединении двух таблиц каждое поле объединения одной таблицы должно соответствовать полю объединения другой.
2. Использование псевдонимов таблиц обеспечивает дополнительную безопасность и упрощает обращение к таблице.
3. Подзапросы можно использовать в операторе FROM выражения SELECT и некоторых DDL-выражениях.
4. Результатом действия оператора EXISTS на подзапрос является значение TRUE или FALSE.
5. Представление — это то же самое, что и таблица, только представление можно определить так, чтобы оно возвращало лишь подмножество столбцов и/или строк базовой таблицы.

### Заполнить пропущенное

1. \_\_\_\_\_ объединение возвращает строки, которые удовлетворяют условию объединения.
2. Когда запрос возвращает все строки одной таблицы и только столбцы другой таблицы, удовлетворяющие условию объединения, это называется \_\_\_\_\_ объединением.
3. \_\_\_\_\_ подзапрос возвращает одну строку и один столбец.
4. \_\_\_\_\_ подзапрос использует значение, получаемое из внешнего запроса, для выполнения внутреннего запроса.
5. Чтобы упростить обращение к таблице посредством уточненного имени, вы можете создать \_\_\_\_\_.

### Варианты ответов

1. Какие из приведенных ниже операторов FROM правильно объединяют таблицы People и Sales по согласующимся значениям столбца PeopleID, присутствующего в обеих таблицах?

- a) FROM People JOIN Sales BY People. PeopleID IN Sales. PeopleID
- б) FROM Sales JOIN People USING PeopleID = PeopleID
- в) FROM People p JOIN Sales s ON PeopleID = PeopleID
- г) FROM Sales JOIN People USING (PeopleID)
2. Какие из приведенных ниже операторов WHERE можно использовать для извлечения всех строк таблицы Employee и выбранных строк таблицы Sales, в которых совпадают значения в соответствующих столбцах (EmpID для Employee и EmployeeID для Sales)?
- а) FROM Employee RIGHT OUTER JOIN Sales ON EmpID = EmployeeID
- б) FROM Employee LEFT OUTER JOIN Sales ON EmpID = EmployeeID
- в) FROM Employee FULL OUTER JOIN Sales ON EmpID = EmployeeID
- г) FROM Employee INNER JOIN Sales ON EmpID = EmployeeID
3. Какие из приведенных условий WHERE отображают строки, в которых значение BaseSalary агента меньше или равно наибольшему значению BaseSalary всех агентов?
- а) WHERE BaseSalary >= ALL (SELECT BaseSalary FROM AgentsHR).
- б) WHERE BaseSalary >= ANY (SELECT BaseSalary FROM AgentsHR).
- в) WHERE BaseSalary <= ALL (SELECT BaseSalary FROM AgentsHR).
- г) WHERE BaseSalary <= ANY (SELECT BaseSalary FROM AgentsHR).
- д) Все варианты неправильные.
4. Какие из приведенных выражений записаны неверно?
- а) CREATE OR REPLACE VIEW Someone AS SELECT \* FROM Agents;.
- б) CREATE VIEW Someone AS SELECT \* FROM Agents WHERE AgentID > 13000;
- в) CREATE VIEW Someone (F1, F2) AS SELECT HireDate FROM Agents;.
- г) CREATE VIEW Someone (AgID) AS (SELECT AgentID FROM Agents);.
- д) Все варианты правильные.
5. Какой из приведенных запросов перечисляет (по именам) созданные вами представления?
- а) SELECT view\_name FROM user\_views;.
- б) SELECT view\_name FROM all\_views;.
- в) SELECT view\_name FROM views;.
- г) SELECT viewname FROM userviews;.
- д) Все варианты неправильные.

## Упражнения

### 1. Readwood Realty

Вам требуется создать новую таблицу и запустить несколько запросов. Новая таблица содержит размер премии для продавцов, имеющих определенное количество домов в таблице Listings. Агенты, работающие с 16–20 домами, получают бонус 300 долларов; с 21–30 домами — бонус 600 долларов; агенты, имеющие более 30 домов, получают премию 1 000 долларов. В таблице указывается нижний и верхний пределы, которые используются для разбивки по суммам премий согласно числу домов. В качестве инструментов применяются команды INNER JOIN, внутреннее представление и объединение не по эквивалентности (> или < вместо =).

Откройте в Блокноте текстовый файл, запишите все, что вы введете, и выполните готовый сценарий в SQL\*Plus. Закончив работу, напечатайте текстовый файл.

Первая часть данного упражнения состоит из следующих этапов. Запустите Блокнот и запишите пустой файл как <имя>Ch06Prob1Part1.sql. Затем введите следующий код в Блокноте, скопируйте его в SQL\*Plus и выполните. Найдите на компакт-диске файл Ch06BuildBonus.sql (папка Chapter06) и скопируйте его на свой диск. Запишите путь к нему.

1. Ведите следующий код, чтобы инициализировать базу данных для данного упражнения, подставив реальный путь к файлу данных вместо <path> и свое имя вместо <имя>.

```
REM <имя>
CLEAR SCREEN
START <path>\Ch06BuildBonus.sql
```

2. Ведите и выполните следующую команду, чтобы отобразить на экране новую таблицу BonusSchedule:

```
SELECT * FROM BonusSchedule;
```

3. Ведите и выполните следующую команду, чтобы задать форматирование столбцов и отобразить размер бонуса:

```
SET PAGESIZE 35
COLUMN Bonus FORMAT $9,999
SELECT a.LastName, a.ListCount, b.Bonus
FROM (SELECT AgentID, LastName, COUNT(*) as
ListCount
      FROM Agents INNER JOIN Listings
      ON AgentID = ListingAgentID
      GROUP BY AgentID, LastName) a
      INNER JOIN BonusSchedule b
      ON a.ListCount BETWEEN b.LowLimit
      AND b.HighLimit
ORDER BY b.Bonus DESC, a.LastName ASC;
```

4. Введите и выполните приведенную ниже команду, чтобы повторить предыдущий запрос и направить его выход в текстовый файл. Подставьте свое имя вместо <имя> и соответствующий путь вместо <path>.

```
SPOOL <path>\<имя>Ch06Prob1P1-spool.txt
SHOW USER
/
SPOOL OFF
```

5. Восстановите значения переменных среды по умолчанию и удалите созданные таблицы, выполнив следующую команду:

```
CLEAR COLUMNS
SET PAGESIZE 14
DROP TABLE BonusSchedule CASCADE CONSTRAINTS
PURGE;
```

6. Запишите все, что вы набрали в Блокноте. Распечатайте полученный текстовый файл и буферный файл, созданный на этапе 4. Далее вы напишете и выполните выражения, перечисляющие всю недвижимость в базе данных Redwood Realty, состояние (SaleStatus) которой — “Sold”. Откройте новый файл в Блокноте, запишите его как <имя>Ch06Prob1Part2.sql и запустите SQL\*Plus, если это требуется. Далее придерживайтесь описанных ниже действий.

7. Введите и выполните следующую команду, подставив свое имя вместо <имя>:

```
REM <имя>
CLEAR SCREEN
CLEAR COLUMNS
COLUMN LastName FORMAT A15
COLUMN City FORMAT A15
COLUMN Address FORMAT A20
COLUMN Owner FORMAT A15
```

8. Введите и выполните следующую команду, чтобы объединить несколько таблиц и перечислить только проданные дома:

```
SELECT p.City, p.Address, p.PropertyID,
c.LastName AS "Owner"
FROM Properties p INNER JOIN Customers c
ON p.OwnerID = c.CustomerID
WHERE PropertyID IN
(SELECT PropertyID
FROM Listings
WHERE SaleStatusID = (SELECT
SaleStatusID
FROM SaleStatus
WHERE SaleStatus
= 'Sold')
)
ORDER BY 1,2;
```

9. Еще раз запишите открытый в Блокноте текстовый файл и распечатайте его. Закройте Блокнот.

Далее вы напишете и выполните выражения, создающие представление MyListings, а затем протестируете это представление. Откройте новый файл в Блокноте, запишите его как <имя>Ch06Prob1Part3.sql и запустите SQL\*Plus (если это требуется). Далее придерживайтесь описанной ниже последовательности действий. В первой строке подставьте нужно имя файла. Вместо <имя> в первой и второй строке подставьте свое имя.

```
SPOOL <path><имя>Ch06Prob1Part3-spoolfile.txt
REM <имя>
CLEAR SCREEN
CREATE OR REPLACE VIEW MyListings (Agent,
Addr, City, Price) AS
  SELECT LastName, Address, City, AskingPrice
    FROM Agents INNER JOIN Listings ON
      AgentID = ListingAgentID
        INNER JOIN Properties USING (PropertyID)
    ORDER BY LastName, City;
REM Описание нового представления
DESCRIBE MyListings
REM Отображение на экране информации об агенте Карлинге
REM с использованием представления
SELECT Agent, City, Price
  FROM MyListings
 WHERE UPPER(Agent) = 'CARLING'
  ORDER BY City, Price DESC;
```

1. “Уберите за собой”, набрав следующую команду, удаляющую представление и закрывающую буферный файл:

```
DROP VIEW MyListings;
SPOOL OFF
```

2. Снова запишите текстовый файл, открытый в Блокноте, распечатайте файл сценария и файл спулинга. Закройте Блокнот.
3. В SQL\*Plus введите exit и нажмите клавишу <Enter>, чтобы выйти из Oracle и SQL\*Plus.

## 2. Coffee Merchant

Менеджер Coffee Merchant попросил вас сгенерировать несколько отчетов.

1. Запустите SQL\*Plus, войдите в Oracle и запустите Блокнот. Запишите новый текстовый файл как <имя>Ch06Problem2Part1.sql, подставив вместо <имя> свое имя. В данном упражнении необходимо вначале набрать все выражения в Блокноте, затем скопировать их в SQL\*Plus и выполнить. Таким образом, по завершении работы вы получите полный файл сценария.
2. Инициализируйте базу данных Coffee Merchant, набрав и выполнив приведенную ниже команду. Указанный сценарий гарантирует возврат всех файлов Coffee Mer-

chant в исходное состояние, независимо от того, что вы делали с ними в предыдущих главах. Итак, прежде всего определите полный путь к файлу BuildCoffee.sql. (Папка CoffeeMerchant на компакт-диске, прилагающемся к книге.) Затем выполните указанную ниже команду START, подставив вместо <path> полный путь к файлу BuildCoffee.sql.

```
START <path>\CoffeeMerchant\BuildCoffee
```

Oracle создаст семь таблиц Coffee Merchants и задаст все внешние ключи.

- Инициализируйте буферизацию выхода в файл, выполнив приведенное ниже выражение (укажите путь к нужной папке и подставьте вместо <имя> свое имя).

```
SPOOL C:\<имя>Ch06Prob2Part1.txt
```

- Введите и выполните приведенный ниже запрос, чтобы перечислить имена клиентов, заказавших кофе из Йемена. Правительство ограничило импорт товаров из этой страны, и вам требуется узнать, с кем из клиентов необходимо связаться, и сообщить об этом. Во второй строке подставьте вместо <имя> свое имя.

```
CLEAR SCREEN
REM <имя>
COLUMN CNAME FORMAT A20 HEADING 'NAME'
COLUMN Street FORMAT A30
COLUMN City FORMAT A15
COLUMN State FORMAT A5
COLUMN Zipcode FORMAT A5 HEADING 'ZIP'
SET PAGESIZE 40
SELECT a.FirstName || ' ' || a.LastName AS
CNAME, a.Street,
      a.City, a.State, a.Zipcode
FROM Consumers a
      JOIN Orders b USING (ConsumerID)
      JOIN OrderLines c USING (OrderID)
      JOIN Inventory d USING
      (InventoryID)
      JOIN Countries e USING (CountryID)
WHERE UPPER(e.CountryName) LIKE 'YEM%'
      AND UPPER(d.ItemType) = 'C'
ORDER BY a.Zipcode;
```

- Ведите в SQL\*Plus следующий код, чтобы отключить буферизацию вывода и обновить форматирование столбцов и размер страницы:

```
SPOOL OFF
CLEAR COLUMNS
SET PAGESIZE 14
```

- Запишите весь набранный код в Блокноте и нажмите комбинацию клавиш <Ctrl+N>, чтобы открыть новый текстовый файл, подготовившись к этапу 2.

Во второй части данного упражнения вы перечислите имена клиентов и номера заказов для всех клиентов, которые сделали заказ в октябре 2005 года. Для извлечения

требуемой информации данный запрос объединит три таблицы. Поскольку вы хотите распечатать результаты, вам потребуется открыть буферный файл.

1. Инициализируйте буферизацию выхода в файл, выполнив приведенную ниже команду (укажите путь к нужной папке и подставьте свое имя вместо <имя>).

```
SPOOL C:\<имя>Ch06Prob2Part2.txt
```

2. Введите приведенный ниже код в Блокноте, чтобы задать форматирование столбцов, заменив <имя> своим именем и фамилией. Затем запишите файл как <имя>Ch06Problem2Part2.sql и скопируйте его в SQL\*Plus для выполнения.

```
REM <имя>
REM Перечисление клиентов (Customers) и сумм их заказов для октября
РЯ 2005 года
CLEAR SCREEN
COLUMN OrderID FORMAT 999999
COLUMN CNAME FORMAT A30
COLUMN Total FORMAT $99, 999.99
SET PAGESIZE 80
```

3. Введите в Блокноте следующий запрос и команды SQL\*Plus, скопируйте их в SQL\*Plus и выполните:

```
SELECT OrderID, a.FirstName || ' ' ||
a.LastName AS "CNAME",
      SUM(c.Quantity*c.Price*(1-
      c.Discount)) AS "Total"
FROM Consumers a JOIN Orders b USING
      (ConsumerID)
      JOIN OrderLines c USING
      (OrderID)
WHERE b.OrderDate BETWEEN '01-OCT-05' AND
      '31-OCT-05'
GROUP BY OrderID, a.FirstName, a.LastName
ORDER BY "Total" DESC;
SPOOL OFF
CLEAR COLUMNS
```

4. Запишите файл, открытый в Блокноте, и распечатайте его. Нажмите комбинацию клавиш <Ctrl+N>, чтобы открыть в Блокноте новый текстовый файл.

5. Следующий запрос определит, какие клиенты сделали заказ через сотрудника Хиллари Флинтстил (Hillary Flintsteel). Введите в Блокноте следующий код, задающий форматирование столбцов, заменив <имя> своим именем и фамилией. Затем запишите полученный файл как <имя>Ch06Problem2Part3.sql и скопируйте его в SQL\*Plus для выполнения.

```
SPOOL C:\<имя>Ch06Prob2Part3.txt
REM <имя>
REM Какие клиенты сделали заказ через Флинтстил?
CLEAR SCREEN
COLUMN FirstName FORMAT A12
COLUMN LastName FORMAT A12
```

```
COLUMN Street FORMAT A27 WORD_WRAP
COLUMN City FORMAT A15 WORD_WRAP
COLUMN State FORMAT A02
COLUMN Zipcode FORMAT A05 HEADING 'ZIP'
SET PAGESIZE 55
```

6. Введите приведенные ниже команды SQL и SQL\*Plus в Блокноте и запишите полученный текстовый файл. Скопируйте новый фрагмент кода в SQL\*Plus и выполните его.

```
SELECT DISTINCT c.FirstName, c.LastName, c.Street,
c.City, c.State, c.Zipcode
FROM Consumers c JOIN Orders USING (ConsumerID)
JOIN Employees USING
(EmployeeID)
WHERE EmployeeID = (SELECT EmployeeID
FROM Employees
WHERE LOWER(LastName)
LIKE 'flint%')
ORDER BY c.Zipcode;
SPOOL OFF
CLEAR COLUMNS
SET PAGESIZE 14
```

7. Закройте SQL\*Plus и Блокнот.  
8. Введите все три файла сценария и три полученных буферных файла.

### 3. Rowing Ventures

Создайте описанные ниже файлы сценариев SQL и соответствующие отчеты. Не забудьте распечатать все файлы сценариев и буферные файлы, указав в их заголовках свои имя и фамилию.

Запустите SQL\*Plus, войдите в Oracle и выполните файл сценария BuildRowing.sql, который находится в папке RowingVentures на компакт-диске. Создайте буферный файл-отчет, руководствуясь описанной ниже последовательностью действий.

**Отчет 1.** Создайте запрос, объединяющий таблицы Rowing Ventures Boat, BoatCrew и Person. В запросе перечислите в указанном порядке следующие столбцы: BoatCategory, BowNumber, Position, LastName и FirstName (объединяются с помощью конкатенации и разделены запятыми). Отсортируйте строки по возрастанию значений в столбцах BoatCategory, BowNumber и Position. Используйте команды форматирования COLUMN, чтобы ограничить объединенную с помощью конкатенации запись имени и фамилии 30 символами, место (в лодке) — тремя символами, категорию лодки — семью символами и номер лодки — тремя символами. Вместо непосредственного перечисления значения в столбце Position отобразите “Cox” (“рулевой”) взамен места с номером 0. Все остальные места обозначьте цифрами 1–8. Установите разрывы страниц: по BoatCategory (SKIP PAGE), затем по BowNumber (SKIP 1 DUP). В конце файла сценария уберите все разрывы и форматирование столбцов.

**Отчет 2.** Напишите запрос, в котором будут перечислены имена всех гребцов, располагающихся в той же лодке, что и Тед Симпсон (Ted Simpson). (*Подсказка.* Объедините таблицы BoatCrew и Person, чтобы определить, в какой лодке находится Симпсон, а затем перечислите имена всех гребцов, имеющих такой же номер BoatID.) Учтите, что Тед Симпсон может участвовать в нескольких заездах на разных лодках. Распечатайте запрос и буферный файл. Используйте форматирование, улучшающее восприятие результатов.

**Отчет 3.** Напишите и запустите файл сценария, создающий (или замещающий) представление RowerOrganizations. Представление должно отображать имена гребцов и организации, к которым они принадлежат (OrganizationName). Представление объединяет четыре таблицы: Person, BoatCrew, Boat и Organization. Перечислите только столбцы OrganizationName, FirstName и LastName (в указанном порядке). Напечатайте запрос, создающий представление, и запустите его, используя представление для отображения на экране трех указанных столбцов. Перед запросом с помощью выражений SQL\*Plus задайте форматирование столбцов и размер страницы. Значение переменной PAGESIZE установите равным 60 (максимальную), ширину столбца OrganizationName — равной 40 символам, столбцы имен должны иметь ширину не более 15 символов. Отсортируйте результаты по OrganizationName, затем по LastName и по FirstName. После запуска запроса снимите форматирование со столбцов, верните размер страницы (14 строк) и удалите представление.

## 4. Broadcloth Clothing

Запустите SQL\*Plus, войдите в Oracle и выполните файл сценария BuildClothing.sql, который находится в папке Chapter06\Broadcloth на компакт-диске. Представление ContactInfo должно иметь статус “только для чтения” и отображать строки и столбцы таблицы Contact. При этом отображаются только столбцы (в указанном порядке) FirstName, LastName, Address, City, State, Nation и PrimaryLanguage и строки, значение поля Nation которых равно China, India, Malaysia или Thailand. Создав представление, откройте буферный файл и запустите команды SQL\*Plus, ограничивающие ширину столбцов Nation, City и LastName 25 символами. Используя представление, запустите запрос, отображающий на экране столбцы LastName, City и Nation (в указанном порядке), отсортированные по Nation, City, LastName (в указанном порядке). Распечатайте сценарий и буферный файл.

Напишите запрос с подзапросом, перечисляющим значения FactoryID, City, Nation и MaxWorkers для всех фабрик, на которых работает больше людей, чем среднее число MaxWorkers. Направьте выход в буферный файл. *Подсказка.* Используется только таблица Factory. Используйте команды COLUMN для форматирования столбцов City и Nation с максимальной шириной 15 символов; формат столбца MaxWorkers — четыре цифры; разряд тысяч отделен запятой. Отсортируйте резуль-

таты по Nation, а затем по City (оба раза — по возрастанию). Распечатайте файл сценария и буферный файл. Не забудьте ввести свои имя и фамилию в заголовках обоих файлов.

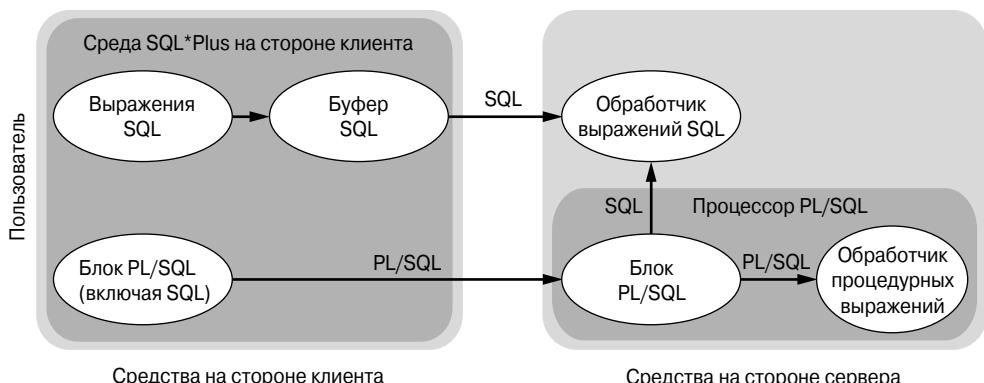
# Использование PL/SQL

**В этой главе...**

- Создание неименованных блоков PL/SQL
- Объявление и использование переменных PL/SQL
- Написание кода PL/SQL
- Создание, открытие, выполнение и закрытие курсоров при обработке нескольких строк из таблицы
- Создание функций и процедур средствами PL/SQL
- Использование циклов и выражений IF для организации итераций и выбора
- Поддержка исключений

## Общие сведения о PL/SQL

PL/SQL — это аббревиатура от *Programming Language/Structured Query Language* (язык программирования/структурированный язык запросов). Он представляет собой расширение известного всем языка SQL и используется только в Oracle. В рамках PL/SQL удается объединить специализированный язык, предназначенный для работы с базой данных с обычным процедурным языком программирования. Основной структурной единицей, используемой в языке программирования, является *блок*, который содержит выражения SQL и PL/SQL. Получив блок PL/SQL, система Oracle может *скомпилировать* его и сохранить в базе. При компиляции блока PL/SQL происходит преобразование высокогоуровневых инструкций PL/SQL в низкоуровневый формат, который более эффективно обрабатывается компьютером. Код PL/SQL можно



**Рис. 7.1.** Обработка блоков SQL и PL/SQL

также сохранить в виде файла сценария и выполнять его в случае необходимости. PL/SQL предоставляет все привычные разработчикам конструкции, типичные для языков программирования третьего поколения, а именно: циклы, операторы выбора, объявление переменных и поддержка ошибок. Считается, что PL/SQL принадлежит к классу объектно-ориентированных языков. На рис. 7.1 показан принцип обработки блоков SQL и PL/SQL системой Oracle.

## Преимущества PL/SQL

Несмотря на то что язык SQL предоставляет все необходимые средства для взаимодействия с реляционными базами данных, некоторые программные конструкции помогли бы более эффективно осуществлять выбор и анализ данных. Так, например, в языке SQL не предусмотрены перечисленные ниже возможности, которые в ряде случаев могут оказаться очень полезными.

- Перехват исключений, возникающих при обработке данных.
- Организация итераций по возвращаемым строкам данных. В процессе перебора можно было бы проверять каждую строку и выявлять требуемые значения или характеристики.
- Защита кода посредством шифрования.
- Хранение кода в базе данных, а не только на клиентской машине.
- Организация принятия решений и альтернативного ветвления при обработке строк таблицы.

SQL — очень мощный и в то же время простой для изучения язык работы с данными. PL/SQL добавляет важные языковые конструкции, отсутствующие в обычном SQL. Интегрированный в систему Oracle, PL/SQL постоянно совершенствуется для обеспечения более полной интеграции с базами данных и высокой эффективности. Достичь подобных результатов позволяют, например, такие меры, как платформенно-ориентированная компиляция кода PL/SQL. Это означает, что код PL/SQL компилируется в код на языке C (язык, на котором была написана сама система Oracle). При этом скорость выполнения скомпилированного кода становится очень высокой. Помимо поддержки выражений SELECT, PL/SQL обеспечивает использование любых команд для работы с данными, например INSERT, UPDATE и DELETE. Однако PL/SQL не поддерживает выражений, предназначенных для определения данных SQL (DDL), таких как CREATE.

PL/SQL уменьшает накладные расходы и повышает производительность. Без PL/SQL система Oracle могла бы обрабатывать лишь последовательности выражений SQL. Обработка каждого нового SQL-выражения означала бы еще одно обращение к серверу. PL/SQL, напротив, может передать серверу целый блок SQL-выражений, уменьшая тем самым объем данных, которым сервер обменивается с клиентом. PL/SQL нетрудно освоить. В простейшем случае можно создать блок PL/SQL, включив выражение SELECT между ключевыми словами BEGIN и END.

PL/SQL тесно интегрирован с сервером Oracle. Большинство типов данных PL/SQL поддерживается также и в Oracle. PL/SQL предоставляет специальный атрибут для присвоения внутренней переменной PL/SQL того же типа, который имеет столбец таблицы.

## Типы блоков PL/SQL

В PL/SQL существуют три типа блоков: неименованные блоки, процедуры и функции. Неименованные, или анонимные, блоки представляют собой наборы выражений PL/SQL, пригодные для компиляции и выполнения. Как следует из названия, неименованные блоки не имеют имени, кроме того, они не могут храниться в базе данных. *Подпрограмма* — это компьютерная программа, которая содержится в составе другой программы. Она вызывается из включающей программы и выполняется относительно независимо. В PL/SQL предусмотрены два типа подпрограмм: процедуры и функции. Они помогают обеспечить более надежный обмен с базой и реализовать интегрированные программные решения. *Процедура* — это именованный блок PL/SQL, который хранится на сервере Oracle и выполняет некоторое действие или набор действий. *Функция* — это именованный блок PL/SQL, который хранится на сервере Oracle и по завершении работы возвращает значение. Более подробно о процедурах и функциях будет сказано далее в этой главе.

## Неименованные блоки

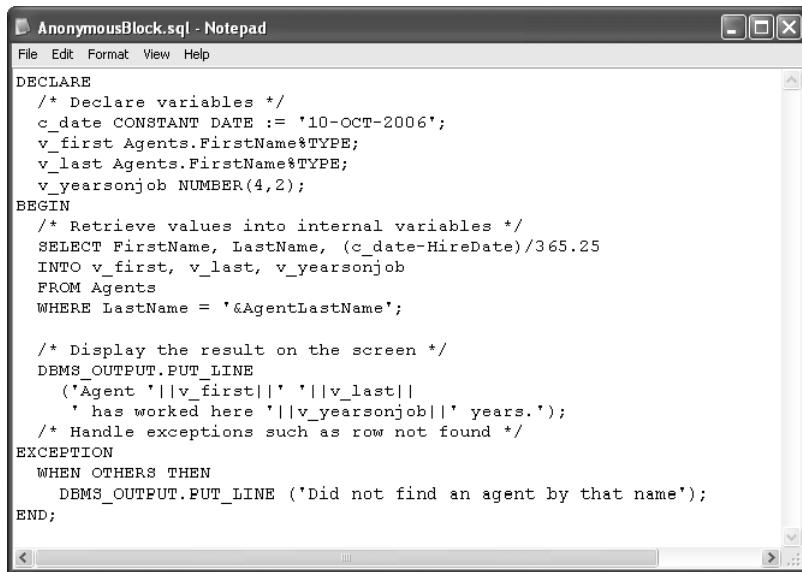
Неименованные блоки языка PL/SQL не находятся на сервере Oracle, обычно они хранятся в виде файлов сценариев SQL; для файлов этого типа принято расширение .sql. Выполнить блок кода можно в любой момент. Для этого надо загрузить SQL\*Plus и запустить файл сценария следующим образом:

```
START <имя_файла_содержащего_блок_PL/SQL>
```

Каждый, кто имеет доступ к компьютеру, диску и файлу, где хранится блок кода, может в случае необходимости запустить сценарий. Неименованные блоки удобно использовать для выполнения действий, которые не повторяются очень часто. В таком блоке могут содержаться три раздела: декларации, исполняемый код и обработчики исключений. Исполняемый код является обязательной частью каждой программы PL/SQL; другие разделы могут отсутствовать. Правила записи неименованного блока приведены ниже.

```
[DECLARE]
BEGIN
[EXCEPTION]
END;
```

Необязательный раздел *деклараций* начинается с ключевого слова DECLARE. В нем объявляются переменные и константы, требуемые для данного блока. В разделе деклараций должна быть определена каждая переменная, необходимая для работы блока. При определении указывается имя переменной, тип данных и необязательное инициализационное значение. Подобно переменным в других языках программирования, переменная PL/SQL резервирует память для хранения значений, используемых исключительно в рамках блока PL/SQL. За пределами блока переменные недоступны, поэтому они по сути являются *закрытыми переменными*. Константа занимает область памяти, существует в течение того времени, когда блок PL/SQL доступен, и содержит значение. Это значение не может быть изменено при выполнении кода. Раздел *исполняемого кода* помещается между ключевыми словами BEGIN и END. В нем содержится код PL/SQL и выражения SQL. Выражения SQL предназначены для организации доступа к таблицам, в то время как выражения PL/SQL манипулируют данными, не обращаясь к таблицам. Необязательный раздел *обработки исключений* начинается с ключевого слова EXCEPTION и позволяет контролировать из программы ситуации, при которых возникают ошибки. Обычно в разделе обработки исключений отображаются сообщения об ошибках и указываются их источники. Управление обработчиками, находящимся в конце блока PL/SQL, передается автоматически. Независимо от того, содержится в блоке раздел обработки исключений или нет, последним выражением в нем является ключевое слово END, за которым следует точка с запятой. Точка с запятой также завершает любое выражение внутри блока PL/SQL, включая



```

AnonymousBlock.sql - Notepad
File Edit Format View Help

DECLARE
    /* Declare variables */
    c_date CONSTANT DATE := '10-OCT-2006';
    v_first Agents.FirstName%TYPE;
    v_last Agents.LastName%TYPE;
    v_yearsonjob NUMBER(4,2);
BEGIN
    /* Retrieve values into internal variables */
    SELECT FirstName, LastName, (c_date-HireDate)/365.25
    INTO v_first, v_last, v_yearsonjob
    FROM Agents
    WHERE LastName = '&AgentLastName';

    /* Display the result on the screen */
    DBMS_OUTPUT.PUT_LINE
        ('Agent'||v_first||' '||v_last||
         ' has worked here '||v_yearsonjob||' years.');
    /* Handle exceptions such as row not found */
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE ('Did not find an agent by that name');
END;

```

Рис. 7.2. Пример блока PL/SQL

те, которые находятся в разделе деклараций или в разделе обработки исключений. Пример блока PL/SQL приведен на рис. 7.2.

## Структура раздела деклараций

Рассмотрим разделы неименованного блока, представленные на рис. 7.2. Раздел деклараций начинается с комментариев. Комментарии поясняют назначение последующих строк кода, содержащихся в разделе. Комментарии помещаются между парами символов /\* и \*/ могут занимать любое количество строк. Существует также способ задать комментарии в одной строке. Признаком их начала является последовательность символов -. В данном примере в разделе деклараций заданы три переменные и одна константа. Имя константы начинается с символов `c_`, а в именах переменных используется префикс `v_`. Для переменных задаются те же типы, что и для соответствующих столбцов таблицы. Указать для переменной PL/SQL тип, совпадающий с типом столбца таблицы, можно следующим образом:

`<имя_таблицы>. <имя_столбца>%TYPE`

На рис. 7.2 видно, что `v_first` имеет тот же тип, что и данные в столбце `LastName` таблицы `Agents` (выражение `Agents.LastName`). Общие правила объявления переменных условно показаны ниже.

`<имя_переменной> [CONSTANT] <тип_данных> [NOT NULL]  
[ := | DEFAULT <значение выражения>];`

Например, объявленная ниже `v_taxrate` — это числовая переменная с начальным значением 18 . 75%. Каждое выражение обязательно должно оканчиваться точкой с запятой.

```
v_taxrate NUMBER (4,4) := .1875;
```

Имя переменной может содержать до 30 символов (букв, цифр и специальных знаков) и должно начинаться с буквы. Нельзя, чтобы имя переменной совпадало с именем столбца, ссылка на который содержится в блоке PL/SQL. Если это правило нарушено, система Oracle посчитает, что имя относится к столбцу таблицы, а не к переменной. Многие программисты придерживаются соглашения об именовании, согласно которому имена констант, переменных и глобальных переменных начинаются соответственно с `c_`, `v_` или `g_`. *Глобальной* называется переменная, к которой возможно обращение из-за пределов блока PL/SQL.

Константа внешне похожа на переменную, но ее значение не может изменяться в процессе выполнения программы. Применять именованные константы вместо конкретных значений (например, числа 0,1875) удобно, так как при этом можно изменить значения в блоке, модифицировав объявление в разделе деклараций. Даже если значение встречается в блоке лишь один раз, желательно объявить его как именованную константу. Это делает программу более простой для восприятия.

Существуют четыре категории переменных PL/SQL: скалярные, составные переменные, ссылки и большие объекты (*large object* — LOB). Скалярный тип данных можно сравнить со скалярным подзапросом, определенным в главе 6. Скалярная переменная содержит единственное значение. Скалярные типы — это те же типы, которые применяются при объявлении столбцов таблиц. Дополнительно в PL/SQL используется тип Boolean. Элемент данных Boolean может содержать значение TRUE, FALSE или NULL.

Выражение NOT NULL, которое может присутствовать в объявлении переменной, указывает на то, что для переменной недопустимо значение NULL. Из этого следует, что такая переменная должна быть инициализирована; в противном случае система Oracle генерирует исключение.

Если после имени переменной вы укажете “`:=`” или другое необязательное выражение, то переменная будет инициализирована перед началом выполнения команд из раздела исполняемого кода. В противном случае начальное значение переменной будет установлено равным NULL.

## Раздел исполняемого кода

Ключевое слово BEGIN помечает начало раздела исполняемого кода. Как было сказано ранее, это единственный раздел, который *обязательно* должен присутствовать в блоке PL/SQL. В него можно включать выражения SELECT, предназначенные для получения информации из таблиц баз данных. Однако в блоке PL/SQL надо использовать

модифицированный вариант оператора SELECT, в котором присутствует также выражение INTO, предназначенное для присвоения переменным значений, извлеченных из столбцов. Другими средствами вы не сможете отобразить извлеченные данные. Как вы помните, данный вариант оператора SELECT допускает упрощенную запись.

```
SELECT <имя_столбца_1> [,<имя_столбца_2>, ... ]  
INTO <имя_переменной_1> [, <имя_переменной_2>, ... ]  
FROM <имя_таблицы> | <имя_представления>  
WHERE <условие>;
```

Когда в блоке PL/SQL встречается выражение SELECT, Oracle выделяет память для хранения и просмотра строк, извлеченных из таблицы. Эта область памяти называется *курсором*. Существуют два типа курсоров. Для выражений SELECT или DML, возвращающих одну строку, Oracle создает *неявный курсор*. Это происходит без вмешательства со стороны разработчика. Если выражение SELECT возвращает больше одной строки, надо вручную создать *явный курсор* и управлять им. Явные курсоры рассмотрены ниже в этой главе. Сейчас же мы будем использовать выражения SELECT, которые либо возвращают одну строку, либо вовсе не извлекают данных. Поэтому в настоящий момент вам не придется заботиться о курсорах.

В блоке PL/SQL (см. рис. 7.2) возвращается только одна строка. Переменная подстановки &AgentLastName вызывает отображение подсказки:

```
Enter value for agentlastname:
```

Получив данное сообщение, надо ввести ответ (например, Carling) и нажать клавишу <Enter>. Это значение заменит переменную подстановки в выражении WHERE команды SELECT. По команде SELECT будет извлечено содержимое двух столбцов и значение виртуального столбца для выбранного имени агента. Эти значения помещаются в переменные, определенные в разделе деклараций и указанные после ключевого слова INTO. Приведенное ниже выражение вычисляет, сколько лет выбранный агент проработал в компании Redwood Realty.

```
(C_date-HireDate)/365.25
```

Поскольку Oracle помещает значения из выбранных столбцов и вычисленное значение в переменные, они отображаются с помощью функции PUT\_LINE пакета DBMS\_OUTPUT. Такой способ можно применить для вывода на экран данных из программы PL/SQL.

В отличие от SQL\*Plus, команда SELECT, находящаяся в блоке PL/SQL, не приводит к непосредственному выводу данных на экран. Вместо этого надо использовать пакет DBMS\_OUTPUT, содержащий различные функции отображения требуемых значений.

**Совет.** Если вы выполняете блок PL/SQL, подобный показанному на рис. 7.2, а выходные данные не отображаются, вам надо установить значение переменной окружения SQL\*Plus SERVEROUTPUT, равное ON. Для этого надо в ответ на приглашение SQL ввести команду SET SERVEROUTPUT ON и нажать клавишу <Enter>.

Параметром функции PUT\_LINE (см. рис. 7.2) является строка, предназначенная для отображения. В данном случае она формируется путем конкатенации выражений и строковых констант.

## Раздел обработки исключений

Необязательный раздел обработки исключений в составе блока PL/SQL определяет действия, которые должны быть предприняты в том случае, если в процессе выполнения блока возникнет исключение. Исключениями являются ошибки или непредвиденные результаты. При возникновении исключения управление получает код его обработки. Предположим, что в составе вашего кода есть следующая команда:

```
SELECT 12/0
INTO v_numbervar
FROM DUAL;
```

Попытка деления на нуль приведет к возникновению исключения. Если в блоке PL/SQL содержится раздел EXCEPTION, управление будет передано ему, система Oracle выполнит любые содержащиеся в нем выражения и работа блока PL/SQL завершится. Если команда SELECT не возвратит ни одной строки, будет сгенерировано исключение другого типа.

Существуют два типа исключений: заданные пользователем и предопределенные. Исключения, заданные пользователем, — это ошибки, который предусмотрел пользователь. Они не обязательно связаны с ошибками Oracle. Такие исключения в некоторых случаях рассматриваются разработчиками как удобный способ передачи управления другому разделу блока PL/SQL. Исключения, заданные пользователем, объявляются в разделе деклараций блока. Для этой цели используется следующее выражение:

```
<имя_исключения> EXCEPTION;
```

При необходимости можно принудительно вызвать исключение, определяемое пользователем. Для этой цели предусмотрено выражение RAISE, которое записывается следующим образом:

```
RAISE <имя_исключения>;
```

Код, предназначенный для поддержки исключений, определяемых пользователем, помещается в раздел обработки исключений. С примерами кода подобного назначения вы познакомитесь далее в этой главе.

Предопределенные исключения связаны с типичными ошибками, возникающими при работе Oracle. К ним относятся попытка модификации константы, деление на нуль и попытка присвоить значение объекту NULL. Предопределенные исключения

**ТАБЛИЦА 7.1.** Некоторые предопределенные исключения Oracle

<i>Код ошибки Oracle</i>	<i>Описание</i>
ORA-0001	Нарушение ограничения
ORA-1001	Недопустимая операция с курсором
ORA-1403	Данные не найдены
ORA-1422	Выражение SELECT INTO возвращает более одной строки
ORA-1476	Деление на нуль
ORA-1722	Недопустимое преобразование в число (например, попытка преобразовать строку, содержащую буквы)
ORA-6502	Усечение, ошибка преобразование или арифметическая ошибка
ORA-6511	Попытка открыть курсор, который уже был открыт

и коды ошибок приведены в табл. 7.1.

Ключевое слово OTHERS, приведенное на рис. 7.2 в разделе исключений, обозначает все исключения, которые не были явно указаны в выражении WHEN команды EXCEPTION. Другими словами, так задается обработчик не предусмотренных исключений. Он используется в основном тогда, когда разработчик не хочет вникать в суть исключений, которые могут возникнуть в блоке PL/SQL.

## Создание неименованных блоков

Задача создания неименованных блоков практически сводится к написанию выражений SQL и SQL\*Plus. Главное отличие состоит в том, что для PL/SQL принятые соглашения, согласно которым разделы следуют в определенном порядке, а каждое выражение оканчивается точкой с запятой. Перед тем как приступить к созданию неименованного блока, восстановим базу данных Redwood Realty в исходном виде.

### Инициализация базы данных Redwood Realty

Перед тем как приступить к написанию блоков PL/SQL, вам надо инициализировать базу данных. Сделать это надо для того, чтобы минимизировать вероятность получения результатов, отличных от приведенных здесь. Все примеры, приведенные в данной главе, ориентированы на использование SQL\*Plus. Описание любых действий предполагает, что вы уже запустили SQL\*Plus и зарегистрировались в системе Oracle. Исключения из этого правила будут специально оговорены. Итак, запустите SQL\*Plus и зарегистрируйтесь в качестве пользователя Oracle.

Для того чтобы инициализировать базу данных, выполните следующие действия.

```

* Oracle SQL*Plus
File Edit Search Options Help
Agents      table populated.
ContactReason table populated.
CustAgentList table populated.
Customers    table populated.
LicenseStatus table populated.
Listings     table populated.
Properties   table populated.
SaleStatus   table populated.

Table          Rows
-----
Agents         29
ContactReason 3
CustAgentList 1,018
Customers      2,500
LicenseStatus  16
Listings       502
Properties     2,000
SaleStatus     3

SQL> START C:\Ch07AgentsHR
SQL>

```

**Рис. 7.3.** Инициализация базы данных Redwood Realty

1. Определите путь к файлу сценария `BuildRedwood.sql` в папке Redwood Realty. Этот сценарий надо найти среди файлов данных для этой книги. Выполните приведенную ниже команду, подставив вместо `<путь_1>` найденный путь.

```
START <путь_1>\RedwoodRealty\BuildRedwood
```

2. Задайте ширину столбцов. Определите путь к файлу `Ch07AgentsHR` для главы 7. Выполните приведенную ниже команду, подставив вместо `<путь_2>` найденный путь.

```
START <путь_2>\Ch07AgentsHR
```

3. На экране должны отобразиться результаты, показанные на рис. 7.3.
4. Оставьте активизированным SQL\*Plus.

## Написание кода для анонимных блоков

Для того чтобы написать блок PL/SQL, надо загрузить привычный вам текстовый редактор (в системе Windows это может быть, например, Блокнот) и ввести код. Затем надо скопировать код в SQL\*Plus и выполнить его. Вы также можете сохранить код в виде сценария (записав его в файл с расширением `.sql`) и выполнить посредством команды `START`. Примеры вызова данной команды приведены ниже.

```
START C:\AnonymousBlockExample.sql
```

или

```
@C:\AnonymousBlockExample.sql
```

Готовый неименованный блок надо записать на диск. Как вы уже знаете, такие блоки не хранятся в базе данных. Поэтому каждый, кто хочет выполнить его, должен, во-первых, иметь доступ к узлу, на котором он находится, во-вторых, знать путь к файлу и, в-третьих, имя файла неименованного блока PL/SQL.

Блок PL/SQL, который мы собираемся написать, будет запрашивать таблицу Agents с целью подсчета числа агентов. Затем он обратится к таблице AgentsHR и вычислит среднюю основную зарплату и среднее время отпуска. И наконец, код, о котором идет речь, использует функцию форматирования Oracle, чтобы представить числовое значение как сумму в долларах, а затем отобразит результаты на экране. Чтобы подготовиться к работе, запустите Блокнот или другой текстовый редактор, подходящий для данной цели. Можно также (но не обязательно) загрузить SQL\*Plus.

Для того чтобы создать неименованный блок PL/SQL, выполните следующие действия.

1. Откройте редактор Блокнот и начните раздел деклараций. Помните, что в конце каждой строки данного раздела (за исключением DECLARE) должна стоять точка с запятой.

```
DECLARE
    v_AvgSal AgentsHR.BaseSalary%TYPE;
    v_AvgVac AgentsHR.Vacation%TYPE;
    v_Count INTEGER NOT NULL := 0;
    v_AvgSalChar VARCHAR2(15);
```

2. Создайте средствами редактора Блокнот раздел исполняемого кода. Пустые строки и лишние пробелы включаются для того, чтобы сделать код более удобным для восприятия. (В данном случае в начале каждой строки, расположенной после BEGIN, включены два пробела. При желании вы можете использовать другую величину отступа.)

```
BEGIN
    /* Подсчет числа агентов, указанных в таблице */
    SELECT COUNT(AgentID) INTO v_Count FROM Agents;

    /* Вычисление средних значений */
    SELECT AVG(BaseSalary), AVG(Vacation)
    INTO v_AvgSal, v_AvgVac
    FROM AgentsHR;

    /* Преобразование в форматированную строку символов */
    v_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');

    /* Отображение результатов */
    DBMS_OUTPUT.PUT_LINE('Average Salary: ' || v_AvgSalChar);
    DBMS_OUTPUT.PUT_LINE('Average Vacation: ' || v_AvgVac || ' weeks');
    DBMS_OUTPUT.PUT_LINE('Based on: ' || v_Count || ' agents');
```

3. Чтобы завершить неименованный блок PL/SQL, введите приведенный ниже код. Не забудьте ввести после END точку с запятой. Заметьте, что в последней строке

```

AnonBlockOne.sql - Notepad
File Edit Format View Help

DECLARE
    v_AvgSal AgentsHR.BaseSalary%TYPE;
    v_AvgVac AgentsHR.Vacation%TYPE;
    v_Count INTEGER NOT NULL := 0;
    v_AvgSalChar VARCHAR2(15);
BEGIN
    /* Count the number of agents in the table */
    SELECT COUNT(AgentID) INTO v_Count FROM Agents;

    /* Compute averages */
    SELECT AVG(BaseSalary), AVG(Vacation)
    INTO v_AvgSal, v_AvgVac
    FROM AgentsHR;

    /* Convert to a formatted character string */
    v_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');

    /* Display results */
    DBMS_OUTPUT.PUT_LINE('Average Salary: ||v_AvgSalChar');
    DBMS_OUTPUT.PUT_LINE('Average Vacation: || v_AvgVac|| weeks');
    DBMS_OUTPUT.PUT_LINE('Based on: || v_Count|| agents');

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/

```

Рис. 7.4. Неименованный блок PL/SQL

содержится косая черта (/), которая указывает Oracle на то, что после загрузки блока в SQL\*Plus необходимо скомпилировать код.

```

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/

```

- Выберите пункт File меню редактора, затем пункт Save As, щелкните в поле редактирования File name и перейдите в папку вашего компьютера, в которой вы собираетесь сохранить код. (Запомните эту папку!) Введите <имя>AnonBlockOne.sql, заменив выражение <имя> вашим именем, откройте раскрывающийся список Save as type и выберите пункт All Files. Щелкните на кнопке Save, чтобы сохранить файл. На рис. 7.4 показан блок PL/SQL, открытый в окне редактора Блокнот.
- Выберите пункт меню File, а затем пункт Exit, чтобы закрыть Блокнот.

Перед тем как выполнить код, обсудим назначение каждой его строки. В разделе деклараций определены три локальные переменные. (Локальной называется переменная, которая доступна только в пределах того блока PL/SQL, где она была создана.) Локальная переменная v\_AvgSal предназначена для хранения средней заработной платы, и для нее использован такой же тип данных, как и для столбца BaseSalary

таблицы AgentsHR. Аналогично, в переменную v\_AvgVac должно быть помещено значение среднего времени отпуска, вычисленное на основании данных из столбца Vacation таблицы AgentsHR. В переменной v\_Count содержится целое число, определяющее количество строк, описывающих агентов в Agents. И наконец, локальная переменная v\_AvgSalChar предназначена для форматированной строки, представляющей среднюю зарплату.

Раздел исполняемого кода содержит комментарии, помещенные между парами символов /\* и \*/. Первое выражение SELECT вычисляет число строк с описаниями агентов в таблице Agents и помещает результат в локальную переменную v\_Count. Второе выражение SELECT определяет среднее значение BaseSalary и Vacation. Полученные величины помещаются в две локальные переменные для дальнейшего использования. И наконец, в процессе присваивания информации о средней заработной плате преобразуется в форматированную строку. Соответствующее выражение приведено ниже.

```
V_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');
```

Последние три строки раздела исполняемого кода отображают результаты, выполняния при этом конкатенацию строковых литералов и значений локальных переменных.

Раздел обработки исключений содержит выражение, отображающее сообщение об ошибке (SQLERRM), которое соответствует условию OTHERS. Если в разделе исполняемого кода возникает ошибка, управление передается этой команде и сообщение появляется на экране. Несмотря на то что мы имеем право удалить раздел обработки исключений, делать этого не стоит, так как при этом информация об ошибках, возникающих при изменении содержимого базы, будет скрыта от нас и мы даже не будем знать о самом факте возникновения ошибки. Желательно включать раздел обработки исключений в каждый блок PL/SQL — не только неименованный, но и любой другой.

Ключевое слово END, за которым следует точка с запятой, является признаком конца неименованного блока PL/SQL. Косая черта сообщает системе Oracle о том, что блок PL/SQL должен быть выполнен. Этот символ можно удалить и вводить его вручную после загрузки блока в SQL\*Plus, но удобнее, чтобы он содержался в коде PL/SQL. При этом не приходится помнить о том, что данный символ надо ввести вручную.

## Настройка среды SQL\*Plus

Если вы попытаетесь запустить созданный блок PL/SQL в SQL\*Plus, ничего не произойдет. Единственным результатом будет строка, которую система Oracle отобразит на экране и которая сообщает о том, что выполнение кода завершилось.

```
PL/SQL procedure successfully completed.
```

Для того чтобы в результате выполнения PUT\_LINE из пакета DBMS\_OUTPUT данные появились на экране (т.е. на консоли), надо установить переменную SERVEROUTPUT из среды SQL\*Plus. Для этого следует выполнить следующую команду:

```
SET SERVEROUTPUT ON
```

Чтобы избавиться от необходимости вводить данное выражение вручную, его можно поместить в состав неименованного блока.

## Выполнение неименованного блока PL/SQL

После того как вы сохранили созданный вами анонимный блок PL/SQL, его можно запустить на выполнение и ознакомиться с полученными результатами. Теперь уже нет необходимости в редакторе Блокнот. Его можно закрыть; вместо этого следует запустить SQL\*Plus и зарегистрироваться в системе Oracle.

Для того чтобы выполнить неименованный блок PL/SQL, выполните следующие действия.

1. Введите приведенную ниже команду. Замените выражение *<путь>* реальным путем, а *<имя>* — вашим именем.

```
START <путь>\<имя>AnonBlockOne
```

Единственное сообщение, появившееся на экране, информирует вас о том, что процедура успешно завершена.

2. Дело в том, что в SQL\*Plus для переменной окружения SERVEROUTPUT установлено значение OFF. Выполните следующую команду:

```
SET SERVEROUTPUT ON
```

3. Поскольку неименованный блок PL/SQL по-прежнему находится в буфере SQL\*Plus, введите косую черту (/) и нажмите клавишу <Enter>, чтобы выполнить его.

На экране отобразятся три строки, представляющие результаты выполнения блока PL/SQL, кроме того, будет снова выведено сообщение об успешном завершении (рис. 7.5).

4. Введите exit и нажмите клавишу <Enter>, чтобы завершить сеанс работы с Oracle и закрыть SQL\*Plus.

В рассмотренном выше примере использовался неявный курсор, так как команда SELECT предоставляла лишь одну строку — среднюю заработную плату и среднее время отпуска для всех агентов. Теперь мы модифицируем неименованный блок так, чтобы в результате его работы возвращалась средняя зарплата и время отпуска группы агентов. Все необходимые для этого данные по-прежнему находятся в таблице AgentsHR. Изменения по сравнению с предыдущим примером состоят лишь в том, что выражение SELECT возвращает пять строк — по одной строке для каждой группы агентов.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> START C:\AnonBlockOne
PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL> /
Average Salary: $3,250.00
Average Vacation: 3 weeks
Based on: 29 agents

PL/SQL procedure successfully completed.

SQL>

```

Рис. 7.5. Выполнение неименованного блока PL/SQL

## Модификация неименованного блока для отображения нескольких строк

Модификация неименованного блока происходит так же, как и модификация любого другого файла сценария SQL. Для этого надо открыть блок в текстовом редакторе, внести требуемые изменения и сохранить блок PL/SQL под тем же или другим именем. Модифицировав неименованный блок, вы можете запустить его в SQL\*Plus, используя команду START, и проверить результаты своей работы. Изменим блок PL/SQL, созданный ранее. Чтобы сохранить имеющийся код, откройте блок и сохраните его под новым именем.

Мы будем модифицировать блок PL/SQL так, чтобы он предоставлял статистические данные о пяти группах агентов Redwood Realty. В частности, вместо того, чтобы вычислять среднее значение для всех риэлторов, неименованный блок будет отображать усредненные величины для каждой группы. В базе зарегистрировано пять групп с номерами от 10 до 14. Прежде чем приступить к модификации блока, запустите Блокнот (или другой текстовый редактор).

Модификация неименованного блока PL/SQL и сохранение его под новым именем предполагает следующие действия.

1. Выберите пункт меню File, затем пункт Open, щелкните на списке Look in, чтобы перейти в папку, в которой вы сохранили файл AnonBlockOne.sql (в качестве префикса, конечно же, указано ваше имя). Затем в списке Files of type выберите пункт All Files и дважды щелкните на файле AnonblockOne.sql, чтобы открыть его.
2. Вставьте пустую строку между объявлениями переменных v\_AvgVac и v\_Count в разделе деклараций и введите в этой строке следующий фрагмент кода:  

```
v_Team AgentsHR.Team%TYPE;
```
3. Модифицируйте выражение SELECT в разделе исполняемого кода следующим образом. Добавьте v\_Team в список INTO команды SELECT в качестве первой переменной, добавьте в конец команды SELECT выражение GROUP BY Team и включите в выражение ORDER BY Team.

чите дополнительную функцию PUT\_LINE. Ваш модифицированный код должен выглядеть так, как показано ниже (добавленные и измененные строки помечены комментариями).

```

DECLARE
    v_AvgSal AgentsHR.BaseSalary%TYPE;
    v_AvgVac AgentsHR.Vacation%TYPE;
    v_Team AgentsHR.Team%TYPE;                                /* новая строка */
    v_Count INTEGER NOT NULL := 0;
    v_AvgSalChar VARCHAR2(15);
BEGIN
    /* Вычисление количества агентов, указанных в таблице */
    SELECT COUNT(AgentID) INTO v_Count FROM Agents;
    /* Вычисление средних значений */
    SELECT Team, AVG(BaseSalary), AVG(Vacation) /* изменено */
        INTO v_Team, v_AvgSal, v_AvgVac           /* изменено */
        FROM AgentsHR                            /* изменено */
        GROUP BY Team;                          /* новая строка */
    /* Преобразование в форматированную строку символов */
    v_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');
    /* Отображение результатов */
    DBMS_OUTPUT.PUT_LINE('Team: '||v_Team);      /* новая строка */
    DBMS_OUTPUT.PUT_LINE('Average Salary: '||v_AvgSalChar);
    DBMS_OUTPUT.PUT_LINE('Average Vacation: '||v_AvgVac|||
        ' weeks');
    DBMS_OUTPUT.PUT_LINE('Based on: '||v_Count||' agents');
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/

```

- Сохраните измененный файл под новым именем: выберите пункт меню **File**, затем **Save As**. Щелкните на списке **Save as type**, выберите **All Files**, щелкните в поле редактирования **File name**, введите <имя>AnonBlockTwo.sql (заменив выражение <имя> своим именем) и щелкните на кнопке **Save**. Редактор Блокнот сохранит измененный файл.
- Загрузите SQL\*Plus (если это необходимо), зарегистрируйтесь в системе Oracle и выполните приведенные ниже команды. Этим вы очистите экран и разрешите вывод из блока PL/SQL.

```
CLEAR SCREEN
SET SERVEROUTPUT ON
```

- Для того чтобы нам было удобнее обсуждать фрагменты блока PL/SQL, скопируйте весь код из редактора Блокнот в SQL\*Plus. В результате строки будут пронумерованы и мы сможем ссылаться на их номера в процессе рассмотрения кода.
- Находясь в окне SQL\*Plus, перейдите к последней строке, содержащей косую черту (/), и нажмите клавишу <Enter>, чтобы выполнить код. Система Oracle сгенерирует сообщение об ошибке, которое будет выведено средствами обработки исключений (рис. 7.6).

```

SQL> DECLARE
  2   v_AvgSal AgentsHR.BaseSalary%TYPE;
  3   v_AvgVac AgentsHR.Vacation%TYPE;
  4   v_Team AgentsHR.Team%TYPE;
  5   v_Count INTEGER NOT NULL := 0;
  6   v_AvgSalChar VARCHAR2(15);
  7 BEGIN
  8   /* Count the number of agents in the table */
  9   SELECT COUNT(AgentID) INTO v_Count FROM Agents;
 10
 11  /* Compute averages */
 12  SELECT Team, AVG(BaseSalary), AVG(Vacation)
 13  INTO v_Team, v_AvgSal, v_AvgVac
 14  FROM AgentsHR
 15  GROUP BY Team;
 16
 17  /* Convert to a formatted character string */
 18  v_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');
 19
 20  /* Display results */
 21  DBMS_OUTPUT.PUT_LINE('Team: '||v_Team);
 22  DBMS_OUTPUT.PUT_LINE('Average Salary: '||v_AvgSalChar);
 23  DBMS_OUTPUT.PUT_LINE('Average Vacation: '|| v_AvgVac||' weeks');
 24  DBMS_OUTPUT.PUT_LINE('Based on: '|| v_Count||' agents');
 25 EXCEPTION
 26   WHEN OTHERS THEN
 27     DBMS_OUTPUT.PUT_LINE(SQLERRM); SQLERRM - это текст сообщения об ошибке
 28 END;
 29 /
ORA-01422: exact fetch returns more than requested number of rows Сообщение об ошибке,
PL/SQL procedure successfully completed. отображаемое посредством
SQL> кода поддержки исключений

```

**Рис. 7.6.** Использование неявного курсора приводит к возникновению ошибки во время выполнения

Что стало причиной ошибки ... more than the requested number of rows ... (... число строк больше, чем было указано ...) ? Ошибка возникла потому, что выражение SELECT (строки с 12 по 15 на рис. 7.6) в модифицированном блоке PL/SQL возвращает несколько строк, точнее, пять. Возникает одно из исключений, которые будут описаны в следующем разделе.

## Обработка исключений

В рассмотренном примере содержался раздел обработки исключений, в котором поддерживалось любое исключение, сгенерированное при выполнении неименованного блока PL/SQL. На это указывало выражение WHEN OTHERS. Указав имена конкретных исключений, можно организовать более избирательную их обработку. Имена и описания исключений приведены в табл. 7.2. Представленный ниже фрагмент кода показывает, как можно организовать индивидуальную обработку именованных исключений.

```

EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE('Value error was generated.')
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Code performs a divide by zero.')
  WHEN TOO_MANY_ROWS THEN
    DBMS_OUTPUT.PUT_LINE('Query returned more than one row.')
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM)
END;

```

**ТАБЛИЦА 7.2.** Некоторые типы исключений

<i>Имя исключения</i>	<i>Описание</i>
ACCESS_INTO_NULL	Попытка присвоить значение атрибуту неинициализированного объекта
CASE_NOT_FOUND	Не были выбраны выражения WHEN структуры CASE и отсутствует выражение ELSE по умолчанию
CURSOR_ALREADY_OPEN	Попытка открыть курсор, который уже был открыт
INVALID_CURSOR	Попытка выполнить недопустимое действие курсора, например, закрыть или открыть курсор
NO_DATA_FOUND	Выражение SELECT INTO возвращает нулевое число строк
ROWTYPE_MISMATCH	Переменная курсора и переменная PL/SQL имеют несовместимые типы данных
TOO_MANY_ROWS	Выражение SELECT INTO возвращает более одной строки
VALUE_ERROR	Ошибка преобразования, арифметическая ошибка, усечение или ошибка, связанная с ограничением размера (например, попытка выбрать столбец, содержащий длинную последовательность символов и поместить в ее в переменную PL/SQL меньшего размера)
ZERO_DIVIDE	Попытка деления на нуль

Здесь первые три выражения WHERE проверяют выполнение конкретных условий, а WHERE OTHERS перехватывает все остальные исключения.

Когда в блоке PL/SQL возникает исключение, управление передается разделу обработки исключений. Если в нем не предусмотрено выражение WHEN для данного конкретного исключения, управление получает обработчик WHEN OTHERS. Если все обработчики, в том числе и универсальный WHEN OTHERS, отсутствуют, сообщение об ошибке выводится на экран. После того как управление было передано разделу обработки исключений, оно уже не возвращается в тело блока PL/SQL. Для того чтобы лучше понять принцип обработки исключений, напишем простой блок PL/SQL. Запустите SQL\*Plus, Блокнот или другой текстовый редактор и следуйте приведенным ниже инструкциям. В результате ваших действий будет создан код для генерации исключений и обработки их.

1. Выполните следующую команду для настройки среды SQL\*Plus:

```
SET SERVEROUTPUT ON
```

2. Введите в текстовом редакторе приведенный ниже код и скопируйте его в SQL\*Plus. После косой черты в последней строке нажмите клавишу <Enter>, чтобы скомпилировать и выполнить блок PL/SQL. Не закрывайте текстовый редактор, так как впоследствии вам придется вносить дальнейшие изменения.

```

CLEAR SCREEN
DECLARE
    v_TeamNo AgentsHR.Team%TYPE;
BEGIN
    SELECT Team INTO v_TeamNo FROM AgentsHR;
    SELECT 123/0 INTO v_TeamNo FROM DUAL;
    SELECT Team INTO v_TeamNo FROM AgentsHR
        WHERE Vacation = 20;
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('Divide by zero');
    WHEN TOO_MANY_ROWS THEN
        DBMS_OUTPUT.PUT_LINE('Multiple rows returned');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Something else went wrong');
END;
/

```

Поскольку система Oracle сгенерировала исключение TOO\_MANY\_ROWS, на экране появится сообщение *Multiple rows returned*. На следующее сообщение Oracle (PL/SQL procedure successfully completed) внимание обращать не следует.

3. Вернитесь к текстовому редактору и включите символы -- в начало строки, содержащей первое выражение SELECT. В результате данная строка будет восприниматься как комментарии. Скопируйте весь измененный сценарий в SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить блок. Теперь будет сгенерировано исключение ZERO\_DIVIDE и на экране отобразится сообщение, соответствующее этому случаю (рис. 7.7).
4. Вернитесь к текстовому редактору и введите символы -- перед вторым выражением SELECT, превратив в комментарии и эту строку. Скопируйте весь сценарий в SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить модифицированный блок. На этот раз возникло исключение OTHERS и отобразилось сообщение *Something else is wrong*.
5. Введите exit и нажмите клавишу <Enter>, чтобы завершить сеанс работы с Oracle и закрыть SQL\*Plus. Перейдите в текстовый редактор и сохраните в соответствующей папке блок PL/SQL, использованный вами для демонстрации работы с исключениями.
6. Закройте Блокнот или тот редактор, с которым вы работали.

Вскоре вы узнаете, как справиться с ошибкой, возникающей при попытке отобразить среднюю заработную плату и среднее время отпуска для группы агентов. Чтобы исключение TOO\_MANY\_ROWS не возникало, надо для получения информации из таблиц Redwood Realty использовать явный курсор.

```

SQL> DECLARE
2   v_TeamNo AgentsHR.Team%TYPE;
3 BEGIN
4   -- SELECT Team INTO v_TeamNo FROM AgentsHR; ← В начало строки включен
5   SELECT 123/0 INTO v_TeamNo FROM DUAL;
6   SELECT Team INTO v_TeamNo FROM AgentsHR
7   WHERE Vacation = 20;
8 EXCEPTION
9   WHEN ZERO_DIVIDE THEN
10    DBMS_OUTPUT.PUT_LINE('Divide by zero');
11   WHEN TOO_MANY_ROWS THEN
12    DBMS_OUTPUT.PUT_LINE('Multiple rows returned');
13 WHEN OTHERS THEN
14    DBMS_OUTPUT.PUT_LINE('Something else went wrong');
15 END;
16 /
Divide by zero

PL/SQL procedure successfully completed.

SQL>

```

Рис. 7.7. Результат обработки исключения

## Явные курсоры

Как вы помните, в блоке PL/SQL выражение SELECT, возвращающее одну строку, использует неявный курсор. Это означает, что средства, действующие по умолчанию, позволяют Oracle передавать в блок PL/SQL одну строку, извлеченную из таблицы, и присваивать значения столбцов переменным. Другими словами, неявный курсор поддерживает не более одной строки, возвращаемой выражением SELECT. Блок PL/SQL, в котором выражение SELECT может возвращать несколько строк, требует модификации; в нем надо использовать явный курсор. Для установки явного курсора надо переместить выражение SELECT в раздел деклараций и преобразовать его в курсор. Затем следует инициализировать курсор, написать код для явного извлечения информации из базы данных и предусмотреть закрытие курсора. Для реализации явного курсора потребуется несколько дополнительных строк кода. Кроме того, придется использовать программную конструкцию, которую мы еще не упоминали в данной главе, а именно цикл.

## Итерации в блоках PL/SQL

В PL/SQL предусмотрены три типа циклов, которые позволяют повторно выполнять фрагмент кода до тех пор, пока не выполнится определенное событие (или до тех пор, пока это событие выполняется). Как вы вскоре увидите, с помощью циклов можно указать Oracle передавать программе при каждой итерации очередную строку. В PL/SQL поддерживаются *простые циклы*, циклы WHILE и циклы FOR. В простом цикле выполнение повторяется до тех пор, пока цикл не будет явным образом завершен. Тело цикла WHILE обрабатывается, пока выполняется заданное условие. Цикл

FOR позволяет повторить фрагмент кода заранее определенное число раз.

Простой цикл, подобно всем остальным вариантам цикла, реализует повторное выполнение фрагмента кода PL/SQL. Для того чтобы завершить цикл, надо включить в его тело выражение EXIT или EXIT WHEN. Простой цикл записывается следующим образом:

```
LOOP
    Выражения
END LOOP;
```

Выражение EXIT немедленно завершает цикл. Выражение EXIT WHEN завершает цикл, когда условие, заданное в этом выражении, становится истинным. Например, следующий набор команд будет выполняться до тех пор, пока переменная v\_Counter не превысит значение 15:

```
v_Counter := 1;
LOOP
    v_Counter := v_Counter + 3;
    EXIT WHEN v_Counter > 15;
    DBMS_OUTPUT.PUT_LINE(v_Counter);
END LOOP;
```

Если вы включите приведенный выше код в блок PL/SQL, то команды в цикле будут выполнены пять раз и отобразят значения 4, 7, 10 и 13. Цикл начинается с ключевого слова LOOP и заканчивается выражением END LOOP, за которым следует точка с запятой. Любой код, содержащийся между этими выражениями, является телом цикла. Выражения EXIT или EXIT WHEN могут присутствовать в любом месте тела цикла. Выражение EXIT WHEN, пример которого был показан выше, определяет, когда цикл будет завершен.

Необходимо заметить следующее. Если вы не включите в простой цикл выражение EXIT WHEN или EXIT, то его выполнение никогда не остановится. Цикл, выполняющийся без остановки, называется бесконечным циклом. Подобные циклы возникают значительно чаще, чем хотелось бы. Поэтому необходимо тщательно следить за наличием выражения EXIT WHEN, а также убедиться, что на определенном этапе работы заданное условие станет истинным. Например, цикл, показанный ниже, является бесконечным, несмотря на то, что в нем есть выражение EXIT WHEN. Наверное, вам понятно, почему цикл никогда не завершится.

```
/* Не пытайтесь выполнить этот код! */
v_Counter := 1;
LOOP
    v_Counter := v_Counter - 1;
    EXIT WHEN v_Counter > 15;
    DBMS_OUTPUT.PUT_LINE('Infinite loop. Help!');
END LOOP;
```

Цикл WHILE выполняется до тех пор, пока заданное условие равно true. Этот цикл записывается следующим образом:

```
WHILE
    <условие>
```

```

LOOP
  Выражения
END LOOP;

```

Например, следующий цикл отобразит пять значений и завершится, потому что значение `v_Counter` превысит 15:

```

v_Counter := 1;
WHILE v_Counter <= 15 LOOP
  DBMS_OUTPUT.PUT_LINE(v_Counter);
  v_Counter := v_Counter + 3;
END LOOP;

```

В отличие от простого цикла, рассмотренного ранее, перед выполнением тела `WHILE` проверяется условие (`v_Counter <= 15`). Это условие задано в начале цикла. Если оно истинно, тело цикла выполняется. В противном случае команда, содержащиеся в цикле, не будут выполнены и управление будет передано выражению, следующему за `END LOOP`.

Цикл `FOR` позволяет повторить фрагмент кода заранее определенное число раз. Его иногда называют циклом со счетчиком. Данный цикл удобно применять тогда, когда вам известно количество итераций, которое должно быть выполнено. Оно вычисляется на основе нижней и верхней границы переменной цикла. Цикл `FOR` записывается следующим образом:

```

FOR
  <переменная_цикла>
  IN [REVERSE]
  <нижняя_граница>
  ..
  <верхняя_граница>
  LOOP
    Выражения
END LOOP;

```

Переменная цикла инициализируется значением нижней границы (или, если указано ключевое слово `REVERSE`, ей присваивается значение верхней границы). Затем при каждой итерации переменная цикла инкрементируется (или декрементируется, если указано `REVERSE`), т.е. ее значение изменяется на единицу. Цикл завершается, когда значение переменной превышает верхнюю границу (или, при наличии ключевого слова `REVERSE`, становится меньше нижней границы). Таким образом, если `REVERSE` не указано, переменная цикла инициализируется значением нижней границы, а верхняя граница определяет максимальное значение, которое данная переменная может иметь. Если указано ключевое слово `REVERSE`, все происходит наоборот: верхняя граница используется в качестве начального значения, а нижняя задает минимальное из возможных значений переменной. Ниже приведен пример цикла `FOR`, тело которого выполняется 15 раз. В отличие от простого цикла и цикла `WHILE` объявлять переменную цикла `FOR` нет необходимости.

```
FOR v_Counter IN 1..15 LOOP
    DBMS_OUTPUT.PUT_LINE(v_Counter);
END LOOP;
```

В предыдущем примере переменная цикла `v_Counter` инициализирована значением 1. Это происходит при первом выполнении выражения `FOR`. Затем выполняется выражение `DBMS_OUTPUT`, после чего управление передается в начало цикла, значение `v_Counter` увеличивается на 1 и сравнивается с верхней границей, т.е. с 15. Если значение переменной не превышает 15, цикл продолжается. Данный процесс повторяется до тех пор, пока значение переменной цикла не достигнет 15. После этого в результате инкрементирования значение переменной цикла становится равным 16. Оно сравнивается с верхней границей, т.е. с числом 15. Поскольку значение переменной цикла превышает 15, управление передается выражению, следующему за `END LOOP`.

## Обработка строк с применением явных курсоров и циклов

Возможно, вы удивились, почему мы стали обсуждать циклы непосредственно перед рассмотрением явных курсоров. Явные курсоры и циклы неотделимы друг от друга. Поскольку по определению явные курсоры представляют собой результат выполнения выражений `SELECT`, возвращающих более одной строки, циклы необходимы для поочередной обработки строк.

### Использование явных курсоров

Если в блоке PL/SQL появляется команда `SELECT`, извлекающая из таблицы больше одной строки, для обработки результатов ее выполнения приходится использовать курсор. Курсор можно рассматривать как набор строк, возвращенный системой Oracle и хранящийся в памяти, причем в каждый момент времени обрабатывать можно только одну строку. Запросом к системе Oracle, указывающим на необходимость поместить строки в курсор, является выражение `SELECT`. После того как строки оказались в курсоре, вы можете извлекать и обрабатывать любую из них. Использование курсора обычно предполагает следующие действия.

1. Объявление переменных для хранения значений, возвращаемых выражением `SELECT`.
2. Определение курсора путем указания выражения `SELECT`, предназначенного для извлечения строк.
3. Открытие курсора. Это действие служит указанием Oracle извлечь строки из базы данных и поместить их в курсор (т.е. в память).

4. Загрузка строки из курсора и обработка ее в блоке PL/SQL. Действия, соответствующие данному этапу, выполняются до тех пор, пока все строки не будут обработаны.
5. Закрытие курсора. Оно приводит к освобождению занимаемой памяти. В то же время строки, извлеченные из таблицы, становятся недоступными.

Наилучший способ понять особенности работы с курсором — создать блок PL/SQL, в котором использовался бы курсор, и написать код для реализации каждого из пяти упомянутых выше этапов. В данном случае модифицировать уже существующие блоки PL/SQL не имеет смысла, лучше создать новый блок с нуля. Он будет похож на блок, который мы сохранили под именем AnonBlockTwo.sql, отличие лишь в том, что он будет возвращать среднюю заработную плату для каждой группы. При этом необходим курсор, так как выражение SELECT возвращает пять строк.

## **Объявление переменных для хранения значений столбцов и курсора**

В первую очередь надо объявить переменные для хранения значений столбцов, возвращаемых командой SELECT. Кроме того, вам потребуется объявить переменную для определения курсора. Курсор определяется с помощью выражения SELECT, которое извлекает строки. Прежде всего откройте Блокнот или другой текстовый редактор, с которым вы предпочитаете работать. В нем вы создадите блок PL/SQL, использующий курсор. В этом разделе описана последовательность действий, которые надо выполнить при работе над блоком PL/SQL. Курсор определяется посредством следующего выражения:

```
CURSOR <имя_курсора>
[ <список_параметров> ]
[RETURN <тип_возвращаемого_значения> ]
IS <запрос>
[FOR UPDATE [OF (
<список_столбцов>
) ] [NOWAIT]];
```

Здесь в качестве имени курсора может быть использован любой допустимый идентификатор Oracle. Список параметров указывать не обязательно. Это может быть любой список аргументов, необходимых для работы. Необязательное выражение RETURN определяет тип возвращаемого значения, если такое предусмотрено. Запросом является любое выражение SELECT. Посредством запроса извлекается один или несколько столбцов из одной или нескольких таблиц базы данных. Необязательное выражение FOR UPDATE блокирует строки базы данных на то время, когда курсор остается открытым. Опция NOWAIT вызывает завершение программы сразу же после открытия курсора в том случае, если выражение SELECT не может получить исключительный доступ к строкам таблицы.

Для того чтобы объявить переменные для хранения значений столбцов и курсора, выполните следующие действия.

1. Откройте Блокнот и создайте пустой файл, чтобы зафиксировать его имя. Сделать это можно следующим образом. Выберите пункт меню File и Save As. Щелкните на раскрывающемся списке Save as type, выберите пункт All files и щелкните в поле File name. Введите <имя>AnonCursorOne.sql (<имя> замените вашим именем) и щелкните на кнопке Save.
2. Для создания локальных переменных, в которых будут храниться значения, извлеченные из столбцов, введите следующий код:

```
DECLARE  
    v_AvgSal AgentsHR.BaseSalary%TYPE;  
    v_TeamNo AgentsHR.Team%TYPE;  
    v_AvgSalChar VARCHAR2(15);
```

3. Введите приведенные ниже строки для того, чтобы определить курсор и пометить начало раздела исполняемого кода в теле блока PL/SQL. Обратите внимание на префикс cv\_ для переменной курсора (сокращенно он означает “cursor variable”). Курсор определяется посредством выражения SELECT.

```
CURSOR cv_TeamCursor IS  
    SELECT Team, AVG(BaseSalary)  
    FROM AgentsHR  
    GROUP BY Team  
    ORDER BY Team;  
BEGIN
```

Не закрывайте Блокнот, так как в дальнейшем вам еще придется работать с ним.

## Открытие курсора

После того как курсор будет определен в блоке PL/SQL, его надо открыть. Сделать это можно посредством выражения OPEN. Оно подготавливает курсор, загружает команду SELECT и извлекает строки из базы данных. Строки помещаются в память. Активная строка курсора — это единственная строка, доступная программе PL/SQL. При выполнении команды OPEN активизируется первая из извлеченных строк.

4. Продолжайте ввод кода. Приведенная ниже строка предназначена для открытия курсора.

```
OPEN cv_TeamCursor;
```

5. Введите следующую строку, чтобы отобразить заголовки:

```
DBMS_OUTPUT.PUT_LINE('Team' || ' Average Salary');
```

Оставьте Блокнот открытым.

## Организация цикла и загрузка строк из курсора

После того как вы откроете курсор, содержащиеся в нем строки данных становятся доступными для обработки программой PL/SQL.

Выражение `FETCH` извлекает строки из курсора, или контекстной области, в переменные, которые, в свою очередь, могут быть обработаны программой. Команда `FETCH` оперирует только с одной строкой. Она записывается следующим образом:

```
FETCH <имя_курсора>
INTO <имена_переменных>;
```

После ключевого слова `FETCH` указывается имя курсора, а после `INTO` задается список имен определенных ранее переменных, разделенных запятыми. Число переменных в списке должно совпадать с числом столбцов в курсоре. Типы переменных должны соответствовать типам столбцов.

Далее вам следует предпринять действия по написанию кода, который извлекал бы по очереди каждую строку и отображал ее. Код будет представлять собой простой цикл, который завершится тогда, когда все строки будут исчерпаны, т.е. после обработки последней строки. Поскольку курсор может содержать произвольное количество строк, необходимо найти способ определить момент окончания цикла. Другими словами, нам надо знать, остались ли строки в курсоре. Логическая переменная `%NOTFOUND` принимает значение `true` после того, как команда `FETCH` достигает последней доступной строки. Имя переменной формируется на основе имени курсора, к которому добавляется последовательность символов `%NOTFOUND`. В данном случае, чтобы определить, надо ли завершать цикл, вам следует анализировать переменную `cv_TeamCursor%NOTFOUND`.

6. Введите приведенный ниже фрагмент кода, чтобы организовать загрузку строк, форматирование значений и отображение на консоли. (Чтобы выровнять значения и расположить их под соответствующими заголовками столбцов, между `v_TeamNo` и `v_AvgSalChar` включено семь пробелов. В общем случае количество пробелов может быть произвольным.)

```
LOOP
    FETCH cv_TeamCursor
    INTO v_TeamNo, v_AvgSal;
    EXIT WHEN cv_TeamCursor%NOTFOUND;
    v_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');
    DBMS_OUTPUT.PUT_LINE(v_TeamNo || ' ' || v_AvgSalChar);
END LOOP;
```

Оставьте Блокнот открытым.

## Закрытие курсора и обработка исключений

Последнее, что надо сделать, — это закрыть курсор. Явные курсоры всегда надо закрывать. В противном случае занимаемую им область памяти нельзя будет использовать для других целей. Выражение, используемое для закрытия курсора, имеет следующий вид:

```
CLOSE <имя_курсора>;
```

После ключевого слова CLOSE указывается имя открытого курсора. Если вы попытаетесь закрыть курсор, который не был открыт, Oracle отобразит следующее сообщение об ошибке (см. табл. 7.2):

```
ORA-01001: invalid cursor
```

Для того чтобы завершить создание кода, выполните следующие действия.

7. Введите команду, с помощью которой открытый курсор cv\_TeamCursor будет закрыт.

```
CLOSE cv_TeamCursor;
```

8. Введите представленный ниже код. В нем реализуется обработка исключений и помечается конец блока PL/SQL.

```
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

9. Работа над блоком завершена. Пришло время сохранить его. В редакторе Блокнот нажмите комбинацию клавиш <Ctrl+S>, чтобы записать текст блока в файл. На рис. 7.8 показан неименованный блок PL/SQL, отображаемый в окне Блокнот. Именно так должен выглядеть созданный вами код.

10. Закройте Блокнот.

Настало время проверить неименованный блок PL/SQL. Запустите SQL\*Plus, зарегистрируйтесь в системе Oracle и следуйте приведенным ниже инструкциям. Если возникнет ошибка, откройте Блокнот, исправьте ее, включите код в SQL\*Plus и нажмите клавишу <Enter>. Если вы вносили исправления в состав блока, сохраните его.

1. Ведите в SQL\*Plus приведенные ниже команды. Таким образом вы очистите экран и создадите условия для того, чтобы выходные данные PL/SQL отображались на консоли.

```
CLEAR SCREEN
SET SERVEROUTPUT ON
```

```

AnonCursorOne.sql - Notepad
File Edit Format View Help

DECLARE
    v_AvgSal AgentsHR.BaseSalary%TYPE;
    v_TeamNo AgentsHR.Team%TYPE;
    v_AvgSalChar VARCHAR2(15);
    CURSOR cv_TeamCursor IS
        SELECT Team, AVG(BaseSalary)
        FROM AgentsHR
        GROUP BY Team
        ORDER BY Team;
    BEGIN
        OPEN cv_TeamCursor;
        DBMS_OUTPUT.PUT_LINE('Team' || ' Average Salary');
        LOOP
            FETCH cv_TeamCursor
            INTO v_TeamNo, v_AvgSal;
            EXIT WHEN cv_TeamCursor%NOTFOUND;
            v_AvgSalChar := TO_CHAR(v_AvgSal, '$99,999.99');
            DBMS_OUTPUT.PUT_LINE(v_TeamNo || ' ' || v_AvgSalChar);
        END LOOP;
        CLOSE cv_TeamCursor;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
/

```

**Рис. 7.8.** Полный текст блока PL/SQL, использующего явный курсор

- Для проверки неименованного блока, который вы сохранили под именем AnonCursorOne.sql, введите представленную ниже команду. Представьте вместо выражения <путь> реальный путь, а вместо <имя> — свое имя.

```
START <путь>\<имя>AnonCursorOne
```

Если вы правильно ввели код, то должны получить такие же результаты, как показаны на рис. 7.9. Другими словами, вы увидите те же номера групп, значения средней заработной платы и сообщение PL/SQL procedure successfully completed.

- Ведите exit и нажмите клавишу <Enter>, чтобы завершить сеанс работы с Oracle и закрыть SQL\*Plus.

## Использование цикла FOR для работы с явным курсором

Если вам надо обработать в цикле строки из явного курсора, лучше всего сделать это, используя специальный цикл FOR, который называется *циклом FORкурсора*. При использовании цикла FOR курсора вам нет необходимости открывать или закрывать курсор, а также не надо выполнять команду FETCH. Эти действия будут произведены

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SET SERVEROUTPUT ON
SQL> START C:\AnonymousCursorOne
Team      Average Salary
10        $3,697.00
11        $3,057.00
12        $3,174.00
13        $3,010.00
14        $3,328.00
PL/SQL procedure successfully completed.
SQL>

```

**Рис. 7.9.** Проверка работы неименованного блока PL/SQL, содержащего явный курсор

автоматически. Таким образом, блок PL/SQL становится проще. Цикл FOR курсора записывается следующим образом:

```

FOR
  <имя_записи>
  IN
  <имя_курсора>
LOOP
  <Выражение 1>;
  <Выражение 2>; ...
END LOOP;

```

Вместо того чтобы извлекать столбцы таблицы и присваивать их переменным PL/SQL, цикл FOR курсора помещает строку в запись. *Запись* — это составная структура, которая построена так же, как строка, извлеченная из явного курсора.

Для того чтобы продемонстрировать отличие от рассмотренного ранее подхода, создадим новый блок PL/SQL, содержащий цикл FOR курсора и явный курсор. В блоке из таблицы будет извлекаться и отображаться несколько строк. В качестве подготовительных действий запустим SQL\*Plus и Блокнот. Введите текст к редакторе Блокнот и скопируйте его в SQL\*Plus, чтобы выполнить код. Для удобства желательно поместить в блок PL/SQL выражения CLEAR SCREEN и SET SERVEROUTPUT ON.

В данном случае мы будем извлекать имена и фамилии всех продавцов, кто живет в городе Лолета, штат Калифорния. В дальнейшем мы сделаем данную процедуру более универсальной, обеспечив поддержку любого города, нужного пользователю. Рассмотрим выражение FOR и особенности его взаимодействия с курсором.

Для того чтобы написать код, в котором используется цикл FOR совместно с явным курсором, выполните следующие действия.

1. Введите приведенные ниже строки. При их выполнении будет очищен экран и установлена требуемая переменная среды SQL\*Plus.

```

CLEAR SCREEN
SET SERVEROUTPUT ON

```

2. Создайте раздел деклараций следующим образом.

**Совет.** При вводе исходного текста нет необходимости в точности воспроизводить отступы и включать такое же количество пробелов, как показано здесь. Они использованы лишь для того, чтобы улучшить восприятие кода.

```
DECLARE
    CURSOR cv_Sellers IS
        SELECT a.FirstName, a.LastName, c.AskingPrice
        FROM Customers a
            INNER JOIN Properties b ON a.CustomerID = b.OwnerID
            INNER JOIN Listings c USING (PropertyID)
            INNER JOIN CustAgentList d USING (ListingID)
        WHERE UPPER(d.ContactReason) = 'SELL'
            AND a.City = 'Loleta'
        ORDER BY LastName, FirstName;
```

3. Продолжая работу над блоком, введите раздел исполняемого кода, в котором реализован цикл FOR курсора, используемый для обработки извлекаемых строк, и раздел обработки исключений, в котором предусмотрены два исключения из многих возможных.

```
BEGIN
    DBMS_OUTPUT.PUT_LINE(RPAD('Seller''s Name', 24, ' ') |
        | ' Asking Price');
    FOR v_Counter IN cv_Sellers LOOP
        DBMS_OUTPUT.PUT_LINE(
            RPAD(v_Counter.FirstName| || ' '| |
            v_Counter.LastName, 30, ' ')| |
            v_Counter.AskingPrice
        );
    END LOOP;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No sellers in that city');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error of unknown type occurred');
END;
/
```

**Совет** Функция RPAD не является необходимой. Она лишь улучшает внешний вид выводимого текста. Вы, наверное, вспомнили, что RPAD добавляет пробел справа от первого параметра, чтобы обеспечить его требуемую ширину.

4. Теперь можно начинать тестирование цикла FOR курсора. Скопируйте код, созданный вами при выполнении пп. 1–3, в SQL\*Plus и нажмите клавишу <Enter>, чтобы выполнить его. Ошибка может возникнуть только в том случае, если вы неправильно ввели код. Скорректируйте текст в редакторе Блокнот, снова скопируйте блок PL/SQL в SQL\*Plus и повторно выполните его. Результаты показаны на рис. 7.10.

```

SQL> SET SERVEROUTPUT ON
SQL> DECLARE
 2   CURSOR cv_Sellers IS
 3     SELECT a.FirstName, a.LastName, c.AskingPrice
 4     FROM Customers a
 5     INNER JOIN Properties b ON a.CustomerID = b.OwnerID
 6     INNER JOIN Listings c USING (PropertyID)
 7     INNER JOIN Custagentlist d USING (ListingID)
 8     WHERE UPPER(d.ContactReason) = 'SELL'
 9     AND a.City = 'Loleta'
10    ORDER BY LastName, FirstName;
11  BEGIN
12    DBMS_OUTPUT.PUT_LINE('Seller's Name' ,24,' '|| Asking Price');
13
14    FOR v_Counter IN cv_Sellers LOOP
15      DBMS_OUTPUT.PUT_LINE(
16        RPAD(v_Counter.FirstName||' '||v_Counter.LastName,30,'')||
17        v_Counter.AskingPrice
18      );
19    END LOOP;
20
21  EXCEPTION
22    WHEN NO_DATA_FOUND THEN
23      DBMS_OUTPUT.PUT_LINE('No sellers in that city');
24    WHEN OTHERS THEN
25      DBMS_OUTPUT.PUT_LINE('Error of unknown type occurred');
26  END;
27 /
Seller's Name      Asking Price
Julie Elliott      282500
Wanda Faust       114000
Frances Gordon    111950
Ty Ha              145000
Jennifer Humphreys 104000
Matthew McGrath    162500
Anthony Stabile    154000

PL/SQL procedure successfully completed.

SQL>

```

**Рис. 7.10.** Использование цикла FOR курсора

5. Сохраните код в какой-нибудь из папок, выбрав подходящее имя файла. Закройте Блокнот.
6. Введите `exit` и нажмите клавишу `<Enter>`, чтобы завершить сеанс работы с Oracle и закрыть SQL\*Plus.

## Выражение IF

По умолчанию команды в блоке PL/SQL выполняются последовательно одна за другой. При наличии цикла содержащиеся в нем выражения повторяются до тех пор, пока выполняется некоторое условие (или до тех пор, пока оно не перестанет выполнятся). Помимо циклов на последовательность выполнения команд влияет также выражение `IF`, которое называют *оператором выбора*. Данное выражение позволяет организовать выполнение альтернативных наборов команд, в зависимости от того, выполняется ли некоторое условие. Конструкция `CASE`, напоминающая `IF`, уже рассматривалась в главе 4. Подобно конструкции `IF`, конструкция `CASE` позволяет организовать выполнение альтернативных наборов команд. Выражение `IF` записывается следующим образом:

```

IF <условие>
THEN   <выражения>;
[ELSIF <условие>
THEN   <выражения> ;]
[ELSE   <выражения> ;]
END IF;

```

Здесь условие — это выражение, в результате вычисления которого получается значение TRUE или FALSE. В простейшем случае в составе конструкции IF имеется только выражение THEN. Если в результате вычисления выражения, которое было задано в качестве условия, получается значение FALSE или NULL, остальная часть конструкции IF игнорируется и никакие действия не предпринимаются. Если условие равно FALSE, Oracle вычисляет первое из указанных выражений ELSIF (заметьте, что в середине данного слова отсутствует буква “E”). Команды в составе необязательного выражения ELSIF представляют собой ветвь выполнения, альтернативную набору команд, следующему за выражением THEN. Если условие, указанное в составе ELSIF, истинно, команды, которые следуют за соответствующим ключевым словом THEN, будут выполнены. В противном случае будет обработано следующее выражение ELSIF или ELSE. Признаком окончания конструкции IF является выражение END IF (два слова), за которым следует точка с запятой. Кроме того, точкой с запятой завершаются все команды, следующие за ключевыми словами THEN, ELSIF и ELSE.

## Реализация выбора посредством выражения IF

Примеры, рассматриваемые в данном разделе, иллюстрируют выполнение альтернативных ветвей кода в зависимости от значения условия. Блок PL/SQL, который мы создадим, определяет, имеется ли в таблице Agents информация об агенте Franklin. Если такая информация отсутствует, она добавляется в таблицу. Выражения DML, такие как UPDATE или INSERT, часто встречаются в составе блоков PL/SQL, как анонимных, так и именованных.

```

DECLARE
    v_Tally INTEGER := 0;
BEGIN
    SELECT COUNT(*) INTO v_Tally FROM Agents
        WHERE LastName = 'Franklin';
    IF v_Tally = 0 THEN
        INSERT INTO Agents (AgentID, FirstName, LastName,
        LicenseStatusID)
        VALUES (23456, 'Ben', 'Franklin', 1001);
    ELSE
        DBMS_OUTPUT.PUT_LINE('Agent already in table: not added');
    END IF;
END;

```

В данном примере выражение SELECT определяет, сколько раз в таблице Agents встречается фамилия Franklin. Результат помещается в переменную v\_Tally. Опе-

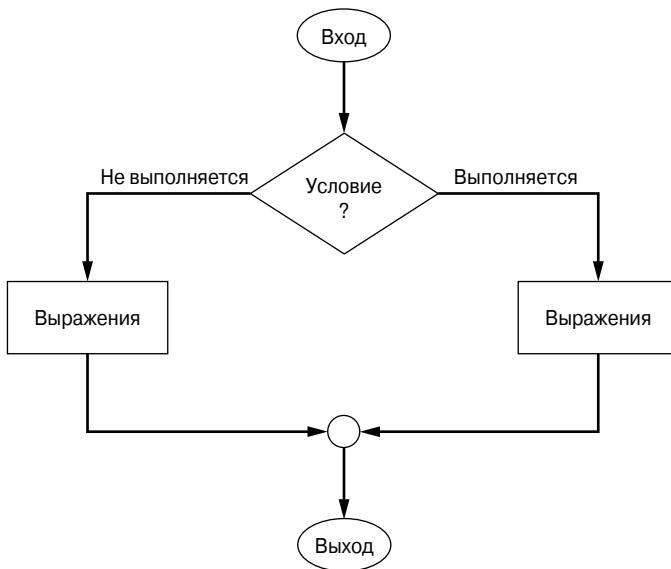


Рис. 7.11. Порядок выполнения выражения IF

ратор `IF` проверяет результат, т.е. определяет, были ли извлечены значения. Если количество значений равно нулю, то выражение принимает значение `TRUE`. В противном случае (для значений больше нуля) выражение равно `FALSE`. Если выражение, указанное в качестве условия, равно `TRUE`, Oracle вычисляет выражение `INSERT INTO`. Если условие равно `FALSE`, выполняется команда, соответствующая ветви `ELSE`, в результате чего отображается сообщение.

Если нет необходимости информировать пользователя о том, найдено ли имя в таблице, выражение `ELSE` можно не указывать. Альтернативный вариант конструкции `IF`, которая включает строку, если она отсутствует в таблице, выглядит так:

```

IF v_Tally = 0 THEN
  INSERT INTO Agents (AgentID, FirstName, LastName,
    LicenseStatusID)
  VALUES (23456, 'Ben', 'Franklin', 1001);
END IF;
  
```

Фрагменты кода, следующие за `THEN` и за `ELSE`, взаимно исключают друг друга. Выполняется лишь один из них. На рис. 7.11 приведено условное графическое представление выражения `IF` с необязательным элементом `ELSE`.

## Выполнение DML-выражений в блоке PL/SQL

Помимо выражений `SELECT` в блоках PL/SQL могут присутствовать команды для обновления данных. Блок PL/SQL может обеспечивать расширенную поддержку операций включения, модификации и удаления записей, однако такой блок должен быть

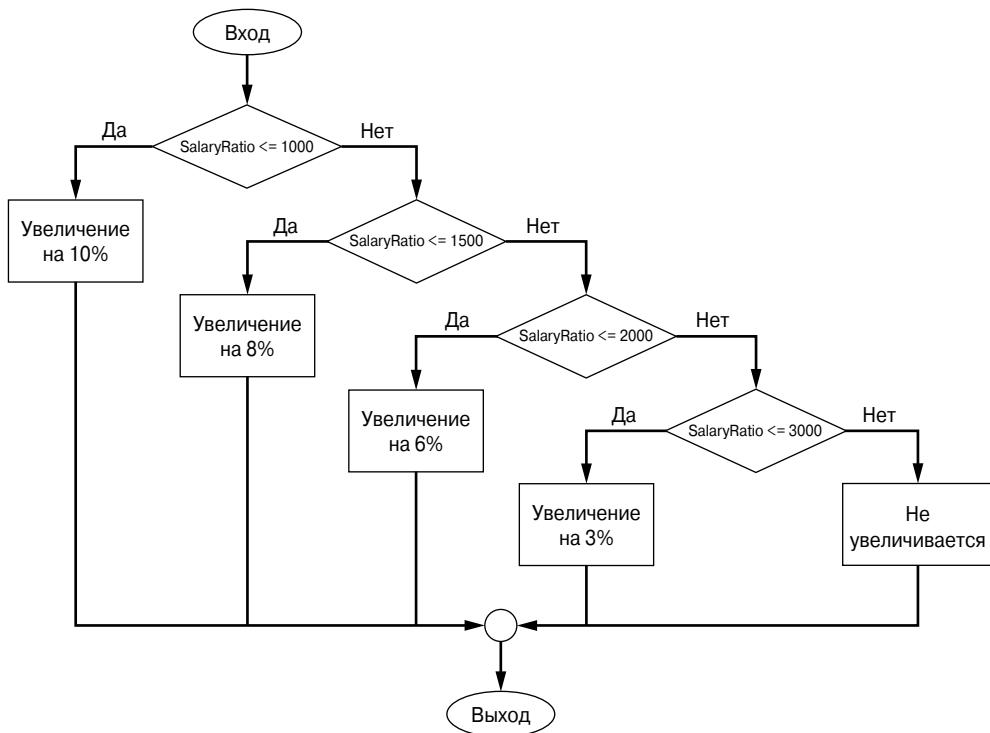
**ТАБЛИЦА 7.3.** Правила, по которым вычисляется приращение базовой зарплаты

<i>Часть зарплаты, приходящаяся на члена семьи (доллары)</i>	<i>Приращение базовой зарплаты (проценты)</i>
0–1,000	10
1,001–1,500	8
1,501–2,000	6
2,001–3,000	3
Более 3,000	Нет приращения

правильно сформирован. Вместо того чтобы непосредственно включать выражения `INSERT`, `UPDATE` или `DELETE`, надо создать код, который предоставит дополнительную информацию о том, должны ли строки быть модифицированы и как они будут изменены. Предположим, например, что управляющий персоналом Redwood Realty заметил, что базовые оклады некоторых агентов, установленные для того, чтобы поддержать сотрудников в те периоды, когда спрос на товары падает, требуют пересмотра. По его мнению, базовую зарплату надо увеличить, выбрав приращение на основании заработной платы агента, с учетом числа членов его семьи. Пусть, например, базовая зарплата агента составляет 3000 долларов, а его семья состоит из пяти человек. Тогда на каждого члена семьи приходится сумма, равная 600 долларов. По мнению управляющего персоналом, коэффициент приращения должен в некоторой степени компенсировать расходы агентов, обремененных большой семьей. Был предложен принцип повышения заработной платы, описанный в табл. 7.3.

Использование шкалы возрастающих коэффициентов — типичная ситуация, в которой оправдано применение выражения `IF`. В конструкции `IF` можно реализовать проверку различных условий, призванную определить, какая из альтернативных последовательностей действий должна быть выполнена. В данном случае возможно пять альтернативных решений, четыре из которых предполагают применение выражений `UPDATE`, действующих на таблицу `AgentsHR`. Данный пример позволяет лучше понять, как работает выражение `IF`. На рис. 7.12 показана диаграмма (блок-схема), которую необходимо реализовать в блоке PL/SQL, содержащем команды `IF` и `UPDATE`.

Начало выполнения программы обозначено на диаграмме овалом, в котором содержится надпись `ENTER`. Первым проверяется условие, не превышает ли часть заработной платы агента, приходящаяся на члена его семьи, значение 1000. Если выражение, используемое для проверки данного условия, равно `TRUE`, то значение `BaseSalary` увеличивается на 10% и выполнение программы завершается. В противном случае выполняется следующая проверка — является ли часть зарплаты, приходящаяся на члена семьи, меньшей или равной 1500. Если данное условие выполняется, значение `BaseSalary` увеличивается на 8% и выполнение блока прекращается. Подобным образом проверяются все условия. Если ни одно из четырех условий не выполняется, это означает, что заработная плата не должна быть увеличена.



**Рис. 7.12.** Диаграмма, представляющая логику бизнес-приложения

Заметьте, что порядок проверки условий очень важен. Проверка типа “меньше или равно” выполняется по возрастанию — от меньших значений к большим. Если бы проверялось условие “больше”, то проверку пришлось бы производить в обратной последовательности, т.е. от больших значений к меньшим. Каждый из этих подходов корректен, нельзя лишь проверять условия в произвольном порядке, так как при этом неизбежны ошибки.

Далее надо включить в состав неименованного блока PL/SQL выражение UPDATE. Правильный выбор команды UPDATE для выполнения — цель выражения IF. Выполните подготовительные работы: запустите SQL\*Plus, зарегистрируйтесь в системе Oracle и загрузите Блокнот или другой текстовый редактор. Сохраните пустой текстовый файл в выбранной вами папке под выбранным именем.

Для написания блока PL/SQL, который бы обновлял значения столбца таблицы в зависимости от выполнения условий, выполните следующие действия.

1. В открытом редакторе Блокнот введите следующие команды:

```
CLEAR SCREEN
SET SERVEROUTPUT ON
```

2. Создайте раздел деклараций, поместив в него приведенный ниже фрагмент кода.

```

DECLARE
    v_Amount NUMBER(2,2);
    CURSOR cv_Agent IS
        SELECT AgentID, BaseSalary/Dependents AS SalaryRatio
        FROM AgentsHR;

```

3. В разделе исполняемого кода введите следующее содержимое:

```

BEGIN
    FOR v_Counter IN cv_Agent LOOP
        IF v_Counter.SalaryRatio <= 1000 THEN
            v_Amount := 0.10;
        ELSIF v_Counter.SalaryRatio <= 1500 THEN
            v_Amount := 0.08;
        ELSIF v_Counter.SalaryRatio <= 2000 THEN
            v_Amount := 0.06;
        ELSIF v_Counter.SalaryRatio <= 3000 THEN
            v_Amount := 0.03;
        ELSE
            v_Amount := 0;
        END IF;
        DBMS_OUTPUT.PUT_LINE(
            (v_Counter.AgentID||' updated by '||
             TO_CHAR(v_Amount*100,'99')||'%'
            );
        UPDATE AgentsHR
            SET BaseSalary = BaseSalary * (1+v_Amount)
            WHERE AgentID = v_Counter.AgentID;
    END LOOP;
    COMMIT;

```

4. Продолжая работу с редактором Блокнот, создайте раздел поддержки исключений. В конце данного фрагмента кода задайте косую черту; она является признаком того, что код должен быть скопирован и выполнен. Это произойдет тогда, когда вы скопируете блок в SQL\*Plus и, установив текущую позицию после косой черты, нажмете клавишу <Enter>.

```

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error; transactions rolled back');
        ROLLBACK;
    END;
/

```

5. Запишите код, который вы только что создали, в файл, а затем скопируйте его в SQL\*Plus. Закройте Блокнот. На рис. 7.13 показано, как должно выглядеть содержимое SQL\*Plus перед выполнением.

6. Находясь в SQL\*Plus, нажмите клавишу <Enter>, чтобы выполнить блок PL/SQL. На рис. 7.14 видна часть готового кода и сообщения, в которых присутствуют значения AgentID, а также сведения о приращении заработной платы. Это свидетельствует об успешном выполнении блока.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
 2   v_Amount NUMBER(2,2);
 3   CURSOR cv_Agent IS
 4     SELECT AgentID, BaseSalary/Dependents AS SalaryRatio
 5     FROM AgentsHR;
 6   BEGIN
 7     FOR v_Counter IN cv_Agent LOOP
 8       IF v_Counter.SalaryRatio <= 1000 THEN
 9         v_Amount := 0.10;
10       ELSIF v_Counter.SalaryRatio <= 1500 THEN
11         v_Amount := 0.08;
12       ELSIF v_Counter.SalaryRatio <= 2000 THEN
13         v_Amount := 0.06;
14       ELSIF v_Counter.SalaryRatio <= 3000 THEN
15         v_Amount := 0.03;
16       ELSE
17         v_Amount := 0;
18       END IF;
19       DBMS_OUTPUT.PUT_LINE(
20         (v_Counter.AgentID||' updated by '||
21          TO_CHAR(v_Amount*100,'99'))||'%'
22       );
23       UPDATE AgentsHR
24         SET BaseSalary = BaseSalary * (1+v_Amount)
25       WHERE AgentID = v_Counter.AgentID;
26     END LOOP;
27     COMMIT;
28   EXCEPTION
29   WHEN OTHERS THEN
30     DBMS_OUTPUT.PUT_LINE('Error; transactions rolled back');
31     ROLLBACK;
32   END;
33 /

```

**Рис. 7.13.** Включение DML-выражений в блок PL/SQL

7. Если вы собираетесь прекратить работу, закройте SQL\*Plus. Если же вы хотите продолжить изучение данного раздела, оставьте его открытым.

В бизнес-приложениях вам вряд ли понадобятся вызовы DBMS\_OUTPUT в цикле FOR. В данном примере они используются лишь для того, чтобы вы могли убедиться, что Oracle выполняет корректные действия.

Используя цикл FOR курсора и условные выражения, мы реализовали увеличение содержимого столбца `BaseSalary` для каждого агента на требуемое число процентов. Число процентов, определяющее увеличение заработной платы, мы записали в переменную `v_Amount`. За пределами выражения IF, но внутри цикла FOR выражение UPDATE использует содержимое `v_Amount` для коррекции сведений о заработной плате выбранного агента. Выражение COMMIT, находящееся за пределами цикла FOR, делает все изменения постоянными. Если при выполнении цикла FOR возникнет ошибка, управление будет передано разделу обработки исключений и содержащееся в нем выражение ROLLBACK отменит все внесенные изменения.

## Именованные блоки

*Именованный блок* – это блок PL/SQL, имеющий имя. Кроме того, структура именованных блоков несколько отличается от анонимных. В них есть четвертый дополнительный раздел, который называется *разделом заголовка*. Этот раздел сообщает

```

* Oracle SQL*Plus
File Edit Search Options Help
23   UPDATE AgentsHR
24     SET BaseSalary = BaseSalary * (1+v_Amount)
25   WHERE AgentID = v_Counter.AgentID;
26 END LOOP;
27 COMMIT;
28 EXCEPTION
29   WHEN OTHERS THEN
30     DBMS_OUTPUT.PUT_LINE('Error; transactions rolled back');
31   ROLLBACK;
32 END;
33 /
10041 updated by 10%
10235 updated by 3%
10629 updated by 10%
10497 updated by 6%
10849 updated by 10%
10913 updated by 6%
11775 updated by 10%
12211 updated by 6%
12389 updated by 6%
12499 updated by 6%
12715 updated by 3%
12765 updated by 8%
12875 updated by 8%
12963 updated by 3%
13353 updated by 8%
13551 updated by 6%
13664 updated by 6%
13771 updated by 10%
14117 updated by 3%
14447 updated by 8%
14599 updated by 8%
14681 updated by 10%
14677 updated by 6%
14888 updated by 8%
15061 updated by 8%
15233 updated by 10%
15293 updated by 8%
15349 updated by 6%
15521 updated by 3%
PL/SQL procedure successfully completed.
SQL>

```

**Рис. 7.14.** Сообщения об изменениях заработной платы агентов

Oracle имя блока, а также информирует о том, является ли блок функцией или процедурой — именно такие типы именованных блоков существуют в Oracle. Функция — это именованный блок PL/SQL, который хранится на сервере Oracle и возвращает одно значение. Процедура — это именованный блок PL/SQL, который хранится на сервере Oracle и выполняет некоторые действия. Функцию или процедуру можно вызывать непосредственно из SQL\*Plus либо из другой функции или процедуры. При вызове функции ее имя указывается в SQL\*Plus либо в другом блоке PL/SQL. Имя функции и параметры можно поместить в выражение SQL. Процедура вызывается из блока PL/SQL либо посредством выполнения команды EXECUTE в SQL\*Plus. В выражении SQL процедуру указывать нельзя.

И функциям, и процедурам обычно передаются параметры — список значений, разделенных запятыми, помещенный в круглые скобки. Параметры передают информацию функции или процедуре и указываются после ее имени. Параметры подразделяются на входные (IN), выходные (OUT) и входные/выходные (IN OUT). Обозначение IN говорит о том, что значение, передаваемое функции или процедуре, не может быть изменено в процессе их выполнения. Параметр обозначается как OUT в том случае, если значение, вычисленное в теле функции или процедуры, может

быть передано посредством этого параметра вызывающему коду. Параметр IN OUT используется в том случае, если значение, переданное функции или процедуре, может быть изменено и с помощью данного параметра доставлено вызывающему коду. Если спецификатор параметра не указан, принимается тип параметра IN. Такое поведение выбрано исходя из соображений безопасности. Любые изменения значения, переданного функции или процедуре, будут автоматически отвергнуты при передаче управления вызывающему коду.

В процессе работы вы можете создавать, модифицировать или удалять функции и процедуры. Очевидно, что для того, чтобы именованный блок можно было использовать, его надо создать. Созданные функции и процедуры хранятся в базе данных. В отличие от файлов сценариев, расположение которых надо помнить, функции и процедуры всегда доступны своим владельцам. Любой пользователь (схема) может выполнять функцию или процедуру, если на это получено разрешение. Рассмотрим, как создавать, модифицировать, вызывать и удалять функции.

---

## Создание, использование и удаление функций

Функция возвращает значение выражению, из которого она была вызвана. В качестве примеров функций, которые может создать администратор базы данных или программист, приведем функцию для извлечения возраста агента из поля BirthDate или функцию, которая определяет стаж работы агента в компании Redwood Realty. Хотя действия, выполняемые обеими функциями, легко реализуются с помощью обычных SQL-выражений, часто используемые операции удобнее вызывать в виде функций. Таким образом, если вам необходимы некоторые данные, например, объекты для продажи в конкретном городе, получить их можно, включив имя соответствующей функции в выражение SELECT. Единожды созданную функцию можно вызывать всякий раз, когда в ней возникнет необходимость.

### Создание функций и сохранение их в базе

Чтобы функцию можно было использовать, ее надо создать. Готовую функцию система Oracle сохранит в базе данных и автоматически обеспечит доступ к ней для схемы, в рамках которой она была создана. Языковая конструкция, используемая для создания функций, имеет следующий вид:

```
CREATE [OR REPLACE] FUNCTION <имя_функции>
[ ( <имя_параметра> [IN|OUT|IN OUT]
  <тип_данных> [, . . . ] )
  RETURN
  <тип_данных> {IS|AS}
BEGIN
  <тело_функции>
[EXCEPTION
```

```
<выражения для поддержки исключений>;
END [ <имя функции> ];
```

Заметьте, что выражение OR REPLACE не является обязательным. Если оно присутствует, разработчик может либо создать новую функцию, либо заменить функцию с совпадающим именем; при этом ошибки не возникают. После имени функции следует необязательный список параметров, для которых указаны спецификаторы IN, OUT или IN OUT. Каждый параметр отделяется от следующего запятой. Функция возвращает значение, поэтому после ключевого слова RETURN указывается тип данных результата. Выражения SQL и PL/SQL, составляющие функцию, располагаются между ключевыми словами BEGIN и END. Необязательный раздел обработки исключений начинается с ключевого слова EXCEPTION и заканчивается ключевым словом END, которое обозначает также конец всей процедуры. Обычно в конце функции ставится косая черта (/); в результате после нажатия клавиши <Enter> система компилирует и сохраняет определение функции. Если в процессе компиляции возникнет ошибка, определение функции не будет сохранено в базе данных. В этом случае надо исправить ошибку, повторно скомпилировать код и поместить определение функции в базу.

Создадим простую функцию, определяющую возраст агента. Функции передается один входной параметр, определяемый как IN. В нем задается идентификационный номер агента. Функция возвращает возраст в виде целочисленного значения. В качестве подготовительных действий загрузите SQL\*Plus, зарегистрируйтесь в системе Oracle и запустите Блокнот. При работе над данным примером нам придется больше, чем прежде, пользоваться текстовым редактором, поскольку мы допустим при создании функции небольшую ошибку, а затем найдем и исправим ее. Когда код скомпилируется корректно, Oracle сохранит функцию в базе данных.

Для того чтобы создать, скомпилировать и сохранить функцию, выполните следующие действия.

1. В SQL\*Plus введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран.
2. Откройте файл в редакторе Блокнот и введите следующую команду:

```
SET SERVEROUTPUT ON
```

3. Задайте заголовок функции и определения переменных. Обратите внимание, что в последней строке не указана точка с запятой. Поступите так же при вводе кода примера.

```
CREATE OR REPLACE FUNCTION Age(InputAgentID IN NUMBER)
  RETURN INTEGER AS
  v_BirthDate Agents.Birthdate%TYPE;
  v_TodaysDate DATE;
  v_Age INTEGER    /* Пропущена точка с запятой */
```

4. Введите оставшуюся часть функции — ее тело и раздел обработки исключений. Не забудьте ввести косую черту в последней строке, но не нажимайте после клавиши <Enter>.

```

BEGIN
    SELECT BirthDate, SYSDATE
    INTO v_BirthDate, v_TodaysDate
    FROM Agents
    WHERE AgentID = InputAgentID;
    v_Age := TRUNC((v_TodaysDate-v_BirthDate)/365.25);
RETURN v_Age;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END Age;
/

```

5. Сохраните текст, введенный в редакторе Блокнот, в удобной для вас папке под именем <имя>Ch07AgeFunction.sql. Как обычно, замените <имя> вашим именем. (Несмотря на то, что функция будет сохранена в базе данных, желательно также записать ее в файл. Это нужно на случай, если вы впоследствии захотите изменить определение функции.)
6. Скопируйте код из редактора Блокнот в SQL\*Plus. Для компиляции кода нажмите клавишу <Enter>.

Система Oracle отобразит сообщение об ошибке, возникшей на этапе компиляции. Как же проверить процесс компиляции и узнать, что не в порядке? Поможет вам в этом команда SHOW ERRORS, которая информирует о проблемах, выявленных компилятором, и сообщает номер строки и позицию, в которой может быть причина ошибки.

7. Введите в SQL\*Plus приведенную ниже команду и нажмите клавишу <Enter>, чтобы выполнить ее.

```
SHOW ERRORS
```

Oracle отобразит информацию, которая поможет вам локализовать проблему (рис. 7.15).

8. Оставьте открытыми SQL\*Plus и Блокнот.

На рис. 7.15 видно, что Oracle не ожидает встретить ключевое слово BEGIN в той позиции, в которой оно находится. Вместо него должен быть указан разделитель. Обычно это означает, что ошибка допущена в предыдущем выражении. Действительно, в последней строке раздела деклараций не указана точка с запятой. Эта строка выглядит так:

```
V_Age INTEGER
```

Если при компиляции функции или процедуры вы получили приведенное ниже универсальное сообщение, выполните команду SHOW ERRORS, чтобы получить более подробную информацию о проблеме.

```
Warning: Function created with compilation errors.
```

The screenshot shows the Oracle SQL\*Plus interface. In the command window, the following PL/SQL code is being entered:

```

SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE FUNCTION Age(InputAgentID IN NUMBER)
  2   RETURN INTEGER AS
  3   v_BirthDate Agents.Birthdate%TYPE;
  4   v_TodaysDate DATE;
  5   v_Age INTEGER
  6 BEGIN
  7   SELECT BirthDate, SYSDATE
  8   INTO v_BirthDate, v_TodaysDate
  9   FROM Agents
 10  WHERE AgentID = InputAgentID;
 11  v_Age := TRUNC((v_TodaysDate-v_BirthDate)/365.25);
 12 RETURN v_Age;
 13 EXCEPTION
 14  WHEN OTHERS THEN
 15    DBMS_OUTPUT.PUT_LINE(SQLERRM);
 16 END Age;
 17 /

```

Below the code, a message indicates compilation errors:

Warning: Function created with compilation errors.

SQL> SHOW ERRORS

Errors for FUNCTION AGE:

LINE/COL ERROR

6/1 PLS-00103: Encountered the symbol "BEGIN" when expecting one of  
the following:  
:= . ( @ % ; not null range default character  
The symbol ";" was substituted for "BEGIN" to continue.

SQL>

**Рис. 7.15.** Информация об ошибке, выявленной в процессе компиляции

Отредактируйте функцию, добавив недостающую точку с запятой. Для того чтобы отредактировать функцию, выполните следующие действия.

- Средствами редактора Блокнот найдите и загрузите файл, содержащий текст созданной вами функции.
- Прейдите в конец строки, содержащей определение переменной `v_Age`, и введите точку с запятой.
- Сохраните модифицированный код функции в файле.
- Скопируйте модифицированный код в SQL\*Plus.
- Закройте Блокнот.
- В SQL\*Plus, находясь в конце строки, содержащей косую черту, нажмите клавишу `<Enter>`. Этим вы укажете Oracle на то, что необходимо скомпилировать функцию.
- Оставьте активизированным SQL\*Plus.

Система отобразит для вас сообщение `Function created`, указывающее на то, что на этот раз функция скомпилирована без ошибок. При отсутствии ошибок Oracle сохранит функцию в базе данных в схеме, соответствующей пользователю, который создал эту функцию. Теперь вы можете обращаться к функции, чтобы определить возраст агента.

## Вызов функции

Вновь созданная функция вызывается так же, как любая встроенная функция Oracle. Как вы помните, встроенные функции указываются в составе команды SELECT. Если вы не предполагаете извлекать значения полей базы данных, указывайте в выражении FROM таблицу DUAL. Протестируем созданную функцию. Вызвать ее можно двумя способами.

Для того чтобы вызвать функцию, выполните следующие действия.

1. Перейдите в SQL\*Plus и введите приведенные ниже команды. Этим вы увеличите значение переменной окружения PAGESIZE и очистите экран. После ввода каждой строки нажмите клавишу <Enter>.

```
SET PAGESIZE 50  
CLEAR SCREEN
```

2. В SQL\*Plus введите представленное ниже выражение. Оно позволит проверить работу функции Age. Очевидно, что результат, который вы увидите, будет зависеть от конкретных данных, хранящихся на вашем компьютере.

```
SELECT Age(10041)  
FROM DUAL;
```

Oracle возвращает значение, зависящее от системного таймера (и определяемой им текущей даты).

3. Для того чтобы вызвать функцию в сочетании с информацией из таблицы Agents, введите следующие строки кода:

```
SELECT AgentID, LastName, Age(AgentID) AS "Agent's Age"  
FROM Agents;
```

Oracle отобразит идентификационные номера, фамилии и возраст 29 агентов (рис. 7.16). Очевидно, что возраст зависит от показаний вашего системного таймера. В нашем случае входным параметром функции Age является идентификационный номер агента.

4. Чтобы проверить надежность функции Age, введите следующее:

```
CLEAR SCREEN  
SELECT Age(12345)  
FROM DUAL;
```

Oracle отобразит сообщение об ошибке, а обработчик исключений выведет отдельное сообщение: ORA-01403: no data found (рис. 7.17).

5. Оставьте SQL\*Plus активизированным.

Почему возникла ошибка? Функции Age был передан параметр, равный 12345. Поскольку агента с таким идентификационным номером нет, Oracle не возвращает ни одной строки. Это порождает проблему. Существует ли способ подавить генерацию

The screenshot shows the Oracle SQL\*Plus interface. The command line displays the following SQL code:

```

SQL> SELECT Age(10041)
  2  FROM DUAL;

AGE(10041)
-----
      35

SQL> SELECT AgentID, LastName, Age(AgentID) AS "Agent's Age"
  2  FROM Agents;

```

Below the command line, the results are displayed in a table:

AGENTID	LASTNAME	Agent's Age
10041	Marcoux	35
10235	Carling	38
10429	Dahlen	36
10497	Rove	52
10849	Sheibani	23
10913	Voss	35
11775	Romero	43
12211	Dann	46
12301	Silverburg	47
12499	Robinson	36
12715	Piperova	27
12765	Townsend	51
12875	Okindo	32
12963	Reed	49
13353	Soltwedel	36
13555	Chong	48
13649	Selby	25
13771	Taylor	44
14117	St-Onge	37
14447	Flamenbaum	37
14599	Herring	41
14681	Weber	45
14677	Kellogg	36
14883	Fernandez	41
15061	Williams	38
15233	Gagnon	47
15293	Ailee	46
15349	Schutz	22
15521	Lewis	46

Below the table, the message "29 rows selected." is displayed. The command line shows "SQL>".

Рис. 7.16. Вызов функции, определенной пользователем

The screenshot shows the Oracle SQL\*Plus interface. The command line displays the following SQL code:

```

SQL> SELECT Age(12345)
  2  FROM DUAL;
SELECT Age(12345)
*
ERROR at line 1:
ORA-06508: PL/SQL: Function returned without value
ORA-06512: at "CH07.AGE", line 16

ORA-01403: no data found
SQL>

```

The error message indicates that the function AGE returned without a value, which triggered ORA-06508, and that the error occurred at line 16 of the function definition. The final message shows that no data was found.

Рис. 7.17. Сообщение о недопустимом идентификаторе агента

сообщения об ошибке и вместо него вывести другое сообщение, более понятное пользователю? Эту задачу можно решить. Следует модифицировать функцию Age и сделать ее более устойчивой к возможным ошибкам пользователей.

## Модификация функции

Для того чтобы улучшить функцию Age, надо добавить проверку, в ходе которой определялось бы, существует ли значение AgentID, введенное пользователем. Лишь при положительном результате можно выполнять выражение для определения возраста. Если значение AgentID, переданное функции, не существует, функция должна отобразить сообщение о том, что идентификационный номер задан неверно, и завершить работу. Если AgentID задан корректно, функция может определить возраст и нормально завершить работу. Пользователь, который указал неправильный идентификационный номер, не увидит непонятного сообщения длиной в три строки.

Для того чтобы реализовать проверку корректности AgentID, выполните следующие действия.

1. Запустите Блокнот и откройте функцию Age, загрузив ее код из файла <имя>Ch07AgeFunction.sql, в котором вы ранее сохранили результаты своей работы.
2. Модифицируйте файл следующим образом. В начале раздела деклараций добавьте строку

```
v_Count INTEGER;
```

3. Между ключевым словом BEGIN и выражением SELECT включите следующий код (обратите внимание, что во второй строке после слова COUNT в скобках указана цифра 1, а не буква l):

```
/* Проверка существования AgentID */  
SELECT COUNT(1)  
INTO v_Count  
FROM Agents WHERE AgentID= InputAgentID;  
IF v_Count = 0 THEN  
    v_Age := -1;  
ELSE
```

4. Сдвиньте вправо каждую строку, относящуюся к выражению SELECT, включив в начало строки два пробела.

5. Сдвиньте на такую же величину строку с выражением присваивания v\_Age :\_TRUNC . . . .

6. После строки, выполняющей присваивание переменной v\_Age, включите новую строку и введите в ней следующее содержимое (также сдвинув его на два пробела):

```
END IF;
```

7. Сохраните модифицированный код в файле. Код, подвергшийся изменениям, показан на рис. 7.18. На нем отмечены внесенные изменения. Сравните ваш код с текстом функции, показанным на рисунке.

8. Оставьте Блокнот открытым.

```

Ch07AgeFunction.sql - Notepad
File Edit Format View Help
SET SERVEROUTPUT ON
CREATE OR REPLACE FUNCTION Age(InputAgentID IN NUMBER)
  RETURN INTEGER AS
  v_Count INTEGER;
  v_BirthDate Agents.Birthdate%TYPE;
  v_TodaysDate DATE;
  v_Age INTEGER;
BEGIN
  /* Test AgentID to see if it exists */
  SELECT COUNT(1)
  INTO v_Count
  FROM Agents WHERE AgentID= InputAgentID;
  IF v_Count = 0 THEN
    v_Age := -1;
  ELSE
    SELECT BirthDate, SYSDATE
    INTO v_BirthDate, v_TodaysDate
    FROM Agents
    WHERE AgentID = InputAgentID;
    v_Age := TRUNC((v_TodaysDate-v_BirthDate)/365.25);
  END IF;
  RETURN v_Age;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SOLERRM);
END Age;
/

```

**Рис. 7.18.** Модификация функции Age

Функция готова к проверке. Код функции надо скопировать в SQL\*Plus и нажать клавишу <Enter>, чтобы скомпилировать его. Поскольку в заголовке функции указано CREATE OR REPLACE FUNCTION, новый ее вариант, свободный от ошибок, заменит старую версию в базе данных.

Для того чтобы модифицированная функция Age была повторно скомпилирована, выполните следующие действия.

1. В SQL\*Plus введите CLEAR SCREEN и нажмите клавишу <Enter>.
2. Скопируйте код функции из редактора Блокнот в SQL\*Plus. Нажмите клавишу <Enter>, чтобы скомпилировать функцию и заменить ее предыдущую версию в базе данных.
- Oracle отобразит сообщение Function created (функция создана). В случае, если будет обнаружена синтаксическая ошибка, вернитесь к редактору Блокнот, внесите необходимые изменения и повторяйте действия, соответствующие данному этапу работы, до тех пор, пока не получите код, свободный от ошибок.
3. Закройте Блокнот. (Если с момента последнего сохранения кода вы вносили в него какие-то изменения, снова сохраните текст функции.)
4. Чтобы проверить новый код, введите приведенную ниже строку и нажмите клавишу <Enter>.

```

* Oracle SQL*Plus
File Edit Search Options Help
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE FUNCTION Age(InputAgentID IN NUMBER)
  2   RETURN INTEGER AS
  3   v_Count INTEGER;
  4   v_BirthDate Agents.Birthdate%TYPE;
  5   v_TodaysDate DATE;
  6   v_Age INTEGER;
  7 BEGIN
  8   /* Test AgentID to see if it exists */
  9   SELECT COUNT(1)
10   INTO v_Count
11   FROM AGENTS WHERE AgentID= InputAgentID;
12   IF v_Count = 0 THEN
13     v_Age := -1;
14   ELSE
15     SELECT BirthDate, SYSDATE
16     INTO v_BirthDate, v_TodaysDate
17     FROM Agents
18     WHERE AgentID = InputAgentID;
19     v_Age := TRUNC((v_TodaysDate-v_BirthDate)/365.25);
20   END IF;
21   RETURN v_Age;
22 EXCEPTION
23   WHEN OTHERS THEN
24     DBMS_OUTPUT.PUT_LINE(SQLERRM);
25 END Age;
26 /
Function created.

SQL> SELECT Age(10497) AS "Agent that exists" FROM DUAL;
Agent that exists
-----
52

SQL> SELECT Age(12345) AS "Agent that does not exist" FROM DUAL;
Agent that does not exist
-----
-1

SQL>

```

Модифицированная функция Age

Проверка функции Age посредством корректных и некорректных значений

**Рис. 7.19.** Проверка модифицированной функции Age

```
SELECT Age(10497) AS "Agent that exists" FROM DUAL;
```

Oracle отобразит вычисленное значение.

- Чтобы проверить реакцию модифицированной функции на получение несуществующего идентификатора агента, введите следующую строку и нажмите клавишу <Enter>:

```
SELECT Age(12345) AS "Agent that does not exist" FROM DUAL;
```

Функция вернет значение -1, свидетельствующее об ошибке. Длинное и непонятное пользователю сообщение уже не отображается. Мы добились поставленной цели. Результаты наших проверок показаны на рис. 7.19.

- Если вы не собираетесь заканчивать работу, оставьте SQL\*Plus открытым.

## Получение информации о функции и удаление функции

Когда компилятор PL/SQL компилирует функцию или процедуру, он генерирует Р-код; это название сокращенно означает псевдокод (pseudocode). Р-код — перено-

смый код функции или процедуры, который может выполняться на любом компьютере, на котором установлен интерпретатор PL/SQL. Если вы хотите получить более эффективную программу, вам надо потребовать, чтобы PL/SQL сгенерировал при компиляции функции или процедуры платформенно-ориентированный код. Платформенно-ориентированный код предназначен для выполнения на машине определенной архитектуры и не требует интерпретации. Чтобы можно было получить платформенно-ориентированный код, на вашем компьютере должен быть инсталлирован компилятор C.

## Вывод определения функции

Если вы создадите функцию или процедуру, в определении которой указано CREATE OR REPLACE, Oracle сохранит ее в словаре. При этом в базу данных будет записан как исходный код, так и его скомпилированный вариант (р-код). При вызове процедуры или функции Oracle читает р-код с диска и помещает его в память, подготавливая тем самым к выполнению. При необходимости вы можете получить доступ к информации о функции и отобразить ее различными способами. Представление словаря USER\_OBJECTS содержит высокоуровневую информацию об объектах. С этим представлением вы уже встречались ранее. Представление с именем USER\_SOURCE содержит исходный код функции или процедуры. Представление USER\_ERRORS содержит информацию об ошибках, возникших в процессе компиляции.

Для того чтобы вывести информацию о функции, выполните следующие действия.

1. В SQL\*Plus введите CLEAR SCREEN и нажмите клавишу <Enter>.
2. Чтобы отобразить имена ваших функций или процедур, хранящихся в базе данных, введите следующие строки кода:

```
COLUMN Object_Name FORMAT A20
COLUMN Object_Type FORMAT A12
SELECT Object_Name, Object_Type
FROM USER_OBJECTS
WHERE Object_Type IN ('FUNCTION', 'PROCEDURE');
```

В ответ Oracle выведет имя и тип Age — единственной созданной вами функции. Если бы вы создали другие функции и процедуры, информация о них также была бы отображена.

3. Введите следующие два выражения SQL\*Plus, в конце каждого из них нажмайте клавишу <Enter>:

```
SET PAGESIZE 40
CLEAR SCREEN
```

4. Отобразите имена и типы данных столбцов User\_Source, указав следующее выражение:

```
DESCRIBE User_Source
```

The screenshot shows the Oracle SQL\*Plus interface. In the command window, the command `SQL> DESCRIBE User_Source` is run, followed by `SQL> SELECT Text FROM User_Source WHERE Name = 'AGE' ORDER BY Line;`. The output shows the structure of the User\_Source object and the source code of the Age function.

Name	Null?	Type
NAME		VARCHAR2(30)
TYPE		VARCHAR2(12)
LINE		NUMBER
TEXT		VARCHAR2(4000)

**Описание User\_Source**

```

FUNCTION Age(InputAgentID IN NUMBER)
  RETURN INTEGER AS
  v_Count INTEGER;
  v_BirthDate Agents.Birthdate%TYPE;
  v_TodaysDate DATE;
  v_Age INTEGER;
BEGIN
  /* Test AgentID to see if it exists */
  SELECT COUNT(1)
  INTO v_Count
  FROM AGENTS WHERE AgentID= InputAgentID;
  IF v_Count = 0 THEN
    v_Age := -1;
  ELSE
    SELECT BirthDate, SYSDATE
    INTO v_BirthDate, v_TodaysDate
    FROM Agents
    WHERE AgentID = InputAgentID;
    v_Age := TRUNC((v_TodaysDate-v_BirthDate)/365.25);
  END IF;
  RETURN v_Age;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END Age;

```

**Исходный код функции Age, хранящейся в базе данных**

Рис. 7.20. Отображение структуры User\_Source и информации о хранимой функции

5. Чтобы получить подробную информацию об исходном коде функции Age, введите следующие строки кода, нажимая в конце каждой из них клавишу <Enter>:

```

SELECT Text
FROM User_Source
WHERE Name = 'AGE'
ORDER BY Line;

```

После короткой паузы Oracle отобразит исходный код функции Age (рис. 7.20).

6. Оставьте SQL\*Plus активизированным.

## Удаление функции

При работе с любым объектом может возникнуть потребность удалить его из базы данных. Удаление функции производится посредством следующего выражения:

```
DROP FUNCTION <имя_функции>;
```

Удалим функцию, созданную в данном разделе. Для этого сделайте следующее.

1. В SQL\*Plus введите следующую команду и нажмите после нее клавишу <Enter>:  

```
DROP FUNCTION Age;
```

Oracle вернет сообщение Function dropped (функция удалена).

2. Для того чтобы проверить, что произошло, снова введите в SQL\*Plus код, посредством которого вы получали информацию о функциях и процедурах:

```
SELECT Object_Name, Object_Type
FROM USER_OBJECTS
WHERE Object_Type IN ('FUNCTION', 'PROCEDURE');
```

Oracle ответит no rows selected (в случае, если в схеме не было других функций или процедур). Вы видите, что теперь в базе данных функций и процедур нет.

3. Введите exit и нажмите клавишу <Enter>, чтобы завершить сеанс работы с Oracle и закрыть SQL\*Plus.

## **Создание, использование и удаление процедур**

Как вы уже знаете, процедура содержит выражения SQL и PL/SQL, выполняющие некоторую задачу, а посредством параметров можно передавать значения процедуре или получать их от процедуры. В отличие от функции процедура не возвращает значение. Однако процедура может предоставлять вызывающему коду одно или несколько значений посредством параметров. Параметры определяются в заголовке процедуры; там же задаются их типы. После заголовка процедуры следует необязательный раздел деклараций, в котором можно определить локальные переменные и курсоры. В отличие от неименованного блока PL/SQL раздел деклараций не начинается с ключевого слова DECLARE. Оно попросту не указывается. После раздела деклараций располагается раздел исполняемого кода и необязательный раздел обработки исключений.

Для того чтобы процедуру можно было использовать, ее надо создать. Созданная и сохраненная в базе данных процедура может быть вызвана из SQL\*Plus или из другой процедуры посредством выражения CALL. Для того чтобы переопределить подпрограмму, надо модифицировать ее текст в текстовом редакторе и повторить компиляцию. Если процедура хранится в базе данных, можно получить дополнительную информацию о ней или удалить ее.

## **Преимущества процедур**

Процедуры, хранящиеся в базе данных, имеют ряд преимуществ перед блоками PL/SQL, выполняющими те же задачи. Процедура компилируется один раз и выполняется при обращении к ней. За счет компиляции в р-код либо в платформенно-ориентированный код эффективность выполнения процедур достаточно высока. Объ-

единение SQL-выражений в единую процедуру снижает трафик при обмене между клиентской машиной и сервером Oracle, так как серверу предоставляется сразу весь SQL-код. Создание набора часто используемых процедур препятствует созданию различными пользователями кода, выполняющего по сути одни и те же действия. Хорошо протестированные процедуры, хранящиеся в базе, быстро завоевывают доверие пользователей, которые применяют их в своих приложениях, что снижает время разработки новых продуктов. Разрешив доступ к важной информации только с помощью хранимых процедур, можно повысить уровень безопасности базы данных. Вместо того чтобы непосредственно предоставлять доступ к синонимам или таблицам, пользователям предлагаются процедуры, которые корректно выполняют для них необходимые действия.

Процедуры позволяют ограничить операции обновления данных в важных таблицах. Для этого пользователю надо запретить все пути доступа к информации, кроме вызова процедуры. Хранимые процедуры особенно полезны в качестве точек входа для доступа к данным. Они позволяют проверить, например, попадают ли значения в допустимый интервал. Использование хранимой процедуры — единственный универсальный и в то же время надежный способ обеспечить все требуемые действия при обновлении, включении или удалении столбцов в одной из взаимосвязанных таблиц. Предположим, например, что хранимая процедура реализует операцию INSERT, предназначенную для создания счета пользователя. В рамках той же процедуры можно реализовать код, который выполнит необходимые дополнительные действия по учету данных, хранящихся в таблицах, связанных посредством первичного и внешнего ключей.

## Создание процедур и сохранение их в базе

Для того чтобы можно было использовать хранимую (именованную) процедуру, ее надо создать. Для создания новой процедуры (или замены существующей) используется следующая конструкция:

```
CREATE [OR REPLACE] PROCEDURE <имя_процедуры>
[ ( <имя_параметра> [IN|OUT|IN OUT]
    <тип_данных> [, . . . ] ) ] {IS|AS}
[ <объявления_локальных_переменных>
]
BEGIN
    <тело_процедуры>
[EXCEPTION
    <код_для_обработки_исключений> ]
END [<имя_процедуры>];
```

Заметьте, что выражение OR REPLACE не является обязательным. Если оно присутствует, разработчик может либо создать новую процедуру, либо заменить процедуру с совпадающим именем; при этом ошибки не возникают. После имени процедуры сле-

дует необязательный список параметров, для которых указаны спецификаторы IN, OUT или IN OUT. Каждый параметр отделяется от следующего запятой. После заголовка процедуры могут объявляться локальные переменные, в том числе курсоры и скалярные переменные. Объявления в разделе деклараций формируются так же, как и в неименованном блоке PL/SQL.

В отличие от неименованного блока в хранимой процедуре не указывается ключевое слово DECLARE. Выражения SQL и PL/SQL, составляющие тело процедуры, надо включать обязательно. Они помещаются между ключевыми словами BEGIN и END. Необязательный раздел обработки исключений начинается с ключевого слова EXCEPTION и заканчивается ключевым словом END, которое обозначает также конец всей процедуры. Как обычно, в конце процедуры ставится косая черта (/); в результате после нажатия клавиши <Enter> система компилирует процедуру и сохраняет ее. Включать косую черту в файл с кодом процедуры удобно, так как при этом не приходится специально заботиться о ее компиляции; достаточно скопировать код в SQL\*Plus и нажать клавишу <Enter>. Если в процессе компиляции возникнет ошибка, определение процедуры не будет сохранено в базе данных. В этом случае надо исправить ошибку, повторно скомпилировать код и поместить определение процедуры в базу.

Создадим хранимую процедуру ForSale, которая будет извлекать и отображать объекты, находящиеся в конкретном городе и предназначенные для продажи. Название города (CityIn) представляет собой входной параметр (IN) процедуры. Он определяет, в каком городе следует искать объекты, выставляемые на продажу. Данная процедура полезна, поскольку выражение SQL, выполняющее эти действия, достаточно сложное. Требовать, чтобы агенты создавали объединение из трех таблиц и использовали выражение WHERE для фильтрации, неоправданно. Лучше поместить требуемую логику в процедуру, чтобы агенты могли вызывать ее и получать необходимую информацию.

Запустите SQL\*Plus и Блокнот; когда код будет готов, вы сможете сохранить его в файле. Зарегистрируйтесь в системе Oracle и следуйте приведенным ниже инструкциям по созданию процедуры ForSale.

1. В редакторе Блокнот введите команды, которые позволят вам очистить экран и установить значение ON переменной окружения SERVEROUTPUT.

```
CLEAR SCREEN
SET SERVEROUTPUT ON
```

2. Продолжая работу с редактором Блокнот, введите заголовок процедуры и раздел деклараций, содержащий лишь объявление курсора для извлечения данных. Курсор объединяет три таблицы Redwood Realty и извлекает из них информацию, обеспечивая получение лишь сведений об объектах для продажи.

```
CREATE OR REPLACE PROCEDURE ForSale(CityIn IN VARCHAR2) AS
CURSOR cv_Properties IS
SELECT PropertyID, AskingPrice, Address, City
```

```
FROM Properties JOIN Listings USING(PropertyID)
    JOIN CustAgentList USING(ListingID)
WHERE UPPER(City) = UPPER(CityIn)
    AND UPPER(ContactReason) = 'SELL';
```

3. Далее вам следует ввести тело процедуры. В нем содержится цикл FOR курсора и несколько строк DBMS\_OUTPUT для отображения заголовка и форматирования строк. Пробелы можно расставлять так, как вам удобно. Конкретное число пробелов важно лишь в строковых литералах, содержащихся в кавычках. Так, в первом выражении DBMS\_OUTPUT после Prop\_ID следуют четыре пробела и три пробела после Asking. Заметьте, что в модели форматирования выражения TO\_CHAR для PropertyID указаны нули, а не буквы O.

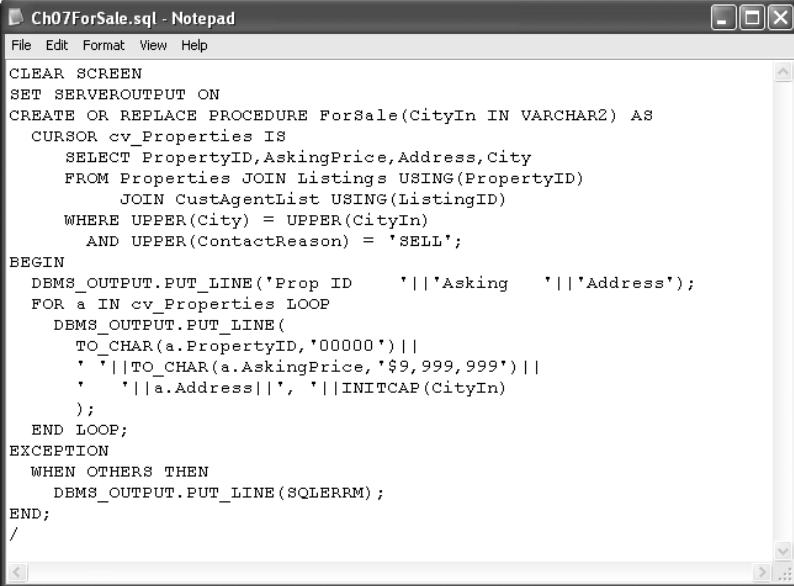
```
BEGIN
    DBMS_OUTPUT.PUT_LINE('Prop ID '||'Asking'||'Address');
    FOR a IN cv_Properties LOOP
        DBMS_OUTPUT.PUT_LINE(
            TO_CHAR(a.PropertyID,'00000')|| |
            ' '||TO_CHAR(a.AskingPrice,'$9,999,999')|| |
            ' '||a.Address||', '|INITCAP(CityIn)
        );
    END LOOP;
```

4. Введите раздел обработки исключений, ключевое слово END, завершающее процедуру, и косую черту.

```
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

Полностью код ForSale должен выглядеть так, как показано на рис. 7.21.

5. Завершив создание кода, сохраните его в файле. Файл вам понадобится в случае, если вы захотите модифицировать процедуру или вывести ее код на печать. Задайте имя файла <имя>Ch07ForSale.sql.
6. Оставьте Блокнот открытым на тот случай, если вам потребуется исправлять синтаксические ошибки. Скопируйте весь код, включая завершающую косую черту, из Блокнота (или из другого текстового редактора) в SQL\*Plus.
7. Скопировав код в SQL\*Plus, нажмите клавишу <Enter>, чтобы скомпилировать процедуру и сохранить ее в базе данных. Если возникнут ошибки, вернитесь в текстовый редактор, исправьте их, сохраните код в файле и повторите пп. 6 и 7. Признаком успешной компиляции кода является сообщение Procedure created (рис. 7.22).
8. Закройте Блокнот или другой используемый вами текстовый редактор, но SQL\*Plus оставьте открытым.



```

Ch07ForSale.sql - Notepad
File Edit Format View Help

CLEAR SCREEN
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE ForSale(CityIn IN VARCHAR2) AS
  CURSOR cv_Properties IS
    SELECT PropertyID, AskingPrice, Address, City
    FROM Properties JOIN Listings USING(PropertyID)
    JOIN CustAgentList USING(ListingID)
    WHERE UPPER(City) = UPPER(CityIn)
      AND UPPER(ContactReason) = 'SELL';
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Prop ID      || Asking      || Address');
    FOR a IN cv_Properties LOOP
      DBMS_OUTPUT.PUT_LINE(
        TO_CHAR(a.PropertyID,'00000') ||
        ' || TO_CHAR(a.AskingPrice, '$9,999,999') ||
        ' || a.Address || ', ' || INITCAP(CityIn)
      );
    END LOOP;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
  END;
/

```

Рис. 7.21. Полный код процедуры ForSale (отображается в окне Блокнота)



```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SET SERVEROUTPUT ON
SQL> CREATE OR REPLACE PROCEDURE ForSale(CityIn IN VARCHAR2) AS
 2  CURSOR cv_Properties IS
 3    SELECT PropertyID, AskingPrice, Address, City
 4    FROM Properties JOIN Listings USING(PropertyID)
 5    JOIN CustAgentList USING(ListingID)
 6    WHERE UPPER(City) = UPPER(CityIn)
 7    AND UPPER(ContactReason) = 'SELL';
 8  BEGIN
 9    DBMS_OUTPUT.PUT_LINE('Prop ID      || Asking      || Address');
10   FOR a IN cv_Properties LOOP
11     DBMS_OUTPUT.PUT_LINE(
12       TO_CHAR(a.PropertyID,'00000') ||
13       ' || TO_CHAR(a.AskingPrice, '$9,999,999') ||
14       ' || a.Address || ', ' || INITCAP(CityIn)
15     );
16   END LOOP;
17  EXCEPTION
18    WHEN OTHERS THEN
19      DBMS_OUTPUT.PUT_LINE(SQLERRM);
20  END;
21 /

Procedure created.

SQL>

```

Рис. 7.22. Сообщение, свидетельствующее о том, что в коде процедуры отсутствуют синтаксические ошибки

Появление сообщения Procedure created свидетельствует о том, что при компиляции кода процедуры система Oracle не обнаружила синтаксических ошибок. Возможны логические ошибки, но на данном этапе мы не обнаружим их. После успешной компиляции Oracle записывает исходный и скомпилированный код процедуры в базу данных. С этого момента она будет доступна в пределах схемы, а также в других схемах, которым были предоставлены соответствующие права доступа.

Процедура работает следующим образом. Когда пользователь вызывает ее, он указывает в качестве единственного параметра название города (например, 'Loleta'). Функция DBMS\_OUTPUT, следующая за ключевым словом BEGIN, формирует заголовки столбцов. В цикле FOR курсора извлекается информация из трех таблиц, причем столбец ContactReason должен содержать значение SELL (оно приводится к верхнему регистру, чтобы исключить ошибки, способные помешать извлечению данных). Название города, заданное пользователем в качестве параметра (CityIn), представляется в выражение WHERE для курсора, содержащееся в разделе деклараций. Таким образом, в цикле FOR курсора извлекаются строки, соответствующие объектам, предназначенному для продажи в городе, название которого было передано процедуре посредством формального параметра CityIn. Название города также присутствует в вызове DBMS\_OUTPUT непосредственно перед ключевыми словами END LOOP. Оно приводится к требуемому регистру посредством INITCAP. Функции TO\_CHAR предназначены лишь для форматирования данных и представления их в более привлекательной форме. Обработчик исключений содержит выражение WHERE OTHERS THEN, посредством которого организуется вывод сообщений об ошибках. В дальнейшем можно модернизировать код с тем, чтобы обнаруживать конкретные ошибки и отображать сообщения, более понятные для пользователей.

## Вызов хранимой процедуры

Для вызова процедуры используется команда CALL. Кроме того, для обращения к именованным подпрограммам предусмотрена команда EXECUTE и ее сокращенная форма EXEC. Мы ограничимся применением CALL. Протестируем процедуру.

Для того чтобы вызвать именованную процедуру в SQL\*Plus, выполните следующие действия.

1. В SQL\*Plus введите приведенные ниже команды. Переменную среды надо устанавливать только в том случае, если вы только что зарегистрировались в системе. Если вы продолжаете сеанс работы и уже выполняли примеры, без первой команды можно обойтись.

```
SET SERVEROUTPUT ON  
CLEAR SCREEN
```

2. Вызовите процедуру ForSale, чтобы получить информацию об объектах, предназначенных для продажи в городе Лолета, штат Калифорния. Введите следующую команду и нажмите клавишу <Enter>, чтобы выполнить ее:

```
CALL ForSale('loleta');
```

Процедура ForSale возвратит информацию о семи продаваемых домах.

3. Вызовите процедуру ForSale, чтобы получить информацию об объектах, предназначенных для продажи в городе Орик, штат Калифорния. Введите приведенную

ниже команду и нажмите клавишу <Enter>, чтобы выполнить ее. (Название города вводите прописными буквами, чтобы исключить ошибки, связанные с регистром символов.)

```
CALL ForSale('ORICK');
```

Вы увидите список объектов и сообщение Call completed.

4. Вызовите процедуру ForSale в третий раз, запросив информацию о домах, выставленных на продажу в Блу Хейвен.

```
CALL ForSale('Blue Haven');
```

На этот раз отображаются только заголовки столбцов. Объектов для продажи нет.

5. Дело в том, что произошла ошибка. Город, который нас интересует, называется Блу Лейк. Попробуем еще раз. Введите представленную ниже строку и нажмите клавишу <Enter>.

```
CALL ForSale('BLUE lake');
```

Теперь все в порядке. В городе Блу Лейк предлагается шесть объектов.

6. Выполним еще одну проверку. Вызовем процедуру ForSale без параметров. Введите представленную ниже строку и нажмите клавишу <Enter>.

```
CALL ForSale();
```

Oracle выведет сообщение об ошибке. Обработчик исключений отображает номер ошибки и полный текст сообщения (рис. 7.23).

Процедуру можно модернизировать: предусмотреть в ней код, выявляющий ситуацию, когда ни одна строка не возвращается, и отображать сообщение, более понятное пользователю, информирующее его, например, о том, что в указанном городе жилье не продается.

## **Модификация, повторная компиляция и сохранение процедуры**

В случае необходимости вы можете модифицировать процедуру. Для этого надо открыть ее исходный код в текстовом редакторе, внести необходимые изменения и повторно скомпилировать код в SQL\*Plus. (В заголовке процедуры должно присутствовать выражение CREATE OR REPLACE. Если ключевые слова OR REPLACE будут отсутствовать, то при попытке повторной компиляции Oracle вернет сообщение об ошибке, указывающее на то, что процедура с таким именем уже существует.)

Поскольку вы предусмотрительно сохранили исходный код процедуры в файле Ch07ForSale.sql, вы можете без труда внести требуемые изменения. Желательно сохранить имеющуюся версию, чтобы впоследствии можно было вернуться к ней. Подготовьтесь к работе, запустив Блокнот или другой текстовый редактор и открыв в нем файл, который вы сохранили под именем <имя>Ch07ForSale.sql.

```

* Oracle SQL*Plus
File Edit Search Options Help
Prop ID Asking Address
00445 $104,000 3146 Copenhagen Rd, Loleta
00638 $111,950 356 Loleta Dr, Loleta
00689 $114,000 296 Spring, Loleta
01458 $145,000 229 Church, Loleta
01574 $154,000 552 Pershing, Loleta
01679 $162,500 119 Bay View Dr, Loleta
01932 $282,500 632 Pershing, Loleta

Call completed.

SQL> CALL ForSale('ORICK');
Prop ID Asking Address
01136 $129,950 124388 US Highway 101 N, Orick
00298 $170,266 128779 Hwy 101, Orick
01806 $184,950 124244 US Highway 101 N, Orick

Call completed.

SQL> CALL ForSale('Blue Haven');
Prop ID Asking Address

Call completed.

SQL> CALL ForSale('BLUE lake');
Prop ID Asking Address
00654 $112,000 549 Broad, Blue Lake
01367 $139,500 2115 Chezum Rd, Blue Lake
01460 $145,000 830 Railroad Ave, Blue Lake
00153 $159,446 833 Blue Lake Bl, Blue Lake
00234 $190,463 741 2nd Ave, Blue Lake
01999 $725,000 5990 Stover Rd, Blue Lake

Call completed.

SQL> CALL ForSale();
CALL ForSale()
*
ERROR at line 1:
ORA-06553: PLS-306: wrong number or types of arguments in call to 'FORSALE'

SQL>

```

Вызов функции ForSale без параметров и полученное в результате сообщение об ошибке

**Рис. 7.23.** Результаты проверки процедуры ForSale

Для того чтобы модифицировать файл сценария и повторно скомпилировать его, выполните следующие действия.

- Сохраните код в файле под другим именем. Выберите пункты меню File и Save As, а затем введите имя файла <имя>Ch07ForSale2.sql, заменив <имя> своим именем.
- Ведите новую строку между заголовком процедуры и определением курсора и введите приведенную ниже последовательность символов, сдвинув ее вправо на два пробела.

```
v_PropCount INTEGER NOT NULL := 0;
```

Локальная переменная v\_PropCount будет хранить количество возвращаемых строк. Если ни одна строка не будет получена, значение переменной останется равным нулю.

- Ведите следующий код непосредственно после заголовка цикла FOR a IN cv\_Properties LOOP:
- ```
v_PropCount := v_PropCount + 1;
```
- После выражения END LOOP: и перед EXCEPTION: введите приведенные ниже строки кода

```

IF v_PropCount = 0 THEN
    DBMS_OUTPUT.PUT_LINE
        ('*****No homes for sale in ' || INITCAP(CityIn));
END IF;

```

5. Закончив внесение изменений, сохраните код в файле.
6. В SQL\*Plus введите CLEAR SCREEN и нажмите клавишу <Enter>, чтобы очистить экран. Скопируйте код процедуры из редактора Блокнот в SQL\*Plus и нажмите клавишу <Enter>, чтобы скомпилировать и сохранить в базе данных модифицированную версию. Oracle вернет сообщение Procedure created. Если система сообщит вам о наличии ошибок, введите SHOW ERRORS, нажмите клавишу <Enter> и найдите причину проблемы.
7. Исправьте в текстовом редакторе ошибки и, если необходимо, повторите действия, приведенные в п. 5.
8. Проверьте модифицированный код. Для этого введите следующие две строки (после каждой из них нажимайте клавишу <Enter>):

```

CALL ForSale('Loleta');
CALL ForSale('Mitchell');

```

Для города Лолета процедура действует, как и раньше. Однако если вы введете название Митчел, то получите новое сообщение, так как в этом городе нет объектов для продажи (рис. 7.24).

9. Закройте текстовый редактор, с которым вы работали.
10. Не закрывайте SQL\*Plus, так как вам предстоит выполнить еще некоторые действия.

## **Получение информации о процедуре и удаление процедуры**

Определение процедуры можно отобразить так же, как вы раньше делали это для функции. Кроме того, если процедура больше не нужна, вы можете удалить ее.

### **Вывод определения функции**

Как было сказано ранее, если вы создадите функцию или процедуру, в определении которой указано CREATE OR REPLACE, Oracle сохранит ее в словаре. При этом в базу данных записывается как исходный код, так и его скомпилированный вариант (р-код). По мере необходимости вы можете получить доступ к информации о процедуре и отобразить ее различными способами. Представление словаря USER\_OBJECTS содержит высокоуровневую информацию об объектах. С этим представлением вы уже встречались ранее. Представление с именем USER\_SOURCE содержит исходный

The screenshot shows the Oracle SQL\*Plus interface with the following content:

```

SQL> CREATE OR REPLACE PROCEDURE ForSale(CityIn IN VARCHAR2) AS
2   v_PropCount INTEGER NOT NULL := 0; ← Новая строка
3   CURSOR cv_Properties IS
4     SELECT PropertyID, AskingPrice, Address, City
5     FROM Properties JOIN Listings USING(PropertyID)
6     JOIN CustAgentList USING(ListingID)
7     WHERE UPPER(City) = UPPER(CityIn)
8     AND UPPER(ContactReason) = 'SELL';
9   BEGIN
10    DBMS_OUTPUT.PUT_LINE('Prop ID    ||' Asking ||'Address');
11   FOR a IN cv_Properties LOOP
12     v_PropCount := v_PropCount + 1; ← Новая строка
13     DBMS_OUTPUT.PUT_LINE(
14       TO_CHAR(a.PropertyID, '00000') ||
15       '||TO_CHAR(a.AskingPrice,$0,999,999') ||
16       '||a.Address||', '||INITCAP(CityIn)
17     );
18   END LOOP;
19   IF v_PropCount = 0 THEN ← Три новых строки
20     DBMS_OUTPUT.PUT_LINE('*****No homes for sale in'||INITCAP(CityIn));
21   END IF; ←
22 EXCEPTION
23   WHEN OTHERS THEN
24     DBMS_OUTPUT.PUT_LINE(SQLERRM);
25 END;
26 /
Procedure created.

SQL> CALL ForSale('Loleta');
Prop ID      Asking      Address
00445      $104,000    3146 Copenhagen Rd, Loleta
00638      $111,950    356 Loleta Dr, Loleta
00689      $114,000    296 Spring, Loleta
01458      $145,000    229 Church, Loleta
01574      $154,000    552 Pershing, Loleta
01679      $162,500    119 Bay View Dr, Loleta
01932      $282,500    632 Pershing, Loleta

Call completed.

SQL> CALL ForSale('Mitchell');
Prop ID      Asking      Address
***** No homes for sale in Mitchell

Call completed.

```

Рис. 7.24. Тестирование модифицированной процедуры ForSale

код функции или процедуры. Если SQL\*Plus не выполняется, запустите его. Код, который вы сейчас введете, вам не придется сохранять. Если же по каким-то причинам вы решите записать его в файл, запустите текстовый редактор. По мере ввода команд копируйте их в окно редактора.

Для того чтобы получить информацию о процедуре, выполните следующие действия.

1. В SQL\*Plus введите CLEAR SCREEN и нажмите клавишу <Enter>.
2. Чтобы отобразить имена ваших функций или процедур, хранящихся в базе данных, введите следующие строки:

```

COLUMN Object_Name FORMAT A20
COLUMN Object_Type FORMAT A12
SELECT Object_Name, Object_Type
FROM USER_OBJECTS
WHERE Object_Type IN ('FUNCTION', 'PROCEDURE');

```

В ответ Oracle выведет имя и тип процедуры ForSale. Это единственная процедура, которую вы создали и не удалили. (Как вы помните, функция Age уже удалена. Если вы не удалили ее, то получите информацию о ней.)

```

File Edit Search Options Help
SQL> SELECT Text
2  FROM User_Source
3  WHERE Name = 'FORSALE'
4  ORDER BY Line;

TEXT
-----
PROCEDURE ForSale(CityIn IN VARCHAR2) AS
  v_PropCount INTEGER NOT NULL := 0;
  CURSOR cu_Properties IS
    SELECT PropertyID, AskingPrice, Address, City
    FROM Properties JOIN Listings USING(PropertyID)
    JOIN CustAgentList USING(ListingID)
    WHERE UPPER(City) = UPPER(CityIn)
      AND UPPER(ContactReason) = 'SELL';
BEGIN
  DBMS_OUTPUT.PUT_LINE('Prop ID    ||| Asking    ||| Address');
  FOR a IN cu_Properties LOOP
    v_PropCount := v_PropCount + 1;
    DBMS_OUTPUT.PUT_LINE(
      TO_CHAR(a.PropertyID, '00000') ||
      ' ||| TO_CHAR(a.AskingPrice, '$9,999,999') ||
      ' ||| a.Address |||','||INITCAP(CityIn)
    );
  END LOOP;
  IF v_PropCount = 0 THEN
    DBMS_OUTPUT.PUT_LINE('*****No homes for sale in '||INITCAP(CityIn));
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
25 rows selected.

SQL>

```

**Рис. 7.25.** Отображение исходного кода процедуры

3. Введите следующие два выражения SQL\*Plus, нажимая в конце каждого из них клавишу <Enter>:

```

SET PAGESIZE 40
CLEAR SCREEN

```

4. Чтобы получить подробную информацию об исходном коде процедуры ForSale, введите следующие строки кода, нажимая после каждой из них клавишу <Enter>. Имя FORSALE в третьей строке должно быть введено прописными буквами:

```

SELECT Text
FROM User_Source
WHERE Name = 'FORSALE'
ORDER BY Line;

```

После короткой паузы система Oracle отобразит 25 строк исходного кода процедуры ForSale (рис. 7.25).

5. Оставьте SQL\*Plus активизированным.

## Удаление процедуры

При необходимости из базы данных можно удалить любой объект. Для удаления процедуры служит следующая команда:

```
DROP PROCEDURE <имя_процедуры>;
```

Удалим процедуру, созданную в этом разделе. Для этого выполните следующее.

1. В SQL\*Plus введите следующую команду и нажмите клавишу <Enter>:

```
DROP PROCEDURE ForSale;
```

Oracle вернет сообщение Procedure dropped.

2. Для того чтобы убедиться, что система Oracle удалила процедуру, снова введите в SQL\*Plus код, посредством которого вы получали информацию о функциях и процедурах.

```
SELECT Object_Name, Object_Type
FROM USER_OBJECTS
WHERE Object_Type IN ('FUNCTION', 'PROCEDURE');
```

Oracle ответит вам no rows selected. Это означает, что в базе данных в текущий момент нет ни функций, ни процедур.

3. Ведите exit и нажмите клавишу <Enter>, чтобы завершить сеанс работы с Oracle и закрыть SQL\*Plus.
4. Если вы загружали Блокнот и копировали в него вводимые команды, сохраните их, выбрав папку и имя файла. Закройте Блокнот или тот редактор, с которым вы работали.

## Резюме

PL/SQL — это процедурный язык, расширяющий SQL. Блоки PL/SQL могут быть неименованными (анонимными) или именованными. Анонимные блоки не имеют имени и не хранятся в базе данных. Именованным блокам присваиваются имена, и они сохраняются в базе данных. Неименованные блоки PL/SQL могут содержать до трех разделов: необязательный раздел деклараций, в котором объявляются локальные переменные, обязательный раздел исполняемого кода, признаком начала которого служит ключевое слово BEGIN, и необязательный раздел обработки исключений. Раздел обработки исключений содержит код, который получает управление, если при выполнении блока возникают ошибки. При этом выполняется та часть раздела, которая соответствует заранее оговоренным условиям. Процедура PUT\_LINE пакета DBMS\_OUTPUT отображает выходные данные на экране, однако чтобы это произошло, переменная среды SQL\*Plus с именем SERVEROUTPUT должна иметь значение ON.

В отличие от SQL в PL/SQL предусмотрены итерации, которые позволяют организовать повторное выполнения кода в зависимости от результатов проверки условия. Простой цикл представляет собой набор выражений, помещенных между ключевыми словами LOOP и END LOOP. Для того чтобы завершить простой цикл, надо выполнить

команду EXIT или EXIT WHEN. Команда EXIT обеспечивает безусловный выход из цикла, а EXIT WHEN завершает цикл при выполнении определенного условия. Цикл WHILE выполняется до тех пор, пока заданное условие равно TRUE. Если нужен счетчик итераций, то для этой цели хорошо подходит цикл FOR. Текло цикла FOR выполняется до тех пор, пока значение переменной цикла не превысит верхнюю границу (прямой счет) либо не станет меньше нижней границы (обратный счет). Цикл FOR курсора — это специальный вариант цикла, который выполняется до тех пор, пока в курсоре остаются записи, извлеченные из базы данных.

Если в блоке PL/SQL из таблицы извлекается лишь одна строка, Oracle создает неявный курсор. Этот курсор автоматически открывается, обрабатывается и закрывается. Для того чтобы извлечь несколько строк из таблицы, необходимо использовать явный курсор. Перед извлечением информации из базы явный курсор должен быть открыт. Команда FETCH, помещенная в тело цикла, поочередно извлекает записи из курсора. Для того чтобы посредством команды SELECT можно было загружать столбцы, она должна содержать выражение INTO и список локальных переменных PL/SQL, в которые Oracle помещает значения столбцов, полученные из базы данных. По окончании работы с курсором его надо закрыть. Цикл FOR курсора автоматически открывает курсор, извлекает строки и закрывает его по окончании работы.

Существуют два типа именованных блоков PL/SQL: функции и процедуры. Функция содержит заголовок и необязательный список параметров, разделенных запятыми. Параметры позволяют передавать данные функции и получать от нее информацию. Именованные блоки PL/SQL компилируются и сохраняются в базе данных. Вызвать их может каждый, кто обладает необходимыми привилегиями. Функция возвращает одно значение. Кроме того, она может передавать информацию посредством параметров. Процедуры имеют ту же структуру, что и функции, но отличаются от функций тем, что не могут возвращать значение. Процедура, ссылающаяся на функцию или другую процедуру, вызывает ее. Поскольку в базе данных хранится исходный, и скомпилированный код (р-код), разработчик может отображать исходные тексты своих функций. Для этого используется представление словаря USER\_OBJECTS.

При необходимости можно удалить функции или процедуры PL/SQL из базы данных. Удаленная функция более недоступна; ее имя и определение стираются из словаря.

## Основные термины

- END
- IN
- IN OUT
- OUT
- Р-код

- Активизированная строка
- Бесконечный цикл
- Блок
- Вызов
- Глобальная переменная
- Именованный блок
- Исключение
- Компиляция
- Константа
- Курсор
- Локальная переменная
- Неименованный (анонимный) блок
- Неявный курсор
- Параметр
- Платформенно-ориентированный код
- Раздел деклараций
- Раздел заголовка
- Раздел исполняемого кода
- Раздел обработки исключений
- Функция
- Цикл FOR
- Цикл FOR курсора
- Явный курсор

## **Повторение пройденного материала**

### **Истина или ложь?**

1. Анонимный блок должен содержать раздел деклараций.
2. В блоке PL/SQL необязательно объявлять переменную цикла, используемую в цикле FOR.
3. Код, помещенный в блок PL/SQL, выполняется эффективнее, чем тот же код, заданный как набор команд в SQL\*Plus, поскольку в составе блока PL/SQL код передается серверу как единое целое.
4. В процедуре можно изменять любой параметр, который был передан ей.

5. Блок PL/SQL, который может извлекать из базы несколько строк, должен использовать явный курсор, посредством которого строки поочередно передаются коду PL/SQL.

### **Заполнить пропущенное**

1. Цикл, тело которого повторяется до тех пор, пока не будет выполнено заданное условие, называется \_\_\_\_\_.
2. Подпрограмма, которая возвращает вызывающей программе одно значение, называется \_\_\_\_\_.
3. \_\_\_\_\_ — это область памяти, которая содержит строки, извлеченные посредством выражения SELECT, содержащегося в блоке PL/SQL.
4. Блок PL/SQL — это либо анонимный блок, либо \_\_\_\_\_ блок.
5. Код, выполнение которого на конкретном компьютере не требует интерпретации, называется \_\_\_\_\_ кодом.

### **Варианты ответов**

1. Когда Oracle \_\_\_\_\_ блок PL/SQL, высокоуровневые инструкции PL/SQL преобразуются в другой формат, обеспечивающий их более эффективное выполнение.
  - а) Идентифицирует.
  - б) Извлекает.
  - в) Выполняет.
  - г) Компилирует.
2. Блок PL/SQL не может содержать:
  - а) Выражения SELECT с подчиненными запросами.
  - б) Выражения, предназначенные для определения данных, например CREATE TABLE.
  - в) Выражения, предназначенные для преобразования данных, например UPDATE.
  - г) Арифметические выражения, например  
`v_Variable := 24;`
3. Неименованный блок PL/SQL не может содержать
  - а) Раздел деклараций.
  - б) Раздел исполняемого кода.
  - в) Раздел заголовка.
  - г) Раздел обработки исключений.

4. Предположим, что вы обрабатываете несколько строк, извлеченных из базы данных. Какой цикл позволяет проще всего сделать это?
  - а) Цикл FOR.
  - б) Цикл FOR курсора.
  - в) Цикл WHILE.
  - г) Простой цикл.
5. Какое выражение используется для вызова хранимой процедуры?
  - а) RUN
  - б) EXECUTIVE
  - в) INVOKE
  - г) CALL

## Упражнения

### 1. Readwood Realty

Роберту Стирлингу необходимо, чтобы каждый его агент мог запустить процедуру и получить список всех объектов, для которых этот агент является инициатором. (Агентом-инициатором считается тот, кто по требованию владельца занес объект в список для продажи.) Каждый агент может вызывать процедуру *MyListings* из SQL\*Plus или iSQL\*Plus. Для этого используется следующее выражение:

```
CALL MyListings ('<фамилия_агента>' | 'all');
```

Здесь в качестве параметра может быть указано имя агента-инициатора. Оно ограничивает выходные данные объектами, соответствующими этому агенту. В качестве параметра можно также задать значение *all*, которое указывает на необходимость вывода списка всех агентов-инициаторов и их объектов. Для того чтобы упростить задачу, сформируем лишь следующие столбцы: *ListingID*, агент покупателя, агент-инициатор, *BidPrice* и *AskingPrice*. В дальнейшем вы сможете расширить процедуру, предусмотрев в ней указание адреса объекта.

Загрузите SQL\*Plus и зарегистрируйтесь в системе Oracle. Затем, следуя приведенным ниже инструкциям создайте процедуру *MyListings* и протестируйте ее. Результатом работы будут исходные коды процедуры и файл, содержащий результат выполнения процедуры для трех сценариев тестирования. Поместите свое имя в начало текста, используя для этого текстовый редактор.

1. Введите в редакторе Блокнот заголовок и раздел деклараций и сохраните код в файле <имя>Ch07Prob1Part1.sql.

```
CREATE OR REPLACE PROCEDURE
MyListings(AgentNameIn IN VARCHAR2) AS
v_Count INTEGER;
```

```

CURSOR cv_AllAgents IS
    SELECT ListingID, d.LastName AS
        LISTAGENT,
        a.LastName AS BUYAGENT,
        BidPrice, AskingPrice
    FROM Agents a JOIN CustAgentList b ON
        a.AgentID = b.AgentID
        JOIN Listings c ON b.ListingID =
            c.ListingID
        JOIN Agents d ON
            c.ListingAgentID = d.AgentID
    WHERE UPPER(ContactReason) = 'BUY'
    ORDER BY d.LastName, a.LastName;
CURSOR cv_OneAgent IS
    SELECT ListingID, d.LastName AS
        LISTAGENT,
        a.LastName AS BUYAGENT,
        BidPrice, AskingPrice
    FROM Agents a JOIN CustAgentList b ON
        a.AgentID = b.AgentID
        JOIN Listings c ON b.ListingID =
            c.ListingID
        JOIN Agents d ON
            c.ListingAgentID = d.AgentID
    WHERE UPPER(ContactReason) = 'BUY'
        AND UPPER(d.LastName) =
            UPPER(AgentNameIn)
    ORDER BY a.LastName;

```

2. Продолжая работу с текстовым редактором, введите следующий фрагмент кода:

```

BEGIN
    /* Присутствует ли имя агента в таблице Agents */
    SELECT COUNT(*)
        INTO v_Count FROM Agents
        WHERE UPPER(LastName) =
            UPPER(AgentNameIn);
    -- Три варианта: ALL, конкретный агент, недопустимое имя
    IF UPPER(AgentNameIn) = 'ALL' THEN
        FOR X IN cv_AllAgents LOOP
            DBMS_OUTPUT.PUT_LINE(
                ' ' || X.ListingID|| |
                ' ' || RPAD(X.LISTAGENT,15)|| |
                ' ' || RPAD(X.BUYAGENT,15)|| |
                ' ' || TO_CHAR(X.BidPrice,
                    '$99,999,99')|| |
                ' ' || TO_CHAR(X.AskingPrice,
                    '$99,999,99')
            );
        END LOOP;
    ELSE IF v_Count > 0 THEN
        FOR X IN cv_OneAgent LOOP
            DBMS_OUTPUT.PUT_LINE(
                ' ' || X.ListingID|| |
                ' ' || RPAD(X.LISTAGENT,15)|| |
                ' ' || RPAD(X.BUYAGENT,15)|| |

```

```

    ' '||TO_CHAR(X.BidPrice,
'$99,999,99')|| |
    ' '||TO_CHAR(X.AskingPrice,
'$99,999,99')
);
END LOOP;
ELSE
    DBMS_OUTPUT.PUT_LINE('Invalid Agent
Name');
END IF;

```

3. Введите приведенные ниже строки определения процедуры. Последняя строка содержит косую черту. Не нажимайте в конце этой строки клавишу <Enter>.

```

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END MyListings;
/

```

4. Сохраните код в файле, а затем скопируйте его в SQL\*Plus.
5. Работая с SQL\*Plus, нажмите клавишу <Enter>, чтобы скомпилировать код. Если при компиляции возникнут ошибки, вызовите команду SHOW ERRORS, чтобы определить, в каких строках содержится неверный код. Исправьте ошибки в редакторе Блокнот и повторите действия, описанные в пп. 4 и 5.
6. Протестируйте вновь созданную процедуру MyListings, для чего введите и выполните в SQL\*Plus следующие команды (как обычно подставьте ваше имя и требуемый путь). Обратите внимание, что размер буфера надо увеличить с помощью параметра SIZE.

```

SPOOL <путь>\<имя>Ch07Prob1Part2.txt
SET SERVEROUTPUT ON SIZE 123456
CALL MyListings('Carling');
CALL MyListings('Quinn');
CALL MyListings('all');
SPOOL OFF

```

7. Удалите процедуру MyListings, выполнив в SQL\*Plus следующую команду:
- ```
DROP PROCEDURE MyListings;
```
8. Найдите файл MyListings и буферный файл, созданный в п. 6. Откройте Блокнот или другой текстовый редактор и добавьте ваше имя в первой строке каждого файла. Выведите оба файла на печать. Закройте SQL\*Plus и используемый вами текстовый редактор.

## 2. Coffee Merchant

Создайте описанные ниже функцию и процедуру. Следуйте приведенным инструкциям по проверке созданного вами кода. Сохраните исходные тексты функции и процедуры в файлах. Выведите файлы с кодом и результаты их выполнения с тем, чтобы

вы могли показать их инструктору. Не забудьте указать в начале каждого документа ваше имя и другие идентификационные данные.

Загрузите SQL\*Plus, зарегистрируйтесь в Oracle и выполните файл сценария `BuildCoffee.sql`, который можно найти в каталоге `CoffeeMerchant`. Перед выполнением сценария все таблицы переводятся в исходное состояние.

Часть 1. Создайте функцию с именем `SalesTax`, которая возвращает налоговою ставку для штата. Название штата передается функции в качестве параметра. Наличие данной функции упростит определение налоговой ставки при вычислении суммы налога. Пример вызова данной функции приведен ниже.

```
SELECT SalesTax('CA')*Price*Quantity FROM OrderLines;
```

В данном случае в качестве входного параметра передается название штата Калифорния, а возвращаемое значение — это налоговая ставка, используемая для вычисления суммы налогов.

1. Введите в редакторе Блокнот представленный ниже код функции, а затем скопируйте его в SQL\*Plus. Не забывайте, что после косой черты в последней строке файла не надо нажимать клавишу <Enter>. В первой строке подставьте свое имя и фамилию, но два знака -- оставьте без изменений.

```
-- <Имя и фамилия>
CREATE OR REPLACE FUNCTION
SalesTax(StateInParm IN VARCHAR2)
    RETURN NUMBER AS
    v_Count NUMBER;
    v_TaxRate NUMBER(4, 4) := -0.05;
BEGIN
    /* Проверка корректности названия штата */
    SELECT COUNT(*)
    INTO v_Count
    FROM States WHERE UPPER(StateID) =
    UPPER(StateInParm)
        OR UPPER(StateName) =
    UPPER(StateInParm);
    IF v_Count > 0 THEN
        SELECT TaxRate
        INTO v_TaxRate
        FROM States
        WHERE UPPER(StateID) =
        UPPER(StateInParm)
            OR UPPER(StateName) =
        UPPER(StateInParm);
    END IF;
    RETURN v_TaxRate;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END SalesTax;
/
```

2. Для компиляции функции нажмите клавишу <Enter>. Если вы получите сообщения о синтаксических ошибках, внесите необходимые исправления. Информацию об ошибках можно получить с помощью команды SHOW ERRORS. Сохраните изменения в текстовом файле средствами редактора Блокнот. Скопируйте исправленный код в SQL\*Plus и повторно скомпилируйте его.

Часть 2. Протестируйте функцию. Введите представленный ниже код в SQL\*Plus. Подставьте вместо <путь> требуемый путь, а вместо <имя> — свое имя. Выражение <имя> содержится в данном фрагменте кода дважды.

3. Введите и выполните следующий код, чтобы проверить функцию SalesTax:

```
SPOOL <путь> \ <имя>Ch07Prob2Part2.txt
-- <имя>
SET SERVEROUTPUT ON
SELECT StateName AS "State",
       TO_CHAR(SalesTax(StateID)
*100,'90.99')|| '%' AS "Tax (%)"
FROM States
WHERE Population/LandArea < 20
ORDER BY 2;
SELECT SalesTax('CaLiFoRnIa')*10*24.95 FROM
DUAL;
SELECT SalesTax('NoSuchState') FROM DUAL;
SPOOL OFF
```

4. Выведите на печать файл, полученный на предыдущем шаге.

Часть 3. Теперь вам следует создать процедуру с именем CustomerOrder, которой передаются два параметра — имя и фамилия потребителя. Эта процедура извлекает все заказы, сделанные данным клиентом, и объединяет четыре таблицы. Как обычно, аккуратно введите текст и скопируйте его в SQL\*Plus. Создав процедуру, проверьте ее, попытавшись получить информацию о заказах, сделанных Джерри Петерсоном (Jerry Peterson).

5. Введите в редакторе Блокнот приведенный ниже код, а затем скопируйте его в SQL\*Plus.

```
CREATE OR REPLACE PROCEDURE CustomerOrder(
    Fname IN VARCHAR2,
    Lname IN VARCHAR2) AS
v_Count INTEGER;
CURSOR cv_GetOrd IS
    SELECT OrderID, OrderDate, Name,
           Quantity, Price, Discount,
           TO_CHAR(Price*Quantity*(1-
Discount), '$9,999.99') AS
ExtPrice,
           TO_CHAR(Price*Quantity*(1-
Discount)*SalesTax(State),
```

```

'99.99') AS Tax
FROM Consumers a
INNER JOIN Orders      b USING
(ConsumerID)
INNER JOIN OrderLines c USING
(OrderID)
INNER JOIN Inventory   d USING
(InventoryID)
WHERE LOWER.FirstName = LOWER(Fname)
AND LOWER(LastName) = LOWER(Lname)
ORDER BY OrderID, LineItem;
BEGIN
/* Проверить существует ли потребитель */
SELECT COUNT(*) INTO v_Count FROM
Consumers
WHERE LOWER.FirstName = LOWER(Fname) AND
LOWER(LastName) = LOWER(Lname);
CASE
WHEN v_Count > 0 THEN
FOR X IN cv_GetOrd LOOP
DBMS_OUTPUT.PUT_LINE(
' | '||X.OrderID|' | '|X.OrderDate| |
' | '||RPAD(X.Name,20)| |
' | '|TO_CHAR(X.Quantity,'999')| |
' | '|TO_CHAR(X.Price,'$9999.99')| |
' | '|X.ExtPrice| ' '||X.Tax
);
END LOOP;
ELSE
DBMS_OUTPUT.PUT_LINE ('No orders for that customer');
END CASE;
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE (SQLERRM);
END CustomerOrder;
/

```

6. Для компиляции кода нажмите клавишу <Enter>. Если будут выявлены ошибки, найдите и исправьте их. Повторяйте компиляцию до тех пор, пока она не осуществляется успешно.

Часть 4. Протестируйте процедуру CustomerOrder. Создайте буферный файл и выведите его после выполнения кода.

7. Если код скомпилирован успешно, проверьте его, выполнив следующие команды в SQL\*Plus. Как обычно, замените <имя> (это выражение встречается дважды) и <путь>.

```

SPOOL <путь>\ <имя>Ch07Prob2Part4.txt
-- <имя>
SET SERVEROUTPUT ON
CALL CustomerOrder('Jerry','Peterson');
CALL CustomerOrder('Abe','Lincoln');
SPOOL OFF

```

8. Удалите созданные вами процедуру и функцию. Для этого используйте следующие команды:

```
DROP FUNCTION SalesTax;
DROP PROCEDURE CustomerOrder;
```

9. Закройте Блокнот и SQL\*Plus.

### 3. Rowing Ventures

Создайте описанную ниже процедуру и выполните предлагаемые действия для ее проверки и отображения информации о ней. Сохраните данные в файле и выведите на печать, чтобы показать инструктору. Не забудьте указать свое имя и другие идентификационные сведения в заголовке каждого документа.

Загрузите SQL\*Plus, зарегистрируйтесь в Oracle и выполните файл BuildRowing.sql, который можно найти в каталоге RowingVentures. Перед выполнением сценария все таблицы переводятся в исходное состояние.

Часть 1. Создайте именованную процедуру RaceInfo, для которой предусмотрены два входных параметра: BoatCat и BowNo типа VARCHAR2. Данная процедура объединяет таблицы Boat, BoatCrew и Person и извлекает следующие столбцы: BoatCategory, BowNumber, Position, LastName и FirstName. Извлеченные строки соответствуют конкретной категории судна (BoatCat) и номеру (BowNo), которые заданы в качестве параметров. Если вы введете значение параметра BoatCat, равное all, а в качестве BowNo укажете строку any, курсором будут извлечены все строки.

Отсортируйте строки по BoatCategory, BowNumber и Position в порядке возрастания значений. Если возникнет нулевое значение Position, преобразуйте его в Sox (сделать это надо в курсоре, а не в цикле FOR). Оставьте остальные значения Position в виде трехзначных цифр. (Для выравнивания чисел по правому краю используйте TO\_CHAR и соответствующую модель формата.)

Для обработки строк примените цикл FOR курсора; в цикл поместите конструкцию IF/THEN/ELSIF, которая позволила бы определить, надо ли отображать все строки или только те, которые определены входными параметрами. Для вывода строк применяйте процедуру PUT\_LINE пакета DBMS\_OUTPUT. Скомпилируйте процедуру.

Часть 2. Протестируйте процедуру RaceInfo. Для этого после успешной компиляции выполните приведенные ниже выражения SQL\*Plus. Замените <путь> значением пути, а <имя> вашим именем. Затем выведите файл на печать.

```
CLEAR SCREEN
SPOOL <путь>\ <имя>Ch07Problem3Part1.txt
PROMPT <имя>
SET PAGESIZE 100
SET SERVEROUTPUT ON
CALL RaceInfo('ALL', 'X');
CALL RaceInfo('Crew-8', '3');
SPOOL OFF
```

Часть 3. Поместите в файл перечисленные ниже данные так, чтобы по окончании работы вы могли вывести результаты.

- Исходный код процедуры `RaceInfo`. Для его вывода применяется механизм представления базы данных.
- Значения `object_name` и `object_type`, сформированные посредством использованного ранее представления базы.

Удалите процедуру `RaceInfo`.

Используйте команду `COLUMN` для установки размеров столбцов, равных 20 и 12 символам. Включите вашу фамилию в начало файла, используя для этого текстовый редактор, и выведите файл.

## 4. Broadcloth Clothing

Создайте описанную ниже функцию. Загрузите SQL\*Plus, зарегистрируйтесь в Oracle и выполните файл `BuildClothing.sql`, который можно найти в каталоге `Broadcloth` с файлами данных для главы 7. Перед выполнением сценария все таблицы переводятся в исходное состояние.

Создайте функцию `ItemTC`, которой при вызове передаются два параметра. Первый параметр — это символ С или S (он может быть как нижнего, так и верхнего регистра). Второй параметр — четырехзначное число. Функция `ItemTC` реализует неявный запрос на основе таблицы `OrderItem` базы данных `Broadcloth`. Она либо подсчитывает, либо суммирует значения всех пунктов таблицы `OrderItem` для соответствующего `ItemID`. Функция отвечает на вопрос “Сколько заказов содержат номер `ItemID`?” или “Каково суммарное значение (`OrderQuantity`, умноженное на `SalePrice`) для заказов с данным `ItemID`?”. Функция `ItemTC` записывается следующим образом:

```
ItemTC(<character>, <itemID_number>)
```

где символ — это С в случае, если вам надо подсчитать пункты, для которых `ItemID` соответствует второму параметру, либо S, если вам надо подсчитать сумму произведений `OrderQuantity` на `SalePrice` для заданного значения `ItemID`. `ItemID`, `OrderQuantity` и `SalePrice` — это строки таблицы `OrderItem`.

Часть 1. Создайте функцию `ItemTC`, которая возвращала бы число, используя неявный запрос, и раздел обработки исключений которой обеспечивал бы отображение сообщения в случае возникновения ошибки. Когда работа будет завершена, включите вашу фамилию в начало файла с текстом функции, оформив ее в виде комментариев. Выполните на печать файл, содержащий код функции.

Часть 2. Протестируйте функцию, включив буферный файл с именем <имя>Ch07Prob4Part2.txt и выполнив приведенные ниже выражения, ссылающиеся на вашу функцию. Второе выражение SELECT генерирует сообщение об ошибке. Поскольку пунктов с таким номером нет, оно будет выглядеть следующим образом:

```
ORA-06503: PL/SQL: Function returned without value
```

В данном случае ошибка внесена намеренно. Первое и третье выражения SELECT должны вернуть реальные данные.

```
-- <имя>
SET SERVEROUTPUT ON
SELECT ItemTC('S',1017), ItemTC('C',1017)
FROM DUAL;
SELECT ItemTC('C',12345)
FROM DUAL;
SELECT ItemTC('c',1005), ItemTC('s',1005)
FROM DUAL;
```

Закройте буферный файл и выведите его на печать. Для третьего выражения SELECT функция должна возвращать значения 2 и 40530.5.

Часть 3. Модернизируйте функцию так, чтобы при получении недопустимого ItemID она возвращала следующее сообщение:

```
Order item with ItemID xxxxxx was not found.
```

Это сообщение должно выводиться в дополнение к сообщению, генерируемому системой Oracle, в том случае, когда ItemID не соответствует ни одному из ItemID в таблице OrderItem. (Введите недопустимое значение ItemID и отобразится вместо xxxxxx в приведенном выше сообщении. Желательно проверить, есть ли хотя бы одна запись в таблице OrderItem.) Выведите на печать модернизированный код функции. Выведите на печать буферный файл:

```
-- <имя>
SET SERVEROUTPUT ON
SELECT ItemTC('C',12345) FROM DUAL;
DROP FUNCTION ItemTC;
```

Сохраните все файлы, созданные в текстовом редакторе, и закройте SQL\*Plus.



# ГЛАВА 8

## Использование Forms Builder

**В этой главе...**

- Три основных типа форм
- Структура форм Oracle
- Создание основной формы с помощью Forms Builder
- Использование формы для фильтрации данных простых запросов
- Модификация форм
- Создание табличной и подчиненной формы

### Общие сведения о формах

Реляционные базы данных разрабатываются для того, чтобы обеспечить эффективное хранение информации. С этой целью посредством столбцов, выполняющих функции ключей, создаются таблицы и формируются ссылки между ними. Такой подход обеспечивает эффективность хранения данных, но при этом пользователям бывает трудно понять структуру базы. Например, несмотря на то, что агент работает с базой данных Redwood Realty, вряд ли он сможет записать информацию непосредственно в таблицу `CustAgentList`. Если мы подробно рассмотрим структуру таблицы, то увидим, что она содержит ключевые значения `CustomerID`, `AgentID` и `ListingID`. Нельзя требовать от пользователя, чтобы он запомнил эти значения. Также подавляющее большинство пользователей не справятся с задачей формирования запросов, позволяющих извлечь из базы и отформатировать требуемые им данные. Решить эту проблему можно путем создания форм.

Формы базы данных — это интерактивные приложения, используемые для ввода и редактирования данных, которые были извлечены из таблиц или предназначены для записи в таблицы. Часто функции форм ограничиваются отображением данных в ответ на определенные действия пользователя. Формы выполняются в среде, позволяющей создавать сложные приложения, реагирующие на ввод информации пользователем, осуществляющие вычисления и обменивающиеся данными с различными таблицами базы. Первые формы выполнялись на тех локальных машинах, на которых работали пользователи. Современные формы Oracle представляют собой Web-приложения. Пользователи открывают формы посредством Web-браузеров, поддерживающих Java.

## Основные типы форм

Для создания многих приложений, предполагающих работу с базами данных, достаточно использовать три типа форм: основную форму, отображающую в каждый момент времени данные из одной строки; табличную форму, которая выводит на одной странице несколько строк таблицы; и комбинированную форму, представляющую собой комбинацию двух предыдущих типов (она отображает первичные данные и подчиненную форму, представляющую несколько строк). Зная, как строить и модифицировать формы этих типов, можно создать практически любое приложение. При создании форм важно помнить, что они нужны лишь для того, чтобы упростить пользователю работу с информацией. Поэтому надо выбирать форму такого типа, которая наилучшим образом соответствовала бы потребностям пользователей.

Одним из самых простых типов форм является основная форма, которая выводит на экран данные из одной строки. На рис. 8.1 показана простая форма Agent для базы данных Redwood Realty. На экране отображаются все данные для одного агента. Когда пользователь, имеющий право работать с данными об агенте, внесет изменения и щелкнет на кнопке Save, информация будет записана в таблицу Agent базы данных. Несмотря на простоту формы, вы можете как угодно изменять расположение компонентов. Например, можно поместить метки и поля редактирования на противоположных сторонах окна. Конечно же, конфигурация формы должна отражать потребности пользователя. Для того чтобы сделать форму более удобной для восприятия, можно использовать цвет и изображения. Допустимо также создавать программный код, который будет реагировать на события, связанные с формой. Например, вы, возможно, захотите, чтобы при добавлении нового агента автоматически генерировался идентификационный номер.

Табличная форма проще основной. На рис. 8.2 показана форма License Status, которая упрощает редактирование списка, определяющего условия лицензирования. Преимущество вывода данных в виде таблицы состоит в том, что пользователь видит сразу несколько записей. Если пользователю надо сравнить строки таблицы или ввести небольшой список, лучше всего предоставить ему именно табличную форму. Ос-

The screenshot shows a Windows application window titled "Agent". Inside, there is a form with various input fields for agent details:

Field	Value
Agent ID	10041
First Name	Kai
Last Name	Marcoux
Hire Date	03-OCT-1996
Birthdate	12-DEC-1970
Gender	Male
Work Phone	(707) 555-0361
Cell Phone	(707) 555-5313
Home Phone	(707) 555-7185
Title	Salesperson
Tax ID	868-26-8846
License ID	107157413
License Date	30-JAN-1997
License Expire	01-FEB-2000
License Status	1001

**Рис. 8.1.** Форма Agent базы Redwood Realty отображает в каждый момент времени сведения об одном агенте

новным недостатком является низкая степень контроля за расположением элементов. Вы можете задавать ширину столбцов или число отображаемых строк либо изменять шрифт. Форму данного типа чаще всего применяют лишь для таблиц с ограниченным количеством столбцов.

В бизнес-приложениях часто приходится применять формы, более сложные, чем были описаны выше. Например, форма для заказов содержит как минимум два основных раздела: в одном отображаются доступные товары и информация о потребителе, а в другом — список товаров, заказанных этим пользователем. На рис. 8.3 показана форма, в которой представлена информация об агенте и данные о его контактах с потребителями. Отношение “один ко многим” между агентом и клиентами поддерживается посредством подчиненной табличной формы, которая отображает для каждого агента несколько контактов.

Подход, предполагающий использование основной и подчиненной форм, обеспечивает большую гибкость в работе с основными данными (в рассматриваемом здесь случае это информация об агентах). По мере необходимости вы можете изменить расположение столбцов, а также добавить строки для представления промежуточных результатов. Заметьте, что подчиненная табличная форма связывается с данными, отображаемыми в основной форме. Когда пользователь выберет нового агента, список будет обновлен и на экране отобразится информация о контактах того агента, который представлен в основной форме.

WINDOW1

License Status	
License Status ID	Status Text
1001	Licensed
1002	Licensed NBA
1003	Canceled Officer
1004	Deceased
1005	Expired
1006	Government Service
1007	Military Service
1008	Conditional Suspension
1009	Restricted
1010	Revoked
1011	Flag Suspended
1012	Voided
1013	Withheld Denied
1014	17520 FC Suspended
1015	11350.6 W and I Suspended
1016	Surrendered

Рис. 8.2. Форма License Status, отображающая несколько строк таблицы

WINDOW1

Agents	
Agent ID	10041
Title	Salesperson
First Name	Kai
Last Name	Marcoux
Work Phone	(707) 555-0361

Contact								
Customer ID	Listing ID	Date	Reason	Bid Price	Commission	Last Name	First Name	
30176	15454	16-SEP-2006	Sell		.05	Champion	George	↑
30243	15815	03-NOV-2006	Sell		.06	Rycus	Richard	↓
25983	15373	13-NOV-2006	Buy	111555	.052	Warnecke	Rick	↑
25983	15373	12-NOV-2006	Buy	111298	0	Warnecke	Rick	↓
25983	15373	24-OCT-2006	Buy	97927	0	Warnecke	Rick	↑
32671	15035	01-JUL-2006	Sell		.06	Kaliszewsk	Steve	↓
30784	15161	26-JUL-2006	Sell		.06	Waines	Frank	↑
31059	15335	24-AUG-2006	Sell		.06	Palkovic	Peter	↓
31693	14992	23-JUN-2006	Sell		.06	Smith	Lowell	↑
31710	16143	21-DEC-2006	Sell		.06	Filoteo	Romeo	↓

Рис. 8.3. Форма Agent Contacts, сочетающая основную и табличную формы

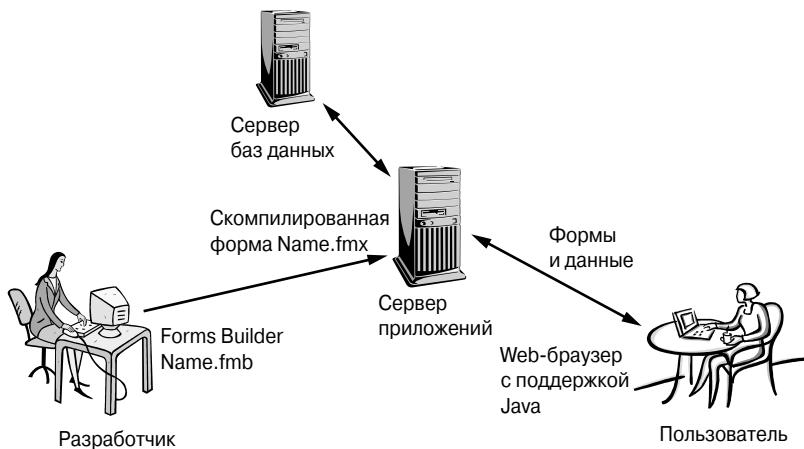
## Архитектура среды для поддержки форм

Создавая приложение, использующее формы, важно представлять себе особенности среды, в которой будут выполняться формы. В ранних версиях Oracle для форм применялась архитектура клиент/сервер, причем формы выполнялись на клиентских машинах и запрашивали данные с сервера. Начиная с версии 9i формы Oracle представляют собой Web-приложения на основе Java. Разработчику надо учитывать сильные и слабые стороны такого подхода.

Преимущество форм на основе Web состоит в том, что программные и аппаратные средства на стороне клиента не являются критичными для работы приложения. Пользователь должен иметь лишь компьютер с Web-браузером, который должен поддерживать Java. В настоящее время Web-браузеры установлены практически на каждом компьютере. Однако в некоторых из них может отсутствовать интерпретатор Java. К счастью, все средства, необходимые для поддержки Java, можно скопировать с сервера Sun ([www.java.com](http://www.java.com)). Версии системы поддержки Java существуют для подавляющего большинства платформ. Настроить клиентскую машину для использования в пределах компании достаточно просто. С другой стороны, невозможно потребовать, чтобы каждый независимый пользователь установил у себя средства поддержки Java, поэтому подобные формы плохо подходят для систем электронной коммерции, рассчитанных на широкую публику.

**Совет.** Инструмент Forms Builder проще будет использовать в том случае, если вы создадите в среде Windows специальную переменную окружения. Щелкните правой кнопкой мыши на пиктограмме My Computer, в окне Properties выберите вкладку Advanced и щелкните на кнопке Environment Variables. Затем щелкните на кнопке New и введите следующие данные: name=FORMS90\_HIDE\_OBR\_PARAMS, value=False.

Как показано на рис. 8.4, формы в системе Oracle создает разработчик, пользуясь инструментом Forms Builder, который входит в состав Developer Suite. Этот инструмент надо установить на той машине, за которой работает разработчик; на сервере баз данных он не нужен. Каждая создаваемая форма хранится в отдельном файле, поэтому для проекта надо создать отдельный каталог, в котором будут находиться все соответствующие формы. По окончании работы вы скомпилируете формы и скопируете результат на сервер приложений, который обеспечит доставку форм и работу с ними посредством Web-интерфейса. В процессе работы форма устанавливает связь с базой данных. В общем случае база данных может находиться на другом компьютере. Если размеры системы невелики, то один сервер может поддерживать и базу данных, и приложения. На пользовательских машинах надо установить Java-приложение Initiator, предоставляемое Oracle. Пользователю (или разработчику) будет предложено инсталлировать это приложение, когда он впервые попытается открыть форму Ora-



**Рис. 8.4.** Архитектура системы на базе Web

cle. Процедура установки очень проста, но она займет несколько минут. Убедитесь, что система безопасности браузера настроена так, что копирование и инсталляция приложений разрешены.

Для того чтобы создать форму, выполните следующие действия.

1. Установите на своем компьютере Developer Suite. Если вы работаете с базой данных, установленной на другой машине, убедитесь, что вы можете обращаться к ней. При необходимости используйте приложение Net Manager для установления локального соединения.
2. Создайте форму с помощью Forms Builder. Этот же инструмент можно использовать для модификации форм. Сохраните результат в папке, доступной для разработчиков, и создайте резервную копию.
3. Протестируйте формы на своем компьютере, запустив приложение OC4J (Oracle Containers for Java). Активизировать его можно посредством меню Start системы Windows; путь к нему имеет такой вид: Oracle Developer/Forms Developer/Start OC4J Instance.
4. Запустите Web-браузер и убедитесь в том, что ваш компьютер включен в список узлов, пользующихся доверием. Для этого надо выбрать в меню браузера Tools⇒Internet Options⇒Security. Откройте или запустите форму и проверьте ее работу. Если возникнет необходимость, установите JInitiator.
5. Перепишите готовую форму на сервер приложений Oracle, сконфигурируйте браузеры и проверьте форму посредством браузера.

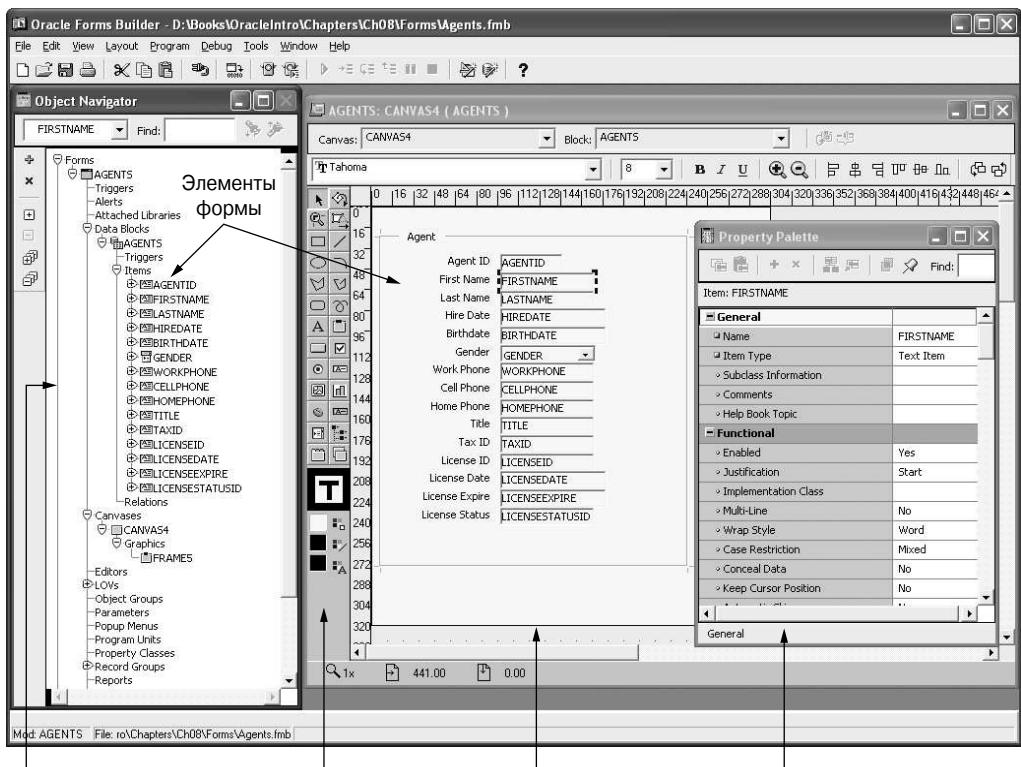


Рис. 8.5. Форма Oracle, представленная в окне Forms Builder

## Структура формы Oracle

Для описания различных элементов форм в Oracle используется специальная терминология. Пользоваться инструментом разработки значительно проще, если вы понимаете общую структуру формы и знаете смысл терминов, которыми она описывается в литературе. На рис. 8.5 показана простая форма, представленная в окне Forms Builder.

Основная область формы, предназначенная для отображения данных, называется **холстом** (canvas). На холсте помещаются **элементы формы**, например текстовые метки и поля редактирования. Они предназначены для отображения и редактирования данных из таблиц. Если вы обратите внимание на набор инструментов, то увидите, что на холсте можно также рисовать линии или размещать изображения. Управлять представлением холста и элементов формы можно путем установки свойств. Обычно палитра свойств скрыта, но ее можно отобразить, щелкнув правой кнопкой мыши на холсте или на конкретном элементе. С помощью палитры можно, например, задавать цвет или шрифт. Некоторые свойства могут быть связаны с меню или панелью инструментов в верхней части формы.

*Объектный навигатор* (панель Object Navigator) представляет элементы формы в виде иерархической структуры. С его помощью можно также отобразить данные для нескольких форм. Объектный навигатор позволяет быстро найти нужный элемент. Блок данных используется для передачи форме данных из базы и их представления. Блок данных создается для каждой таблицы, с которой должна работать форма. В рассматриваемом здесь примере мы используем только таблицу Agents, поэтому блок данных лишь один. Элементы формы связываются со столбцами таблицы. Для большинства элементов можно установить свойства, выбрав элемент либо в форме, либо на панели Object Navigator. Некоторые действия можно выполнить лишь посредством списка, который предоставляется объектным навигатором. Если надо выбрать несколько элементов либо на панели Object Navigator, либо на холсте, это можно сделать, щелкнув на них мышью, одновременно нажав клавишу <Ctrl>. Свойства можно устанавливать одновременно для всех выбранных элементов.

*Обработчик события*, или *триггер*, — это код, который выполняется при возникновении определенного события, связанного с формой. Для формы предусмотрено большое количество событий, и вы можете создавать программы PL/SQL, призванные помогать пользователям в работе и решать возникающие проблемы. Например, включая в состав формы кнопку, надо предусмотреть также фрагмент кода, который будет выполняться при ее активизации. Организовать работу с обработчиками событий помогает объектный навигатор. Для каждого объекта можно отобразить список обработчиков. Как вы увидите в дальнейшем при рассмотрении примеров, чтобы быстро создать триггер, можно щелкнуть правой кнопкой мыши на объекте и выбрать динамический список обработчиков (smart trigger list). В динамическом списке представлены лишь наиболее часто используемые события для текущего объекта. Выбрав опцию Other, можно отобразить все триггеры, доступные для данного объекта. Модель событий позволяет создавать сложные приложения, реагирующие на различные действия пользователя.

## Создание основной формы с помощью Forms Builder

Первое, что нужно сделать, — это удалить и повторно сформировать базу данных.

1. Запустите SQL\*Plus и выполните файл BuildRedwood.sql. Таким образом вы удалите базу данных и вновь создадите ее.

Поскольку вы имеете сведения о формах и представляете себе принципы работы Forms Builder, попробуем создать несложную форму. Можно было бы начать с пустой формы и включить в нее холсты, блоки данных и элементы, однако гораздо эффективнее использовать инструменты типа “мастер”, предоставляемые Oracle. Они ав-

томатически генерируют требуемую форму, которую затем можно модифицировать, например, изменить стили или добавить новые элементы.

## Использование инструментальных средств

Перед тем как запускать Forms Builder, убедитесь, что успешно установлено соединение с базой данных Oracle. База данных может находиться на локальной машине или на удаленном сервере. Если вы сомневаетесь, где находится база, выполните проверку с помощью SQL\*Plus. Приступая к созданию формы, надо представлять себе, как она будет выглядеть. Это может быть форма для одной записи, табличная форма или основная форма, связанная с подчиненной формой. Определите, какие данные должны быть представлены в форме. Еще раз подумайте над назначением формы. Большинство из них предназначено для обновления информации в одной таблице (подчиненная форма, как правило, работает с отдельной таблицей). Возможно, имеет смысл нарисовать эскиз формы и указать таблицы, с которыми она будет работать.

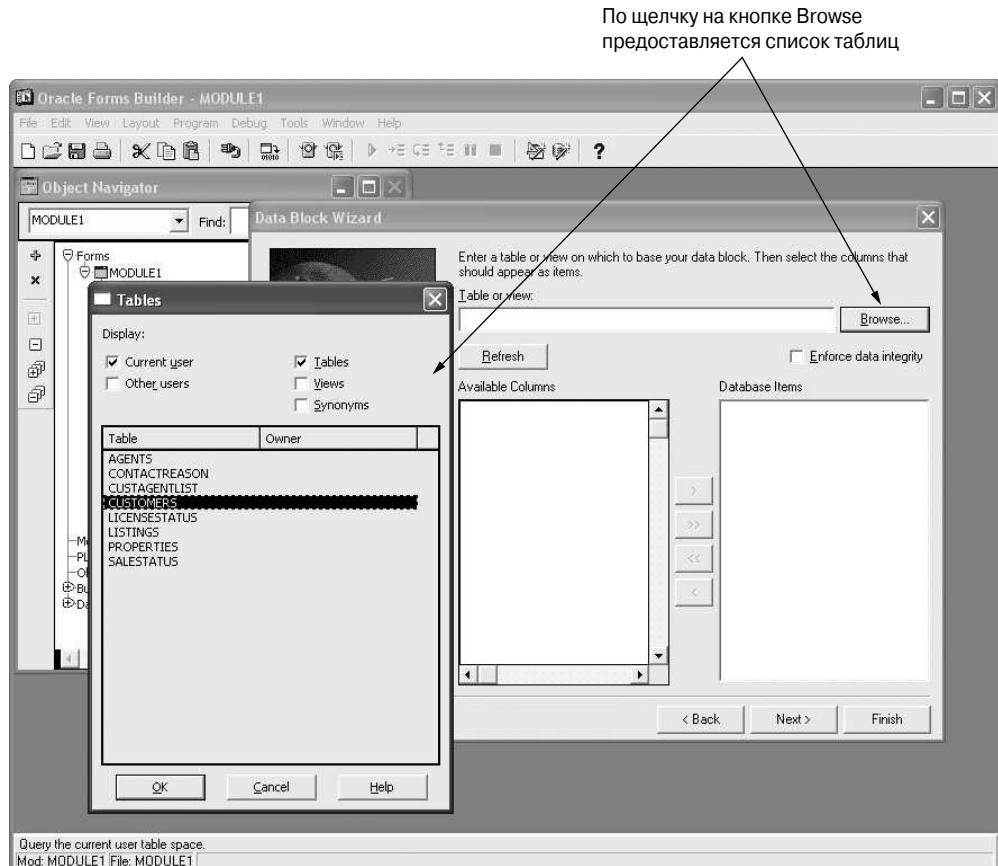
Для создания форм Oracle предоставляет два специальных инструмента: мастер блоков данных и мастер компоновки. Начнем работу с мастером блоков данных. По окончании работы с ним будет автоматически открыт мастер компоновки. Действия, которые необходимо выполнить, перечислены ниже.

1. Запустите Forms Builder из главного меню. Вам будет предоставлена пустая форма.
2. Запустите мастер блоков данных.
3. Выберите таблицы и столбцы в них.
4. Примите опцию по умолчанию, предполагающую запуск мастера компоновки.
5. Выберите элементы формы, задайте размер и заголовки, чтобы получить начальную форму.
6. Уточните расположение элементов, перемещая их и задавая их свойства.
7. Добавьте необходимые элементы, например раскрывающиеся списки.
8. Проверьте форму и подумайте о том, как улучшить ее применимость.

Очевидно, большинство из описанных выше действий можно выполнить несколькими способами. Разработчик, имеющий хотя бы минимальный опыт, без труда справится с данной задачей. Определенное время может занять корректировка расположения элементов и добавление специальных средств.

Действия, касающиеся непосредственно управления мастером блока данных, приведены ниже.

1. Запустите мастер, выбрав пункт меню Tools⇒Data Block Wizard.
2. Ничего не делайте в первом окне.

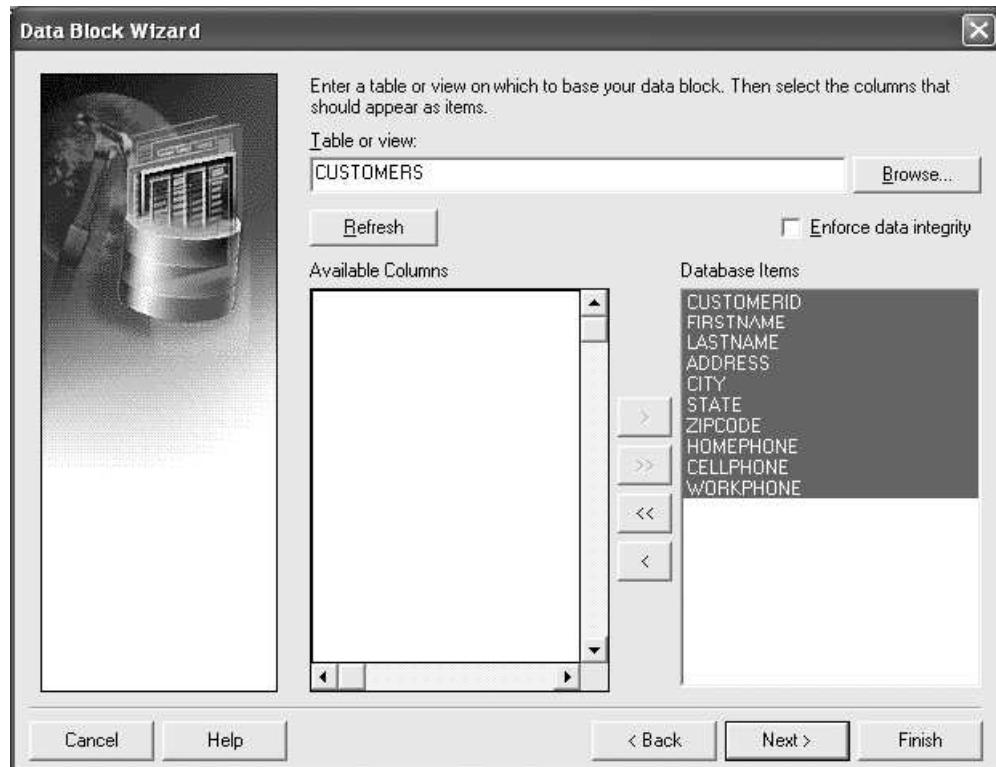


**Рис. 8.6.** Выбор таблицы с помощью мастера блоков данных

3. Примите опцию по умолчанию, которая задает выбор данных Table or View.
4. Щелкните на кнопке Browse, чтобы установить соединение с базой данных и выбрать таблицы.
5. Для данного примера выберите таблицу Customers.
6. Щелкните на двойной стрелке (»), чтобы выбрать для формы все столбцы.

На рис. 8.6 показан список таблиц. Выбрав таблицу Customers, вы получите в левом поле список всех столбцов. Нам надо, чтобы все столбцы присутствовали в форме Customer, поэтому следует выбрать их и переместить в правую область (рис. 8.7).

Перед завершением работы с мастером блоков данных вам будет предложено задать имя блока. Как правило, ему присваивают имя, совпадающее с именем таблицы.



**Рис. 8.7.** Выбор столбцов таблицы с помощью мастера блоков данных

7. Щелкните на кнопке **Next**, введите имя блока данных *Customers* и щелкните на кнопке **Finish**. Таким образом вы примете опцию по умолчанию, предполагающую запуск мастера компоновки. При желании вы сможете впоследствии запустить мастер компоновки вручную, выбрав пункт меню *Tools⇒Layout Wizard*.

Обычно мастер компоновки отображает начальную страницу. Ее надо пропустить, а для того, чтобы она не отображалась в дальнейшем, сбросьте флажок опции. Основные действия с мастером компоновки сводятся к следующему.

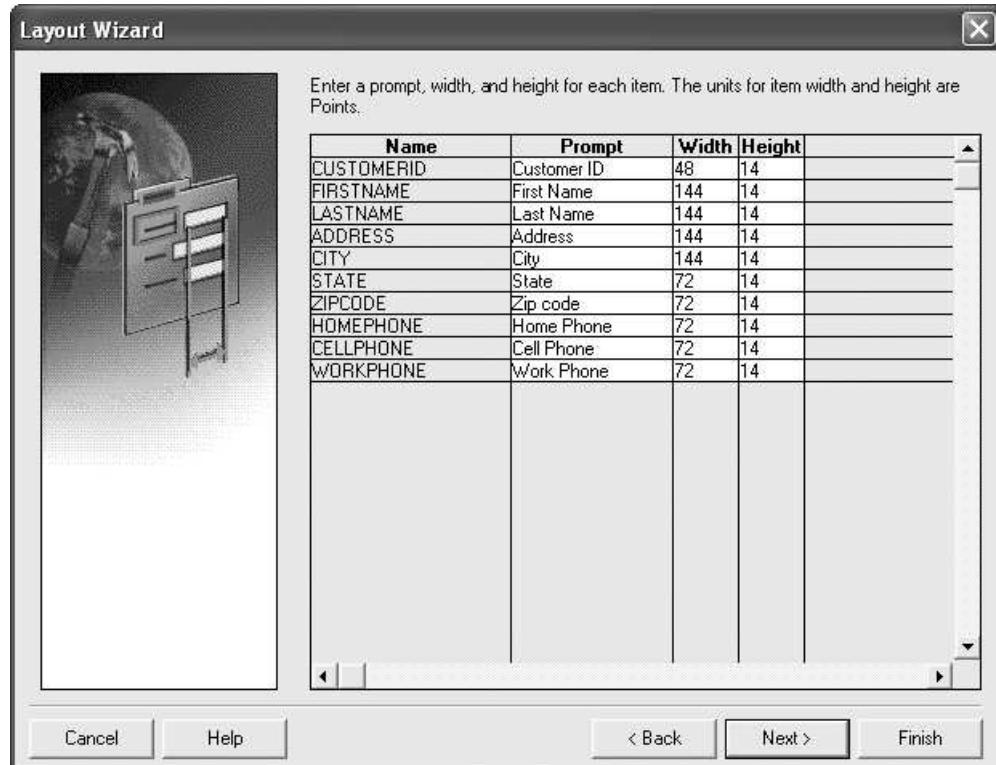
1. Примите опцию *New Canvas* и щелкните на кнопке **Next**.
2. Щелкните на кнопке **»**, чтобы выбрать все столбцы для отображения на холсте. Щелкните на кнопке **Next**.
3. Установите ширину столбцов. Для большинства из них желательно выбрать ширину 72, что соответствует приблизительно одному дюйму. Для элементов, отображающих больший объем данных, например *Address*, можно выбрать значение, равное 144. Щелкните на кнопке **Next**.

4. Выберите тип формы (*Form* или *Tabular*) и щелкните на кнопке *Next*.
5. Установите число записей, которые должны отображаться на экране. Для стандартного типа формы примите значения по умолчанию. В качестве имени фрейма введите *Customers*, а затем щелкните на кнопке *Next*. В последнем окне щелкните на кнопке *Finish*.
6. Скорректируйте внешний вид и расположение элементов, используя редактор компоновки.

Создавая новую форму, вы, вероятно, захотите включить в нее новый холст. Как вы помните, холст — это область формы для отображения элементов и другой информации. В большинстве случаев приходится выбирать все столбцы блока данных и помещать их на холст так, чтобы они могли отображаться и были доступны для редактирования. Подобно тому, как вы делали это в мастере блока данных, щелкните на двойной стрелке », чтобы переместить все столбцы в правую область.

Достаточно много времени занимает упорядочение пояснительных надписей и выбор ширины столбцов. На рис. 8.8 показаны значения, которые можно использовать для форм. Пояснительные надписи представляют собой текстовые метки, отображаемые в составе формы. Выровнять их можно, дополняя пробелами. Правильно выбрать ширину элементов несколько сложнее, так как необходимо угадать размер каждого поля редактирования. Утешает лишь то, что принятое решение можно в любой момент изменить. На данном этапе устанавливать размеры надо лишь потому, что первоначальные значения обычно слишком велики и портят внешний вид формы. Выполнив определенную подготовительную работу, можно сэкономить время и усилия в дальнейшем. Размеры большинства элементов задаются в пунктах. Пункт — это единица измерения, используемая в типографском деле (72 пункта составляют один дюйм). Очевидно, что дюйм на бумаге и дюйм на экране могут отличаться друг от друга. Для того чтобы оценить, какое количество символов может быть отображено в поле редактирования, надо помнить, что при использовании шрифта размером 12 пунктов в одном дюйме помещается 12 цифр. Благодаря тому что некоторые буквы имеют небольшую ширину, текст, как правило, занимает меньше места, чем цифры.

Мастер компоновки предлагает выбрать тип формы "Form- или "Tabular-. Различие между этими типами состоит в том, что основная форма обычно используется для представления одной строки; при этом значения полей отображаются посредством элементов формы. Табличная форма выводит данные в виде таблицы, отображая в общем случае несколько строк и столбцов. Для формы *Customer* следует принять значение "Form-", предлагаемое по умолчанию. Следующее окно мастера, показанное на рис. 8.9, демонстрирует, что если вы выбрали табличную форму, вам придется ввести число строк, которые должны отображаться в форме. Для формы *Customer* можно принять значение по умолчанию, которое задает вывод одной строки.



**Рис. 8.8.** Установка пояснительных надписей и выбор ширины полей с помощью мастера компоновки

Остальные два элемента формы — заголовок окна и полоса прокрутки — также важны. Убедитесь, что окну присвоено содержательное имя. Его можно изменить в дальнейшем, но выбрать это имя надо так, чтобы оно давало представление о назначении формы. В данном случае мы выбираем заголовок *Customers*. С полосой прокрутки дело обстоит несколько сложнее. Она позволяет перемещаться вперед и назад по строкам с данными. Как правило, полоса прокрутки бывает необходима в табличных формах, поскольку записи, отображаемые в них, могут не поместиться в окне. В формах, отображающих одну запись, без полосы прокрутки чаще всего можно обойтись, но вначале все-таки следует включить ее. Дело в том, что на данном этапе поместить ее в состав формы легко, еще легче удалить впоследствии, но с определенного этапа работы над формой включение в нее полосы прокрутки становится достаточно сложной задачей. Поэтому, если вы сомневаетесь в том, нужна ли для формы полоса прокрутки, следует включить ее. Этой рекомендации мы и последуем при работе над формой *Customer*. Последнее окно мастера сообщает вам о том, что работа окончена и что требуемая форма будет сгенерирована щелчком на кнопке *Finish*.

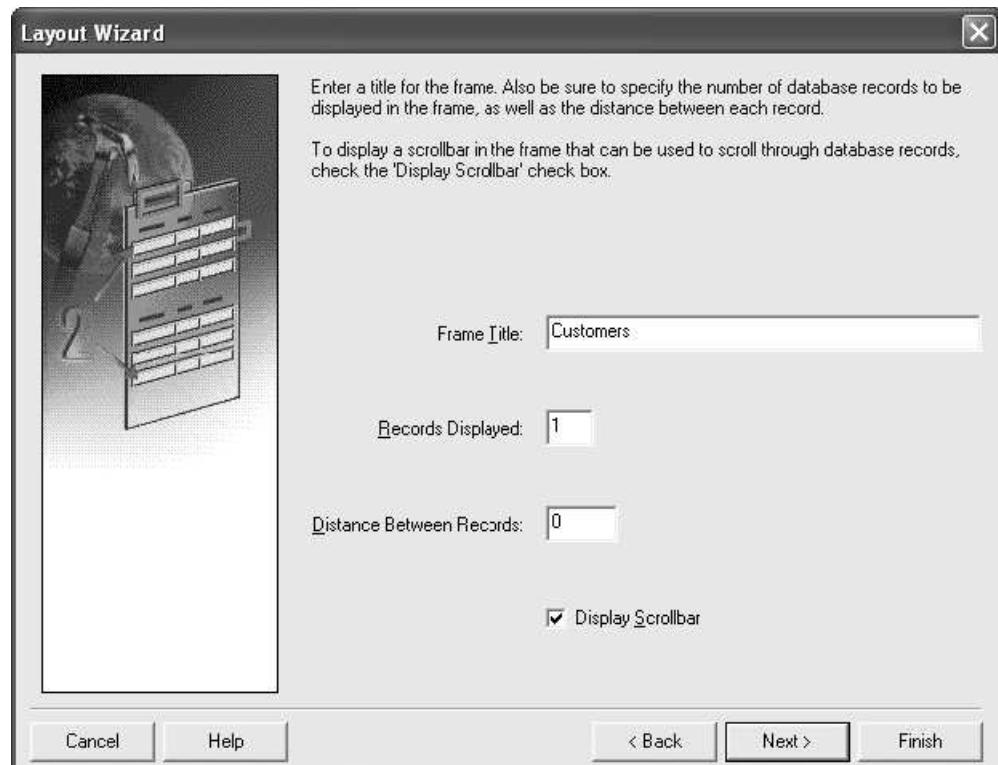
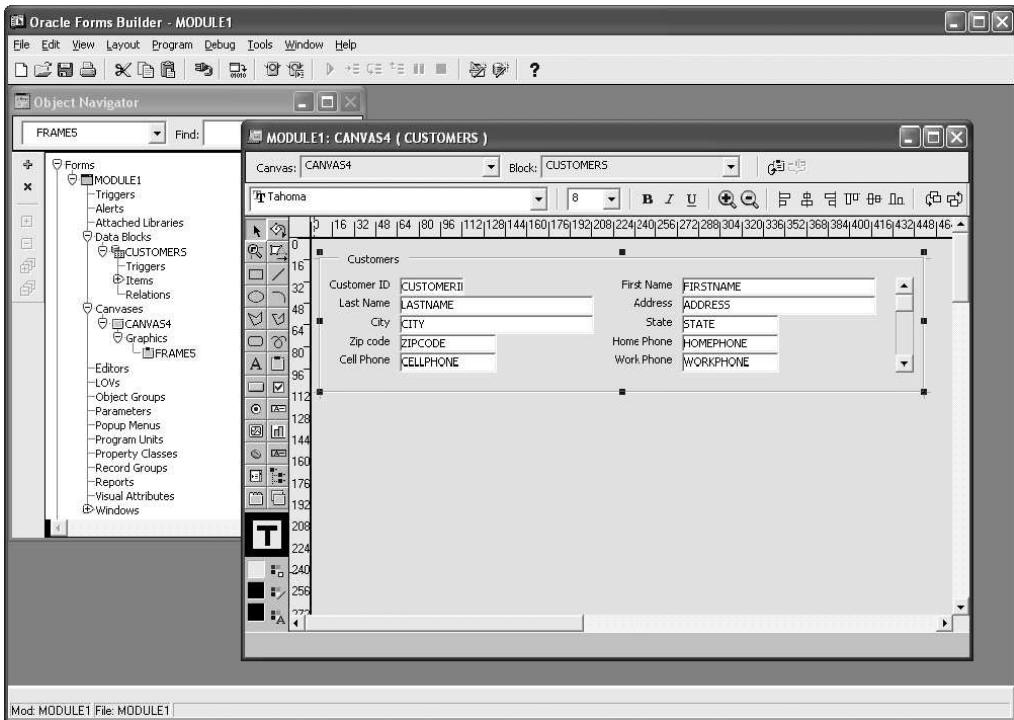


Рис. 8.9. Выбор числа записей с помощью мастера компоновки

На рис. 8.10 показано первоначальное размещение элементов формы *Customer*, созданной двумя инструментами типа “мастер”. Форму можно использовать в неизменном виде, но, скорее всего, вы захотите либо разместить все элементы в один столбец, либо расположить их на холсте в требуемом вам порядке. Выстроить все пункты в один столбец несложно. Щелкните на обрамлении вокруг элементов, тем самым выделив их. Затем перетащите правый нижний угол обрамления вниз и влево. Все элементы в составе фрейма автоматически изменят свое расположение в соответствии с новыми размерами доступной для них области. В одних случаях такое автоматическое упорядочение дает нужные результаты, в других — нет. Для фрейма определено свойство *Update Layout*, которое по умолчанию имеет значение “*Automatically*”. Эта установка указывает на то, что местонахождение отдельных элементов управляет окном. Если вы хотите, чтобы элементы разместились так, как это надо вам, измените значение указанного выше свойства на “*Manually*”, и вы получите возможность перемещать любой элемент в пределах окна. Пока мы ограничимся созданием одного столбца, включающего все пункты. Пока же сохраняйте форму; если вам не понравятся результаты вашей работы, вы сможете вернуться к одному из промежуточных вариантов.

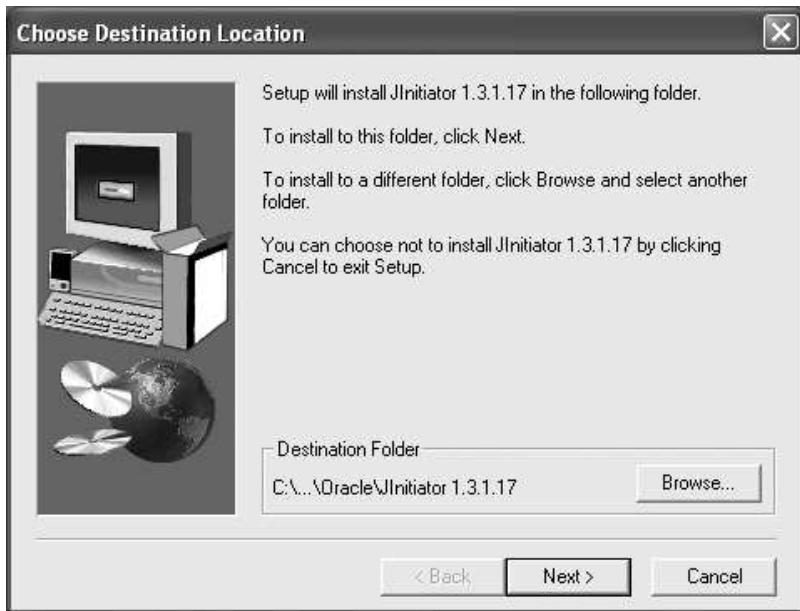


**Рис. 8.10.** Первый вариант формы Customer

## Проверка формы

Теперь, когда форма создана, можно запустить ее для проверки. В принципе формы предназначены для работы на сервере приложений. Однако, чтобы избавить разработчика от необходимости постоянно копировать формы, Oracle предоставляет программу, позволяющую тестировать формы на локальном компьютере.

1. Посредством главного меню системы Windows запустите OC4J, выбрав Start OC4J Instance. Для загрузки программы потребуется несколько секунд. Система защиты вашего компьютера запросит полномочия на выполнения программы. Вам следует подтвердить ваше согласие.
2. В окне конструктора форм щелкните на пиктограмме с изображением светофора , чтобы запустить форму. Тот же результат можно получить, активизировав пункт меню Program⇒Run Form.
3. При необходимости установите JInitiator. Если на вашей машине используется браузер с развитыми средствами защиты (такая ситуация типична для системы Windows XP с SP2), вам придется установить несколько флажков опций, разрешив в том числе отображение раскрывающихся окон.



**Рис. 8.11.** Первый этап инсталляции JInitiator

4. После запуска формы вы можете отобразить реальные данные, щелкнув на кнопке Execute Query или выбрав пункт меню Query⇒Execute.

**Совет.** Если перед запуском формы вы забудете вызвать OC4J, то получите сообщение об ошибке. В этом случае надо запустить OC4J и выполнить форму снова.

**Совет.** Для того чтобы остановить работу службы OC4J, выберите в меню Windows пункт Shutdown OC4J Instance. Если вы просто закроете окно OC4J, служба останется активизированной.

Первый запуск формы Oracle предполагает ряд действий по инсталляции JInitiator. Чем выше уровень средств защиты в браузере, тем сложнее будет процедура установки. На рис. 8.11 показано первое из последовательности окон, отображаемых в процессе установки JInitiator (в вашем случае номер версии может отличаться от представленного на рисунке). Следуйте рекомендациям, отображаемым на экране. В конце концов система откроет форму, но перед этим вы получите ряд сообщений, связанных с защитой. Вам придется устанавливать значения опций, разрешающих работу приложения и снимающих блокировки. При появлении ошибки следует завершить работу, перезапустить Forms Builder и снова попытаться выполнить форму.

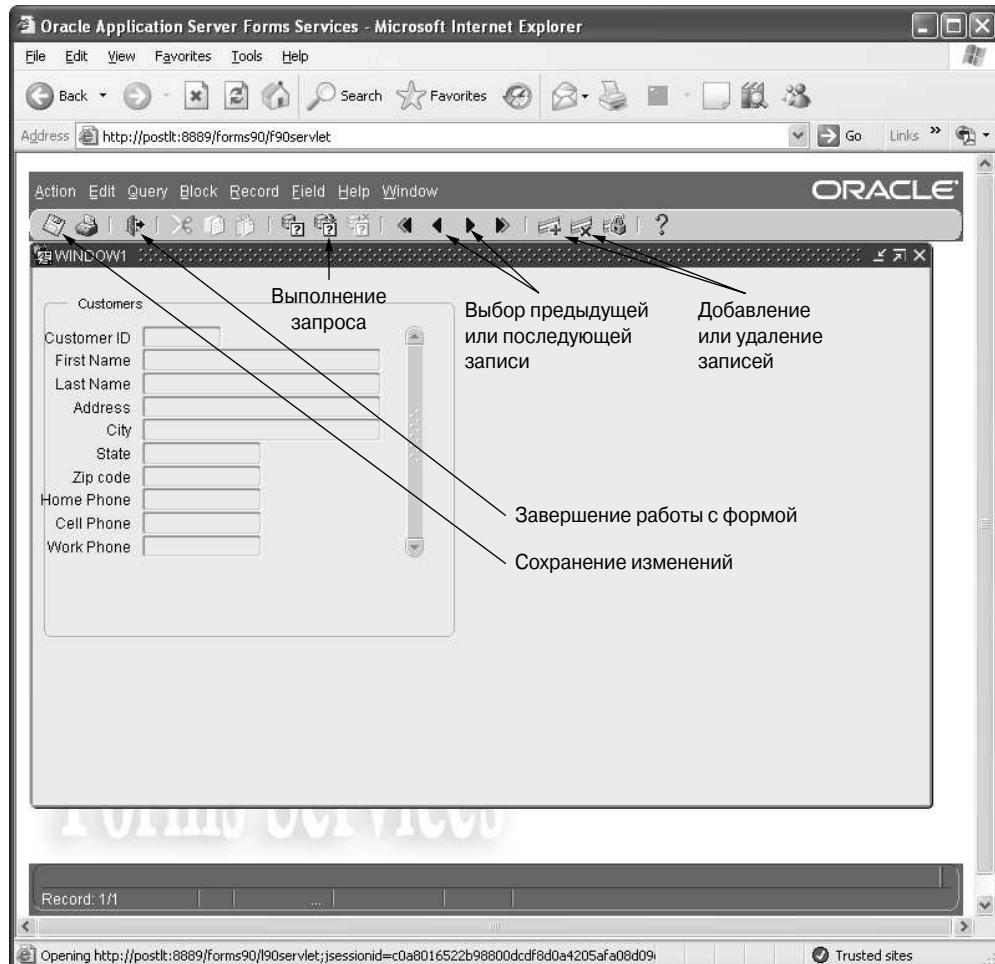


Рис. 8.12. Форма Customer и элементы управления

На рис. 8.12 показана форма, которую вы увидите на экране. На панели инструментов присутствует ряд кнопок, предназначенных для управления формой. С их помощью можно получать данные (Execute Query), переходить к другой записи, добавлять или удалять записи и сохранять внесенные изменения. Тех же результатов можно добиться с помощью меню, но, разобравшись один раз с кнопками панели инструментов, вы сможете выполнять поставленные задачи несколько быстрее, чем с помощью меню. Проверьте работу формы, щелкнув на кнопке Execute Query. Просмотрите несколько записей, пользуясь для этого кнопками Next Record (▶), Previous Record (◀), прибегая по мере необходимости к полосе прокрутки. Попробуйте изменить данные или добавить новую запись. Заметьте, что, для того, чтобы сохранить

изменения в базе данных, вам (а впоследствии и конечному пользователю) придется активизировать кнопку Save . Можно, конечно, отказаться от сохранения изменений. По завершении работы не забудьте закрыть форму с помощью кнопки Close .

Не стоит закрывать окно браузера. При активизации кнопки Close будут выполнены некоторые действия по очистке ресурсов, поэтому, если вы не сохранили изменения, отобразится предупреждающее сообщение. До завершения работы с формами желательно оставить программу OC4J активизированной.

## Получение данных с помощью формы

По умолчанию при первом запуске формы она переходит в режим запроса. Пользователю предоставляется возможность ввести условия поиска и, щелкнув на кнопке, выполнить запрос. Условия, введенные в полях редактирования, используются для формирования запроса, в результате возвращаются лишь записи, соответствующие этим условиям. Для того чтобы проверить систему запросов, выполните следующие действия.

Перезапустите форму Customer, но не щелкайте на кнопке Execute Query. Если форма находится в режиме редактирования, то, щелкнув на кнопке Enter Query , можно перевести ее в режим запросов. На рис. 8.13 показан пример простого запроса.

Введите в поле редактирования, соответствующем фамилии потребителя, символы B%. Обработчик запросов распознает знак процента и будет интерпретировать выражение как шаблон. В терминах SQL это будет соответствовать условию WHERE Lastname LIKE 'B%'.

Щелкните на кнопке Execute Query . Результаты, которые будут получены, показаны на рис. 8.14.

Условия выполняют функции фильтра. Просмотрите несколько записей, переходя от одной записи к другой с помощью кнопки Next Record . Заметьте, что отображаются только те строки, в которых фамилия заказчика начинается с буквы “B”. При необходимости можно добавить дополнительные условия; они будут объединены с существующими с помощью операции AND. Стока будет возвращена только в том случае, если она удовлетворяет всем условиям. Помните, что в выражениях, задающих условия, учитывается регистр символов. Инструктируя конечных пользователей, стоит заметить, что при поиске имен надо указывать прописные буквы. Для числовых значений допускаются операторы “больше” или “меньше”. Полученные данные можно редактировать или удалять. Фильтр – удобное средство, позволяющее находить требуемые строки в больших таблицах.

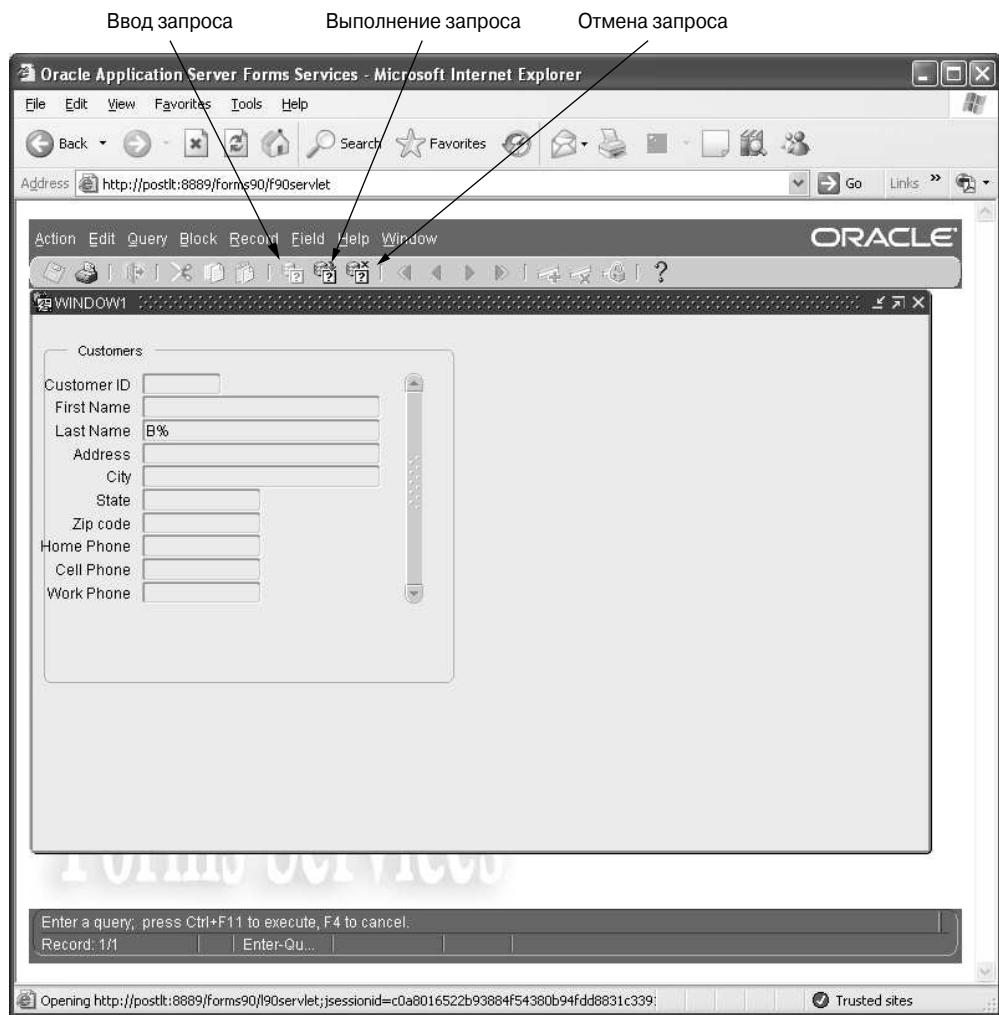


Рис. 8.13. Простой запрос для поиска потребителей, фамилии которых начинаются с буквами В

## Модификация форм

Инструментальные средства типа “мастер” выполняют большой объем работы по формированию структуры и компоновке форм. Однако в ряде случаев может возникнуть необходимость модифицировать форму, чтобы привести ее в соответствие с изменившимися требованиями пользователя или просто улучшить ее внешний вид. В принципе можно создать форму, не прибегая к услугам мастера.

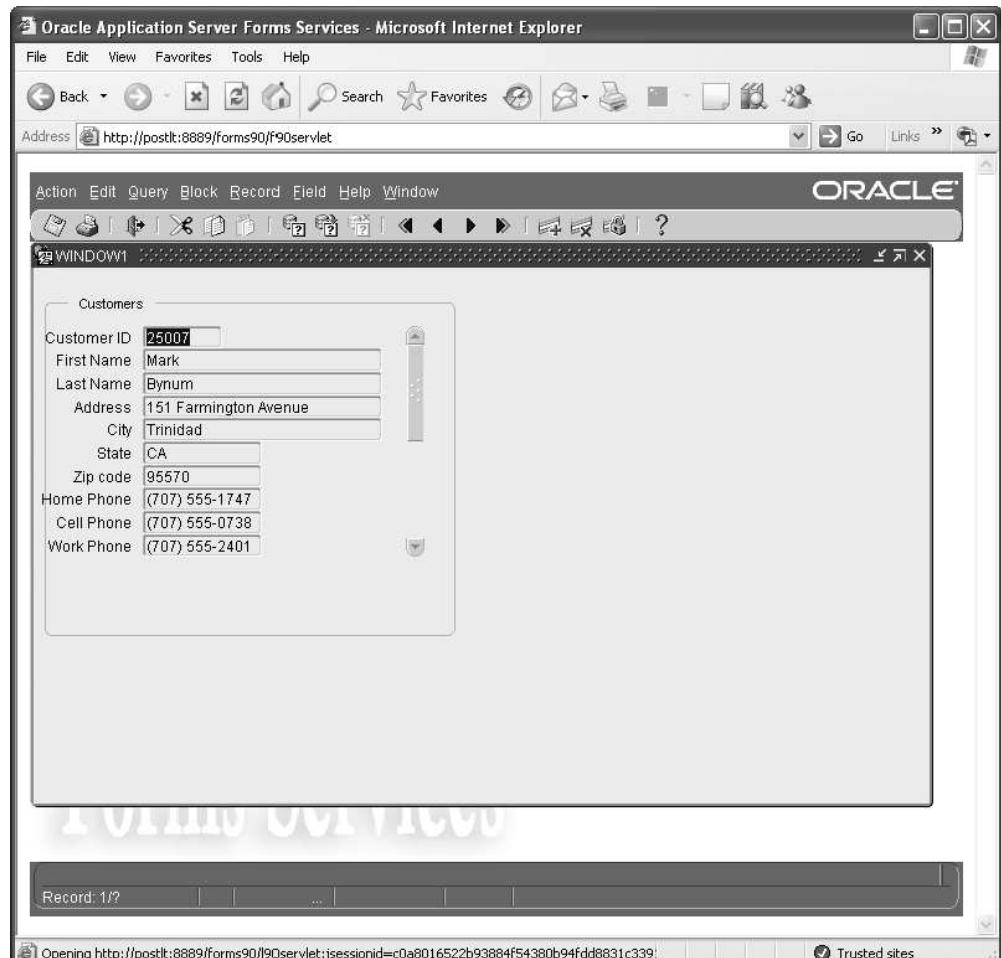


Рис. 8.14. В процессе выполнения запроса осуществляется фильтрация строк

## Редактор компоновки

Редактор компоновки очень часто используется при работе над формами. Если, завершив работу с Form Builder, вы снова запустите этот инструмент для редактирования формы, редактор компоновки не начнет выполняться по умолчанию. Открыв форму, вам надо нажать клавишу <F2> или выбрать пункт меню Tools⇒Layout Editor. Сделав это, вы перейдете в режим редактирования формы.

1. Откройте форму *Customers*, чтобы изменить расположение ее элементов.
2. В данном случае надо отказаться от режима обновления *Automatically* и установить режим *Manually*. Щелкните правой кнопкой мыши на форме, выберите

пункт меню **Property Palette** и установите значение **Manually** свойства **Update Layout**.

Теперь вы можете перемещать элементы в окне, и они останутся в тех позициях, где вы установите их. Для того чтобы форма была более пригодна для восприятия, между элементами надо оставить некоторый зазор. Для этого надо сделать следующее.

3. Переместите поле редактирования **WorkPhone** вниз на несколько пунктов.
4. Щелкнув кнопкой мыши и нажимая клавишу <Ctrl>, выделите все поля и выберите **Layout⇒Align Components**.
5. В группе **Vertically** выберите опцию **Distribute**.
6. Щелкните на кнопке **Align Left** , чтобы выровнять левые края полей редактирования.

Выбирая поля редактирования, следите за тем, чтобы выделены были только поля без текста, поясняющего их назначение. Нарисуйте с помощью мыши прямоугольник, включающий только поля редактирования. Если сделать это не получается, выбирайте каждое поле отдельно, щелкнув на нем мышью и нажимая клавишу <Ctrl>. На рис. 8.15 показаны опции, управляющие выравниванием выделенных полей. Несколько кнопок, управляющих выравниванием, присутствует на панели инструментов, но некоторые возможности, например опция **Distribute**, доступны только посредством меню. Именно их мы используем для изменения расстояния между элементами.

## Объектный навигатор

Как видно на рис. 8.16, объектный навигатор представляет иерархическую структуру, включающую все объекты формы. В основном он предназначен для того, чтобы облегчить задачу поиска и выбора элементов. Некоторые объекты можно найти только посредством навигатора. Создавая форму, часто приходится задавать порядок перехода от одного поля к другому, нажимая клавишу <Tab>. Он определяется последовательностью включения элементов в блок данных объектного навигатора. Например, в форме **Customers** фокус ввода будет передаваться от **CustomerID FirstName**, затем **LastName** и т.д. до **WorkPhone**. Для того чтобы изменить порядок передачи фокуса ввода, надо перетащить элемент на другую позицию в списке. Это не повлияет на расположение элементов в форме, а затронет лишь поведение формы при нажатии клавиши <Tab>.

С помощью объектного навигатора можно также задать для главного окна более подходящее имя.

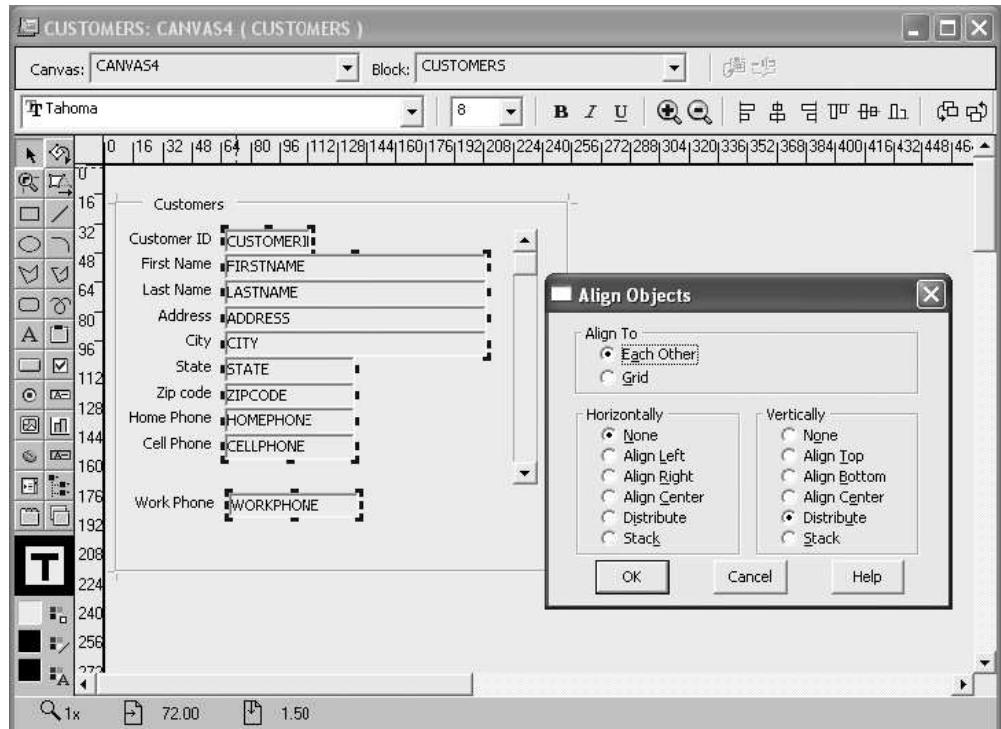


Рис. 8.15. Опции выравнивания

1. В окне Object Navigator разверните раздел Windows и найдите единственное окно WINDOW1.
2. Дважды щелкните на нем мышью и переименуйте его в REDWOOD\_REALTY.

При выполнении формы заданное имя отображается в верхней части окна браузера. Вы можете щелкнуть правой кнопкой мыши на элементе, воспользоваться подходящей палитрой и изменить характеристики этого элемента. Однако в ряде случаев проще не выбирать элементы в редакторе компоновки, а делать это посредством объектного навигатора. Позже вы узнаете, как использовать навигатор для создания и проверки программного кода, связанного с событиями формы.

## Основные свойства элементов

Для установки различных свойств используется палитра свойств (окно Property Palette). С помощью Property Palette вы можете ознакомиться с основными свойствами, доступными для каждого типа элемента. Весь список свойств запоминать не обязательно, достаточно иметь представление о главных свойствах и о том, как они воздействуют на внешний вид и поведение формы. Например, свойства раздела Database отображают элементы формы в столбцы. В разделе физических свойств

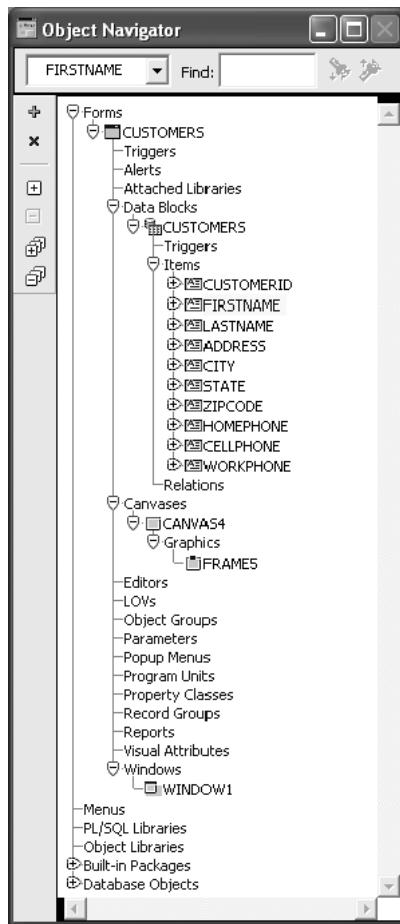


Рис. 8.16. Объектный навигатор

стоит отметить свойства `Height` и `Width`, которые дают возможность достаточно просто устанавливать конкретные значения для нескольких элементов. Существуют также средства, позволяющие задавать точную позицию каждого элемента по координатам `X` и `Y`. Кнопки на панели инструментов, управляющие выравниванием, также позволяют работать с группами элементов. Необходимо выбрать в навигаторе объектов все элементы, подлежащие выравниванию, щелкнув мышью на каждом из них, нажав одновременно клавишу `<Ctrl>`, а затем активизировать соответствующую кнопку.

На рис. 8.17 показан пример одного из этапов работы по модификации формы. Многие из описанных ниже действий необязательны; возможно, вы разработаете для каждого проекта свой стандарт, определяющий внешний вид формы. Желательно лишь обеспечить согласованность всех форм в рамках одного приложения. Работа над формой может происходить следующим образом.

Варианты выравнивания

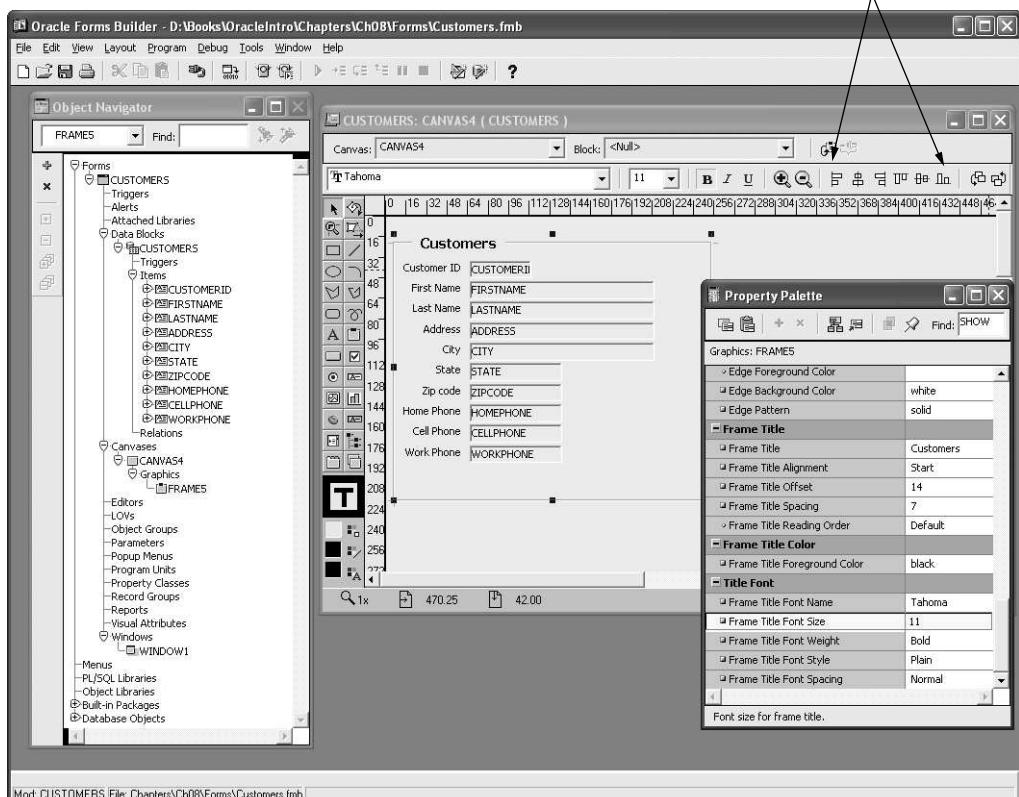


Рис. 8.17. Модификация формы

- Удалите полосу прокрутки. Щелкните правой кнопкой мыши на данном элементе и выберите **Property Palette**, чтобы иметь возможность отредактировать параметры блока данных *Customers*.

Сейчас самое время обратить ваше внимание на удобную возможность, предоставляемую палитрой свойств. Если вы знаете имя свойства, щелкните в поле **Find**, расположенном в верхнем правом углу окна. Начинайте вводить имя свойства; по мере ввода палитра выполнит прокрутку и отобразит тот пункт, имя которого соответствует заданной вами последовательности символов.

- Ведите в поле редактирования **Find** последовательность символов **scr**. Если возникнет необходимость, выполните прокрутку в окне **Property Palette** и установите значение **No** свойства **Show Scroll Bar**.
- Щелкните правой кнопкой мыши на фрейме, чтобы изменить шрифт заголовка. Задайте для шрифта размер 10 пунктов.

Вы можете также применить средства управления шрифтами, которые доступны на панели инструментов в верхней части окна, или воспользоваться свойствами заголовка в палитре свойств.

#### 4. Измените имя основного окна на Redwood\_Realty.

Это имя не появляется в редакторе компоновки, но отображается при выполнении формы. Для того чтобы изменить имя, откройте поддерево Windows в объектном навигаторе. Протестируйте форму; в частности, в процессе проверки обратите внимание на выравнивание элементов.

## Включение изображений в состав формы

Если вы хотите включить в форму логотип, сначала убедитесь в том, что он хранится на вашем компьютере и представлен в стандартном формате (например, JPEG, GIF или TIFF). Вы можете воспользоваться готовым файлом или создать собственный. Специальный инструмент для помещения логотипа на странице не предусмотрен.

1. Щелкните в окне редактора компоновки, затем выберите пункт меню **Edit⇒Import**, чтобы найти и импортировать файл.
2. Установите размер изображения, соответствующий вашим замыслам. Изменяя размер статического изображения с помощью мыши, удерживайте нажатой клавишу **<Shift>**.

Нажатая клавиша **<Shift>** позволяет при перетаскивании мышью границ изображения сохранить постоянным соотношение его сторон. Статическое изображение можно расположить в любой позиции. Кроме того, вы можете подобрать яркость и контрастность и поместить изображение позади остальных элементов формы так, чтобы оно походило на водяной знак. Реализуя подобные эффекты, надо соблюдать осторожность, так как при этом есть риск ухудшить восприятие элементов формы. На рис. 8.18 показана форма с изображением в правом нижнем углу.

Изображение можно также включить непосредственно в базу данных. В этом случае оно станет не просто элементом оформления формы, а частью информации, хранящейся в базе данных. Так, например, вы, возможно, захотите поместить в базу фотографии сотрудников. Для рассматриваемого здесь примера Redwood Realty в базе пригодятся фотографии домов, предназначенных для продажи. Помимо фотографий в базе данных можно хранить любые двоичные объекты. Чтобы воспользоваться этой возможностью, сначала надо создать столбец таблицы. Одно из преимуществ грамотно спроектированной базы данных состоит в том, что в таблицу в любой момент можно без труда добавить новый столбец. Добавление столбца не затрагивает существующие формы и отчеты, поэтому можно работать с ним, не беспокоясь о том, как это скажется на внешнем виде приложения.

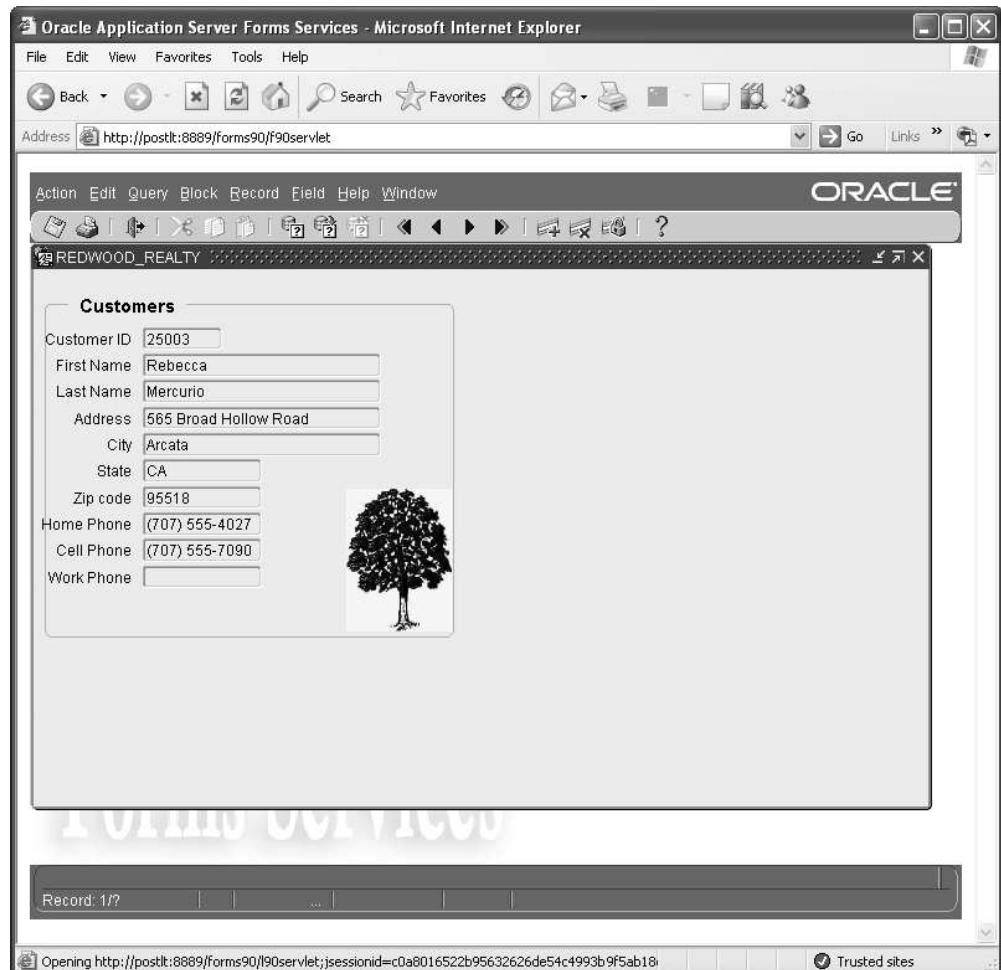
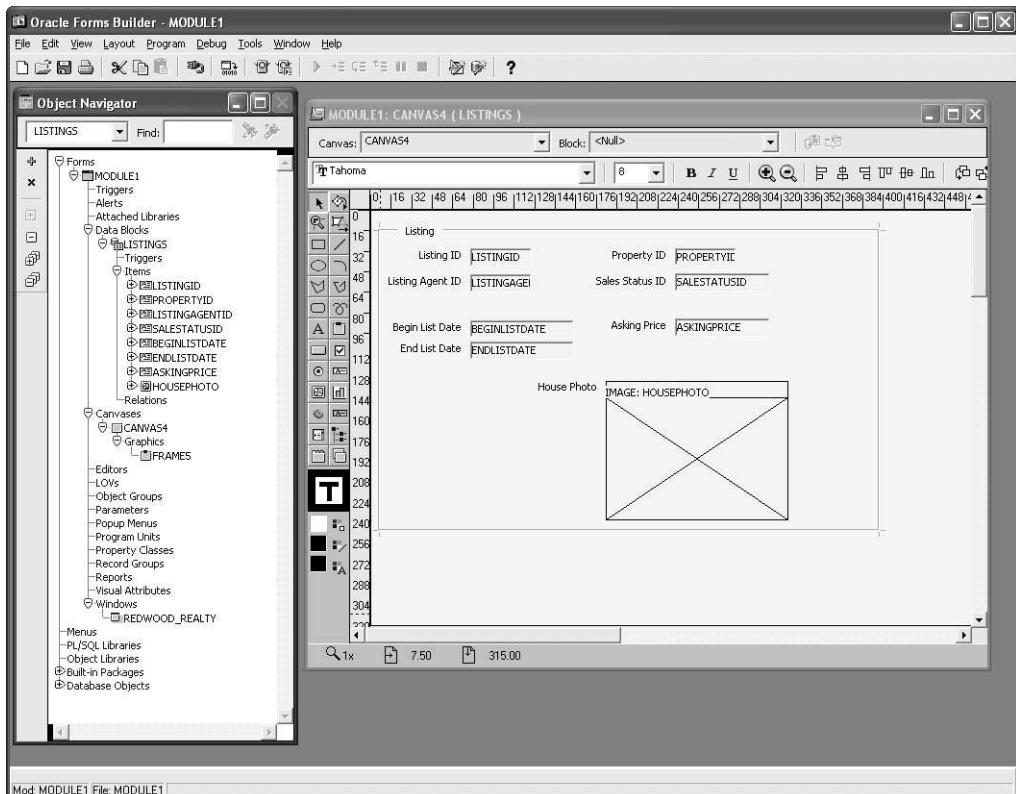


Рис. 8.18. Включение статического изображения

1. Для того чтобы включить в таблицу Listings столбец HousePhoto, запустите SQL\*Plus, установите соединение с базой данных и введите следующие команды:
 

```
ALTER TABLE Listings
ADD HousePhoto LONG RAW;
```
2. В результате каждая строка таблицы Listings получит возможность хранить одно изображение. Первоначально значения данного поля во всех строках будут равны null. Но если агенты вооружены цифровыми фотоаппаратами, они вскоре заполнят базу фотоснимками. В результате вам придется создавать форму, способную отображать фотографии.



**Рис. 8.19.** Расположение элементов формы Listings

3. Используйте мастер блоков данных и мастер компоновки для создания формы *Listings*, предназначеннной для отображения одной строки таблицы *Listings*. Выполните те же действия, которые предпринимались при разработке формы *Customer*.

На рис. 8.19 показано расположение элементов новой формы. Вероятнее всего, вы захотите установить ширину столбцов с идентификаторами равной 48. Как вы помните, чтобы сделать это, надо установить значение *Manually* свойства *Update Layout* фрейма. Оставьте в форме пустое место, чтобы впоследствии можно было включить в состав формы несколько новых пунктов. Используя команды выравнивания, добейтесь приемлемого внешнего вида существующих элементов.

Включение фотоснимка или любого двоичного объекта в базу данных Oracle требует загрузки изображения из файла. Файл должен храниться на том же компьютере, на котором выполняются формы. Для тестирования файлы надо расположить на той машине, на которой ведется разработка. Когда вы разместите базу данных на сервере приложений, вам придется скопировать на этот сервер все фотоснимки. Скопируйте

файл с изображением, которое вы будете использовать в качестве примера, на свой компьютер и запомните, в какой папке вы его разместили. Для того чтобы поместить в базу содержимое файла, вам надо включить в форму Listings независимое поле редактирования, в котором пользователь сможет указать полный путь к файлу. Вам также понадобится кнопка, предназначенная для запуска процесса копирования.

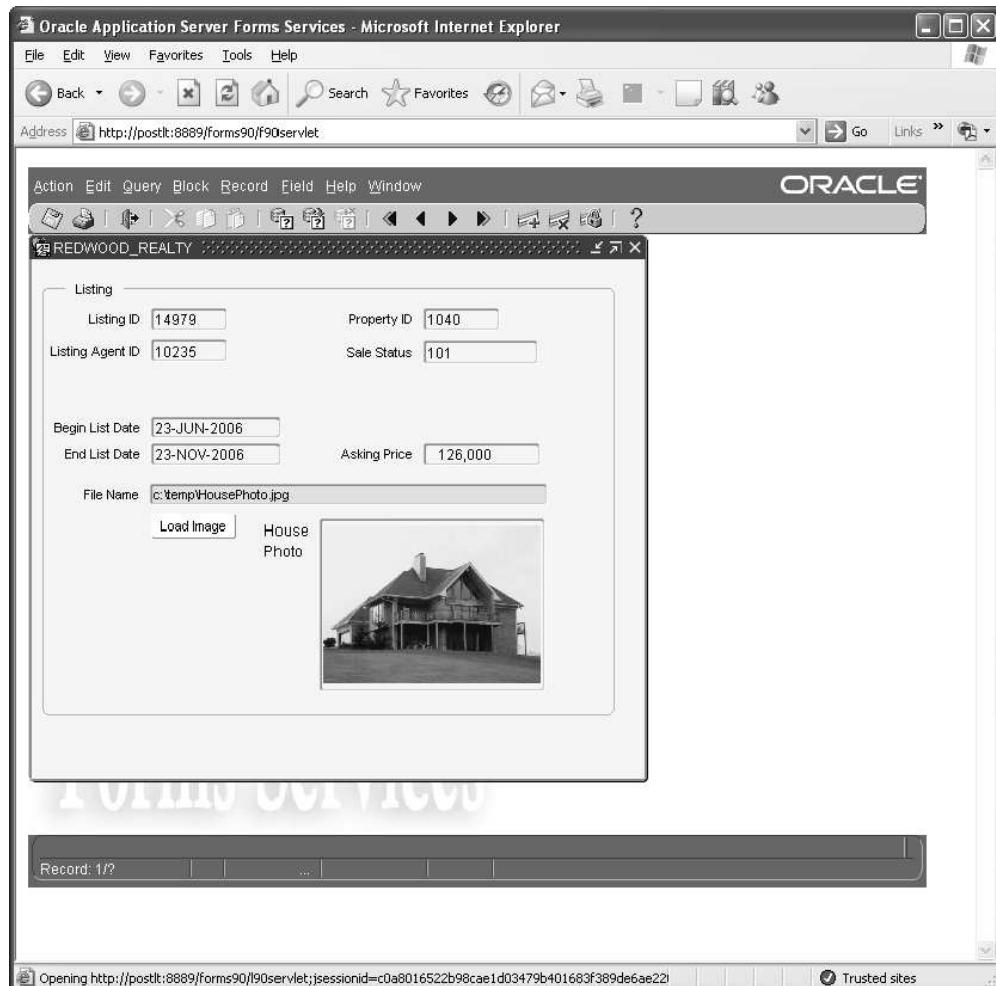
1. Включите в форму простое поле Text Item . Назовите его ImageFileName. Значением Prompt должно быть File to Load.
  2. Добавьте кнопку к форме. Назовите ее CmdLoadImage и убедитесь, что значением Label является Load Image.
  3. Щелкните правой кнопкой мыши на новой кнопке, выберите пункт Triggers, затем установите событие WHEN-BUTTON-PRESSED.
  4. В окне редактора PL/SQL введите следующую строку кода:
- ```
READ_IMAGE_FILE(:IMAGEFILENAME, 'ALL', 'HOUSEPHOTO');
```
5. Щелкните на кнопке , чтобы скомпилировать код PL/SQL. Закройте окно с кодом.
  6. Сохраните форму и проверьте ее работу. Выберите один из списков, введите полный путь к файлу с изображением, затем щелкните на кнопке Load Image. Щелкните на кнопке Save формы , чтобы обновить базу данных.

На рис. 8.20 показана форма с изображением, загруженным в базу данных. Из папки, предназначеннной для временного хранения, можно удалить исходный фотографий. Если вы закроете форму и вернетесь к ней позже, изображение будет присутствовать, поскольку теперь оно хранится в базе данных.

## Добавление средств поиска

Обратите внимание на то, что в форме Listings представлен только идентификатор агента. Если брокер просматривает список, он, возможно, захочет узнать реальное имя агента. В принципе это возможно, так как имя хранится в таблице Agents. Не имеет смысла сохранять имя агента много раз, при необходимости его можно извлечь, поскольку таблицы связаны через ListingAgentID. Но как отобразить значение из таблицы Agents в форме Listings? Для этого надо сформировать запрос к блоку данных. При создании формы на основании запросов надо соблюдать осторожность. Форма может записывать данные только в одну таблицу, соответствующую блоку данных. С помощью запросов можно отобразить дополнительные столбцы, но они будут доступны только для чтения. Изменить их значения невозможно.

Для того чтобы сформировать запрос к блоку данных, выполните следующие действия.



**Рис. 8.20.** Загрузка изображения в базу данных

1. В объектном навигаторе выберите блок данных Listings. Щелкните правой кнопкой мыши и откройте окно Property Palette. Найдите пункт Query Data Source Name. Первоначально значением этого свойства является имя таблицы (Listings).
2. Замените имя запросом SELECT. Не забудьте поместить запрос в круглые скобки.
 

```
(SELECT ListingID, PropertyID, ListingAgentID, SaleStatusID,
BeginListDate, EndListDate, AskingPrice, HousePhoto,
FirstName, LastName
FROM Listings INNER JOIN Agents ON
Listings.ListingAgentID=Agents.AgentID)
```

**ТАБЛИЦА 8.1.** Значения свойств для использования запроса

| <i>Свойство</i>          | <i>Новое значение</i> |
|--------------------------|-----------------------|
| Name                     | FIRSTNAME             |
| Prompt                   | First Name            |
| Prompt Attachment Offset | 7                     |
| Database Item            | Yes                   |
| Column Name              | FIRSTNAME             |
| Primary Key              | No                    |
| Query Only               | Yes                   |

Перед тем как задавать запрос в качестве значения свойства, его надо проверить в SQL\*Plus. В этом случае вы сможете найти и исправить ошибки, не дожидаясь запуска формы. В данном примере обратите внимание то, что из таблицы Agents извлекаются значения FirstName и LastName. Поскольку форма реально выполняет запрос как подчиненный, его следует поместить в скобки. Отсутствие скобок — это ошибка, часто допускаемая специалистами с недостаточным опытом работы.

3. Откройте свойство Query Data Source Columns, щелкнув на кнопке More ... . Добавьте FIRSTNAME и LASTNAME в качестве новых строк в списке Column Names. После того как будет сформирован новый запрос, откройте свойство Query Data Source Columns. Вы увидите список столбцов, используемых в текущий момент. Переместитесь в конец списка и добавьте в него FIRSTNAME и LASTNAME. Закройте окно свойств. Теперь в объектном навигаторе должны появиться два дополнительных элемента. Мы готовы отображать информацию в форме.
4. Добавьте к форме два поля Text Item. Присвойте им имена FirstName и LastName и соответствующим образом сформируйте пояснительные надписи.
5. Теперь вам надо установить для обоих полей несколько свойств. В табл. 8.1 показаны установки для поля FIRSTNAME. Убедитесь в том, что значение Primary Key равно No. Выполните те же установки для поля LASTNAME.
6. Для новых текстовых элементов задайте свойства, связанные с базой данных: Database Item = Yes и Column Name = FIRSTNAME. Повторите те же действия для LASTNAME.

Сохраните форму и проверьте ее работу. Переходя от одного списка к другому, убедитесь, что имя агента изменяется. Попытайтесь отредактировать несколько значений и добавьте новую запись, чтобы убедиться в том, что основная форма по-прежнему работоспособна. Запросы можно использовать практически в любой форме. Однако при этом необходимо соблюдать осторожность. Перед добавлением пунктов, связанных

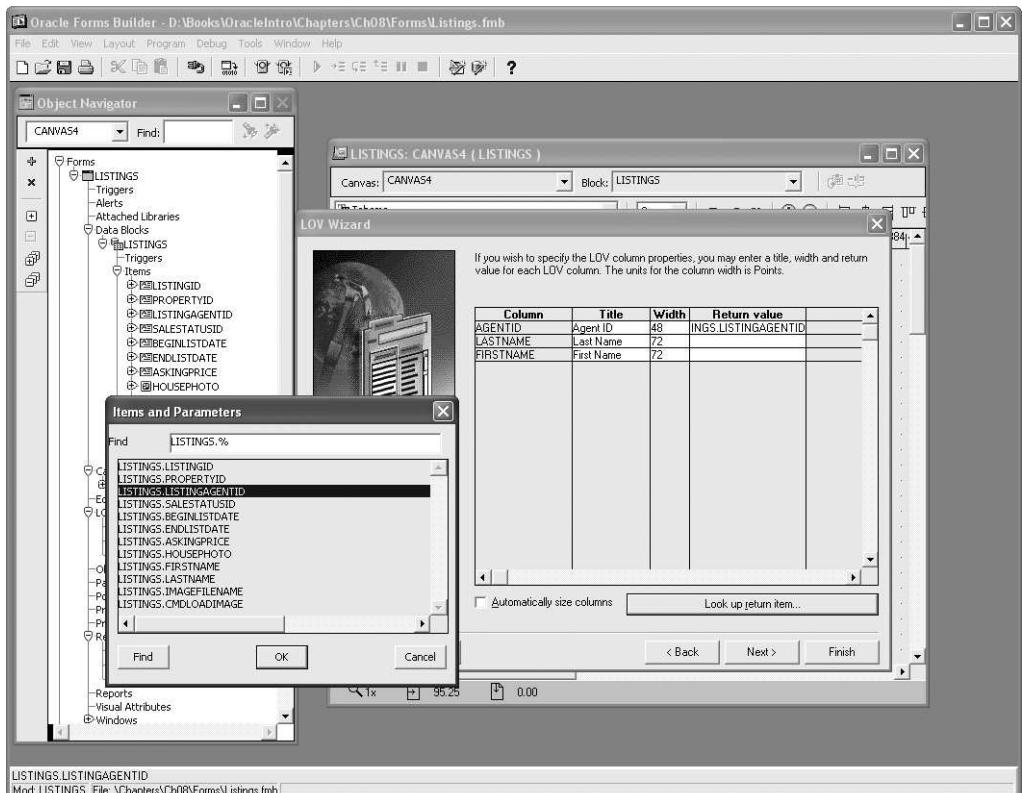
ных с запросами, убедитесь в том, что форма работает. Если впоследствии возникнут проблемы, возвратитесь к исходному варианту формы и опять попытайтесь включить новые элементы.

## Создание списка значений

Включив имена агентов в форму, вы упростили пользователям работу с ней. Вряд ли брокеры будут помнить номера агентов. Но предположим, что вам надо добавить новый список. При этом придется ввести значение идентификатора агента. Значит ли это, что пользователям придется запоминать идентификационные номера? Даже если вы потребуете, чтобы агенты вводили свои собственные списки, они должны будут помнить свои идентификаторы. Неплохо было бы создать список поиска, из которого можно было бы выбрать агента. Наилучшим решением данной проблемы было бы формирование *списка значений*(list of values — LOV). Список значений — это список всех возможных вариантов выбора для конкретного поля, который допускает фильтрацию, а пользователь с его помощью может выполнять поиск. Когда выбор сделан, соответствующий идентификационный номер извлекается из списка и помещается в таблицу. Заметьте, что в данный процесс вовлекаются две таблицы: список всех агентов извлекается из таблицы Agents, а выбранное значение хранится в таблице Listings.

Проще всего создать список значений, воспользовавшись мастером под названием List of Values Wizard, доступным посредством меню Tools. Мастер создает группу записей, содержащую выражение SELECT, которое позволяет извлекать данные. Работа с мастером LOV сводится к следующему.

1. Запустите мастер, выбрав пункт меню Tools⇒List of Values Wizard.
2. Установите опцию, разрешающую создание новой группы записей.
3. Создайте SQL-запрос, используя специально предназначенный для этой цели инструмент, либо, если вы уже создали и проверили запрос в SQL\*Plus, введите его вручную.
4. Инструмент Query Builder предоставляет визуальные средства для выбора таблиц и столбцов. Пользуясь результатами выбора, он формирует и записывает SQL-выражение SELECT. В данном примере вам надо выбрать AgentID, LastName и FirstName. Щелкните на кнопке Sort и выберите LastName, FirstName в качестве условия сортировки.
5. Закройте Query Builder, и вы увидите команду SQL. Щелкните на кнопке Check Syntax.
6. Включите все столбцы в LOV, переместив их в правое окно.



**Рис. 8.21.** Выбор возвращаемого значения с помощью List of Values Wizard

7. Ограничите ширину отображаемых значений столбцов: 48 для идентификаторов и 72 для имён. При желании вы также можете воспользоваться опцией *Automatically size columns*.
8. Поместите курсор в столбец *Return Item* строки *AgentID*, щелкните на кнопке *Look up return item* и выберите *ListingAgentID*. В этот столбец будет помещен выбранный пункт.

На рис. 8.21 показаны результаты выполнения описанных выше действий. Убедитесь в том, что *Listings.ListingAgentID* помещен в столбец для возвращаемого значения из *AgentID*. На данном этапе требуемое значение (*AgentID*) извлечено из списка и помещено в нужный столбец основной формы (*Listings.ListingAgentID*). Теперь, если после закрытия мастера будет обнаружена ошибка, вы сможете выбрать в объектном навигаторе LOV и перезапустить мастер, используя пункт меню *Tools⇒List of Values Wizard*. Можно также непосредственно внести изменения в LOV или записать значения свойств. Заметьте, что LOV связан с формой, так как свойству LOV присвоено имя поля формы (*ListingAgentID*).

9. Введите в качестве заголовка `List of Agents` и установите высоту, по умолчанию равную 235. Примите предлагаемое по умолчанию значение, предполагающее одновременное отображение 20 строк.
10. Последний этап аналогичен этапу 8. Вам необходимо выбрать возвращаемое значение (`Listings.ListingAgentID`) и поместить его в правое окно.
11. По окончании работы мастера откройте раздел `LOVs` объектного навигатора и переименуйте `LOV` в `LOV_AGENTS`. Кроме того, переименуйте новую группу записей в `RG_LOV_AGENTS`.
12. Запустите форму. Она будет иметь тот же вид, что и ранее. Перейдите к полю, соответствующему идентификатору агента. В строке состояния в нижней части окна формы вы увидите надпись `List of Values`. Это означает, что выбранное поле поддерживает список значений. Чтобы открыть его, нажмите комбинацию клавиш `<Ctrl+L>`.

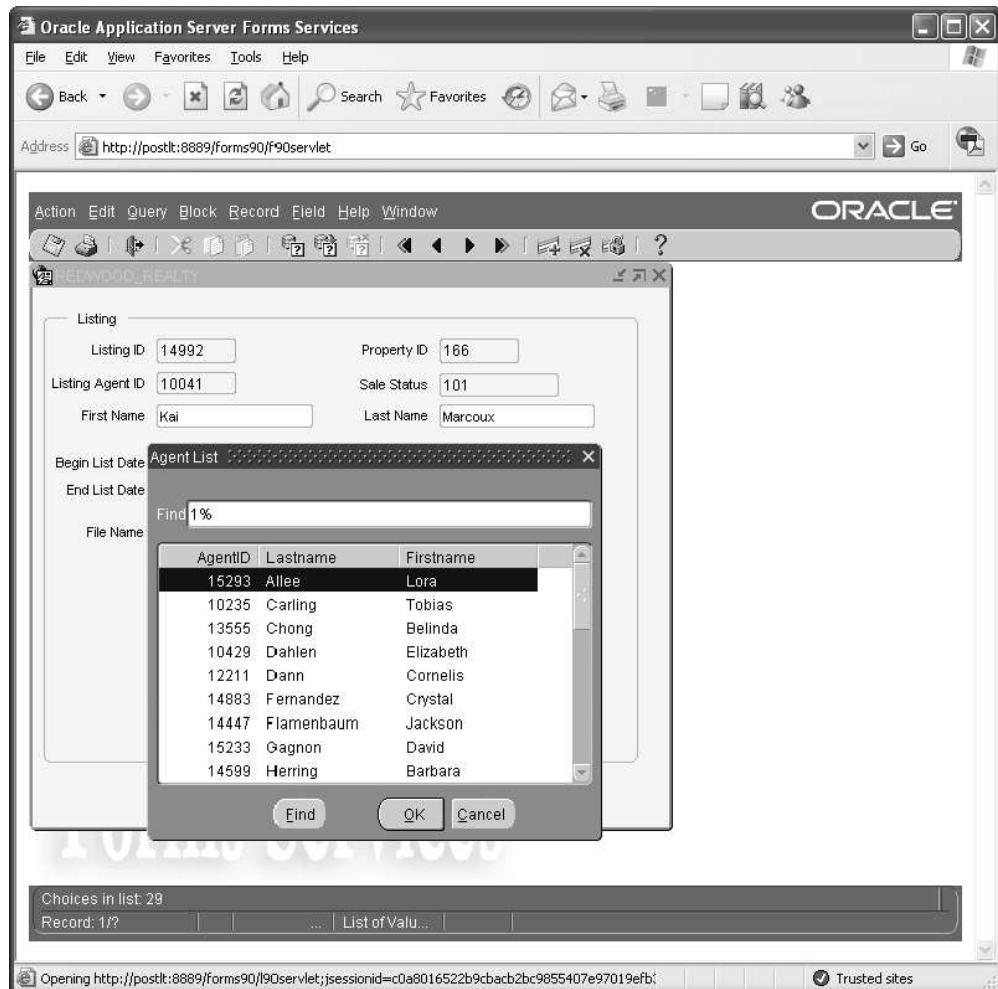
Исходный список показан на рис. 8.22. Пользователь может просмотреть список и найти требуемый пункт. Поле `Find` в верхней части окна существенно упрощает работу. Пользователь может ввести условия поиска, организовав таким образом фильтрацию строк. В данном случае отображаются поля `AgentID`, `Lastname` и `Firstname`. Для того чтобы найти идентификатор по фамилии агента, например `Marcoux`, надо ввести выражение `%Mar%`. Первый знак процента соответствует любому идентификатору, последний символ означает конец фамилии и любое имя.

Подход, связанный с использованием списка значений, очень удобен, поскольку в этом случае пользователь может оперировать не с абстрактными номерами, а с конкретными именами. Эффективность работы повышается, так как в окне `LOV` не отображается весь список, а лишь часть его, удовлетворяющая условиям фильтрации. Чтобы упростить работу со списком значений, можно поместить рядом с каждым полем редактирования небольшую кнопку `LOV`. Этим вы сообщите пользователю о том, что список значений доступен после щелчка на данной кнопке. Если вы поступите таким образом, для кнопки надо будет задать две строки кода:

```
Go_Item('LISTINGAGENTID');  
List_Values;
```

## Автоматическое выполнение запроса

На данном этапе можно несколько оптимизировать работу пользователя с формами. Как вы помните, когда форма открывается в первый раз, она пуста. Пользователь должен щелкнуть на кнопке `Execute Query`, чтобы извлечь данные, необходимые для заполнения формы. Если форма используется часто, ее заполнение можно осуществлять автоматически так, чтобы сразу после ее открытия пользователь видел первую



**Рис. 8.22.** Список значений, позволяющий выбрать агента

запись. Сделать это несложно — потребуется лишь одна строка кода. Поместить ее надо в соответствующий обработчик события.

1. Найдите в верхней части окна объектного навигатора поддерево Forms, а в нем раздел LISTINGS. Щелкните правой кнопкой мыши на пункте Triggers и выберите подменю для динамического списка обработчиков. Выберите пункт WHEN-NEW-FORM-INSTANCE, после чего откроется редактор PL/SQL. Введите следующий код:  

```
EXECUTE_QUERY;
```
2. Скомпилируйте код и закройте окно редактора. Проверьте форму.

На этот раз при открытии формы будет отображаться первая строка. Теперь пользователь сможет просмотреть список, не прибегая к помощи кнопки запроса. По мере необходимости он может перейти к окну запроса и ввести условия для фильтрации списка. Возможно, вам имеет смысл поговорить с пользователями и выяснить, предпочтут ли они открывать форму с отображаемыми данными или сначала вводить условия фильтрации.

## Переключатели и флагки опций

В большинстве случаев для представления значений столбцов базы данных подходят обычные поля редактирования. Свойства полей обеспечивают контроль информации в процессе ввода. Для того чтобы упростить ввод, можно использовать список значений; в этом случае пользователю не обязательно запоминать данные — он может выбрать пункт списка. Однако в некоторых случаях необходимы дополнительные средства для работы с данными. В частности, могут понадобиться переключатели или флагки опций. В принципе единственное различие между переключателями и флагками опций состоит в том, что переключатель отображается в виде круга, заполняемого при выборе данного элемента, а флагок опции имеет вид квадрата, в котором при выборе элемента отображается метка. Однако большинство разработчиков следуют соглашениям о том, что несколько переключателей, объединенных в группу, действуют как кнопки с зависимой фиксацией, т.е. реализуют взаимоисключающий выбор. Так, например, вы можете задать синхронный или асинхронный обмен данными, но не оба режима одновременно. Флагки опций применяются тогда, когда допустим одновременный выбор нескольких пунктов. Так, например, при передаче по последовательной линии можно независимо устанавливать наличие дополнительного стопового бита и контроль четности. Данные соглашения удобны, поэтому желательно следовать им, так как пользователь ожидает от ваших приложений именно такого поведения. Однако имейте в виду, что обработчик форм не требует от вас соблюдения этих соглашений, поэтому, организуя группы переключателей, будьте внимательны.

Чтобы проиллюстрировать использование переключателей, создадим форму, отображающую в каждый момент времени одну строку таблицы *Properties*. Для этого потребуется выполнить действия, типичные для других форм.

1. Запустите Forms Builder, чтобы создать новую форму, и убедитесь в том, что выполняется ОС4J. Активизируйте мастер блока данных и выберите все столбцы таблицы *Properties*.
2. В мастере компоновки выберите все столбцы, переместив их в правое окно, но не щелкайте на кнопке *Next*.
3. Выберите столбец *Zone*, щелкнув на нем мышью. Обратитесь к полю *Item\_type* и измените его значение с *Text Item* на *Radio Group*.

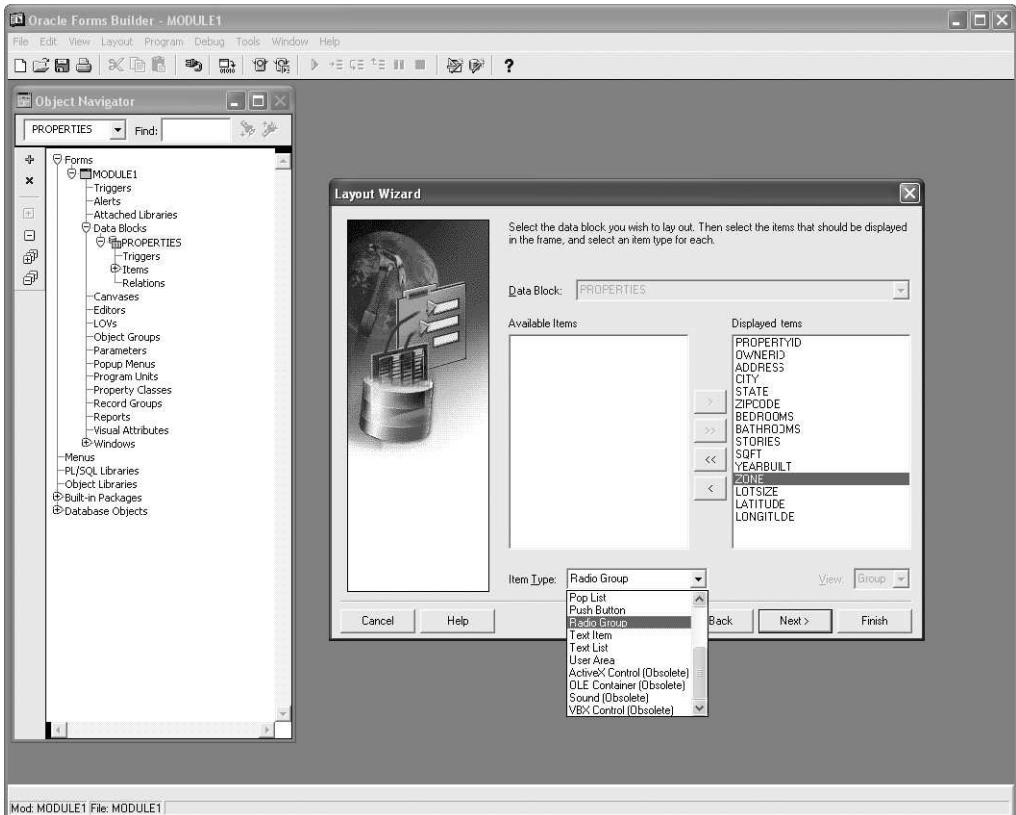


Рис. 8.23. Изменение типа элемента

Поскольку создается стандартная форма, вы начинаете работу с мастером блоков данных и выбираете таблицу *Properties* и все ее столбцы. Отличия от привычной вам последовательности действий начинаются при переходе к мастеру компоновки, поэтому будьте внимательны. После того как вы переместите все столбцы в правое окно, вы выбираете пункт *Zone* и изменяете его тип *Text Item*, предложенный по умолчанию, на *Radio Group*. На рис. 8.23 показано расположение раскрывающегося списка, с помощью которого вы вносите необходимые изменения. Выполнив требуемые действия, завершите обычным образом работу с мастером, не забыв при этом изменить размеры полей редактирования.

4. Завершите работу с мастером компоновки, упорядочьте элементы по вертикали и задайте обновление фрейма вручную.
5. Добавьте к форме элемент *Radio Button*. Задайте его свойства следующим образом: *Name=RAD\_R1*, *Label=Residential R1*, *Radio Button Value=R1*. Путем копирования и вставки добавьте кнопки *R2*, *R3* и *C1*.

В мастере компоновки вы определили группу для столбца *Zone*. Отдельные кнопки переключателя создаются вручную. Данные в столбце *Zone* представляют одно из свойств объекта. В большинстве городов для описания значений данного свойства используется следующая терминология: R1 — дом на одну семью; R2 и R3 — дом на несколько семей; C1 — объект для коммерческого использования. Возможны и другие варианты, но для простоты примем, что компания Redwood Realty имеет дело только с этими четырьмя типами объектов.

Первая кнопка добавляется посредством меню. Система предложит вам выбрать для нее группу. Поскольку существует только группа *Zone*, выберите ее и продолжайте работу. Для того чтобы элемент работал так, как вам необходимо, придется задать несколько свойств. На рис. 8.24 показаны значения трех основных свойств, которые вам придется установить. Присвойте кнопке узнаваемое имя (*RAD\_R1*); оно потребуется вам на случай, если к ней придется обращаться в дальнейшем. Установите пояснительный текст, который будет представлен пользователю (*Residential R1*). И наконец, присвойте данной кнопке значение (*R1*). Это значение должно соответствовать данным, хранящимся в таблице. Когда пользователь выберет конкретную кнопку, значение будет помещено в столбец *Zone* таблицы *Properties*.

Закончив установку свойств для первой кнопки, скопируйте ее и вставьте в форму новую кнопку. Поместите новую кнопку ниже первой и установите ее свойство *Residential R2*. Сделайте то же для *Residential R3* и *Commercial C1*. Обратившись к пункту меню *Layout⇒Align Component*, установите одинаковое расстояние между кнопками и выровняйте их по левому краю.

Найдите пункт *Zone* в объектном навигаторе. Откройте палитру свойств и перейдите к свойству *Initial Value*. Первоначально оно пусто; установите для него значение *R1*. Это значение будет использоваться по умолчанию, так как в большинстве случаев компания имеет дело с домами для одной семьи. Чтобы выделить группу кнопок в форме, вы можете обвести их контуром и снабдить пояснительной надписью *Zone*.

6. Используя *Object Navigator*, установите значение *R1* свойства *Initial Value* для *Zone*.
7. Выровняйте кнопки. При необходимости обведите их прямоугольным контуром. Сохраните форму и запустите ее. Заметьте, что все существующие объекты в базе данных имеют значение *R1* рассмотренного здесь свойства, поэтому при переходе от одной записи к другой будет выбрана одна и та же кнопка. Измените одну из записей, присвоив значение *R2*. Не забудьте сохранить изменения. Затем снова просмотрите несколько записей, и вы увидите, что обновленное значение отображается корректно. Внешний вид формы показан на рис. 8.25.

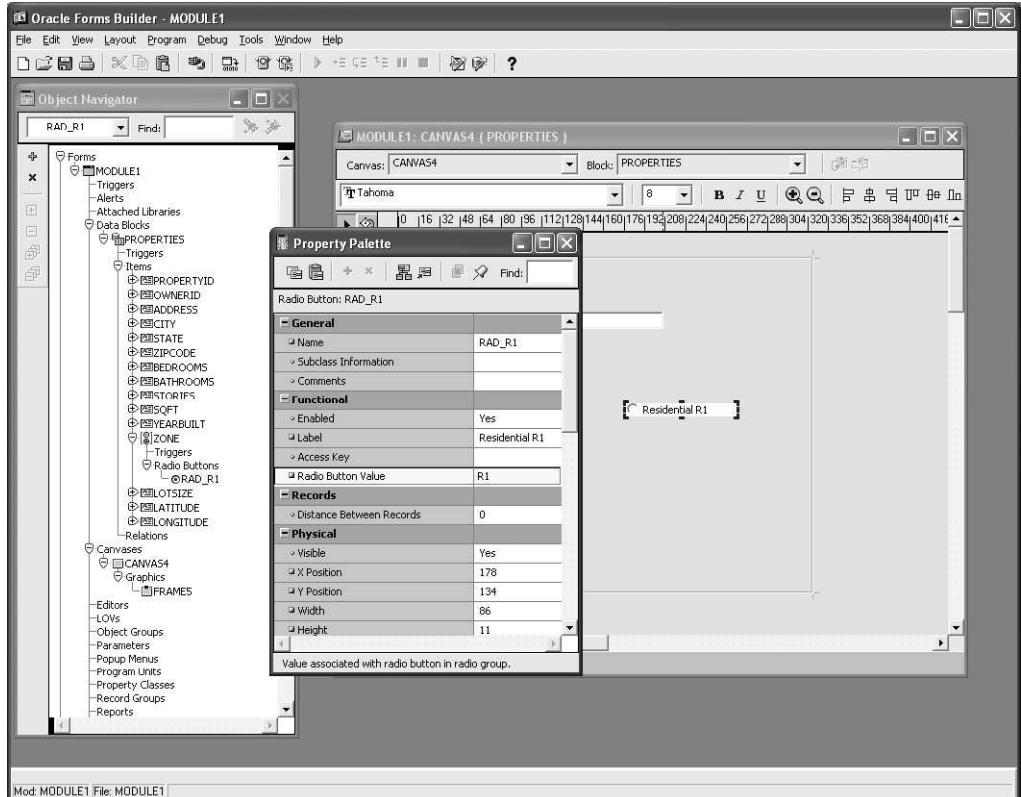


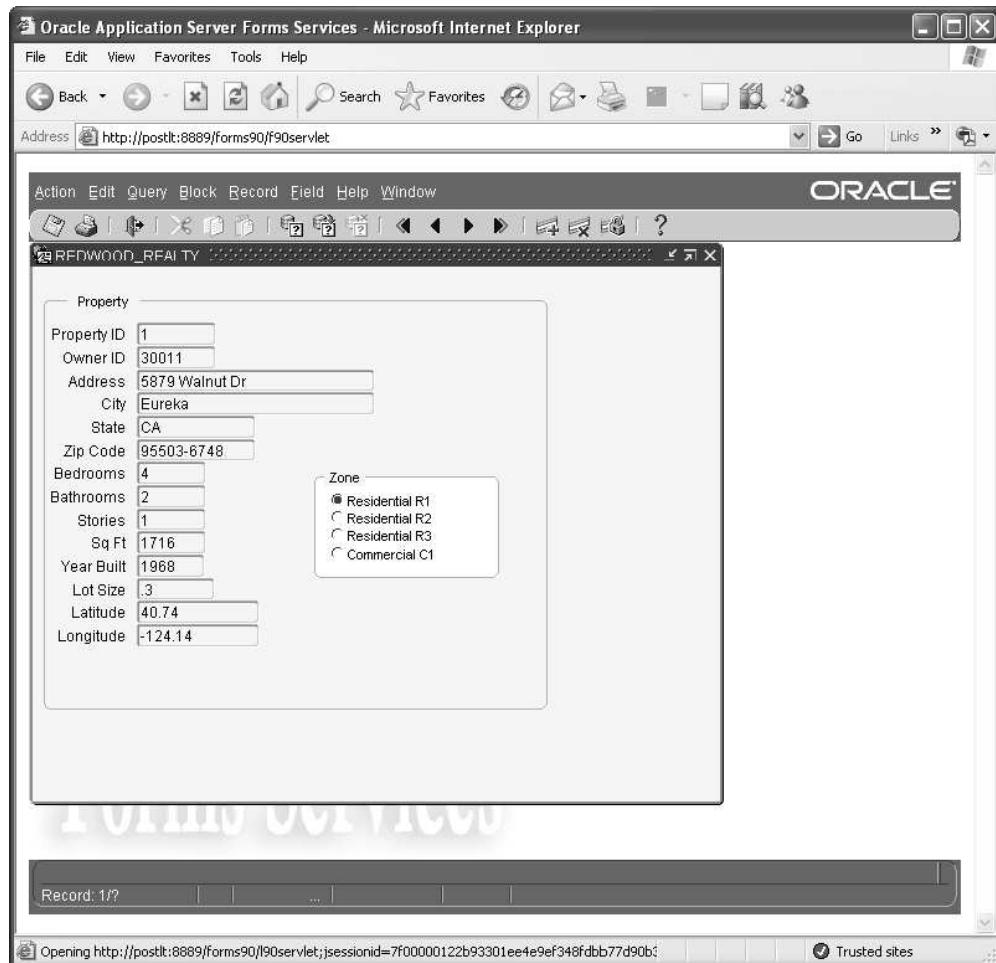
Рис. 8.24. Свойства переключателя

## Создание табличных и подчиненных форм

Формы, ориентированные на работу с одной строкой, очень популярны. Они представляют полный контроль над расположением элементов. Однако иногда пользователю необходимо видеть одновременно несколько строк данных. Для очень простых таблиц можно создавать табличные формы, которые лишь отображают строки и столбцы. Однако в более сложных ситуациях (они типичны для приложений электронной коммерции) потребуется реализовать подчиненную форму и связать ее с основной.

### Создание табличных форм

Табличная форма проста в использовании, и при ее создании, как правило, не возникают трудности. С точки зрения пользователя, табличная форма выглядит как обычный редактор таблиц, в котором данные располагаются по строками и столбцам.



**Рис. 8.25.** Переключатели в составе формы

Табличные формы оправданы лишь в тех случаях, когда следует отобразить ограниченное количество столбцов. Если число столбцов велико, пользователю для просмотра данных придется задействовать горизонтальную прокрутку. При этом трудно отследить, какая именно строка редактируется. Таким образом, создавать табличные формы имеет смысл, только если все столбцы помещаются на экране. Табличные формы используются при решении некоторых задач администрирования, например, для включения новых значений в списки поиска. Форма для отслеживания состояния лицензий — хороший пример табличной формы. Она содержит ограниченное число столбцов, и работать с ней удобно, если одновременно на экране отображается несколько строк. Несмотря на то что табличные формы просты в работе, используются они относительно редко.

Основные действия по созданию табличной формы в основном такие же, как и для формы, отображающей одну строку (по окончании каждого этапа необходимо щелкать на кнопке **Next**).

1. Запустите мастер блоков данных и выберите таблицу **LicenseStatus** и все ее столбцы.
2. Запустите редактор компоновки и выберите все столбцы.
3. Установите для полей, отображающих идентификаторы, ширину 48, а поля, предназначенные для представления текста, должны иметь ширину 176.
4. Вместо **Form** задайте тип компоновки **Tabular**.
5. Укажите число строк (20) и включите полосу прокрутки.

На рис. 8.26 показано одно из самых важных окон, используемых в процессе настройки. В нем задается число строк и включается полоса прокрутки. В табличной форме полоса прокрутки присутствует почти всегда, так как чаще всего строки, предназначенные для отображения, не помещаются на экране. Число одновременно отображаемых записей определяет размер формы. Это значение можно изменить в дальнейшем, установив соответствующее свойство блока данных.

На рис. 8.27 показан окончательный вид формы **License Status**. Запустив форму и щелкнув на кнопке **Execute Query**, вы увидите, что выполняющаяся форма выглядит практически так же, как и на рисунке. Пользователь может изменить данные в нескольких столбцах или строках, а также добавлять строки. Для этого достаточно ввести новые значения в требуемых позициях. Как вы помните, для того, чтобы сохранить изменения в базе данных, надо щелкнуть на кнопке **Save**. Если количество строк в таблице превышает число строк формы (в данном случае 20), необходимо воспользоваться полосой прокрутки. Чтобы проверить работу полосы прокрутки, вы можете уменьшить число отображаемых записей и повторно запустить форму.

## **Создание основной и подчиненной форм**

Сама по себе табличная форма используется лишь для очень простых таблиц. Однако она может входить в состав более сложных форм. Как разработчик приложений вы должны создавать формы, которые помогали бы пользователям решать стоящие перед ними задачи. Нередко это означает необходимость объединения в одной форме нескольких различных элементов. Так, например, в системе электронных платежей часто применяется форма, содержащая информацию о доступных товарах и, возможно, о потребителе. С ней связывается подчиненная форма, которая отображает выбранные товары. В качестве подчиненной обычно используется табличная форма. Данные в подчиненной форме связаны с основной формой. Для случая электронных платежей таблица **Sale Item** связывается с таблицей **Sale** посредством столбца **SaleID**.

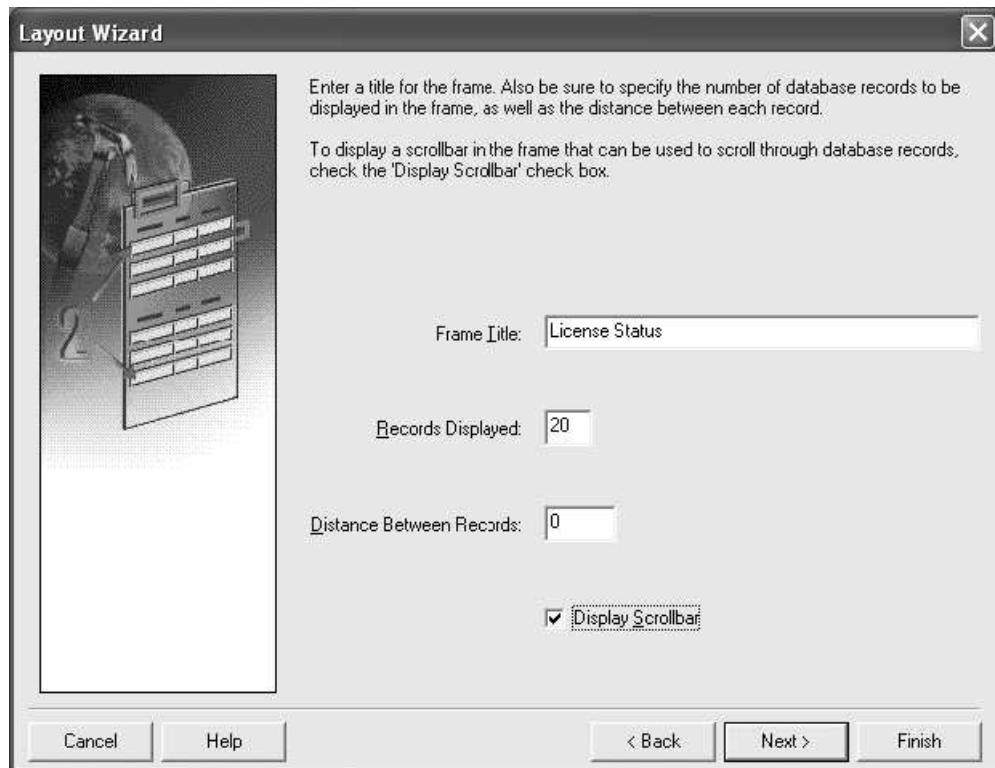
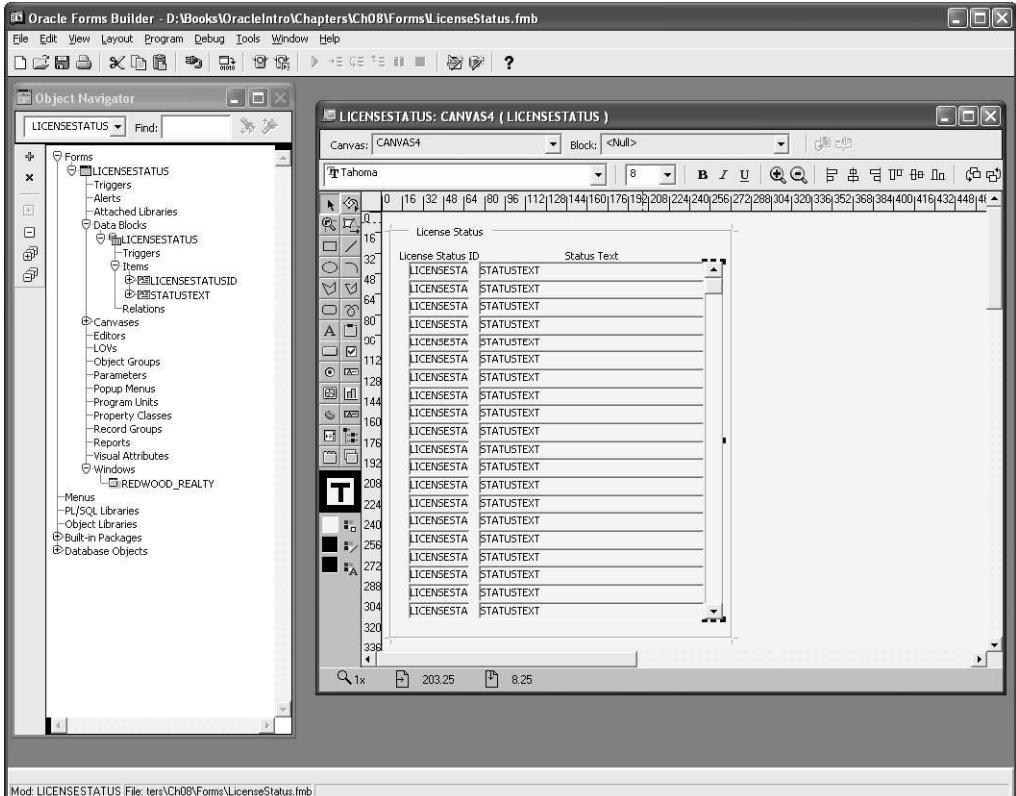


Рис. 8.26. Установка свойств табличной формы

Для приложения Redwood Realty можно использовать основную и подчиненную формы так, чтобы агенты и менеджеры могли записывать и просматривать информацию о контактах каждого агента. Просмотрев список таблиц, вы увидите две связанные таблицы: Agents и CustAgentList. Они связаны посредством AgentID. Таблица CustAgentList предназначена для хранения записей о контактах конкретного агента в процессе его профессиональной деятельности. Например, когда агент предоставляет потребителю список строений, в таблицу CustAgentList включается запись, содержащая поля AgentID, CustomerID и ListingID. Столбец ContactReason определяет цель контакта; в данной ситуации это продажа дома. Аналогично, в случае предложения другого потребителя агент записывает данные в поля CustomerID и ListingID, а также регистрирует предложенную цену. Поскольку большинство записей создается конкретными агентами, логично создать основную форму, в которой была бы представлена информация об агенте, и подчиненную форму, отображающую данные о контактах в виде таблицы.

Как и ранее, мастер блоков данных и мастер компоновки выполняют основную работу. Сначала надо создать основную форму, затем вернуться к мастеру блоков данных и добавить подчиненную табличную форму. Создайте начальную форму, за-



**Рис. 8.27.** Представление данных о лицензировании с помощью табличной формы

пустив мастер блоков данных и выбрав несколько столбцов из таблицы Agents. Как минимум, необходимо включить столбцы, содержащие идентификатор и имя агента. Выполните необходимые действия с диспетчером компоновки и упорядочьте требуемым образом выбранные столбцы. Оставьте на холсте свободное место ниже фрейма Agents. Здесь вы разместите подчиненную форму, отображающую десять строк, поэтому вам понадобится как минимум пара дюймов пространства.

**Совет.** Перед тем, как вызывать мастер блоков данных, щелкните на холсте, в противном случае мастер перейдет в режим редактирования исходного блока данных. Если это произойдет, завершите работу мастера и запустите его снова.

1. Создайте основную форму, отображающую одну строку таблицы Agents. Включите в нее столбцы AgentID, Title, FirstName, LastName и WorkPhone.
  2. Если возникнет необходимость, увеличьте размер холста. Запустите мастер блоков данных для выбора новой информации. Выберите таблицу CustAgentList и все ее столбцы.

3. Щелкните на кнопке **Create Relationship**, чтобы объединить таблицы **CustAgentList** и **Agents** посредством столбца **AgentID**.

Щелкните на холсте и снова запустите мастер блоков данных. На этот раз выберите таблицу **CustAgentList** и все ее столбцы. Система запросит у вас информацию о связи между новым блоком данных и существующим блоком **Agents**. Таблицы **CustAgentList** и **Agents** связываются посредством столбца **AgentID**. На рис. 8.28 показан результат установки связей. Чтобы добиться его, вам надо выполнить ряд действий. В случае ошибки щелкните на кнопке, удаляющей связи, и начните снова. Ваша задача — установить условия объединения так, чтобы в области **Join Condition** отображалась строка **CUSTAGENTLIST.AGENTID = AGENTS.AGENTID**. Если вы имеете дело только с двумя таблицами, требуемого результата можно добиться посредством опции **Auto-join data blocks**. Щелкните на кнопке **Create Relationship**. Если условия объединения созданы корректно, вы можете переходить к следующему окну. В противном случае удалите условия и выполните вручную следующие действия.

1. Сбросьте флажок **Auto-join data blocks**.
2. Щелкните на кнопке **Create Relationships**.
3. В контекстном меню выберите значение по умолчанию для объединения.
4. В списке выберите таблицу **Agents**.
5. В раскрывающемся списке **Detail Item** выберите столбец **AgentID**.
6. В раскрывающемся списке **Master Item** выберите столбец **AgentID**.

В мастере компоновки вам надо задать отображение всех столбцов таблицы **CustAgentList**. Установите начальную ширину столбцов с идентификаторами равной 48 пунктам. Убедитесь в том, что выбрана опция, соответствующая табличной форме. Задайте число записей, равное 10, и установите флажок, управляющий отображением полосы прокрутки.

7. Если связи установлены правильно, убедитесь еще раз, что все столбцы таблицы **CustAgentList** включены. Впоследствии вы сможете скрыть столбец **AgentID**, установив его ширину равной нулю. Задайте число записей, равное 10, и включите полосу прокрутки.

На рис. 8.29 показан исходный вариант новой формы. Заметьте, что вам, возможно, придется сократить некоторые из заголовков, чтобы таблица помещалась на экране. Сохраните форму и запустите ее для проверки. Обратите внимание, что при выборе новой записи в основной форме (**Agents**) содержимое подчиненной формы автоматически обновляется и в форме отображается информация о контактах выбранного агента. Такая зависимость имеет место потому, что вы связали два блока данных.

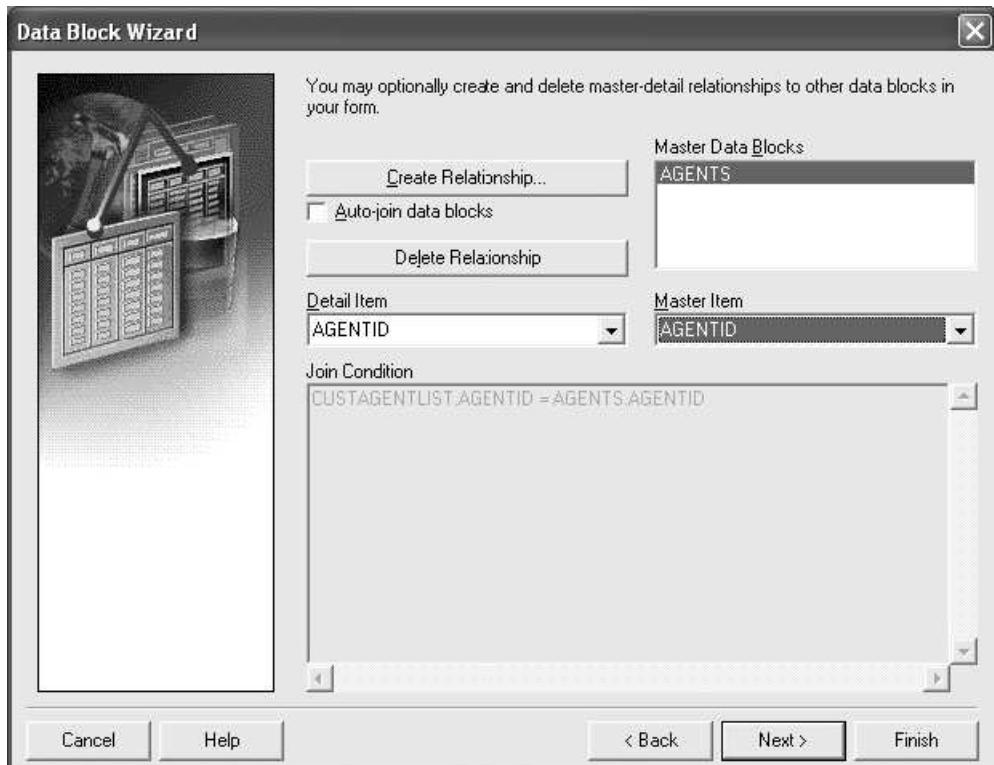


Рис. 8.28. Создание связи между основной и подчиненной формами

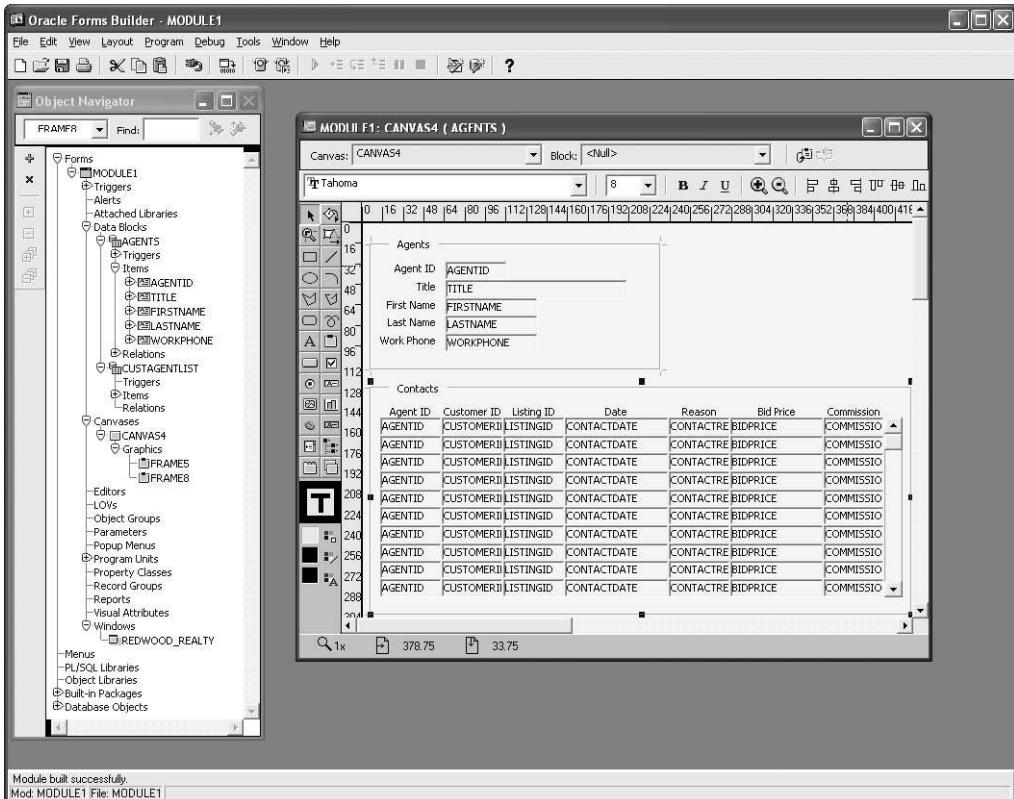
## Изменение внешнего вида таблицы

Несмотря на то что наша форма работоспособна, она требует внесения некоторых изменений. Обратите внимание, например, на то, что значение AgentID отображается в каждой строке. Поскольку данный столбец представлен в основной форме, в подчиненной его выводить не обязательно.

1. Установите ширину столбца AgentID равной нулю и удалите его заголовок.

Возможно, вы захотите реализовать поиск значений CustomerID и ListingID посредством списка значений, но мы не будем рассматривать здесь этот вопрос. Вы можете реализовать эту возможность самостоятельно. Мы же поговорим о том, что в форме целесообразно отображать имя потребителя, чтобы агенты видели не только идентификаторы, но и имена тех, с кем они ведут переговоры.

Подобно тому, как это было с формой, отображающей только одну строку таблицы, включить имя потребителя можно, используя запрос.



**Рис. 8.29.** Внешний вид основной и подчиненной форм

2. Откройте палитру свойств для блока данных `CustAgentList` и измените значение свойства `Data Source` так, чтобы оно выглядело следующим образом:

```
(SELECT CUSTAGENTLIST.CUSTOMERID, AGENTID, LISTINGID,
CONTACTDATE, CONTACTREASON, BIDPRICE, COMMISSIONRATE,
LASTNAME, FIRSTNAME FROM CUSTAGENTLIST INNER JOIN CUSTOMERS
ON CUSTAGENTLIST.CUSTOMERID=CUSTOMERS.CUSTOMERID)
```

Убедитесь в том, что выражение `SELECT` помещено в круглые скобки.

3. Поскольку запрос предоставляет фамилию и имя потребителя, вам надо добавить два соответствующих пункта в список `Query Data Source Columns` блока данных.

После того как столбцы будут включены в блок данных, их можно добавить к форме. Проще всего скопировать существующий столбец и вставить его в конец таблицы. Для копирования больше подходит столбец `ContactReason`, поскольку он содержит текстовые данные и вам придется изменить лишь некоторые свойства.

**ТАБЛИЦА 8.2.** Свойства, которые надо изменить, чтобы отобразить столбец на основе запроса

| Свойство    | Значение   |
|-------------|------------|
| Name        | FIRSTNAME  |
| Enabled     | No         |
| Column Name | FIRSTNAME  |
| Query Only  | Yes        |
| Prompt      | First Name |

- Выберите столбец ContactReason и нажмите комбинацию клавиш <Ctrl+C>, чтобы скопировать его. Нажав комбинацию клавиш <Ctrl+V>, вставьте копию столбца. Мы превратим этот новый столбец в FirstName; для этого надо установить свойства так, как это показано в табл. 8.2.
  - Повторите п. 4, чтобы создать столбец LastName, скопировав столбец FirstName.
  - Для свойства Enable столбцов с именами надо установить значение No, чтобы пользователи не пытались изменить их содержимое.
- Запустите форму и проверьте ее работу. Убедитесь, что вы имеете возможность добавлять строки и редактировать существующие данные.

## Установка маски формата

При отображении данных в формах обычно используется формат, принятый по умолчанию, аналогичный тому, который был получен в результате запроса. В некоторых случаях возникает необходимость изменить порядок отображения данных. Например, при выводе сведений о контактах агента желательно представлять цены в виде денежных единиц. Сделать это несложно, понадобится лишь изменить два свойства.

- Для элемента BidPrice установите значение End свойства Justification.
- Установите значение FML99G999G999 свойства Format Mask.

Установка значений свойств не составит труда. Несколько сложнее понять, что такое *маска формата* (format mask). Нажмите клавишу <F1> в тот момент, когда курсор находится в поле свойства. Oracle предоставит вам текст с объяснением основных понятий. На первый взгляд маска для вывода данных в виде денежных единиц выглядит несколько странно. Сразу же возникает вопрос, что означает буква “L” и почему не использовать вместо нее знак доллара. Буква “L” включает локальный символ валюты для той страны, где в текущий момент находится компьютер (это может быть тот же знак доллара). Аналогично, вы можете использовать запятую для разделения

**ТАБЛИЦА 8.3.** Символы, используемые в масках формата

| Символ форматирования | Действие                                |
|-----------------------|-----------------------------------------|
| x                     | Любая буква или цифра                   |
| A                     | Только буквы                            |
| 9                     | Только цифры                            |
| 0                     | Только цифры; ведущие нули отображаются |
| YYYY                  | Год, задаваемый четырьмя цифрами        |
| MM                    | Год, задаваемый двумя цифрами           |
| DD                    | День месяца                             |

групп цифр, однако буква “G” задает автоматический выбор разделителя в соответствии с соглашениями, принятыми в конкретной стране. Если вам надо отделить дробную часть, то лучше вместо десятичной точки ввести букву “D”. Согласно стандарту США, числа будут иметь вид наподобие следующего: 123,456.78. В некоторых европейских странах то же значение будет выведено как 123.456,78. Префикс FM означает, что числа вводятся в режиме заполнения, т.е. пользователь может вводить числа короче, чем это указано в маске. Символы FM присутствуют в большинстве масок; их нет только тогда, когда необходимо, чтобы пользователь ввел определенное количество цифр.

Как видно из табл. 8.3, маску формата можно задать почти для любого типа данных, например чисел, символьных последовательностей и дат. Использовать маски формата удобно, но при этом надо соблюдать осторожность, так как они накладывают ограничения на вводимые данные. Например, вы можете предусмотреть ZIP-код, состоящий из пяти цифр, а пользователю потребуется ввести 9-значный код для Канады. Кроме того, в европейских странах этот же код может включать буквы и пробелы. Аналогичные проблемы могут возникать при вводе телефонных номеров.

## Резюме

Большинство пользователей не справляются с задачей непосредственного ввода данных в таблицу. В реляционных базах таблицы связаны посредством первичных ключей. Запомнить такой ключ и корректно ввести его достаточно трудно. Для упрощения работы пользователя создаются формы. Пользователи вводят данные в формы приблизительно так же, как они поступают с формами, напечатанными на бумаге. Данные, отображаемые в формах, хранятся в таблицах. При необходимости можно реализовать специальные возможности поиска данных пользователями, например, применять списки значений. Такой список включает первичный ключ в требуемую таблицу, и пользователь избавляется от необходимости запоминать его. В формах можно также использовать флагги опций и переключатели, которые в ряде случаев упрощают

ввод данных. В формах Oracle для поиска записей могут быть использованы запросы, можно также организовать фильтрацию строк с данными.

Существуют три основных типа форм: основная форма, отображающая одну строку данных, табличная форма и сочетание основной и подчиненной формы. Более сложные формы можно создавать, объединяя элементы перечисленных выше типов. Например, можно создать основную форму, с которой будут связаны две или три подчиненные формы.

Процесс создания форм существенно упрощают мастер блоков данных и мастер компоновки. Мастер LOV позволяет создавать списки значений путем выбора нескольких элементов базы данных. После генерации исходного варианта формы вы имеете возможность изменить расположение элементов и формат вывода данных путем установки различных свойств. Существуют инструменты для выравнивания и организации объектов в составе формы. Необходимо лишь помнить, что для произвольного размещения элементов надо выбрать режим *Manual*, в противном случае внесенные изменения будут отвергнуты.

Блок данных, соответствующий форме, поддерживает запись и редактирование только для одной таблицы. Если при заполнении формы используется запрос, соответствующие поля будут доступны только для чтения. Благодаря наличию первичных ключей появляется возможность извлекать данные из связанных таблиц. Например, в большинстве случаев предпочтительнее отображать имя пользователя вместо его идентификатора. В качестве источника данных для блока можно применять запрос, помещенный в круглые скобки. Есть возможность включать в состав формы дополнительные столбцы, в которых можно отображать, например, имена потребителей. Следует однако помнить, что дополнительные столбцы не могут входить в состав первичного ключа другой таблицы и что отображаемые в них данные доступны только для чтения.

Процесс создания исходных форм упрощают два инструмента типа “мастер”. Начиная работу над проектом, необходимо выделить время на упорядочение всех деталей так, чтобы форма была понятной для пользователя и простой в работе.

## Основные термины

- JInitiator
- OC4J
- Блок данных
- Группа записей
- Динамический обработчик
- Маска формата
- Мастер блоков данных
- Мастер компоновки

- Обработчик события
- Объектный навигатор
- Палитра свойств
- Панель инструментов
- Переключатель
- Пункт (единица измерения для шрифтов)
- Список значений (list of values – LOV)
- Табличная форма
- Флажок опции
- Форма
- Холст
- Элемент (формы)

## **Повторение пройденного материала**

### **Истина или ложь?**

1. Мастер компоновки надо запустить перед началом работы с мастером блоков данных.
2. Со столбцами внешних ключей в формах должны быть связаны списки значений.
3. Табличная форма обычно используется для ввода данных в таблицы, содержащие большое количество столбцов.
4. Флажки опций используются для представления взаимоисключающих значений.
5. Пользователи могут работать с формами Oracle посредством любого стандартного браузера.

### **Заполнить пропущенное**

1. Свойство \_\_\_\_\_ запрещает автоматическое упорядочение элементов формы.
2. Когда создается список значений, запрос SELECT для списка хранится в разделе \_\_\_\_\_ объектного навигатора.
3. Создавая группу переключателей с помощью мастера компоновки, для элемента, который содержит выбранное значение, надо изменить значение свойства \_\_\_\_\_ с Text Item на Radio Group.
4. Для отображения даты по соглашениям США (1/31/2006) надо использовать маску формата \_\_\_\_\_.
5. Основная и подчиненная формы создаются в том случае, когда между таблицами существует отношение \_\_\_\_\_.

## Варианты ответов

1. Какой объект связывает элементы формы Oracle с базой данных?
  - а) Группа записей
  - б) Холст
  - в) Блок данных
  - г) JInitiator
  - д) Фрейм
2. Какой тип формы лучше всего подходит для редактирования таблицы базы данных с несколькими столбцами (например, таблицы Customer)?
  - а) Основная форма, отображающая одну строку.
  - б) Табличная форма
  - в) Основная и подчиненная формы
  - г) Форма LOV
  - д) Ни одна из перечисленных выше форм.
3. Процесс создания основной формы с помощью инструментов типа “мастер” состоит из следующих этапов.
  - а) Мастер компоновки, упорядочение элементов формы, мастер блоков данных.
  - б) Редактирование содержимого холста, мастер блоков данных, мастер компоновки.
  - в) Мастер компоновки, мастер блоков данных, упорядочение элементов формы.
  - г) Мастер блоков данных, мастер компоновки, упорядочение элементов формы.
  - д) Редактирование содержимого холста, мастер компоновки, мастер блоков данных.
4. Процесс создания основной и подчиненной форм включает следующие этапы.
  - а) Создание основной формы для отображения одной строки, добавление подчиненной формы посредством нового мастера блоков данных, создание связи между формами.
  - б) Создание блока данных для основной формы, размещение элементов основной формы, создание связи между формами.
  - в) Размещение элементов основной формы, размещение элементов подчиненной формы, создание блоков данных для обеих форм, связывание блоков данных.
  - г) Размещение элементов основной формы, создание блока данных подчиненной формы, связывание блоков данных.
  - д) Размещение элементов основной формы, размещение элементов подчиненной формы, создание списка значений для связывания форм.

5. Для того чтобы включить в форму столбцы, отображающие данные из другой таблицы, надо:
  - а) Запустить мастер списков значений.
  - б) Заменить значение Query Data Source Name тщательно подготовленным запросом SELECT.
  - в) Добавить новый холст
  - г) Создать блок данных на основании запроса, включающего столбцы из обеих таблиц.
  - д) Добавить подчиненную форму посредством связи “многие ко многим”.

## Упражнения

### 1. Readwood Realty

Агенты из компании Redwood попросили вас создать форму, которая отображала бы состояние указанного листинга. Им надо искать конкретные листинги и просматривать основную информацию о ценах и объектах для продажи. Они также хотят получать дополнительные данные о жилых домах, например, число спальных и ванных комнат, а также площадь дома, выраженную в квадратных футах. Кроме того, им нужны списки предлагаемых объектов. Форма, которую вам придется создать, основана на таблице Listings, но в ней также должны присутствовать некоторые поля таблицы Properties. Вам понадобится подчиненная форма, в которой будут отображаться данные из таблицы CustAgentList. Поиск будет осуществляться посредством запросов. Например, агент будет иметь возможность ввести максимальную цену и искать дома, которые еще не были проданы.

1. Запустите Forms Builder и активизируйте мастер блоков данных. С его помощью вы создадите исходную форму, базой для которой послужат все столбцы таблицы Listings.
2. Упорядочьте элементы. В частности, расположите начальную и конечную даты одну за другой.
3. Снова запустите мастер блоков данных и добавьте все столбцы таблицы CustAgentList. Убедитесь, что вы задали связь посредством столбца ListingID. Также удостоверьтесь в том, что выбрана табличная форма с несколькими отображаемыми записями и полосой прокрутки. Отредактируйте свойства блока данных и установите значение свойства ORDER BY равным CONTACTDATE. Сохраните форму под именем ListingActivity.
4. Для блока данных Listings модифицируйте Query Record Source Name так, чтобы вы могли включить информацию о некоторых свойствах. Перед тем как устанавливать запрос в качестве значения свойства, проверьте его в SQL\*Plus.

```
(SELECT ListingID, Listings.PropertyID, ListingAgentID,
SaleStatusID, BeginListDate, EndListDate, AskingPrice,
HousePhoto, Bedrooms, Bathrooms, Stories, SQFT, YearBuilt
FROM Listings INNER JOIN Properties
ON Listings.PropertyID=Properties.PropertyID)
```

5. Добавьте к Query Data Source Columns значения BEDROOMS, BATHROOMS, STORIES, SQFT и YEARBUILT. Задайте для них тип данных Number.
6. Создайте новые поля редактирования для новых элементов. Установите следующие свойства: Name, Enabled (No), Database Item (Yes), Column Name, Query Only (Yes), Prompt.
7. В основной форме Listings добавьте LOV к Property ID. Запустите мастер LOV. Выберите из таблицы Properties PropertyID, Address, City, Bedrooms, Bathrooms и YearBuilt. При необходимости измените размеры столбцов. Убедитесь, что значение Return для столбца PropertyID установлено равным Listings.PropertyID. Переименуйте LOV и Record Group.Save и проверьте работу формы.
8. Создайте список значений для поля AgentID формы Listings. Затем создайте список значений для поля CustomerID подчиненной формы. Убедитесь в том, что LOV и группы записей переименованы. В поле AgentID подчиненной формы установите для свойства List of Values созданный вами LOV\_AGENT.
9. Запустите форму на выполнение. Введите запрос, например 1990 для Year Built. Выбрав Action⇒Print, выведите форму на печать. Преподаватель, возможно, предложит вам поместить файл ListingActivity.fmb в папку, предназначенную для совместного использования.

## 2. Coffee Merchant

Поскольку вы не собираетесь предлагать пользователю вводить данные непосредственно в таблицы, вам надо создать формы для приложения Coffee Merchant. Первое, что надо сделать, — это выбрать тип формы, которая лучше всего соответствовала бы каждой таблице. Проанализировав таблицы, легко увидеть, что две из них (*States* и *Countries*) — это обычные поисковые таблицы, которые редко претерпевают изменения. Для них можно создать табличные формы. Кто-то из сотрудников будет отвечать за изменения, которые осуществляются приблизительно раз в год. Данные в таблице *Employee* будут изменяться гораздо чаще. Очень важно также, что эта таблица содержит несколько столбцов и что необходимость редактировать одновременно несколько записей возникает крайне редко. Таким образом, форма *Employee* должна быть основной и отображать одну строку таблицы. Аналогичные рассуждения можно провести для таблиц *Consumers* и *Inventory*. Остаются таблицы *Orders* и *OrderLines*. Они типичны для бизнес-приложений и связаны отношением “один

ко многим” посредством столбца OrderID. Для поддержки данной зависимости вам потребуются основная и подчиненная формы.

1. Удалите и установите снова базу данных, запустив файл BuildCoffe.sql.
2. Начните работу с простых форм. Запустите Forms Builder и с помощью мастера блоков данных и мастера компоновки создайте табличные формы для таблиц States и Countries. Вероятно, вы захотите отображать в каждой форме двадцать строк или даже больше, однако, принимая решение, учитывайте размер и разрешающую способность вашего монитора и используемый браузер.
3. Основная форма, отображающая одну строку таблицы, создается достаточно просто. При построении форм для таблиц Employees и Consumers используйте мастер блоков данных. Для выбора некоторых данных, например возрастной группы сотрудника, надо использовать переключатели опций. Создавая группы переключателей, учитывайте формат данных в столбце.
4. Таблица Inventory достаточно мала и может отображаться с помощью табличной формы, но подойдет также форма, представляющая только одну строку. Работая над реальным приложением, имеет смысл создать оба типа форм и предложить пользователям выбрать, какое представление больше подходит для них. Можно также оставить в приложении обе формы и дать пользователю возможность выбирать конкретное представление данных. Исходя из этих соображений, создайте форму Inventory дважды: один раз в виде табличной формы, а второй — как основную форму для вывода одной строки. Создайте список значений, чтобы помочь пользователям выбирать значение CountryID.
5. Чтобы создать форму Orders/OrderLines, начните с основной формы, которая будет основываться на всех столбцах таблицы Orders. Добавьте списки значений для ConsumerID и EmployeeID.
6. Снова запустите мастер блоков данных, чтобы добавить подчиненную табличную форму для OrderLines. Сохраните форму под именем Orders.
7. Для блока данных Orders создайте запрос и свяжите его с Query Record Source Name. С помощью этого запроса будут извлекаться имя, фамилия и название города. Отобразите новые значения в форме Orders в виде полей, предназначенных только для чтения.
8. Для блока данных OrderLines создайте запрос и свяжите его с Query Record Source Name. Этот запрос будет извлекать значения Name и Price из таблицы Inventory. Добавьте новые столбцы к подчиненной форме и отформатируйте столбец, представляющий цену.
9. Выведите форму Orders на печать и предъявите преподавателю. Возможно, он потребует копии всех форм.

### 3. Rowing Ventures

Вам необходимо создать формы, которые упростят ввод данных в таблицы базы Rowing Ventures. В основном действия с базой данных выполняются либо при регистрации участников, либо после гонки. При регистрации потребуются формы для таблиц Person и Organization. Каждая из этих форм будет отображать одну строку. Проанализировав определения таблиц, вы увидите, что между Boat и BoatCrew существует отношение “один ко многим”. Таким образом, вам надо создать для этих двух таблиц основную и подчиненную формы. Обеспечьте использование LOV для выбора членов экипажа. Кроме того, в списке вам надо представлять не только PersonID, но и имя конкретного члена экипажа. В основной форме для гонки организуйте выбор значения RaceCategory с помощью группы переключателей. Перед тем как приступать к созданию переключателей, напишите запрос, предназначенный для выбора реальных данных.

Результаты проще вводить отдельно для каждой гонки. Кто-то из сотрудников должен заранее определить исходные данные для гонки и выбрать лодки. Если такая подготовительная работа проведена, то пользователь, ответственный за регистрацию результатов, видит все лодки и ему остается лишь задать время финиша и место участника. Вам необходимо создать основную и подчиненную формы для таблиц Race и RaceTimes. В подчиненной форме организуйте выбор значения BoatID посредством LOV. Также необходимо вывести значения BowNumber и Organization для каждой лодки. Для выбора лодок следует создать список значений. Если вам надо включить в список значений название организации, то придется создать представление данных из таблиц Boat и Organization и сформировать LOV на их базе.

### 4. Broadcloth Clothing

Приложение Broadcloth Clothing оперирует с несколькими таблицами. Для того чтобы создать все формы, необходимые в данном случае, потребуется достаточно много времени, поэтому мы сосредоточим внимание лишь на некоторых из них. Начнем с форм на стороне потребителя. Создайте формы для Customer, CustomerOrder, OrderItem, Item и Model. Вам также придется создать табличные формы для поиска в таблицах PrimaryLanguage и ColorList. Перед созданием таблиц Item и Model убедитесь в том, что вы правильно представляете себе связи между ними. Как вы помните, таблица Model ссылается на тип товара, а таблица Item — на конкретную разновидность модели. Например, женская блузка соответствует определенной модели, для которой существуют разновидности, полученные путем изменения цвета и размера. В тех случаях, когда это оправдано, создайте списки значений и группы переключателей.

# Специальная настройка форм

**В этой главе...**

- Создание и модификация форм без использования мастеров
- Поддержка событий, связанных с формой
- Вычисление промежуточных результатов
- Использование последовательностей в формах
- Включение в форму нескольких холстов

Инструменты типа “мастер”, предоставляемые Oracle, удобны в использовании. Они позволяют быстро создавать формы, извлекать данные из таблиц и редактировать их. Однако в некоторых случаях формы необходимо дополнительно настраивать с учетом специфики приложений. При этом форма по-прежнему должна взаимодействовать с базой, например, извлекать данные для заполнения списка значений или формировать отображаемые строки посредством запросов. Исходный вариант формы можно создать с помощью мастеров, а затем настроить его в соответствии с конкретными требованиями. В некоторых случаях проще создать форму с нуля и включить в нее необходимые элементы. В любом случае необходимо представлять себе общую структуру формы Oracle, знать, какие ее части должны подвернуться изменениям и как поместить в нее необходимые элементы.

Предположим, например, что сотрудникам компании Redwood Realty потребовалась простая поисковая страница, в которой они могли бы находить списки объектов по заданным свойствам, таким как число ванных или спальных комнат либо цена. Для этого можно организовать курсы по обучению правилам составлению запросов для поиска списков. Однако обычно агенты по продаже недвижимости очень заняты,

|                          |                                   |                          |                          |                          |                          |                          |                          |
|--------------------------|-----------------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Bedrooms                 | 2 3 4 5 or more                   |                          |                          |                          |                          |                          |                          |
| Bathrooms                | 1 1.5 2 2.5 3 4 or more           |                          |                          |                          |                          |                          |                          |
| Sq Feet                  | 1000 2000 3000 4000 or more       |                          |                          |                          |                          |                          |                          |
| Age/Built                | 1900 1970 1980 1990 2000 or later |                          |                          |                          |                          |                          |                          |
| Price                    | Min:                              | Max:                     |                          |                          |                          |                          |                          |
| Sale Status              | For sale Pending                  | Sold                     |                          |                          |                          |                          |                          |
| <b>Matches</b>           |                                   |                          |                          |                          |                          |                          |                          |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| <input type="checkbox"/> | <input type="checkbox"/>          | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

**Рис. 9.1.** Эскиз поисковой формы

поэтому более логичным решением будет создать простую в работе форму, которая предоставит именно те возможности, которые им нужны и для пользования которой не требуется длительное обучение. Перед тем как приступить к работе с Forms Builder, желательно создать упрощенную схему, которая давала бы представление об основных элементах формы. Желательно показать ее потенциальным пользователям и при разработке формы учесть их отзывы. Включать в форму новые элементы и вносить изменения удобнее на первых этапах разработки. Впоследствии можно скорректировать взаимное расположение элементов; передвигать их в пределах формы несложно. Пересмотр же запросов и логики работы формы обычно требует много времени. Пример наброска, который дает представление о внешнем виде формы, показан на рис. 9.1.

Для того чтобы создать форму и настроить ее так, как это требует специфика приложения, необходимо понимать структуру форм Oracle. Для создания исходной конструкции можно применять мастера, но модифицировать форму, не зная назначения ее основных частей, невозможно. На рис. 9.2 условно показаны основные компоненты формы: холст, блок данных и отображаемые элементы. Все они рассматривались в главе 8, там же мы обсуждали их создание средствами мастеров. Сейчас же пришло время для более детального рассмотрения этих компонентов.

Холст служит полем для элементов, которые представляются пользователю. В составе элементов может быть текст, например заголовки, и даже изображения. Очень важно помнить, что формы содержат разнообразные элементы, которые могут быть

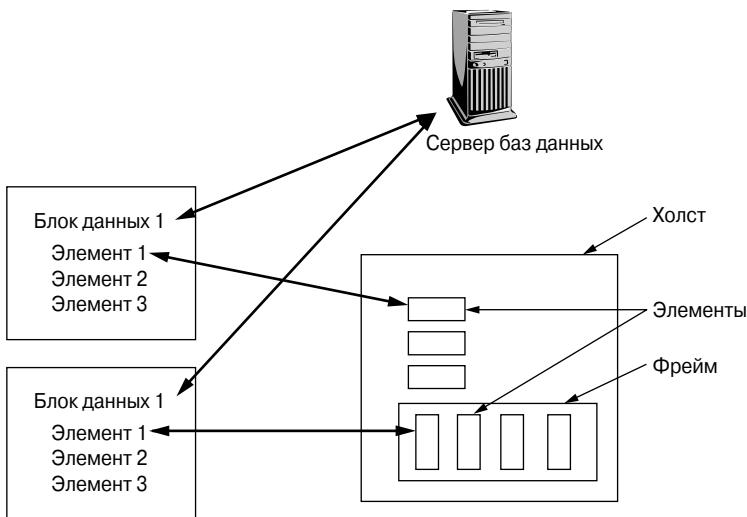


Рис. 9.2. Структура формы Oracle

связаны с базой данных. Эти элементы позволяют отображать и редактировать содержимое таблиц. Самый простой из подобных элементов — поле редактирования; оно непосредственно выводит данные из базы. В форме также часто имеются флажки опций, переключатели и списки. В каждой форме имеется по крайней мере один холст; элементы формы выбираются на панели инструментов и размещаются на холсте. Для настройки внешнего вида холста (например, указания цвета фона) и управления элементами применяется палитра свойств.

Фреймы служат для группировки элементов. Группировка влияет только на внешний вид формы и не затрагивает таблицы с данными. Автоматическая компоновка упрощает выравнивание групп полей редактирования; часто для достижения требуемого результата разработчику достаточно изменить размеры фрейма. Фреймы нередко используются для объединения связанных между собой переключателей так, чтобы с первого взгляда стало ясно, что они составляют одну группу. С помощью фреймов можно также распределять данные по различным разделам. В частности, с их помощью удобно выделять подчиненные табличные формы.

Холсты и фреймы — основные инструменты, управляющие внешним видом формы. Форму можно связать с базой данных, и извлекать из базы информацию. Для этой цели используются блоки данных, которые обеспечивает связь с базой. Связь может быть установлена непосредственно с таблицей, и для этой цели можно применить хранимые представления и даже хранимые процедуры. Конкретное решение зависит от того, планируется ли обновление данных в базе. Если возможно, следует использовать непосредственное соединение с таблицей. Представления и запросы применимы тогда, когда надо отображать в форме взаимосвязанные данные из нескольких таблиц и при этом данные не должны изменяться.

Необходимость извлечения информации из нескольких таблиц или редактирования данных определяет структуру многих форм Oracle. Основное правило таково: если надо записывать данные в базу, необходимо иметь блок данных, связанный с соответствующей таблицей. Необходимо помнить, что каждый блок данных может обновлять информацию только в одной таблице. Таким образом, если вам надо обновлять содержимое нескольких таблиц, для каждой из них нужен блок данных. Если вам необходимо лишь отображать информацию, можно написать запрос для получения требуемых сведений, но в этом случае данные нельзя будет редактировать и сохранять в базе.

Одна из особенностей форм Oracle состоит в том, что каждый элемент, помещенный в форму, связывается с одним блоком данных. Установить связь с блоком или изменить ее можно путем установки соответствующего свойства. Тот же результат можно получить более простым способом. Перед тем как включать в форму новый элемент, установите текущий активизированный блок данных, выбрав его в раскрывающемся списке, который расположен в верхнем правом углу окна конструктора форм. Этот подход позволяет быстро связать с блоком данных несколько элементов.

## Создание холста и простого блока данных

Создание поисковой формы для компании Redwood Realty начинается с выполнения следующих действий.

1. Запустите Forms Builder и создайте новую форму. Присвойте модулю имя `Search`.
2. Разверните раздел `Windows` и переименуйте `Window1` в `Search`, откройте палитру свойств и задайте заголовок `Redwood Realty`.
3. Добавьте к форме новый холст, выбрав в объектном навигаторе раздел `Canvases` и щелкнув на кнопке `Add` (на ней изображен знак +). Присвойте новому холсту имя `Search_Canvas`. Возможно, вы захотите изменить цвет фона, так как по умолчанию предлагается слишком темный цвет. Попробуйте сделать фон белым.
4. Добавьте новый блок данных, который будет использоваться при поиске. Выберите раздел `Data Blocks` в объектном навигаторе и щелкните на кнопке `Add`. Вместо того чтобы пытаться воспользоваться мастером, убедитесь в том, что выбрана опция `Build a new data block manually`. Этот блок данных не связан с базой, поэтому измените свойства, задав значение `Name` равным `Search_Block`, а `Database Data Block` — `No`.

На данном этапе в нашем распоряжении есть простая форма с пустым холстом и блоком данных. Мы можем включать в форму основные элементы, предназначенные для поиска. Блок данных не связан с базой, поскольку пользователи будут лишь выбирать пункты заранее определенного списка. После того как пользователь сделает

выбор, необходимо сформировать запрос, который извлечет значения, соответствующие заданным условиям. Эти значения будут отображаться посредством нового блока данных, который мы создадим немного позже. Сейчас же сосредоточим внимание на элементах, предоставляющих пользователю возможность выбора.

Большинство элементов, которые мы разместим в верхней части формы, достаточно просты, так как они не связаны с базой данных. Чтобы контролировать действия пользователя, элементы выполнены в виде групп переключателей. Таким образом, пользователь может выбрать только одно значение в каждой категории. Прежде всего сформируем заголовок, включив в форму обычную текстовую метку.

Процедура создания группы переключателей одна и та же для четырех основных категорий (количество спален, количество ванных комнат, площадь в квадратных футах и год, в котором дом был построен). Создание фрейма не вызовет затруднений. Гораздо сложнее выбрать для элементов согласованные имена.

1. Создайте фрейм. Если вы хотите пользоваться им в дальнейшем, присвойте ему имя `FRM_BEDROOMS`. Для свойства, определяющего текстовую метку, задайте значение `Minimum Bedrooms`.
2. Поместите внутрь нового фрейма кнопки переключателя. Включая первую кнопку новой категории, удостоверьтесь, что она будет принадлежать соответствующей группе. С помощью объектного навигатора переименуйте группу переключателей и присвойте ей имя `RADIO_BEDROOMS`.
3. Установите для новой кнопки переключателя следующие свойства: `Name=RAD_BED_2`, `Label=2`, `Radio Button Value=2` и `Width=20`.
4. Создайте копию кнопки переключателя. Поместите ее в требуемую позицию и задайте такие же свойства, которые были указаны выше, за исключением того, что в первых трех свойствах символ 2 надо заменить на 3. Сделайте то же самое для остальных кнопок, чтобы сформировать группу, позволяющую выбрать 2, 3, 4 и 5 спальных комнат. Внимательно проверьте значения и текстовые метки. Убедитесь, что все кнопки помещаются внутри фрейма.
5. Выберите каждую из кнопок и, используя `Layout⇒Align Components`, установите одинаковое расстояние между ними по горизонтали и выровняйте их нижние края.
6. С помощью объектного навигатора откройте окно свойств для группы переключателей. Установите свойство `Initial Value=3`.

Создав группу переключателей для выбора количества спален, повторите ту же процедуру для количества ванных, площади в квадратных футах и года постройки. Приступая к работе с первым элементом категории, убедитесь, что вы выбрали опцию, определяющую создание новой группы переключателей. Также не забудь-

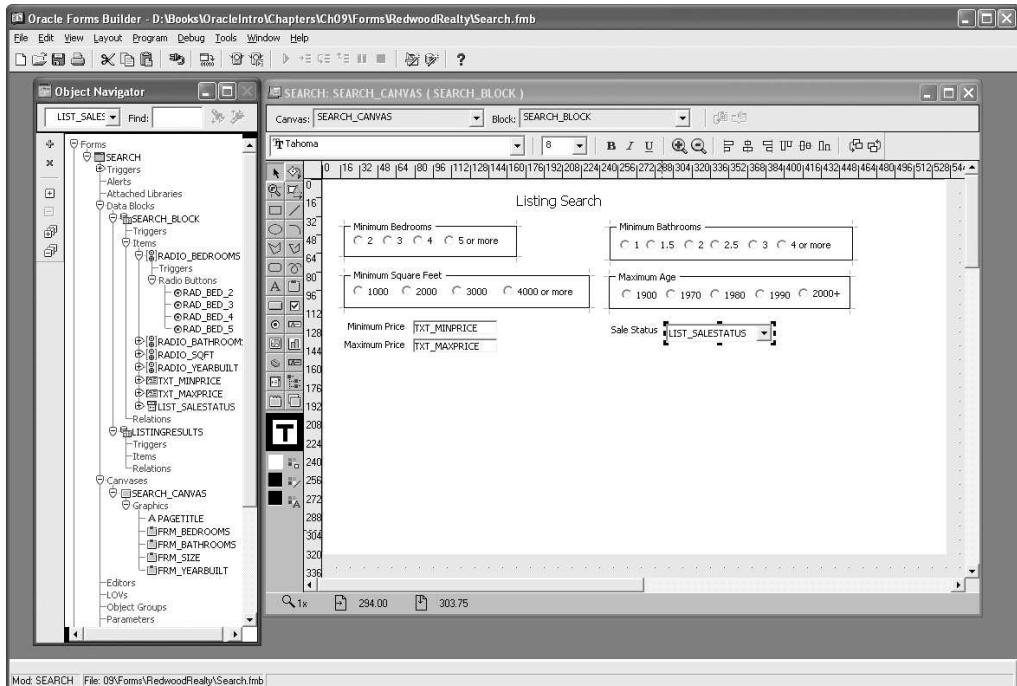
те установить начальное значение для каждой группы. Создав таким образом одну или две категории, вы потеряете интерес, и дальнейшая работа покажется вам утомительной, но она поможет вам лучше запомнить все особенности формирования групп переключателей.

Следующий шаг — включение полей редактирования, определяющих минимальную и максимальную цену. Возможно, вы посчитаете более уместным применить и в этом случае группу переключателей или воспользоваться списком. Однако поступать так не стоит. Каждый человек по-своему устанавливает диапазоны цен, и вам вряд ли удастся определить категории, которые устраивали бы всех. Лучше всего в такой ситуации прибегнуть к обычным полям редактирования. Но для того, чтобы упростить работу с ними, установите начальную величину, а также максимальное и минимальное значения.

1. Добавьте поле редактирования и назовите его `TXT_MinPrice`.
2. Установите следующие свойства: `Data Type=Number`, `Initial Value=0`, `Lowest Allowed Value=0`, `Highest Allowed Value=10000000`, `Prompt=Minimum Price`, `Prompt Attachment Offset=7`.
3. Аналогичным образом создайте поле `TXT_MaxPrice`.

Элемент `Sale Status` несколько отличается от предыдущих. В таблице `SaleStatus` содержится список возможных значений. Ключом для каждого из трех значений является число. Например, 101 означает, что объект выставлен на продажу. Для работы в данной ситуации можно было бы создать еще одну группу переключателей. Однако агентам чаще всего требуется одно и то же значение, поэтому переключатели будут зря занимать место в окне. Вместо них лучше создать раскрывающийся список и установить для него в качестве начального значение `101/For Sale`. Как вы узнаете из следующего раздела, пункты списка можно загружать, используя запрос к базе данных. Таким образом, значения формы могут автоматически обновляться в результате изменения содержимого таблицы. Создание раскрывающегося списка не составит труда.

1. Поместите на холст элемент списка и назовите его `LIST_SaleStatus`.
2. Используя палитру свойств, установите для свойства `Elements in List` значение, задающее три пункта. Для пунктов списка укажите следующие значения `List Item Value: For Sale — 101, Pending — 102, Sold — 103`. Для свойства `Initial Value` установите значение, равное 101.
3. Установите следующие свойства: `Prompt=Sale Status`, `Prompt Attachment Offset=7`. Возможно, вы захотите изменить цвет фона так, чтобы он соответствовал цвету холста (т.е. задать белый цвет).



**Рис. 9.3.** Холст, блок данных и элементы поисковой формы

По мере работы над формой время от времени сохраняйте результаты. Создайте папку и поместите в нее файл формы (Search.fmb). На рис. 9.3 показан вариант исходного расположения элементов. Возможно, вы захотите выбрать другой вариант компоновки, но следите за тем, чтобы в нижней части формы осталось достаточно места для отображения записей, удовлетворяющих условиям поиска.

## Создание блока данных для запроса

Для того чтобы убедиться в том, что переключатели работают корректно, следует протестировать имеющийся вариант формы поиска. Однако в настоящий момент форма не выполняет никаких полезных действий. Следующим шагом будет создание запроса, который будет извлекать данные и отображать их в виде таблицы. Поскольку запрос передается базе, вам придется создать новый блок данных. Для того чтобы упростить часть работы с блоком данных, вам следует сначала создать представление, а затем воспроизвести требуемую информацию. Создадим представление средствами SQL.

```
CREATE VIEW ListingResults AS
SELECT Listings.ListingID, Listings.PropertyID, Listings.SaleStatusID,
Listings.AskingPrice, Listings.EndListDate, Properties.Address,
Properties.City, Properties.State, Properties.Zipcode,
```

```

Properties.Bedrooms, Properties.Bathrooms, Properties.SqFt,
Properties.YearBuilt, Agents.LastName, SaleStatus.SaleStatus
FROM Listings INNER JOIN Properties ON
Listings.PropertyID=Properties.PropertyID
INNER JOIN Agents ON Agents.AgentID=Listings.ListingAgentID
INNER JOIN SaleStatus ON
SaleStatus.SaleStatusID=Listings.SaleStatusID;

```

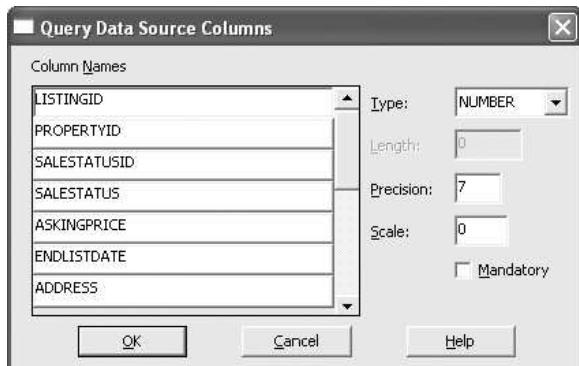
Необходимо включить как минимум те столбцы, которые будут использоваться для установки критериев поиска в выражении WHERE. Поэтому нам обязательно понадобятся таблицы Listings и Properties. Вы можете включить дополнительные столбцы, если у пользователей возникнет потребность в этом. Однако следует помнить, что поместить более 10 столбцов на стандартном экране дисплея не всегда возможно. Заметьте, что в составе запроса нет выражений WHERE и ORDER BY. Соответствующие действия будут выполнены в дальнейшем в блоке данных. Проверьте представление, чтобы убедиться в его работоспособности. Например, задайте следующее выражение: SELECT \* FROM ListingResults WHERE rownum<5.

Теперь, когда готов запрос, предназначенный для извлечения информации из базы, добавим к форме блок данных, который будет содержать этот запрос и выполнять роль своеобразного канала для передачи информации.

1. Вручную добавьте к форме новый блок данных. Назовите его ListingResults и убедитесь, что в объектном навигаторе он расположен ниже блока SEARCH\_BLOCK. Расположение объектов в окне навигатора определяет порядок выбора элементов формы после нажатия клавиши <Tab>. Нам надо, чтобы SEARCH\_BLOCK вызывался первым. Путем перетаскивания можно изменить взаимное расположение элементов в окне Object Navigator.
2. Обратившись к палитре свойств, убедитесь, что установлены следующие значения: Database Data Block=Yes, Query Allowed=Yes и Query Data Source Type=Table.
3. Задайте Query Data Source Name в качестве запроса, который будет получать данные из созданного представления. Выражение надо поместить в скобки:

```
(SELECT ListingID, PropertyID, SaleStatusID, SaleStatus,
AskingPrice, EndListDate, Address, City, State,
Zipcode, Bedrooms, Bathrooms, SqFt, YearBuilt,
LastName FROM ListingResults)
```

4. Для свойства Query Data Source Columns включите в список имена всех пятнадцати столбцов. Убедитесь в том, что для каждого столбца правильно установлены тип данных и длина.
5. Запретите редактирование свойств блока данных. Сами свойства должны выглядеть так: Allowed=No, Update Allowed=No, Delete Allowed=No. Установите также Number of Records Displayed=7 и Show Scrollbar=Yes.



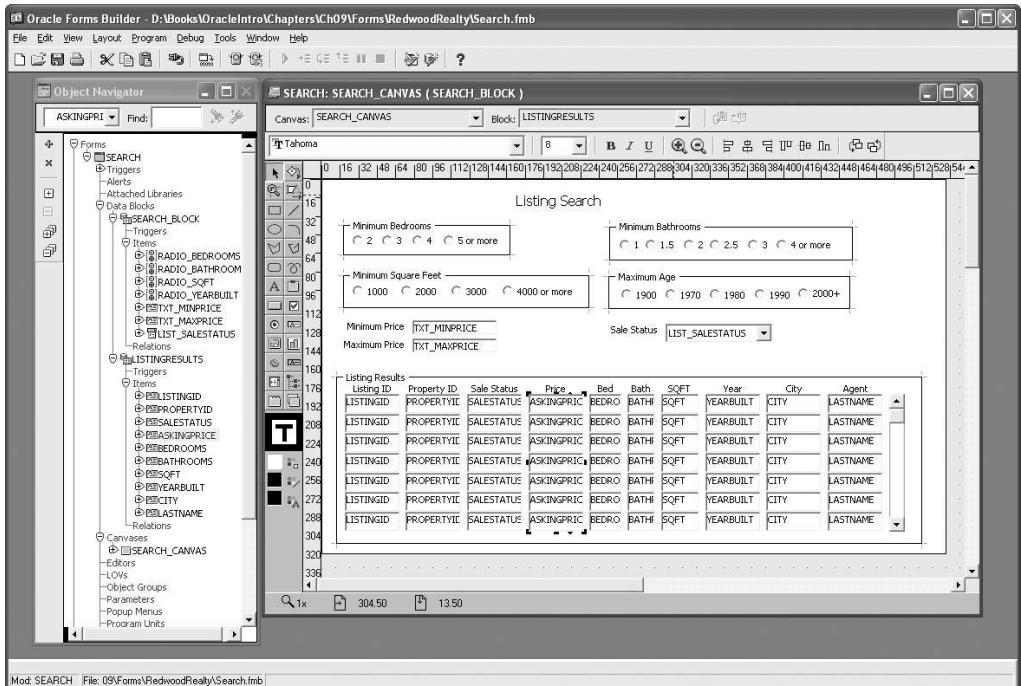
**Рис. 9.4.** Окно Query Data Source Columns

Действия на четвертом этапе требуют много времени и довольно утомительны, однако их надо выполнить и убедиться в том, что вы не допустили ошибок. На рис. 9.4 показано окно для ввода данных. Если вы ошибитесь при указании имени столбца, эту ошибку будет трудно найти. Поскольку блок данных содержит запрос и столбцы определены, вы можете добавлять элементы на холст. Вначале надо создать фрейм и определить вновь созданный блок *ListingResults* в качестве блока данных по умолчанию.

1. В редакторе компоновки задайте в качестве блока данных по умолчанию *ListingsResults*.
2. Создайте в нижней части формы фрейм большого размера. Установите для него следующие свойства: Name=FRM\_ListingResults, Layout Data Block=ListingResults, Layout Style=Tabular, Number of Records Displayed=7, Show Scroll Bar=Yes, Frame Title=Listing Results, Background Color=white.

Теперь можно помещать в новый фрейм поля, в которых будет отображено содержимое различных столбцов. Тщательно подбирая ширину, можно разместить в стандартном окне 10 столбцов. В большинстве случаев проще создать один-два элемента, а затем, копируя их, формировать остальные.

1. Поместите в новый фрейм текстовый элемент. Он автоматически растянется на семь строк.
2. Установите его свойства так, чтобы он соответствовал первому столбцу (ListingID): Name=ListingID, Data Type=Number, Maximum Length=5, Database Item=Yes, Column Name= LISTINGID, Width=48, Background Color=white, Prompt=Listing ID, Prompt Attachment Edge=Top.



**Рис. 9.5.** Внешний вид формы поиска в процессе работы над ней

- Повторите те же действия для ListingID, PropertyID, SaleStatus, AskingPrice, Bedrooms, Bathrooms, SQFT, YearBuilt, City и Lastname.

На рис. 9.5 показан внешний вид формы, полученной на данном этапе работы. Обратите внимание на сокращенные надписи и небольшую ширину полей. Это сделано для того, чтобы форма помещалась на экране. Возможно, вы решите не включать столбцы PropertyID и ListingID, однако они могут оказаться полезны, если агент захочет найти детальную информацию. Возможно, наилучшим решением было бы включить в состав формы ссылку, указывающую на другую форму, которая представляла бы подробные данные. Эта форма должна открываться после щелчка на кнопке или ссылке.

## Обеспечение поиска

На данном этапе форма почти готова. Ее можно запустить, проверить работу, но этого недостаточно. Необходимо организовать реальный поиск. Сама по себе форма выполняется на пользовательском компьютере. Поиск же предполагает передачу запроса на сервер, получение результатов и отображение их в виде таблицы. Большую часть этой работы выполняет блок данных. Однако вам надо ответить на важный вопрос:

когда форма должна получать данные и обновлять таблицу? В некоторых случаях найти ответ на него очень трудно; приходится экспериментировать, пробовать различные варианты обработки событий и выбирать тот из них, который больше всего устроит пользователей. Для поисковой формы принять решение относительно просто. На данный момент мы не можем получить информацию о том, что пользователь сделал свой выбор и ожидает результатов. Необходима кнопка, щелчком на которой пользователь сообщит, что он задал критерии поиска. Добавьте кнопку и убедитесь, что она соответствует блоку данных `Search_Block`. Пометьте ее как `Search`, чтобы пользователю было ясно назначение этой кнопки.

Перед тем как переходить к рассмотрению кода, выполняющегося при активизации кнопки, необходимо предпринять очень важное действие. На данном этапе запрос извлекает всю информацию из представления `ListingResults`. Вам надо модифицировать его так, чтобы он возвращал лишь те данные, которые соответствуют условиям, заданным в форме. Откройте палитру свойств для блока данных `ListingResults`. В разделе `Database` найдите свойство `WHERE clause`. Вам надо добавить сравнительно длинное выражение, которое сравнивает столбцы в запросе с элементами формы.

```
(ASKINGPRICE BETWEEN :SEARCH_BLOCK.TXT_MINPRICE AND  
:SEARCH_BLOCK.TXT_MAXPRICE)  
AND (BEDROOMS >= :SEARCH_BLOCK.RADIO_BEDROOMS)  
AND (BATHROOMS >= :SEARCH_BLOCK.RADIO_BATHROOMS)  
AND (SQFT >= :SEARCH_BLOCK.RADIO_SQFT)  
AND (YEARBUILT >= :SEARCH_BLOCK.RADIO_YEARBUILT)  
AND (SALESTATUSID = :SEARCH_BLOCK.LIST_SALESTATUS)
```

Выражение получается длинным, потому что приходится проверять большое количество столбцов. Все части выражения имеют один и тот же формат. Чтобы упростить отладку, можно начать с одного фрагмента. Если при вводе значения свойства вы допустите ошибку, будет выведено короткое сообщение. В этом случае вам надо будет тщательно проверить составленное выражение. Рассмотрим фрагмент, предназначенный для проверки цены: (`AskingPrice BETWEEN :Search_Block.TXT_MinPrice AND :Search_Block.TXT_MaxPrice`). Во-первых, заметьте, что выражение помещено в круглые скобки, так как в дальнейшем могут быть добавлены новые составляющие. Скобки гарантируют, что этот фрагмент будет интерпретирован как отдельное выражение, независимо от других фрагментов. Во-вторых, обратите внимание на синтаксис. Ссылаясь на текстовый элемент, надо указать блок данных (`Search_Block`), поскольку элементы содержатся в разных блоках. Обратите также внимание на использование двоеточия (:). Оно указывает на то, что система извлекает значение из внешних переменных, а не ищет имена столбцов. Разобравшись в синтаксисе, вы увидите, что выражение `WHERE` очень похоже на любую другую команду SQL. При выполнении формы интерпретатор читает из нее значения и помещает в переменные выражения `WHERE`, затем возвращает строки, удовлетворяющие условиям.

Работая со свойствами блока данных, следует также определить выражение ORDER BY. На данном этапе введите AskingPrice, так как большинство пользователей предпочтут, чтобы результаты были отсортированы по цене. Впоследствии можно подумать о том, чтобы предоставить пользователю возможность выбирать вариант сортировки. Свойства WHERE и ORDER BY удобны тем, что можно писать программный код, изменяющий их значение, в ответ на действия пользователя.

Последний этап работы над формой — написание кода обработчика, получающего управление после щелчка на кнопке Search. Поскольку основная часть работы выполняется выражением WHERE, код для кнопки достаточно прост. Этот код объявляется как обработчик формы. Большинство обработчиков для кнопок связывается с событием When-Button-Pressed. Щелкните правой кнопкой мыши на кнопке Search и выберите данное событие. После этого вы получите возможность ввести код в специальном окне.

```
GO_BLOCK('LISTINGRESULTS');
EXECUTE_QUERY;
```

Первая строка (GO\_BLOCK) сообщает интерпретатору формы о том, что надо использовать блок данных ListingResults. Как вы помните, кнопка Search принадлежит блоку данных Search\_Block, а вам надо, чтобы результаты отображались посредством блока ListingResults. Чтобы это стало возможным, надо сообщить системе о необходимости переключить активизированный блок. Вторая строка (EXECUTE\_QUERY) говорит о том, что система должна выполнить запрос, хранящийся в блоке данных ListingResults. Сначала интерпретатор форм выберет текущие значения и включит данные в выражение WHERE. Затем он извлечет соответствующие строки и поместит их в элементах формы.

Проверьте форму и убедитесь, что она работает корректно. Измените некоторые параметры поиска; возвращаться должны лишь значения, соответствующие указанным критериям. Заметьте, что задав нереальные условия (например, дом с пятью спальными комнатами ценой 20000 долларов), вы не получите информации ни об одном объекте. На рис. 9.6 показаны результаты выполнения запроса. Очевидно, что полученные вами результаты могут отличаться от показанных на рисунке.

Подведем вкратце итоги того, что было сделано в этом разделе. Мы создали форму с нуля, не пользуясь инструментами типа “мастер”. Данная форма содержит несколько элементов, упрощающих пользователям процесс указания критериев поиска. Затем при выполнении запроса извлекается список объектов, удовлетворяющих условиям, и результаты отображаются в виде небольшого списка. Для создания соединения добавляется блок данных на основе запроса. Мы сформировали выражение WHERE, включающее значения из формы в качестве условий. С кнопкой Search мы связали две строки кода, выполняющих запрос. Блок данных выполняет всю работу по поддержке соединения с базой данных, передаче запроса и возврату строк, удовлетворяющих условиям.

The screenshot shows a web browser window for Oracle Application Server Forms Services. The title bar reads "Oracle Application Server Forms Services - Microsoft Internet Explorer". The address bar shows the URL: "http://postlt:8889/forms90/f90servlet?form=D:\Books\OracleIntro\Chapters\Ch09\Forms\RedwoodRealty\Search." The page itself is titled "Redwood Realty" and features a "Listing Search" form. The form includes fields for "Minimum Bedrooms" (radio buttons for 2, 3, 4, or 5 or more), "Minimum Bathrooms" (radio buttons for 1, 1.5, 2, 2.5, 3, or 4 or more), "Minimum Square Feet" (radio buttons for 1000, 2000, 3000, or 4000 or more), "Maximum Age" (radio buttons for 1900, 1970, 1980, 1990, or 2000+), "Minimum Price" (text input: 125000), "Maximum Price" (text input: 175000), "Sale Status" (dropdown menu: "For Sale"), and a "Search" button. Below the form is a table titled "Listing Results" with columns: Listing ID, Property ID, Sale Status, Price, Bed, Bath, SQFT, Year, City, and Agent. The table contains 10 rows of data. At the bottom of the page, there is a status bar with "Record: 1/1" and a "Trusted sites" icon.

| Listing ID | Property ID | Sale Status | Price  | Bed | Bath | SQFT | Year | City     | Agent      |
|------------|-------------|-------------|--------|-----|------|------|------|----------|------------|
| 15258      | 1029        | For Sale    | 125000 | 4   | 2    | 2041 | 1990 | Eureka   | Williams   |
| 15043      | 1281        | For Sale    | 135000 | 4   | 2    | 2161 | 1990 | Fortuna  | Sheibani   |
| 15899      | 1452        | For Sale    | 144500 | 4   | 2    | 2300 | 1991 | Eureka   | St-Onge    |
| 16132      | 1481        | For Sale    | 146500 | 3   | 2    | 2086 | 1990 | Eureka   | Silverburg |
| 15149      | 1521        | For Sale    | 149990 | 4   | 2    | 2348 | 1996 | Eureka   | Townsend   |
| 15588      | 1574        | For Sale    | 154000 | 3   | 2    | 2060 | 1991 | Loleta   | Weber      |
| 15243      | 1603        | For Sale    | 155000 | 4   | 2    | 2348 | 1996 | Trinidad | Flamenbaum |

Рис. 9.6. Результаты работы поисковой формы

## Создание обработчиков событий

Как вы могли сделать вывод, проанализировав примеры, рассмотренные в данной книге, обработчики событий позволяют реализовать многие полезные функции. Они дают возможность контролировать действия, предпринимаемые пользователем. Написав одну-две строки кода, можно связать разделы формы и обеспечить выполнение запросов. Понятно, что путем создания кода, можно реализовать сколь угодно сложные действия, но действительно ли необходимы обработчики событий, связанных с формами? Вы уже знаете, что практически все действия программиста направлены

на то, чтобы упростить работу пользователей. Действительно, нужна ли компании Redwood Realty поисковая форма? Агенты могут составлять SQL-запросы и извлекать из базы любые требуемые им данные. И тем не менее мы потратили время на создание такой формы. Сделали мы это лишь для того, чтобы агент мог получать нужную ему информацию, не затрачивая свое время на изучение правил составления запросов к базе. SQL-запрос всегда возвращает строку данных, и разработчик должен следить, действительно ли эта строка содержит ответ на вопрос, интересующий пользователя. Создавая форму и программный код, надо внимательно составлять запросы и тщательно проверять результаты. Детали выполнения запроса скрыты от пользователя. Ваша задача как разработчика — упростить работу пользователей и снизить вероятность возникновения ошибки. Для того чтобы достичь этой цели, надо везде, где возможно, использовать запросы и встроенные возможности форм. Но в некоторых случаях приходится написать несколько строк кода вручную.

Программный код, выполняющий роль обработчика, имеет две особенности. Во-первых, он связан с тем или иным событием, относящимся к форме, например, щелчком мышью или нажатием клавиши на клавиатуре. Во-вторых, он может содержать практически любой код PL/SQL. Сказанное означает, что для эффективной работы с базой данных надо знать правила написания кода и основные функции, предоставляемые Oracle. Кроме того, необходимо представлять себе, какие события могут происходить при работе пользователя с формой. В ряде случаев самым сложным оказывается принятие решения о том, какое именно событие следует использовать.

## Определение событий, связанных с формой

Создавая форму, разработчики часто связывают фрагменты кода с различными событиями. При возникновении события можно решать различные задачи, например, анализировать входные данные, выполнять запрос, запускать новую форму, передавать сообщения другим пользователям и т.д. Имея воображение и опыт, можно придумать сотни задач, которые целесообразно решать обработчиками событий. Одна из положительных особенностей среди разработки форм — это поддержка сотен событий. При работе с формами Oracle, блоками данных и отдельными элементами могут возникать десятки событий. Полный набор событий формы никогда не используется. События можно разделить на несколько уровней; такое разделение позволяет создавать более эффективный код обработчиков и осуществлять контроль над формой. Если вы используете события слишком низкого уровня (например, свяжете обработчик с нажатием клавиши), код обработчика будет вызываться слишком часто. Это не только снижает производительность, но и увеличивает вероятность возникновения ошибки. Если вы подниметесь слишком высоко по иерархии событий, может случиться так, что обработчик не будет вызван ни разу или, например, получит управление лишь единожды при загрузке формы. Таким образом, правильно выбрав уровень события, вы получаете возможность контролировать именно те действия, которые вам нужны.

| Форма                    | Блок                        | Элемент                   |
|--------------------------|-----------------------------|---------------------------|
| 1.Pre-Logon              |                             |                           |
| 2.On-Logon               |                             |                           |
| 3.Post-Logon             |                             |                           |
| 4.Pre-Form               |                             |                           |
|                          | 5.When-CREATE-Record        |                           |
|                          | 6.Pre-Block                 |                           |
|                          | 7.Pre-Record                |                           |
|                          |                             | 8.Pre-Text-Item           |
| 9.When-New-Form-Instance |                             |                           |
|                          | 10.When-New-Block-Instance  |                           |
|                          | 11.When-New-Record-Instance |                           |
|                          |                             | 12.When-New-Item-Instance |
|                          |                             | 1.Post-Text-Item          |
|                          | 2.Post-Record               |                           |
|                          | 3.Post-Block                |                           |
| 4.Post-Form              |                             |                           |
| 5.On-Rollback            |                             |                           |
| 6.Pre-Logout             |                             |                           |
| 7.On-Logout              |                             |                           |
| 8.Post-Logout            |                             |                           |

Рис. 9.7. Последовательность первичных событий

Поскольку число событий достаточно велико, бывает трудно решить, с каким из них следует связать код обработчика. Правильно сделать выбор поможет вам рис. 9.7, представляющий последовательность первичных событий, которые генерируются в начале и в конце работы пользователя с формой. Form Builder предоставляет полный список событий, но содержимое этого списка отсортировано по алфавиту. Определить, в какой последовательности вызываются обработчики, можно по их именам. Если в имени есть префикс PRE, обработчик вызывается до возникновения события, а префикс POST указывает на то, что обработка начинается в тот момент, когда событие уже завершилось. Например, имя PRE-FORM указывает на то, что форма будет открыта, а POST-FORM — на то, что форма уже была закрыта. Благодаря рис. 9.7 можно составить представление о взаимосвязи событий блока и элемента с событиями формы.

Механизм событий в действии демонстрирует инструмент формы Oracle — сообщение Alert. Это форма небольшого размера, которая отображается на экране и содержит фрагмент текста и кнопки. Некоторые разработчики применяют данную форму для представления информации. Однако такие сообщения отвлекают пользователя, поэтому по возможности их надо избегать. Если подобное сообщение появилось на экране, пользователь не сможет продолжить работу, не отреагировав на него. Оперативную информацию лучше отображать в виде текстовой метки в составе формы.

В этом случае пользователь сможет беспрепятственно продолжать работу. Однако в ряде случаев, например при возникновении серьезной ошибки, не остается ничего другого, как прибегнуть к сообщению `Alert`. Они также полезны при отладке. С помощью подобного сообщения можно получить сведения о том, какой обработчик получил управление.

Для того чтобы продемонстрировать часть последовательности событий, создайте новую форму для таблицы `Agents` и добавьте три новые формы `Alert`.

1. Используя мастер блоков данных и мастер компоновки, создайте простую формы для таблицы `Agents`.
2. Выберите раздел `Alerts` в объектном навигаторе и щелкните на кнопке `Add` (на ней изображен знак +).
3. Установите свойства для отображения простого сообщения: `Name=ALERT_TRIGGER`, `Title=Trigger Fired`, `Message=test`, `Alert Style=Note`.
4. Создайте три обработчика с почти одинаковым кодом, изменения для каждого события лишь имя сообщения.

```
DECLARE
    btn NUMBER;
BEGIN
    SET_ALERT_PROPERTY('ALERT_TRIGGER', ALERT_MESSAGE_TEXT,
        'Pre-Form trigger event');
    btn := SHOW_ALERT('ALERT_TRIGGER');
END;
```

5. Запустите форму и выполните запрос. Просмотрев сообщения, запишите порядок возникновения событий.

Функция `Show_Alert` возвращает значение, соответствующее кнопке, на которой щелкнул пользователь. В данном простом примере нас не интересует, какая кнопка была выбрана, поэтому возвращаемое значение не используется. Однако поскольку вызывается именно функция, то согласно синтаксическим правилам значение должно быть получено. На рис. 9.8 показано сообщение, отображаемое при выполнении запроса. Необходимо отметить, что при закрытии формы событие `Post-Item` снова будет сгенерировано. Сообщение о данном событии вы будете получать каждый раз, когда попытаетесь перейти к новой строке или переместите курсор за пределы поля `Agent ID`. Возможно, у вас возникнет вопрос, зачем нам нужно событие, связанное с выходом из поля редактирования. В принципе его можно использовать для проверки изменений, внесенных пользователем. Однако обработчик `Post-Text-Item`, примененный для данной цели, обладает существенным недостатком: он вызывается каждый раз, независимо от того, вносил ли пользователь изменения в поле. С другой стороны, поля редактирования Oracle поддерживают событие `When-Validate-Item`. Обработчик этого события вызывается только в том случае, если пользователь из-

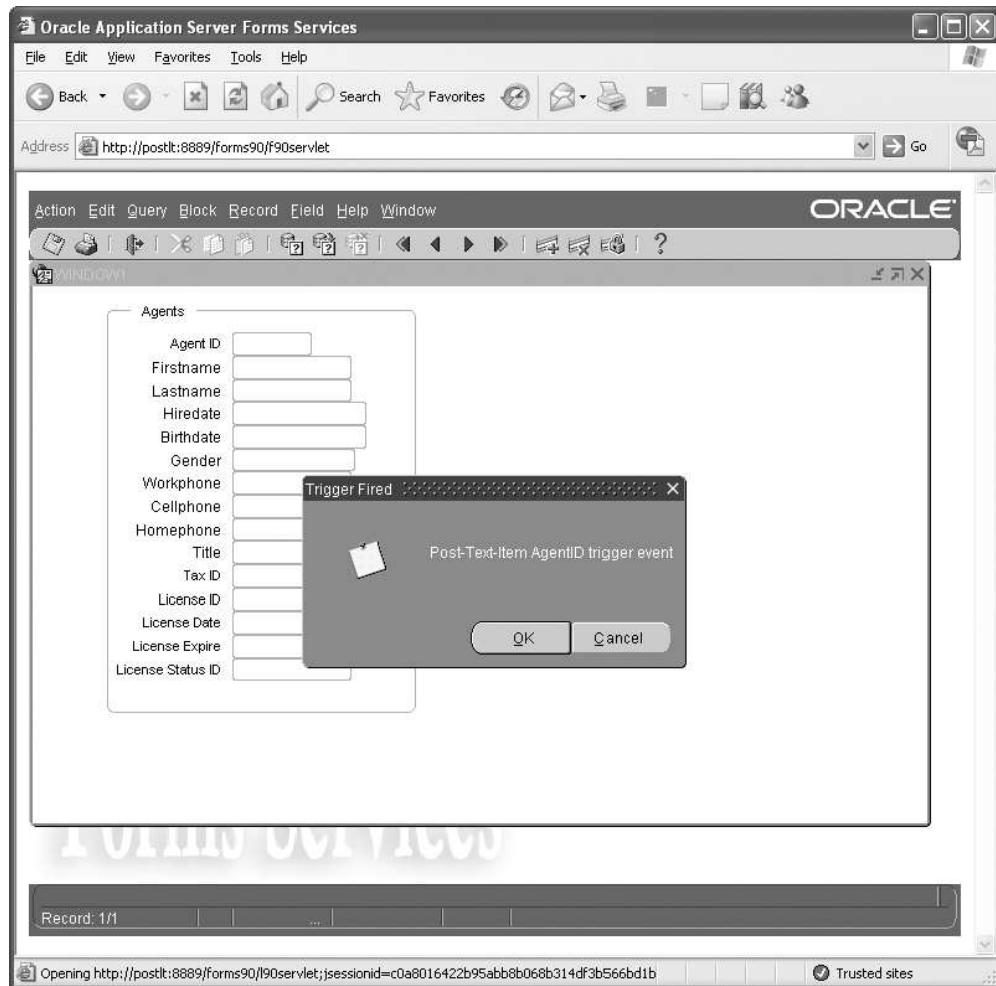


Рис. 9.8. Первый вызов обработчика Post-Item

менил значение поля, а затем покинул его. Если вы сомневаетесь, какое из событий применить в том или ином случае, можете прибегнуть к помощи сообщений Alert и проверить, как обработчик будет вести себя в различных ситуациях. Поработайте с формой в качестве пользователя и выясните, какое событие возникает именно в тех случаях, которые интересуют вас. Если ни одно из событий вас не устраивает, вы можете написать код для двух или более событий. Однако при этом могут возникнуть трудности с совместным использованием данных.

## Создание и редактирование обработчиков

Когда вы разберетесь в последовательности событий и выясните, какое из них целесообразно использовать в конкретном случае, настанет время написания кода обработчика. Создать новый обработчик и внести изменения в уже существующий несложно. Для создания обработчика найдите в объектном навигаторе пункт Triggers, щелкните на нем правой кнопкой мыши, выберите в контекстном меню пункт Smart Triggers, а затем укажите требуемое событие. Введите код в окне редактирования. Для того чтобы внести изменения в существующий обработчик, вам надо таким же способом открыть окно редактора либо дважды щелкнуть на пиктограмме обработчика в объектном навигаторе.

Гораздо интереснее выяснить, какой код написать в окне и как структурировать его. Преимуществом механизма событий, связанных с формами Oracle, является тот факт, что код имеет те же синтаксис и структуру, что и код, применяемый для обработки данных. Как вы помните, обработчики данных создаются для событий, затрагивающих данные (Update, Insert и Delete), и хранятся в базе. Структура обработчика события формы имеет следующий вид:

```
DECLARE
    -- Определение переменных
BEGIN
    -- Код PL/SQL
    -- (обычно содержит вызовы стандартных функций)
EXCEPTION
    -- Выражения, выполняемые в случае ошибки
END;
```

Следуя хорошему стилю программирования, следует определить все переменные в разделе деклараций. Из рассмотренных ранее примеров вы видели, что синтаксис достаточно прост. Надо лишь указать имя переменной, а после него — тип данных. Не забывайте завершать каждое выражение точкой с запятой.

Если у вас будут возникать вопросы, касающиеся структуры и синтаксиса кода, обращайтесь к руководству пользователя по PL/SQL. Там же вы найдете список команд. Вероятнее всего, вам потребуются выражения, реализующие условный оператор (IF/ELSE/END), и циклы (WHILE ... LOOP и FOR ... LOOP). Как правило, в обработчиках событий приходится создавать переменные, выполнять вычисления, извлекать данные из базы, реализовывать ветви, выполняющиеся в зависимости от некоторого условия, формировать циклы и обрабатывать ошибки. На рис. 9.9 условно показаны основные синтаксические конструкции PL/SQL, используемые для выполнения этих действий.

Заметьте, что присваивание значений переменным осуществляется посредством оператора :=. Кроме того, обратите внимание, что с помощью конструкции SELECT ... INTO можно непосредственно присваивать значения отдельным переменным. Для непосредственной передачи данных из формы в базу можно использовать

```

DECLARE
    tax NUMBER(9,2); price NUMBER(9,2);
    AvgSalary NUMBER(10,3);
    tax_rate CONSTANT NUMBER := 0.07;
BEGIN
    Вычисления
    tax := price * tax_rate;

    Извлечение данных
    SELECT Avg(Salary) INTO AvgSalary FROM Employee;

    Проверка условий
    IF (condition)
        --statements
    ELSE
        --statements
    END IF;

    Цикл
    WHILE (j < 10)
    LOOP
        j := j + 1;
    END LOOP;

    Обработка ошибок
    EXCEPTION
        WHEN (error event) THEN ...
    END;

```

**Рис. 9.9.** Основные команды PL/SQL

команды DML (*Insert*, *Update* и *Delete*). Ссылаясь на элементы формы, не забывайте указывать двоеточие, например :Search\_block.Txt\_MinPrice.

Основные команды DML и выражение *SELECT INTO* хорошо подходят для извлечения из базы или модификации одиночных данных. Для того чтобы получить или отредактировать несколько строк, надо создать курсор и с его помощью обрабатывать результаты построчно. Эти возможности можно использовать для решения некоторых проблем в поисковой форме Redwood Realty. Откройте форму в редакторе и обратите внимание на пункты списка SaleStatus. Как вы помните, эти пункты (*For Sale*, *Pending* и *Sold*) были созданы вручную. Они “жестко” закодированы в форме, а это значит, что добавить новые значения или изменить уже существующие можно, только отредактировав форму. На данный момент это не составит труда, но представьте себе, что прошло два года, за это время были созданы десятки форм и данные используются в различных частях приложения. Специалисту, который занимается сопровождением, надо будет найти форму, изменить ее, повторно скомпилировать и разослать приложение пользователям. Вы уверены, что, внося изменения, он не допустит ошибку?

Поскольку описания и идентификаторы уже хранятся в базе данных (таблица SaleStatusID), ваша форма может извлечь их оттуда. Если кто-нибудь решит изменить данные или добавить новые, ему достаточно отредактировать значения в таблице с помощью простой формы, предназначенней для администрирования. Будучи запущенной, поисковая форма получит текущие значения и отобразит их. Осталось решить, как извлечь значения из таблицы и поместить их в список.

Попробуем ответить на вопрос о том, когда следует загрузить значения списка. Проще всего сделать это при открытии формы, а это значит, что надо включить соответствующий код в обработчик Pre-Form. Код, предназначенный для получения данных запроса, имеет одну особенность. Он использует курсор базы, и с его помощью реализует построчный просмотр результатов запроса. Логика этой простой программы, которую нам предстоит написать, выглядит следующим образом.

1. Создание запроса для извлечения данных из таблицы SaleStatus.
2. Просмотр в цикле наборов результатов.
3. Включение описания и идентификатора в список.

Процедура создания запроса и курсора проста, детали описаны в документации на PL/SQL. Надо лишь найтистроенную функцию, которая упростит задачу связывания значений с пунктами списка. Используя курсор, можно применить команду ADD\_LIST\_ELEMENT.

```

DECLARE
    CURSOR cStatus IS
        SELECT SaleStatusID, SaleStatus
        FROM SaleStatus
        ORDER BY SaleStatusID;
BEGIN
    CLEAR_LIST('LIST_SALESTATUS');
    FOR stats IN cStatus
    LOOP
        IF (stats.SaleStatusID <> 101) THEN
            ADD_LIST_ELEMENT('LIST_SALESTATUS', 99,
                stats.SaleStatus, TO_CHAR(stats.SaleStatusID));
        END IF;
    END LOOP;
END;

```

Курсор определяется в разделе деклараций и описывается посредством запроса SELECT. Заметьте, что обычно в запросе присутствует выражение ORDER BY, размещающее данные в нужном порядке. В следующем разделе кода используется внутренняя функция. Она очищает данные, которые могут присутствовать в списке. Основной частью кода является список, в теле которого анализируется каждая строка. Функция ADD\_LIST\_ELEMENT — внутренняя. Она упрощает включение данных в список. В качестве параметров ей передается имя пункта списка, расположение нового пункта (в данном случае значение 99 помещает элемент в конец списка), описание пункта, отображаемое пользователю, и идентификатор, который возвращается, когда пользователь выберет данный пункт (идентификатор преобразуется в символьное значение посредством функции TO\_CHAR).

Для проверки созданного кода откройте свойства списка и удалите все пункты за исключением пункта по умолчанию 101/For Sale. Для того чтобы установить значение по умолчанию, вам надо, чтобы значение с таким статусом уже присутствовало

в списке. Обратив внимание на код, представленный в окне на рис. 9.12, вы увидите, что значение по умолчанию создает еще одну проблему. Команда `Clear_List` не удаляет его. Следовательно, код, используемый для загрузки списка, должен содержать выражение `IF`; оно необходимо для того, чтобы пункт не был записан дважды. Если описание пункта по умолчанию может впоследствии измениться, вам придется создать более сложный код, который удалял бы первоначальное значение, установленное функцией `SET_ITEM_PROPERTY`. Проанализировав доступные функции Oracle, вы увидите, что код для работы со списком можно упростить, использовав для извлечения информации из базы функцию `POPULATE_GROUP_WITH_QUERY`, а для передачи полученных данных списку — функцию `POPULATE_LIST`, объединив их в одно выражение. Однако сначала надо создать группу записей, которая хранила бы данные, и написать код для работы с курсором, который отвергал бы значение по умолчанию, предотвращая тем самым дублирование пунктов списка.

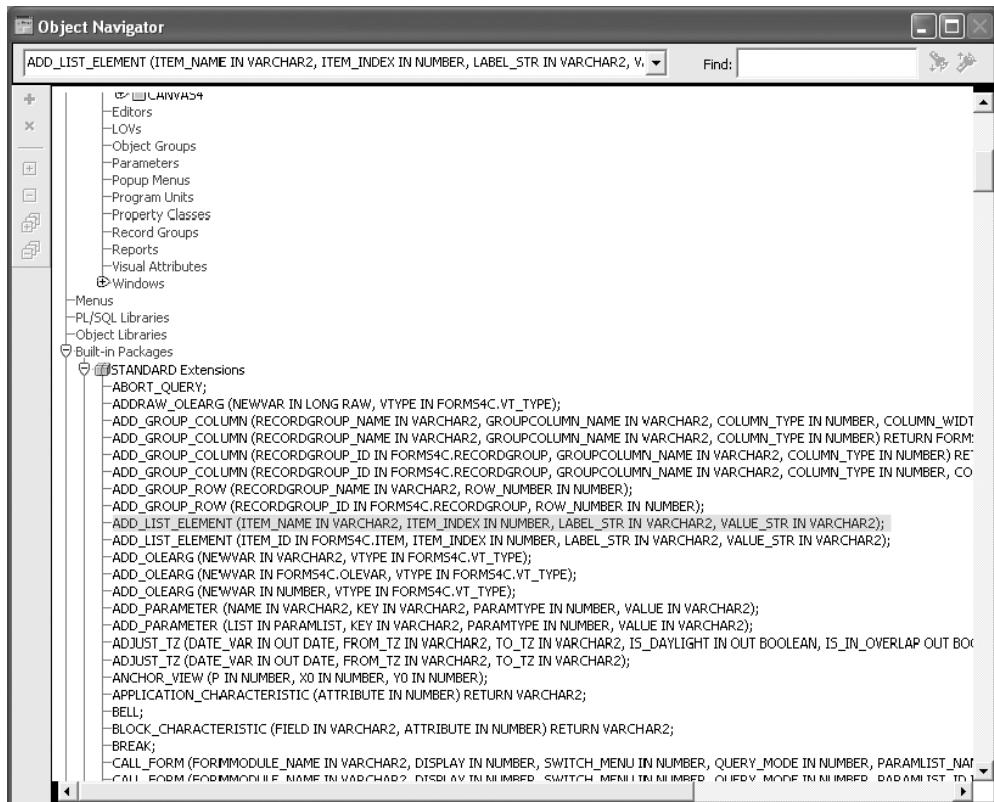
Код обработчика может быть настолько сложен, насколько это необходимо для решения конкретной задачи. Некоторые обработчики содержат сотни строк. Желательно написать как можно более простой код. Если код становится сложным, вам следует пересмотреть общий подход к решению проблемы. Если более простого решения найти не удается, постарайтесь хотя бы переместить часть кода в базу данных, чтобы его было легче найти.

На данный момент вы, наверное, уже заметили, что в большей части приведенных здесь примеров используются встроенные функции. Они выполняют основные задачи, такие как переход к другому блоку, изменение свойств элементов формы или отображение различных записей. Как узнать, какие функции доступны? Как показано на рис. 9.10, список функций можно отобразить в объектном навигаторе. Разверните раздел `Built-in Packages`, а затем раздел `STANDARD Extensions`. В списке, отсортированном по алфавиту, приведены имена и параметры функций.

На рис. 9.11 показано, как получить детальную информацию о каждой функции, используя интерактивную систему подсказки. Выберите пункт меню `Help` либо нажмите комбинацию клавиш `<Ctrl+H>`. Выберите вкладку поиска и введите в поле имя функции. Отметьте подходящий для вас раздел и щелкните на кнопке `Open`, чтобы отобразить справочную информацию. Как правило, справочная система содержит описания всех параметров функции и предлагает код примера, демонстрирующий ее использование. Потратьте некоторое время на ознакомление с функциями, и вы получите представление об инструментах, которые есть в вашем распоряжении.

## Отладка обработчиков событий

Иногда вполне работоспособный код получается с первой попытки, но такое случается крайне редко. Но даже если код выполняется, вам все равно надо проверить значения переменных на различных этапах работы. Существуют три основных способа поиска ошибок в формах Oracle.



**Рис. 9.10.** Некоторые стандартные функции для форм Oracle

1. Компиляция кода обработчика и поиск синтаксических ошибок.
2. Отображение на различных этапах работы сообщений, которые информируют о ходе выполнения программы и отображают несколько значений переменных.
3. Использование отладчиков форм для пошагового выполнения строк кода.

В состав Oracle 10g входит мощный отладчик форм, который позволяет организовать построчное выполнение кода. Можно устанавливать точки останова и выполнять код до тех пор, пока не будет достигнута одна из них. Предоставляется также кнопка, позволяющая приостановить выполнение цикла. При необходимости можно проверять значения переменных в составе обработчиков. В процессе отладки доступны значения элементов, выбранных в форме. Можно также потребовать, чтобы программа прекращала работу в том случае, когда значение определенной переменной будет изменено.

Для того чтобы проверить работу отладчика, выполните следующие действия.

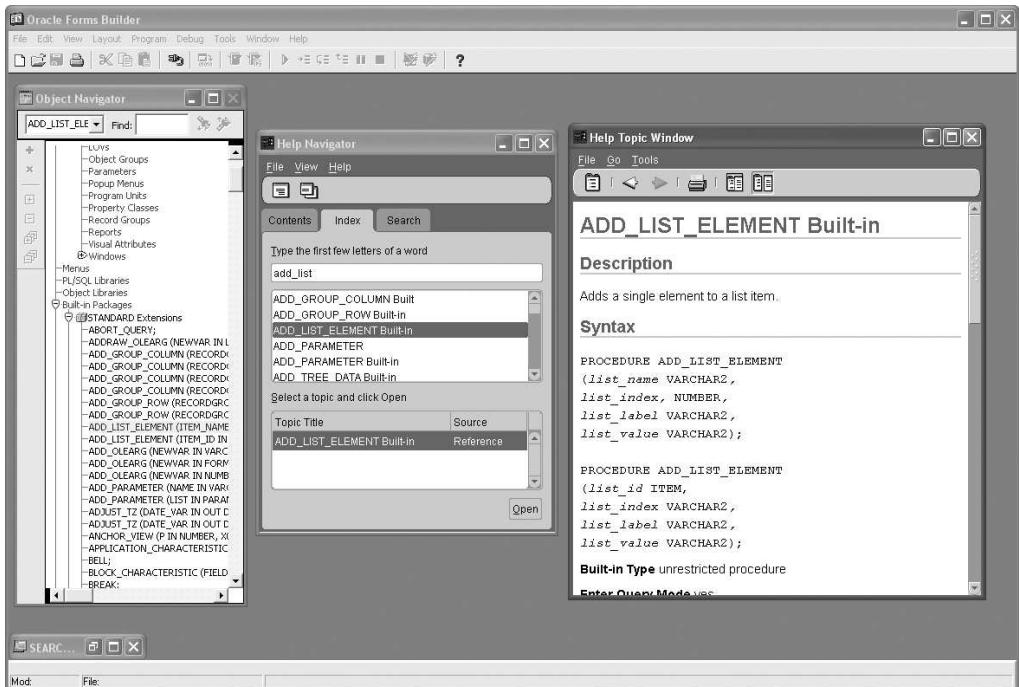


Рис. 9.11. Описание стандартных функций

1. Откройте форму Search и код обработчика события Pre-Form.
2. Дважды щелкните на серой полосе, расположенной вдоль левой границы окна. При этом курсор должен быть расположен перед строкой, содержащей имя функции Clear\_List.
3. Щелкните на пиктограмме на панели инструментов, чтобы начать процесс отладки, или запустите отладчик из главного меню.
4. В результате начнется загрузка формы и откроется консольное окно отладчика. Чтобы увидеть консоль, надо щелкнуть мышью в окне Forms Builder.

Устанавливать и удалять точки останова можно также, щелкнув правой кнопкой мыши. Какой бы способ вы ни использовали, точка останова будет отображаться в виде небольшого маркера красного цвета слева от выбранной строки. Программа остановится на этой точке, причем строка, помеченная маркером, выполнена не будет.

На рис. 9.12 показаны некоторые инструменты, доступные в режиме отладки. Щелкнув на кнопках в верхней части окна, можно отображать или скрывать различную информацию. В окне стека отображается текущая процедура и номер строки, а также все процедуры, которые были выполнены перед этим. Предлагается также окно с информацией о переменных, объявленных в текущей процедуре. Кроме того, отображаются значения элементов формы.

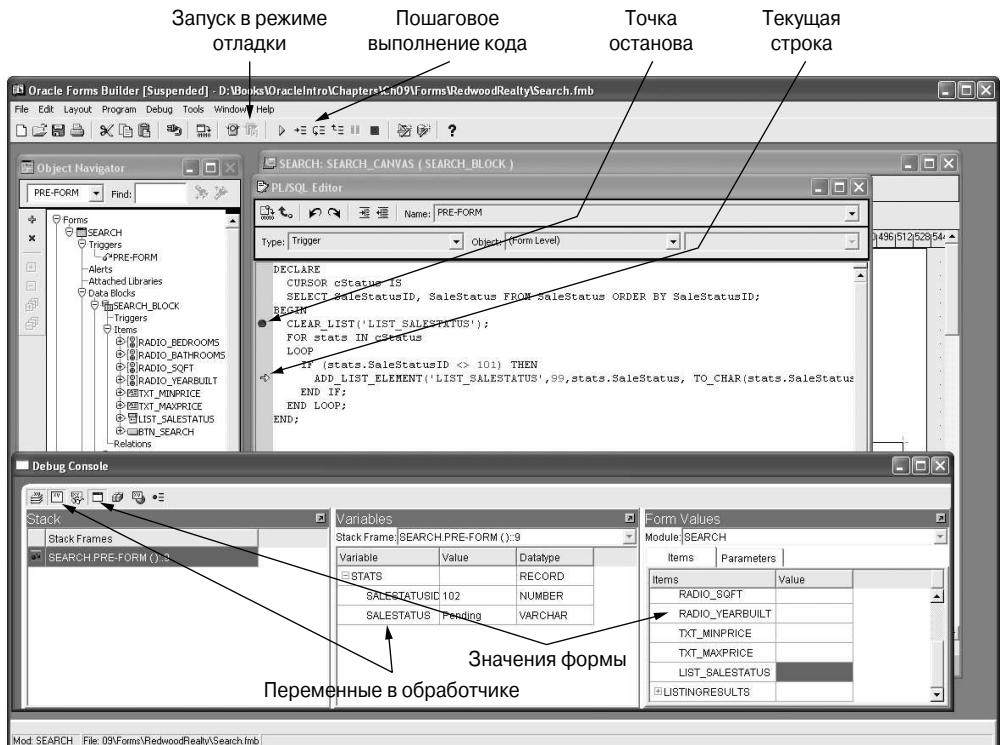


Рис. 9.12. Возможности, предоставляемые консолью отладчика форм

Кнопки, предназначенные для отладки (они расположены на основной панели инструментов) Forms Builder, позволяют выполнять код в пошаговом режиме либо проходить всю подпрограмму за один шаг. Текущая строка помечается маркером в виде стрелки. Щелкните несколько раз на кнопке, ответственной за выполнение одной строки программы, и вы увидите, что система выполнила несколько команд в цикле. Вы можете также проверить текущие значения, извлеченные из базы данных. Для этого надо просмотреть содержимое локальных переменных. При наличии сложных проблем можно открыть окно Watch и связать точки останова с конкретными переменными. Когда значение такой переменной изменится, отладчик приостановит работу и отобразит код, ответственный за эти изменения. В таком режиме программа выполняется гораздо медленнее, поэтому, прибегая к данному средству, соблюдайте осторожность.

**Совет.** Если вы работаете с системой Windows XP и консоль отладки не открывается, закройте Forms Builder, отключите брандмауэр с помощью Windows Control Panel, затем создайте обычным образом сеанс отладки для удаленного компьютера.

Отладчик также имеет возможность работать с формами, выполняющимися на сервере. Однако для этого надо знать имя сервера и порт, используемый процессором форм. Предусмотрена специальная команда, предоставляющая такую информацию. Чтобы проверить данную возможность, откройте форму `Search` и модифицируйте код обработчика `When-Button-Pressed` для кнопки `Search`. Добавьте строку `Attach.Debug;` (не забудьте указать в конце строки точку с запятой) и установите в следующей строке (`Go_Block`) точку останова. Запустите форму, не переходя в режим отладки. Щелкните на кнопке `Search`, и в окне отобразится имя компьютера и используемый порт. В конструкторе форм выберите пункт меню `Debug⇒Attach` и введите имя сервера и порт. Теперь вы сможете установить соединение с формой и отлаживать ее удаленно, выполняя те же действия, что и при работе с формой, находящейся на локальной машине. Пользоваться системой отладки просто. Более подробно вы сможете ее изучить, просмотрев обучающий видеоролик, находящийся на Web-узле Oracle.

## Обработка ошибок

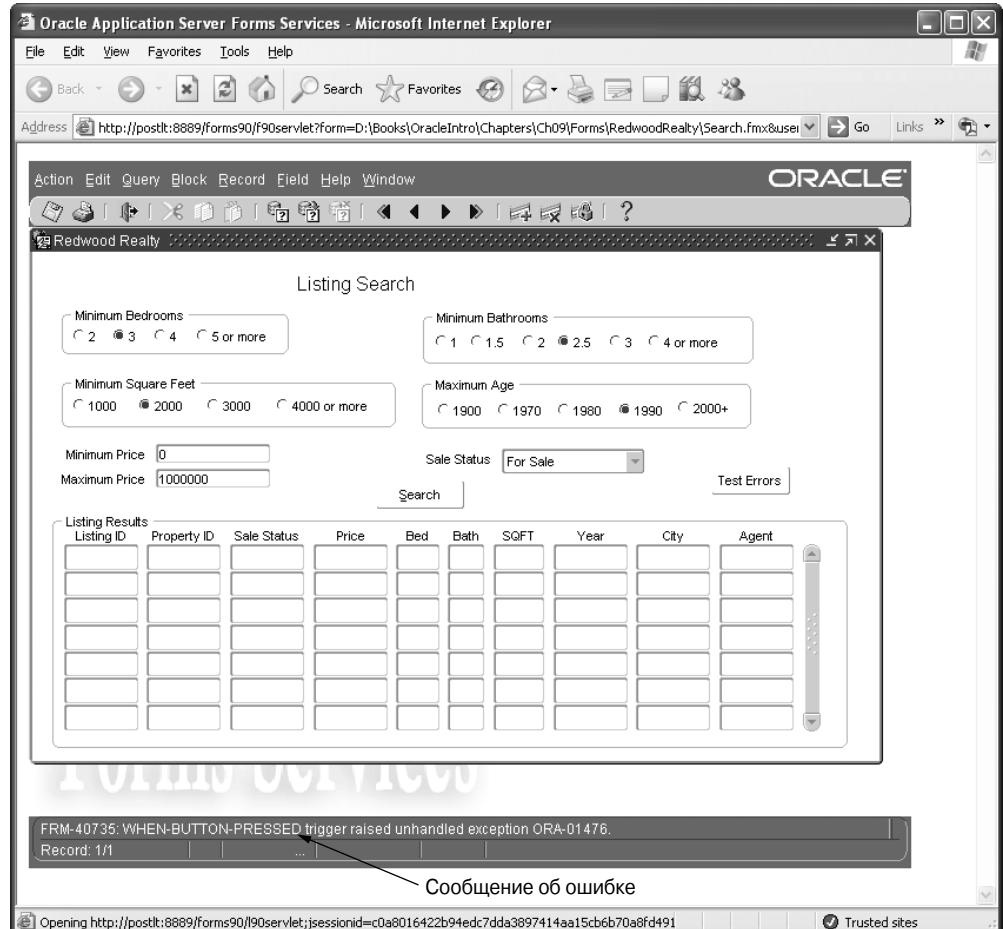
После проверки синтаксиса на этапе компиляции можно воспользоваться отладчиком для того, чтобы найти ошибки в логике программы и устраниТЬ другие проблемы. Однако, несмотря на все усилия, могут возникнуть непредвиденные ситуации. При работе формы иногда появляются *ошибки выполнения*(runtime error). Причиной их может быть недопустимое значение переменной, в результате чего, например, предпринимается попытка деления на нуль, а подобные значения, в свою очередь, имеют место потому, что разработчик не ограничил должным образом возможности пользователей по вводу данных. Если вы не предпримете мер для устранения таких ошибок, в один прекрасный момент ваша программа прекратит работу, отобразив пользователю непонятное для него сообщение.

Для того чтобы подробнее рассмотреть данную проблему и научиться бороться с ошибками выполнения, создадим новую кнопку в форме `Search`. Эта кнопка (`BTN_TEST`) и код обработчика будут нужны нам лишь временно. Напишите код, содержащий явно недопустимые операции.

```
DECLARE
    i NUMBER;
    j NUMBER;
BEGIN
    i := 0;
    j := 10/i;
END;
```

Скомпилируйте данный фрагмент программы и убедитесь, что в нем нет синтаксических ошибок.

На рис. 9.13 показано сообщение, отображаемое при попытке активизировать новую кнопку. Сообщение отображается в строке, специально предназначеннай для



**Рис. 9.13.** Сообщение, генерируемое при попытке деления на нуль

этой цели. Как видите, оно не слишком информативно. Большинство пользователей попросту проигнорируют его. Если вы добиваетесь именно такой реакции пользователей, никаких мер предпринимать не надо. Однако в большинстве случаев вы захотите в большей степени контролировать ситуацию при возникновении ошибки. Сделать это можно, определив обработчик исключений.

```

DECLARE
    i NUMBER;
    j NUMBER;
BEGIN
    i := 0;
    j := 10/i;
EXCEPTION
    WHEN OTHERS THEN Message(sqlerrm);
END;

```

Добавьте выражение EXCEPTION перед последней командой END. Если в процессе выполнения возникнет ошибка, интерпретатор форм передаст управление разделу исключений и найдет соответствующую категорию ошибок. Если вы предвидите некоторые ошибочные ситуации, то можете указать их в обработчике. В нашем примере обработчик исключений имеет следующий вид:

```
EXCEPTION
  WHEN ZERO_DIVIDE THEN j:=0;
  WHEN OTHERS THEN Message(sqlerrm);
```

В документации по PL/SQL приведен список исключений, которые можно специально обрабатывать. В большинстве случаев самым простым решением будет использование выражения WHEN OTHERS, которое перехватывает все ошибки. Для того чтобы гарантировать обработку каждой ошибки, необходимо всегда включать данное выражение в раздел исключений. Заметьте, что если исключение возникло, управление уже не вернется в раздел кода. В некоторых случаях после обработки ошибки желательно повторно выполнить команду. В документации на PL/SQL приводится ряд примеров, содержащих решения этой проблемы. Обычно для этого включают дополнительный код BEGIN/EXCEPTION/END и перехватывают ошибку непосредственно после строки, в которой она возникает. Можно также поместить весь блок в цикл или использовать условный оператор.

На данном этапе можно удалить кнопку Test и связанный с ней код, однако впоследствии вам придется добавить раздел EXCEPTION, в котором будут обрабатываться ошибки, к обработчику Pre-Form.

Раздел EXCEPTION имеет дело с ошибками, которые возникают при выполнении обработчика. Что же произойдет, если ошибка возникнет в самой форме? Предположим, например, что пользователь случайно удалил первичный ключ и пытается сохранить результаты своих действий. В таком случае интерпретатор форм перехватит ошибку и вернет сообщение пользователю. Эти ошибки обрабатываются обработчиком ON\_ERROR формы. Закройте форму Search и откройте форму Agents.

1. Создайте для формы обработчик OnError.
2. Добавьте простой код, отображающий номер ошибки: `Message('Error Code is: '|| ERROR_CODE||' DBMS Error Code is: '||DBMS_ERROR_CODE);`.
3. Запустите форму и выполните запрос, чтобы загрузить данные.
4. Добавьте букву “A” в конец идентификатора и попытайтесь сохранить запись.

На рис. 9.14 показано сообщение, генерируемое новой программой обработки ошибок. Возможно, вы захотите сохранить код ошибки. Кроме того, вы можете создать дополнительный код, позволяющий отображать более содержательное сообщение. Пример такого кода приведен ниже.

```
IF (ERROR_CODE = 50016) THEN
  Message('You must enter numbers only, not letters.');
```

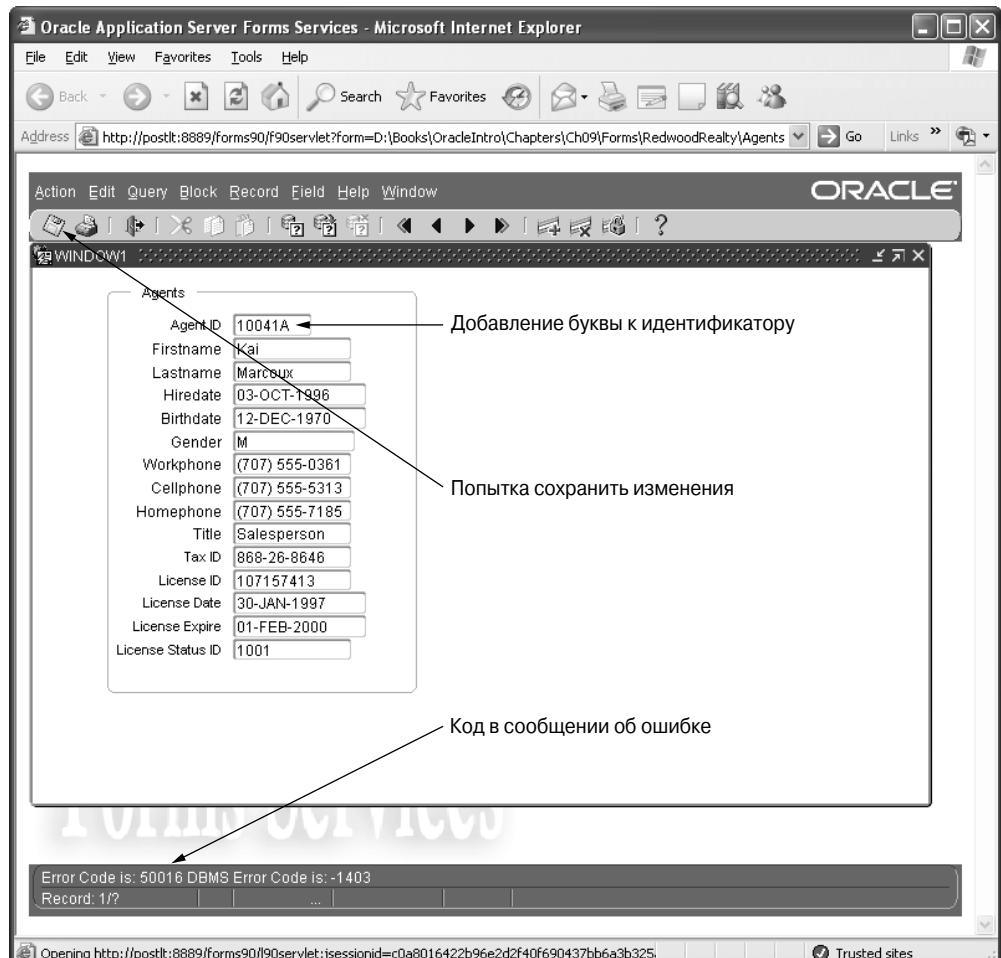


Рис. 9.14. Перехват ошибки, генерируемой формой

```

ELSE
  Message('Error Code is: ' || ERROR_CODE
  || ' DBMS Error Code is: ' ||
  DBMS_ERROR_CODE);
END IF;

```

Подобным образом можно определить действия по обработке различных ошибок. По мере необходимости вы можете написать сложный код, регистрирующий ошибку в файле протокола, отображающий подробное сообщение или перенаправляющий пользователя к другой форме. Заметьте, что для некоторых операций DBMS\_ERROR\_CODE возвращает более подробные сообщения. Некоторые коды ошибок описаны в справочной системе. Например, DBMS\_ERROR\_CODE = -1400 означает, что пользователь забыл ввести значение для столбца.

Написание обработчиков ошибок требует определенного навыка. Необходимо представлять себе, что думает пользователь и проблемы какого типа могут возникать в его работе. Надо также найти наилучший способ обработки ошибок. Выполнения данную работу, следует помнить, что основная цель — упростить пользователю решение стоящих перед ним задач. Не следует утомлять пользователя непонятными сообщениями. Где возможно, необходимо предусмотреть автоматическое исправление ошибок. Создавая некоторые приложения, приходится подолгу наблюдать за работой пользователя с формами. Таким образом, можно выявить потенциальные проблемы и предложить наилучший способ их решения.

## Область видимости и время жизни

При разработке приложений важно представлять себе область видимости и время жизни переменных в составе форм. Область видимости определяет, когда переменные будут доступны. Время жизни — это интервал между моментом создания переменной и ее удалением. Основное правило таково: переменная создается и доступна лишь в подпрограмме или разделе, в котором она определена. Для форм переменные чаще всего объявляются в обработчиках событий. Рассмотрим следующий код:

```
DECLARE
    i1 NUMBER;
BEGIN
    i1 := 5;
    IF (i1 > 3) THEN
        DECLARE
            i2 NUMBER;
        BEGIN
            i2 := i1 + 10;
        END;
    END IF;
END;
```

Вы можете создать новую кнопку в форме и включить данный код в состав обработчика. Установите в начале обработчика точку останова и выполните его код в пошаговом режиме. Это поможет вам определить область видимости двух переменных (*i1* и *i2*). Заметьте, что *i1* объявлена в начале обработчика, поэтому она создается сразу же после возникновения события. Ее значение доступно только обработчику. Код обработчик завершает свое выполнение (последнее выражение *END*), данная переменная разрушается. Если вы щелкнете на кнопке, чтобы снова выполнить код обработчика, все начнется сначала, причем в начале кода переменным не будут присвоены значения. Вторая переменная (*i2*) объявлена в блоке, находящемся в составе выражения *IF*. Ее значение доступно только в этом блоке. Если вы попытаетесь обратиться к ней после выражения *END IF*, возникнет ошибка выполнения, так как переменная уже не существует.

Проблемы, связанные с временем жизни переменных, особенно важны в тех случаях, когда необходимо обеспечить совместное использование данных несколькими обработчиками формы. Например, когда вы тестируете значение, введенное пользователем, вам надо знать исходное содержимое поля редактирования. Выполните следующие действия с формой Search.

1. Создайте новый обработчик Pre-Text-Item для элемента Txt\_MinPrice.

```
:GLOBAL.MinPrice := :TXT_MINPRICE;
```

2. Создайте новый обработчик Post-Text-Item для элемента Txt\_MinPrice.

```
IF (:GLOBAL.MINPRICE <> :TXT_MINPRICE) THEN
    Message('Minimum price has been changed.');
END IF;
```

3. Запустите форму и измените минимальную цену с 0 до 50000. Обратитесь к строке сообщений и убедитесь, что в ней отображается информация о том, что значение было изменено.

Проблема, возникающая в данном случае, связана с тем, что любая переменная, созданная в составе обработчика Pre-Text-Item, удаляется, как только обработчик завершает свое выполнение. Другими словами, к тому времени, как обработчик Post-Text-Item получит управление, все локальные переменные перестанут существовать. Таким образом, вам надо хранить исходное значение (0) в той переменной, время жизни которой будет больше, чем время выполнения обработчика. Другими словами, необходимо создать глобальную переменную. Время жизни глобальной переменной определяется временем жизни сеанса, в течение которого может быть запущено несколько форм. Область видимости глобальных переменных позволяет обращаться к ним из любого обработчика любой формы в пределах одного сеанса. Таким образом, глобальные переменные подходят для разделения данных между различными обработчиками и различными формами.

Заметьте, что сеанс уникален для каждого пользователя, поэтому глобальная переменная, созданная одним пользователем, недоступна другому, даже если они оба работают одновременно с одной и той же формой. Это очень удобно, так как не приходится беспокоиться о конфликтах, связанных с использованием глобальных переменных. Если же вам надо обеспечить разделение данных между пользователями, то придется записывать их в базу.

Глобальные переменные хорошо подходят для решения многих сложных задач, связанных с формами. Их нельзя определить заранее. Имя глобальной переменной предваряется префиксом GLOBAL (GLOBAL.имя), и ей присваивается значение.

---

# Создание инструментальных средств

Как вы помните из предыдущих примеров, создание форм предполагает написание кода для обработки некоторых событий, например активизации кнопок. Во многих случаях для этого достаточно нескольких строк кода. Чтобы решить задачи, которые реже встречаются при работе с базами данных, код, возможно, будет более сложным. В этом разделе мы поговорим о реализации основных функций, встречающихся во многих приложениях. Каждому примеру посвящен отдельный раздел; вы можете обратиться к ним, если одна из подобных задач возникнет при работе над вашим проектом. В предлагаемых здесь примерах также уделяется внимание синхронизации событий и другим проблемам.

## Формирование последовательностей

Как вы помните, одна из самых важных особенностей реляционных баз данных — это наличие в каждой таблице первичного ключа. Во многих случаях самым простым решением оказывается создание нового ключевого столбца, который выполняет роль идентификатора при работе с базой данных. Например, в таблице `Agents` есть столбец `AgentID`, который играет роль первичного ключа. Может возникнуть вопрос: откуда получать номера, которые гарантированно были бы уникальными? В больших организациях всегда есть отдел кадров, и в корпоративных приложениях обычно выделяется отдельный процесс, который отвечает за присвоение идентификаторов сотрудникам, вновь принятым на работу, и включение соответствующих данных в таблицы базы. Но все же кто-то должен отвечать за присвоение идентификационных номеров и обеспечивать их уникальность в пределах базы данных.

Вместо того чтобы возлагать подобные обязанности на сотрудников, лучше реализовать в базе данных генерацию номеров, используемых в качестве ключевых значений. Для подобных целей в Oracle используются последовательности. *Последовательность* — это именованный набор номеров, который Oracle использует при генерации новых значений. Последовательность имеет следующие особенности: во-первых, ее надо установить заранее, а во-вторых, для того, чтобы реализовать запрос на генерацию нового значения, нужно написать код. Последовательные номера не создаются автоматически при включении новой строки. Для генерации соответствующего значения и включения его в соответствующий столбец нужен специальный код. Можно написать обработчик события `Insert` для таблицы, который автоматически генерировал бы номер. Однако в Oracle чаще используется другой подход — ключевые значения генерируются по мере необходимости. Обычно это делается в форме.

Как правило, последовательность определяется при создании исходной таблицы. Но это не обязательно. Поскольку между последовательностью и таблицей нет непосредственной связи, последовательность можно определить в любой момент перед

созданием номеров. Она создается единожды, и для этого можно воспользоваться SQL\*Plus.

- Поскольку данные уже имеются в таблице, найдем максимальное значение.

```
SELECT Max(AgentID) FROM Agents;
```

- Максимальное значение равно 15521, поэтому новые номера надо начинать с больших величин. Значение 15522 вполне подошло бы, но для надежности примем его равным 20000.

```
CREATE SEQUENCE seq_Agents
INCREMENT BY 1
START WITH 20000;
```

- Для получения новых значений используются средства SQL, автоматически увеличивающие счетчик на величину, указанную при определении. Сгенерируем значение, чтобы убедиться, что мы правильно понимаем работу команды.

```
SELECT seq_Agents.NEXTVAL FROM dual;
```

Данная команда должна вернуть значение 20000. При повторном вызове она вернет 20001 и т.д. Заметьте, что последовательность лишь предоставляет номера. Она не выполняет с ними никаких действий. Использовать полученные значения в качестве первичных ключей — это наш выбор. При желании вы можете делать с номерами что угодно, но использовать последовательность надо для решения лишь одной задачи. Тщательно выбирайте имя последовательности, чтобы ее назначение было понятно всем.

Теперь, когда последовательность задана, надо добавить к форме новый код, который автоматически генерировал бы новое значение при добавлении информации о новом агенте. Запустите Forms Builder и откройте форму Agents. Лучше всего использовать в качестве исходного вариант формы, созданный в главе 8, поэтому скопируйте файлы Agents.fmb и Agents.fmx в папку, предназначенную для работы с примерами, рассматриваемыми в этой главе. Если файлы у вас не сохранились, можно использовать более простой вариант формы, который был разработан в этой главе.

Как обычно, в первую очередь надо решить, обработчик какого события будет использоваться. На первый взгляд, подходящим кажется On-Insert, но это событие происходит слишком поздно. Как вы помните, когда пользователь работает с формой, он использует Insert Record для создания пустой страницы. При этом возникает событие When-Create-Record. Событие же On-Insert не будет сгенерировано до тех пор, пока пользователь не щелкнет на кнопке Save.

- Создайте новый обработчик для события When-Create-Record.
- Воспользовавшись пунктом меню File⇒Connect, зарегистрируйтесь для работы с базой данных.

- Добавьте код, генерирующий новый последовательный номер для Agents.

```
SELECT seq_Agents.NEXTVAL INTO :AGENTS.AGENTID FROM dual;
```

- Щелкните на кнопке Compile, чтобы убедиться, что вы не допустили синтаксических ошибок.

**Совет.** Если вы получите сообщение об ошибке взаимодействия с таблицей, вероятнее всего, причина в том, что вы не установили соединение с базой данных.

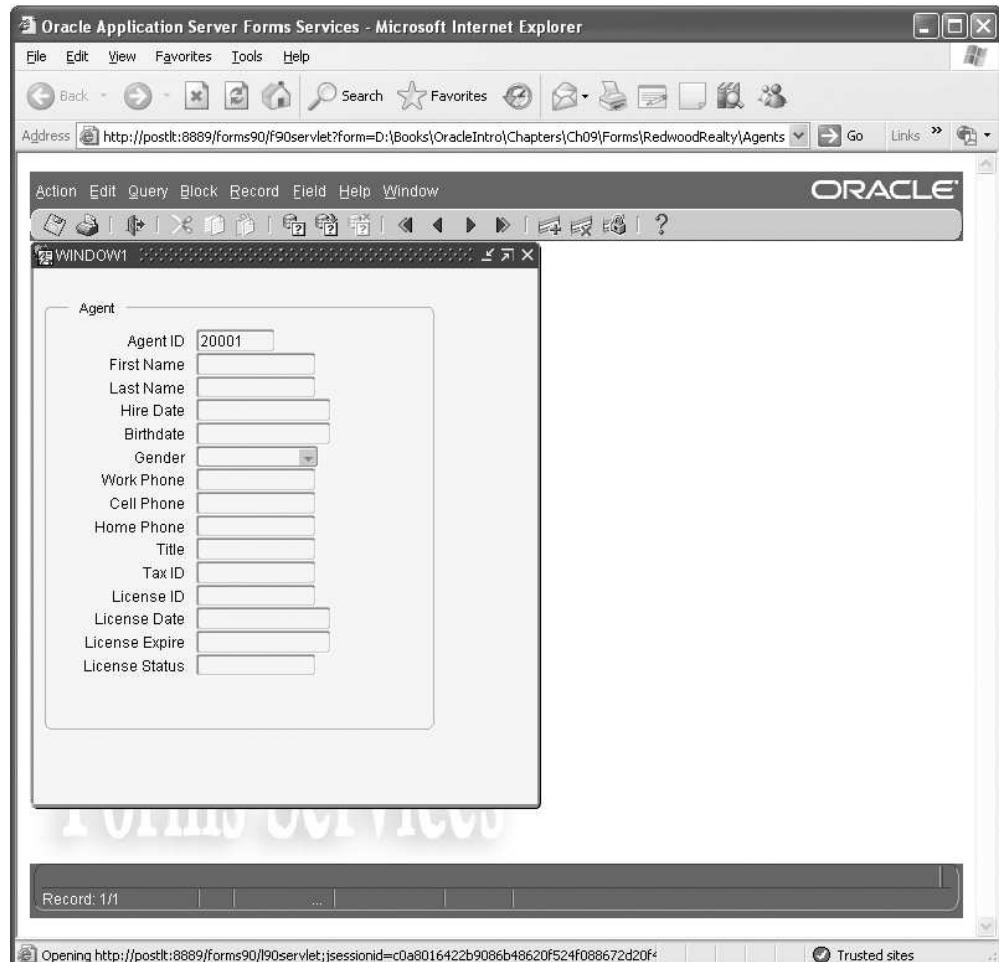
- Запустите форму и убедитесь, что новый идентификатор сгенерирован.

На рис. 9.15 показана форма Agents. Заметьте, что событие When-Create-Record генерируется при первом открытии формы. Следовательно, новый идентификатор генерируется каждый раз, когда кто-либо запускает форму. Если пользователь не планирует включать сведения о новом агенте, щелчком на кнопке Execute Query будет загружена форма для существующих агентов. При этом сгенерированный идентификатор не будет использоваться. Поскольку доступны более миллиарда номеров, потеря некоторых из них не представляет серьезной проблемы. Все же, если вы предполагаете, что пользователи будут редко вводить новые записи посредством формы, желательно изменить программу, чтобы она использовала номера менее расточительно. Добавьте к обработчику When-New-Form-Instance выражение Execute\_Query. Этим вы скроете от пользователя пустое исходное окно, но последовательность будет по-прежнему генерировать значения.

Для того чтобы запретить создание ненужных номеров, рассмотрим события, представленные на рис. 9.7. Заметьте, что событие When-Create-Record происходит между событиями Pre-Form и When-New-Form-Instance. Это событие позволит нам создать глобальную переменную, выполняющую функции индикатора.

- В обработчик Pre-Form включите выражение :GLOBAL.DoNotAdd := 1;.
- В обработчике When-New-Form-Instance сбросьте то же самое значение :GLOBAL.DoNotAdd :=0;.
- Включите в обработчик When-Create-Record выражение IF, с помощью которого вы будете определять, следует ли генерировать идентификатор: IF (:GLOBAL.DoNotAdd = 0) THEN . . . .

Эти действия покажутся вам рутинными. Однако благодаря им форма станет проще в использовании, следовательно, выполнить их необходимо. Подобные детали делают приложения более профессиональными.



**Рис. 9.15.** Автоматическая генерация значений AgentID

## Проверка вводимых данных

При разработке пользовательских интерфейсов необходимо придерживаться важного правила: ошибки следует выявлять и идентифицировать как можно ближе к источнику их возникновения. Если пользователь вводит данные в форму, он всегда может допустить неточность. По мере возможностей программное обеспечение формы должно идентифицировать проблему и сразу же информировать об этом пользователя, чтобы он смог внести исправления.

Существуют различные способы проверки данных, вводимых пользователем в форме Oracle. Некоторые из этих способов учтены в свойствах элементов формы. Вернемся к рассмотрению формы Agents. Как вы, наверное, помните, значение

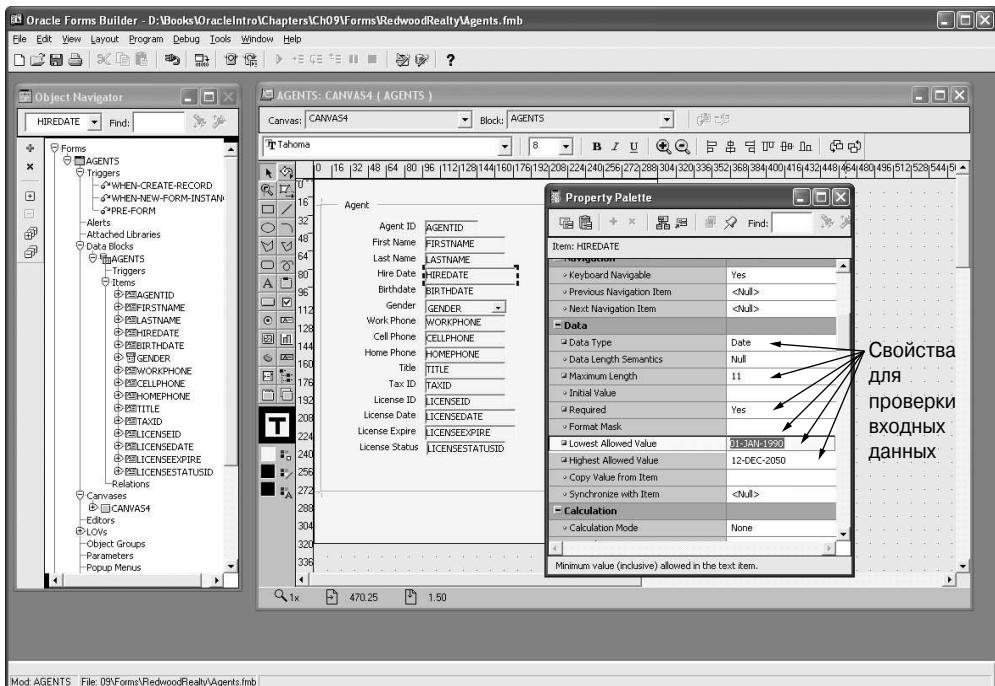
Gender вводится посредством раскрывающегося списка. Пользователь должен лишь выбрать требуемый пункт. Такой подход гарантирует, что для свойства будет задано лишь одно из допустимых значений. Конечно, невозможно предотвратить неправильный ввод пользователем поля сотрудника, но пока не существует программных средств, позволяющих выявить такую ошибку.

Откройте палитру свойств для поля License Status. Прокрутите список и найдите LOV, созданный для этого элемента. Для надежности вы установили свойство, согласно которому пользователь может вводить значения, лишь выбирая их из списка. Если вы обратите внимание на исходную структуру базы данных, то увидите, что данный столбец связан с таблицей LicenseStatus посредством внешнего ключа. Такая зависимость также предотвращает ввод пользователем недопустимого значения.

Другие свойства также предоставляют средства проверки. Например, вы можете задать тип данных, максимальную длину, определить, обязательно ли должно быть указано значение, и установить минимальную и максимальную величину.

1. Откройте палитру свойств для элемента HireDate. Убедитесь в том, что указан тип данных Date.
2. Задайте значение Yes свойства Required, установите свойство Lowest Allowed Value равным 01-Jan-1990, для свойства Highest Allowed Value задайте значение 30-Dec-2050.
3. Запустите форму и попытайтесь изменить значение в поле HireDate для кого-либо из сотрудников на любой день 1980 года. Интерпретатор форм отвергнет изменение и проинформирует вас об этом в строке сообщений.
4. При желании вы можете установить исходным значением поля HireDate текущую дату: \$\$DATE\$\$. В результате при добавлении информации о новом сотруднике по умолчанию будет установлен тот день приема на работу, когда заполнялась форма. Можно задать и другую дату.

На рис. 9.16 показаны значения основных свойств, используемых для проверки входных данных. Существует еще одно свойство, Format Mask, с помощью которого можно контролировать информацию, вводимую пользователем в поле редактирования. Маска формата (Format Mask) позволяет устанавливать допустимые сочетания символов. Например, вы можете задать для поля, в котором вводится номер рабочего телефона сотрудника, значение свойства Format Mask, равное FM"("999")" "999"- "9999". Заметьте, что символы-разделители (круглые скобки и дефис) помещаются в двойные кавычки. Данный формат ограничивает возможности пользователя. Задавая телефонный номер, он вынужден указать код региона и семь цифр номера. Маски формата можно задавать также для полей, содержащих символьные последовательности и даты. Прежде чем задавать маски форматов для каждого поля, следует задуматься о том, какие последовательности символов на самом деле доступны. Возьмем



**Рис. 9.16.** Основные свойства, используемые для проверки входных данных

в качестве примера почтовые индексы. Ограничив данные, вводимые пользователем, пятью цифрами, вы исключите возможность ввода кодов, состоящих из девяти цифр, а также запретите ввод кодов для Канады и некоторых других стран. Подобные проблемы могут возникнуть и с телефонными номерами. Ограничив номер десятью цифрами, вы не дадите возможности вводить многие номера других стран и указывать дополнительный номер, поддерживаемый АТС предприятия. В результате можно сделать вывод, что маски форматов имеют ограниченное применение. Возможно, для какого-нибудь идентификатора, используемого внутри компании, можно выбрать подходящую маску формата, но полагаться на эту возможность для данных, получаемых из внешних источников, нежелательно.

Третий способ проверки допустимости данных — написание кода PL/SQL, контролирующего корректность информации. Для каждого элемента формы Oracle поддерживает событие When-Validate-Item.

1. Щелкните правой кнопкой мыши на поле Birth Date и добавьте обработчик события When-Validate-Item.
2. Напишите код, который будет проверять, достиг ли сотрудник восемнадцатилетнего возраста.

```
IF (ADD_MONTHS(:BIRTHDATE, 18*12) > CURRENT_DATE) THEN
  Message('The person must be at least 18 years old.');
  RAISE FORM_TRIGGER_FAILURE;
END IF;
```

- Запустите форму и проверьте действия обработчика, задавая различные даты в поле Birth Date.

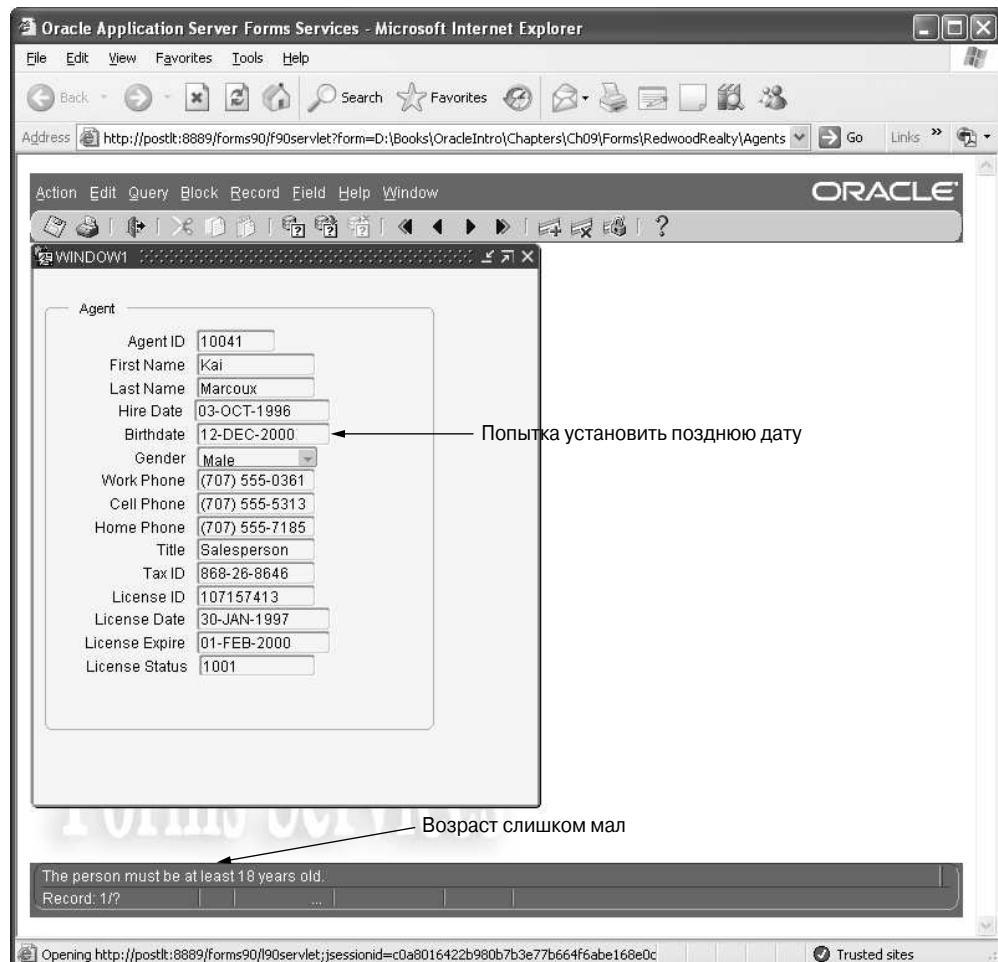
На рис. 9.17 показано, что произойдет, если вы запустите форму и попытаетесь ввести слишком позднюю дату рождения. Обработчик, выполняющий проверку, выявляет тот факт, что претендент на занятие должности не достиг 18 лет. Он отображает сообщение и генерирует ошибку (FORM\_TRIGGER\_FAILURE), которая блокирует работу с формой. Обработчики, имеют существенное преимущество перед другими средствами проверки: вы можете написать сколь угодно сложный код для контроля любых параметров. Можно также написать код PL/SQL, который будет обращаться к базе и сравнивать данные из нескольких таблиц. Для проверки сложных условий используются вложенные выражения IF/THEN. К счастью, большинство условий, встречающихся в бизнес-приложениях, достаточно просты, но полезно знать о наличии возможностей для более детальной проверки. Однако имейте в виду: написав сложный код большого объема, вы затрудните работу специалиста, который будет сопровождать ваш продукт. Следите за тем, чтобы код был хорошо документирован.

## Вычисление итоговых значений для табличных данных

Инструмент, о котором пойдет речь, широко используется в приложениях электронной коммерции. Представьте себе, что в подчиненной форме отображается список товаров, выбранных пользователем, и цены на них. Основная форма содержит общую стоимость изделий, выбранных в данном сеансе. В приложении для Redwood Realty подобная форма не используется. Однако в форму Search было бы неплохо включить счетчик полученных объектов. Счетчик реализуется практически так же, как и суммирование. Единственное отличие — в используемой функции (Count вместо Sum).

Проблема с определением итоговых значений состоит в том, что все вычисления должны выполняться в блоке данных подчиненной формы, поскольку именно там содержится требуемая информация. Но данные, отображаемые в табличной форме, повторяются в каждой строке, что создает некомфортные условия для работы пользователя. Таким образом, все вычисления, связанные с определением значения счетчика, должны выполняться в блоке данных табличной формы, а результаты должны отображаться в основной форме.

- Откройте форму Search и добавьте поле к блоку данных ListingResults. Надо также уменьшить ширину нового поля редактирования и расположить его возле правой границы фрейма.



**Рис. 9.17.** Сообщение о том, что при проверке входных данных обнаружена ошибка

2. Установите свойства поля редактирования так, как показано в табл. 9.1.
3. Откройте палитру свойств для блока данных ListingResults. Установите Query All Records равным Yes.
4. Задайте в качестве блока данных по умолчанию Search\_Block и добавьте поле редактирования непосредственно над фреймом для табличных данных.
5. Установите свойства так, как показано в табл. 9.2.
6. Сохраните форму и запустите ее.

**ТАБЛИЦА 9.1.** Свойства поля редактирования подчиненной формы для выполнения вычислений

| <i>Свойство</i>    | <i>Значение</i> |
|--------------------|-----------------|
| Name               | MatchCount      |
| Enable             | No              |
| Keyboard Navigable | No              |
| Data Type          | Number          |
| Calculation Mode   | Summary         |
| Summary Function   | Count           |
| Summarized Block   | LISTINGRESULTS  |
| Summarized Item    | LISTINGID       |
| Database Item      | No              |
| Visible            | No              |
| Width              | 5               |
| Height             | 1               |
| Prompt             | Count           |

**ТАБЛИЦА 9.2.** Свойства поля редактирования для отображения итогового значения

| <i>Свойство</i>  | <i>Значение</i>            |
|------------------|----------------------------|
| Name             | MatchCount                 |
| Enabled          | No                         |
| Justification    | Right                      |
| Data Type        | Number                     |
| Format Mask      | 9,990                      |
| Calculation Mode | Formula                    |
| Formula          | :LISTINGRESULTS.MatchCount |
| Database Item    | No                         |
| Prompt           | Number of Matches          |

**Совет.** Если счетчик не отображается в поле основной формы, выведите формулу для этого поля, дважды щелкнув мышью, и убедитесь, что вы задали двоеточие в начале выражения.

Значение счетчика должно отображаться в новом поле. Оно выделено серым цветом, поскольку для свойства Enabled установлено значение No.

На рис. 9.18 показаны результаты поиска. Если вы выбрали другие условия поиска, ваши данные будут отличаться от приведенных на рисунке.

The screenshot shows a web browser window for Oracle Application Server Forms Services. The address bar contains the URL `http://postlt:8889/forms90/f90servlet?form=D:\Books\OracleIntro\Chapters\Ch09\Forms\RedwoodRealty\Search.`. The page title is "Redwood Realty". The main content area displays a "Listing Search" form with various input fields and a results grid. The search criteria include:

- Minimum Bedrooms:  3
- Minimum Bathrooms:  2
- Minimum Square Feet:  2000
- Maximum Age:  1990
- Minimum Price: 100000
- Maximum Price: 300000
- Sale Status: For Sale
- Number of Matches: 20

The "Listing Results" section displays a table with 8 rows of data:

| Listing ID | Property ID | Sale Status | Price  | Bed | Bath | SQFT | Year | City     | Agent      |
|------------|-------------|-------------|--------|-----|------|------|------|----------|------------|
| 15258      | 1029        | For Sale    | 125000 | 4   | 2    | 2041 | 1990 | Eureka   | Williams   |
| 15043      | 1281        | For Sale    | 135000 | 4   | 2    | 2161 | 1990 | Fortuna  | Sheibani   |
| 15899      | 1452        | For Sale    | 144500 | 4   | 2    | 2300 | 1991 | Eureka   | St-Onge    |
| 16132      | 1481        | For Sale    | 146500 | 3   | 2    | 2086 | 1990 | Eureka   | Silverburg |
| 15149      | 1521        | For Sale    | 149990 | 4   | 2    | 2348 | 1996 | Eureka   | Townsend   |
| 15588      | 1574        | For Sale    | 154000 | 3   | 2    | 2060 | 1991 | Loleta   | Weber      |
| 15243      | 1603        | For Sale    | 155000 | 4   | 2    | 2348 | 1996 | Trinidad | Flamenbaum |

At the bottom, a status bar shows "Record: 1/1" and the URL "Opening http://postlt:8889/forms90/f90servlet?jsessionid=c0a8016422b900a92d11529140fab830b81d".

Рис. 9.18. Результаты поиска со счетчиком записей

Как видите, для того, чтобы реализовать счетчик элементов, удовлетворяющих критерию поиска, пришлось выполнить довольно много работы. Конечно, данная возможность не является критичной для работы формы, но она удобна для пользователя. Еще более важен тот факт, что аналогичный способ можно использовать для вычисления суммарных и даже средних значений. Преимущество данного подхода состоит в том, что все вычисления осуществляются на клиентском компьютере. Если табличная форма используется для ввода новых данных, итоговое значение обновляется практически мгновенно. Для этого не приходится лишний раз обращаться к серверу.

## Использование нескольких холстов

Все формы, которые мы рассмотрели до сих пор, были созданы так, что все данные отображались в них одновременно. Например, подчиненная форма выводилась наряду с основной. С технической точки зрения вся информация отображалась на одном холсте. Для многих приложений такой подход оправдан, поскольку пользователи видят сразу все необходимые им сведения. При работе над большими проектами можно распределить обязанности так, что каждый разработчик будет отвечать за отдельную форму. Различные формы будут работать независимо друг от друга, а благодаря связям между ними они составят единое приложение.

Однако объем данных, которые можно поместить на один экран, ограничен. В этой ситуации Oracle предлагает специальное решение. Данные можно разместить на отдельных холстах. Пользователь получает контроль над отображаемой информацией и может переключаться с одной группы данных на другую. Существуют три способа организации холстов: в виде отдельных областей отображения, в виде слоев и в виде вкладок. С точки зрения разработчика, создание любого из трех вариантов холстов предполагает выполнение одной и той же последовательности действий. Различия между ними заключаются лишь в стилях и способе взаимодействия пользователя с формой. Некоторые организации принимают решение о единой структуре пользовательского интерфейса и стремятся, чтобы все приложения выглядели одинаково. В ряде случаев целесообразно реализовать несколько вариантов и предоставить пользователю возможность выбирать наиболее подходящий из них.

### Добавление холста

Чтобы проиллюстрировать работу с несколькими холстами, создадим форму использующую два отдельных холста. В вашем распоряжении уже есть форма Search. Чтобы не повредить исходный вариант формы поиска, создайте ее копию.

1. Найдите форму средствами Windows и скопируйте файлы Search.fmb и Search.fmx.
2. Присвойте копиям имена Search2.fmb и Search2.fmx.
3. Откройте новую форму Search2.fmb и, если отображается кнопка Test Scope, удалите ее. Также удалите обработчики для элемента `Txt_MinPrice`.
4. Сохраните форму Search2 и создайте две дополнительные копии, Search3 и Search4, которые будут использованы в последующих разделах.

Форма Search отображает PropertyID для списков элементов, удовлетворяющих условиям поиска, а также представляет основные сведения о свойствах. Предположим, что пользователи решили, что им необходима вся информация о некоторых свойствах. В принципе можно включить дополнительные столбцы в табличную фор-

му, однако она и так уже перегружена информацией. Вместо этого создадим новый холст, на котором будет полностью отображаться запись для одного свойства. Добавить холст сравнительно просто, к тому же вы уже умеете создавать блок данных и изменять компоновку элементов.

1. Добавьте новый холст, выбрав пункт **Search\_Properties** в объектном навигаторе и щелкнув на кнопке **Create (+)**. Присвойте новому холсту имя **Properties** и убедитесь, что в списке, содержащемся в навигаторе, он располагается ниже **Search\_Canvas**. Холст, расположенный первым, отображается при открытии формы; это должен быть **Search\_Canvas**.
2. Добавьте блок данных, используя соответствующий мастер. Выберите таблицу **Properties** и все ее столбцы. Создайте объединение, используя следующее выражение:

```
Properties.PropertyID= ListingResults.PropertyID.
```

**Совет.** Если при попытке создать связь, система отобразит сообщение об ошибке, убедитесь, что флажок опции **Auto Join** сброшен.

3. При работе с мастером компоновки очень важно выбрать вновь созданный холст **Properties**. Если вы пропустили первый этап, то можете создать новый холст сейчас. Как обычно, установите ширину столбцов, чтобы данные были доступны пользователю.
4. На данный момент на новом холсте есть информация, но необходимо обеспечить ее отображение. Перейдите к **Search\_Canvas** и добавьте в основную форму новую кнопку. Пометьте ее как **Property**. Создайте обработчик события **When-Button-Pressed**, который будет выглядеть следующим образом:

```
GO_ITEM('PROPERTIES.PROPERTYID');
```

Можно было бы поместить кнопку в таблицу с результатами, но для таблицы выделено ограниченное пространство, поэтому оставим ее в верхней части формы. Для того чтобы несколько упростить работу пользователей, создайте обработчик для элемента **PropertyID**. Этот обработчик связывается с событием **When-Mouse-DoubleClick**. Пользователь получает возможность отобразить свойства, либо щелкнув на кнопке **Property**, либо дважды щелкнув на **PropertyID** в списке результатов.

5. На холсте **Properties** должна быть кнопка, с помощью которой пользователь сможет вернуться к странице **Search**. Добавьте кнопку и пометьте ее как **Search**. Создайте обработчик события **When-Button-Pressed**, который будет выглядеть следующим образом:

```
GO_ITEM('SEARCH_BLOCK.BTN_SEARCH');
```

6. Скомпилируйте код и проверьте его работу.



**Рис. 9.19.** Форма с двумя холстами

На рис. 9.19 показан внешний вид формы с двумя холстами. При запуске она выглядит так же, как и исходная форма поиска. Щелкните на кнопке Search, и вы получите список элементов, удовлетворяющих критерию поиска. Теперь выберите один из пунктов списка, щелкнув в поле редактирования, принадлежащем соответствующей строке. Затем щелкните на новой кнопке Property. Произойдет переключение холстов, и вы увидите данные для выбранного свойства, в частности PropertyID. Щелкнув на кнопке Search, вы вернетесь к холсту Search и получите возможность осуществить новый поиск. Находясь на странице поиска, можно дважды щелкнуть в поле PropertyID, чтобы отобразить холст Property, содержащий подробную информацию о свойствах. Основная часть работы скрыта от пользователя, благодаря тому, что вы создали связь между блоками данных Properties и ListingResults. Такой подход можно применить для любых таблиц данных, допускающих связывание.

## Холсты в виде слоев

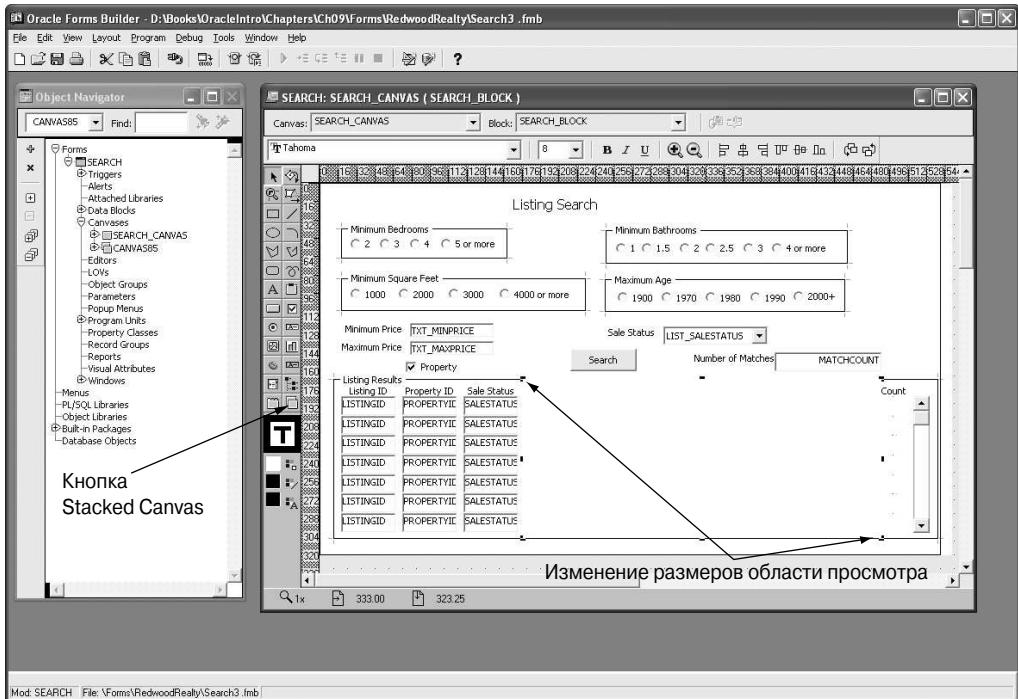
Несколько холстов в составе одной формы используются для того, чтобы дать пользователю возможность переходить от одного набора данных к другому. Но можно ли переключаться более оперативно? И что делать, если пользователь хочет видеть и кри-

терии поиска, и список, и подробную информацию о свойствах? Пытаясь отобразить большой объем данных в одном окне, есть риск перегрузить его информацией. Для решения этой проблемы можно использовать холсты в виде слоев. В данном случае слово слой означает, что один холст отображается поверх другого. При работе с такими холстами формируется *область просмотра*. В результате один холст частично перекрывает другой. Написав код, управляющий отображением или скрытием холстов, вы предоставите пользователю возможность решать, следует ли отображать дополнительные данные.

Холсты, расположенные в виде слоев, создаются практически так же, как и холсты других типов.

- Если необходимо, закройте и удалите все формы в Forms Builder. Откройте копию исходной формы Search. Если вы выполняли копирование так, как было рекомендовано в предыдущем разделе, используйте файл Search3.fmb.
- Щелкните на кнопке **Stacked Canvas**, расположенной на панели инструментов. Сформируйте область просмотра в правой части таблицы, при этом пользователю должны быть видны по крайней мере столбцы ListingID и PropertyID. На рис. 9.20 показан новый холст, частично перекрывающий список результатов поиска. Присвойте холсту имя **Properties**.
- Активизируйте мастер блоков данных и выберите все столбцы в таблице **Properties**. Создайте объединение: **Properties.PropertyID=ListingResults.PropertyID**.
- Запустите мастер компоновки и убедитесь, что выбран холст **Properties**. Когда мастер завершит работу, подберите размеры фрейма, которые соответствовали бы области просмотра. Область просмотра отмечается прямоугольником темного цвета. Видимыми будут только данные, попадающие в нее. Желательно выровнять заголовок фрейма по центру.
- Сохраните результаты ваших действий, запустите форму и проверьте ее работу. Новый холст **Properties** постоянно закрывает некоторые данные из списка, что очень неудобно, поскольку пользователю надо время от времени обращаться к ним.
- Добавьте на холст **Search\_Block** флажок опции, установив для него следующие свойства: **Name=CHK\_PROPERTY, Label=Property, Value when Checked=1, Value when Unchecked=0, Data Type=Number, Initial Value=0, Database Item=No**.
- Если вы хотите, чтобы при запуске формы холст **Properties** был скрыт, отредактируйте обработчик события **Pre-Form**. Перед началом и после окончания кода (перед выражением **EXCEPTION**) добавьте следующую строку:

```
HIDE_VIEW('PROPERTIES');
```



**Рис. 9.20.** Добавление холста, выполненного в виде слоя

8. Создайте новый обработчик When-Checkbox-Changed для флашка опции.

```
IF (:CHK_PROPERTY > 0) THEN
    SHOW_VIEW('PROPERTIES');
ELSE
    HIDE_VIEW('PROPERTIES');
END IF;
```

9. Проверьте форму и выберите такой цвет фона для холста Properties, при котором данные воспринимались бы наилучшим образом.

На рис. 9.21 показан описанный в данном разделе холст в действии. Флажок *Property* используется для отображения и скрытия холста, выполненного в виде слоя. Если холст будет видимым, то выбор очередной строки в списке приведет к обновлению данных на этом холсте. Если взаимодействующие компьютеры находятся в одной локальной сети, то обновление происходит практически мгновенно. Если же обмен данными происходит по Интернету, следует тщательно проверить работу формы. Правда, если обновление будет осуществляться слишком медленно, пользователь может попросту скрыть холст *Properties* и не обращаться к нему до тех пор, пока не понадобится подробная информация о свойствах.

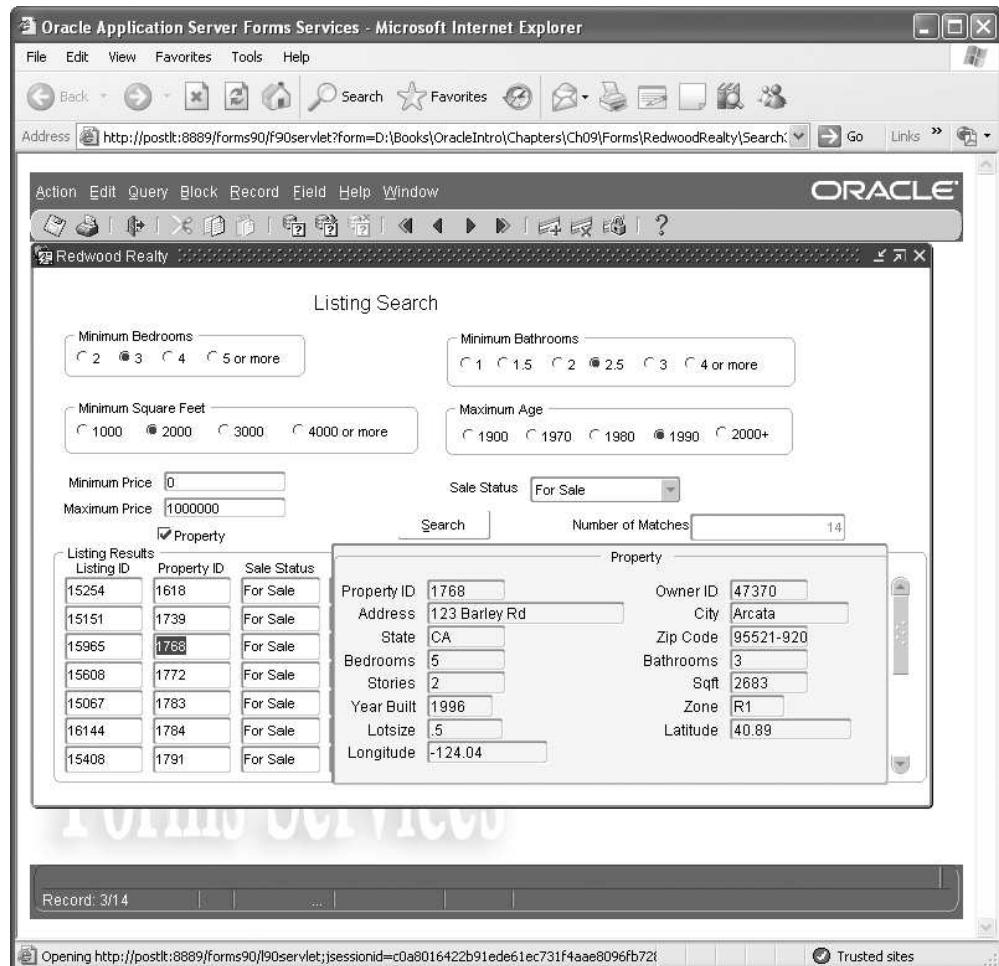


Рис. 9.21. Отображение холстов в виде слоев

## Холсты в виде вкладок

Вкладки часто используются в интерфейсах прикладных программ. Например, в системе Windows таким образом отображаются наборы свойств, связанные между собой. Некоторые разработчики приняли данный подход в качестве основного принципа формирования интерфейсов. Наборы данных отображаются на отдельных страницах. Ярлыки вкладок, находящиеся в верхней части окна, позволяют без труда переходить от одной страницы к другой. Подобный принцип используется также для организации данных на бумажных носителях. Для удобства поиска документов папки снабжаются ярлыками.

Создавать холсты-вкладки несколько сложнее, чем независимые области или холсты в виде слоев. Обычно решение об использовании холстов в виде вкладок принимают в самом начале работы над приложением. Поскольку повторно создавать всю форму Search слишком долго, мы в данном разделе продемонстрируем, как реализуются холсты в виде вкладок в готовой форме. Сначала надо создать новый холст и переместить на него элементы из Search\_Canvas.

1. Начнем работу над копией исходной формы Search. Будем использовать файл Search4.fmb, созданный в предыдущем разделе.
2. Создайте новый холст, выбрав узел Canvas в объектном навигаторе и щелкнув на кнопке Create. Установите основные свойства: Name=MainTab, Canvas Type=Tab, Background=white.
3. Первоначально холст содержит две отображаемые страницы (вкладки). С помощью палитры свойств переименуйте каждую страницу и создайте для них метки. Для первой вкладки используйте свойства Name=PG\_SEARCH, Label=Search, а для второй — Name=PG\_PROPERTY, Label=Property.

**Совет.** Для того чтобы установить свойства страницы холста, щелкните правой кнопкой мыши на странице (но не на ярлыке) либо выберите страницу в объектном навигаторе.

4. Необходимо перенести графические элементы с Search\_Canvas на холст MainTab. Сделать это проще всего, выбрав их как группу в составе узла навигатора Search\_Canvas/Graphics и перенеся в раздел MainTab/PG\_Search/ Graphics. В результате будут также перенесены заголовок и фреймы.
5. Теперь надо связать все элементы блока данных с новым холстом и страницей. Их также можно перенести как группу. Откройте палитру свойств, затем выберите все элементы в объектном навигаторе (для этого можно выбрать их мышью, нажав клавишу <Shift>, либо выделить на исходном холсте). Найдите свойства Canvas и Page и установите для Canvas значение MainTab, а для Page — PG\_Search. Затем перейдите к блоку данных ListingResults и выполните те же действия для каждого из его элементов. Работая с группой элементов, не модифицируйте другие свойства, так как изменения одновременно затрагивают все элементы. На рис. 9.22 показаны два основных свойства, которые вам необходимо установить.
6. Тщательно проверьте результаты ваших действий и убедитесь, что вы перенесли все элементы, присутствующие на холсте Search\_Canvas. Проще всего сделать это, просматривая Search\_Canvas, щелкая правой кнопкой мыши на каждом оставшемся элементе и изменения свойства, определяющие холст и страницу.
7. Удостоверьтесь, что холст MainTab расположен в объектном навигаторе над Search\_Canvas и, следовательно, будет открыт по умолчанию. Запустите форму

и проверьте ее работу. Если все сделано корректно, вы можете удалить исходную форму `Search_Canvas`.

- На данном этапе мы лишь воспроизвели исходную поисковую форму. Новые возможности станут доступны, когда мы добавим блок данных `Properties`. Перейдите на страницу `Property` и, используя мастер блоков данных, добавьте новый блок. Как и ранее, включите все столбцы из таблицы `Properties` и создайте объединение.

```
Properties.PropertyID= ListingResults.PropertyID.
```

- Запустите мастер компоновки и убедитесь, что вы выбрали холст `MainTab` и страницу `PG_Property`.
- Теперь можно запустить форму и переключаться между страницами, щелкая на их ярлыках, расположенных в верхней части окна. Упростить работу пользователя можно, добавив строку кода, что позволит переходить к странице `Property` двойным щелчком на `PropertyID`. В таблице результатов создайте обработчик свойства `When-Mouse-DoubleClick` для `PropertyID`.

```
GO_ITEM('PROPERTIES.PROPERTYID');
```

Проверьте работу формы. Просматривая страницу `Property`, заметьте, что на ней содержатся данные для `OwnerID`. Желательно, чтобы пользователи могли также получать дополнительную информацию о владельцах. Используя холст с вкладками, можно достаточно просто реализовать эту возможность, не перегружая окно информацией. Необходимые для этого действия похожи на те, которые предпринимались для свойств.

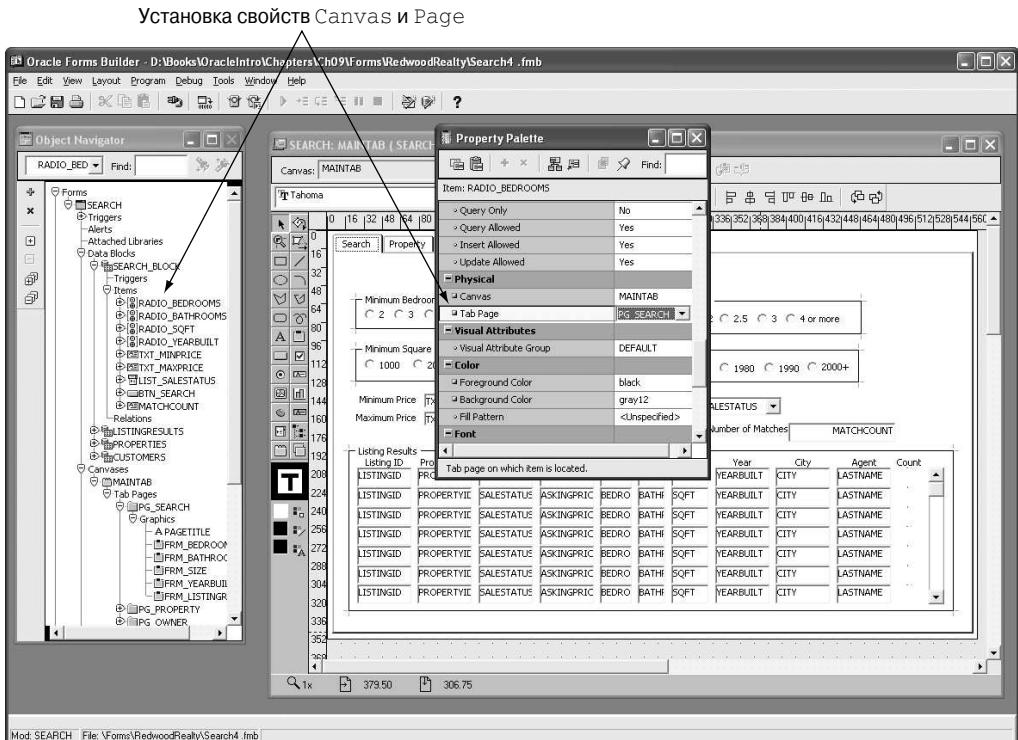
- Добавьте новую страницу к холсту `MainTab`. Выберите в объектном навигаторе пункт `PG_PROPERTY` в разделе `MainTab/Tab Pages`. Щелкните на кнопке `Create`, чтобы создать новую страницу. Установите ее свойства: `Name=PG_OWNER`, `label=Owner`.
- С помощью мастера блока данных добавьте новый блок, выбрав все столбцы из таблицы `Customers`. Свяжите его с блоком данных `Properties`.

```
Customers.CustomerID= Properties.OwnerID.
```

- Запустите мастер компоновки и убедитесь, что вы выбрали страницу `PG_OWNER`.
- Форма уже работоспособна, но вы можете модернизировать ее, установив обработчик двойного щелчка мышью для `OwnerID` на странице `Property`.

```
GO_ITEM('CUSTOMERS.CUSTOMERID');
```

На рис. 9.23 показана готовая форма с тремя вкладками. Проверьте ее работу, выполнив поиск и выбрав свойство. Дважды щелкните на `PropertyID` либо щелкните на вкладке `Property`, чтобы получить подробную информацию о свойстве. Несложно



**Рис. 9.22.** Установка свойств Canvas и Page для элементов данных

также получить данные о владельце. Используя данную форму, агент может быстро выполнить поиск объектов с требуемыми атрибутами, ознакомиться с детальной информацией о свойстве, при необходимости вывести страницу на печать и быстро узнать имя владельца. Очевидно, что необходимо консультироваться с агентами и выяснить, все ли требуемые данные содержатся в форме.

Создание форм с холстом в виде вкладок требует не больше времени и усилий, чем создание форм в виде отдельных областей. Необходимо помнить, что если вы хотите использовать холст с вкладками, желательно предпринимать действия для его создания с самого начала, чтобы избежать необходимости переноса элементов. Реализовывать ли обработчик двойного щелчка мышью — решать вам и вашим пользователям. Ярлыки вкладок уже сами по себе удобны в работе.

## Выбор стиля для формы

Мы рассмотрели три способа отображения связанных данных в составе формы: использование холстов в виде отдельных областей, холстов в виде слоев и холстов с вкладками. Строго говоря, существует и четвертый способ: можно создать формы,

The screenshot shows a web browser window for Oracle Application Server Forms Services. The title bar reads "Oracle Application Server Forms Services - Microsoft Internet Explorer". The address bar shows the URL: "http://postlt:8889/forms90/f90servlet?form=D:\Books\OracleIntro\Chapters\Ch09\Forms\RedwoodRealty\Search". The main content area displays a search interface titled "Listing Search" for "Redwood Realty". The interface includes several search criteria: "Minimum Bedrooms" (radio buttons for 2, 3, 4, or 5 or more), "Minimum Bathrooms" (radio buttons for 1, 1.5, 2, 2.5, 3, or 4 or more), "Minimum Square Feet" (radio buttons for 1000, 2000, 3000, or 4000 or more), "Maximum Age" (radio buttons for 1900, 1970, 1980, 1990, or 2000+), "Minimum Price" (text input field "0"), "Maximum Price" (text input field "1000000"), "Sale Status" (dropdown menu "For Sale"), and a "Search" button. Below the search form is a table titled "Listing Results" with columns: Listing ID, Property ID, Sale Status, Price, Bed, Bath, SQFT, Year, City, and Agent. The table contains 10 rows of data. At the bottom of the page, there is a navigation bar with "Record: 2/14" and a "Trusted sites" link.

| Listing ID | Property ID | Sale Status | Price  | Bed | Bath | SQFT | Year | City          | Agent    |
|------------|-------------|-------------|--------|-----|------|------|------|---------------|----------|
| 15254      | 1618        | For Sale    | 156000 | 6   | 3    | 2727 | 1990 | Mckinleyville | Robinson |
| 15151      | 1739        | For Sale    | 170000 | 4   | 3    | 2688 | 1992 | Arcata        | Romero   |
| 15965      | 1768        | For Sale    | 176000 | 5   | 3    | 2683 | 1996 | Arcata        | Selby    |
| 15608      | 1772        | For Sale    | 177000 | 3   | 3    | 2202 | 1992 | Eureka        | Herring  |
| 15067      | 1783        | For Sale    | 179900 | 4   | 3    | 2042 | 1996 | Fortuna       | Weber    |
| 16144      | 1784        | For Sale    | 179950 | 3   | 3    | 2204 | 1993 | Fortuna       | Okindo   |
| 15408      | 1791        | For Sale    | 180000 | 5   | 4    | 2845 | 1993 | Eureka        | Voss     |

Рис. 9.23. Холст, содержащий связи с Property и Owner

полностью независимые друг от друга, и переходить к следующей форме посредством команды OPEN\_FORM. Этот способ более сложен, так как он требует применения глобальных переменных. К тому же при этом нет средств объединения, позволяющих связывать данные. Данный подход будет более подробно описан в главе, посвященной созданию приложений.

Разрабатывая форму, необходимо решить, какой из способов лучше использовать. Иногда при выборе приходится учитывать вопросы производительности и организации деятельности рабочей группы. Однако главным фактором все же остается удобство работы пользователя. Каждый из трех описанных выше способов имеет свои преимущества. Они отличаются друг от друга в основном особенностями навигации между различными холстами. В ряде случаев предпочтительными оказываются

вкладки, так как пользователь всегда видит возможные варианты выбора. В некоторых случаях, например, тогда, когда надо сравнивать данные с содержимым исходной формы, удобнее просматривать новые данные на холсте, выполненном в виде слоя. Возможны случаи, когда целесообразно будет скрыть все холсты. Тогда придется использовать отдельный холст для группы элементов. Данный подход позволяет пользователю в каждый момент сосредоточить внимание только на одном действии. При необходимости можно контролировать порядок, в котором страницы будут представлены пользователю.

Заметьте, что в большинстве вновь созданных форм таблица `Property` (а впоследствии `Owner`) активизирована. Пользователь может изменить содержащиеся в них данные. Не исключено, что вам потребуется ограничить возможности пользователя по модификации информации посредством форм. Для этого следует лишь не указывать в свойствах блока данных `UPDATE`, `INSERT` и `DELETE`.

При создании форм с несколькими холстами вы столкнетесь с рядом ограничений. Например, вы не сможете поместить холст в виде слоя непосредственно на холст с таблицей. Оба они должны быть созданы на базе обычного холста с содержимым. В некоторых случаях можно обойти это ограничение, сначала создав базовый холст, а затем разместив холст с вкладками поверх него. Затем можно поместить холст в виде слоя на основной холст с содержимым. Для того чтобы создать эффективный пользовательский интерфейс, надо поработать с компоновкой страницы. В большинстве случаев желательно выбрать лишь один из способов работы с холстами и не применять при решении задачи несколько способов одновременно.

Таким образом, мы рассмотрели использование трех различных подходов при работе над одной и той же формой. Эту форму можно использовать как демонстрационный пример, на примере которого удобно объяснить различия между разными типами интерфейсов. Данную форму также можно применять для проверки производительности, обеспечиваемой различными решениями (это важно в случае обмена данными по сети с низким быстродействием).

---

## Создание Web-форм с помощью JDeveloper

---

Данный раздел, наверное, заинтересует не всех программистов, поскольку он посвящен совершенно иному инструменту разработки. Однако не исключено, что этот инструмент будет широко использоваться наряду с конструктором форм и даже вместо него, поэтому желательно представлять себе его возможности и основные особенности. Как вы помните, формы, созданные с помощью конструктора, требуют, чтобы на клиентской машине были установлены Java и JInitiator. По этой причине выполнять доставку таких форм средствами Web не совсем удобно как для поставщиков, так и для потребителей. Oracle предлагает новую технологию, которая может быть использована в традиционных Web-приложениях. Возможно, она даже полностью за-

менит конструктор форм. Новая система предполагает выполнение Java-программ на сервере и генерирует относительно низкоуровневые HTML-страницы, которые могут отображаться практически на любом компьютере. Данная технология интегрирована с Java 2 Enterprise Edition (J2EE) и использует многие мощные средства этой системы. Однако J2EE — большой и сложный продукт, для изучения которого требуется длительное время. Поэтому Oracle предлагает несколько инструментов, интегрированных в рамках одной системы и составляющих набор базовых средств разработки приложений (application development framework — ADF). Система основана на интегрированной среде разработки JDeveloper, содержащей визуальные средства и инструменты типа “мастер”, которые упрощают создание форм, управляемых базой данных.

Вам необходимо установить в вашей системе JDeveloper. Процесс установки относительно прост, но при этом не используются стандартные инструменты инсталляции Oracle. JDeveloper быстро развивается, поэтому лучше всего скопировать последнюю реализацию с Web-узла OTN. Пакет представляет собой Zip-файл небольшого размера, который можно найти в разделе JDeveloper OTN.

1. Скопируйте с сервера OTN последнюю реализацию JDeveloper.
2. Распакуйте файлы в папку в вашей системе. Запомните расположение файлов.
3. Найдите главный исполняемый файл (*panka/jdev/bin/jdevw.exe*), щелкните на нем правой кнопкой мыши и, используя пункт меню *Send To⇒Desktop*, поместите ярлык на рабочий стол.

## Модификация таблицы и настройка файлов на сервере

Инструменты типа “мастер” в составе JDeveloper позволяют без труда создать базовую форму для отображения данных. Несложно также объединить основную форму с подчиненной. Однако, для того, чтобы продемонстрировать возможности системы, надо создать простую поисковую систему, ориентированную на работу с недвижимостью. Потребители могут использовать ее для просмотра доступных строений. Первый шаг — создание основной формы, в которой будет отображаться базовая информация о списке и некоторые свойства, например, начальная цена, число спален и размер дома. Если есть фотоснимок дома, неплохо отображать также и его. Как вы помните из предыдущей главы, фотоснимки можно хранить в базе данных. К сожалению, соответствующий инструмент несовместим с JDeveloper. Вместо этого мы пойдем по пути, который выбирают большинство разработчиков Web-приложений, и поместим изображения в файлы на сервере. Имя файла можно хранить в базе и организовать страницу для воспроизведения изображения каждого дома. В первую очередь надо установить на сервере базу данных.

1. На компьютере, на котором выполняется JDeveloper, создайте папку для хранения изображений (C:\Temp).

2. Скопируйте в эту папку файл HousePhoto.jpg, содержащий пример графических данных, или воспользуйтесь своими картинками.
3. С помощью SQL\*Plus добавьте в таблицу Listings столбец, в котором будет храниться информация о местонахождении файла.

```
ALTER TABLE Listings  
ADD PhotoFile NVARCHAR2(250);
```

4. Используя SQL\*Plus, свяжите фотоснимок с одним из списков характеристик; при этом следите за тем, чтобы папка была указана корректно.

```
UPDATE Listings  
SET PhotoFile = 'C:\Temp\HousePhoto.jpg'  
WHERE ListingID=14979;
```

Теперь можно создать простую Web-форму, отображающую основной список и информацию о свойствах. Для того чтобы ускорить работу, в данном примере будем использовать лишь несколько столбцов. Запустите JDeveloper, щелкнув на созданном ранее ярлыке. Создание Web-страницы включает три основных этапа: создание нового соединения с данными и новой рабочей области Application Workspace; создание модели данных для выбора используемой таблицы; создание Web-формы, организующей отображаемые объекты.

Соединение нужно для того, чтобы JDeveloper мог найти базу данных. Его установку надо выполнить лишь единожды. То же самое соединение можно использовать для других страниц и даже в других проектах. После запуска JDeveloper вы увидите на экране несколько окон. Левое верхнее окно принадлежит навигатору приложений (Application Navigator). В этом окне содержится список созданных компонентов. Используйте его для выбора страницы или элемента. Центральное окно большего размера вначале не содержит данных, но потом в него будут помещены страницы приложения. Основное окно справа содержит различные палитры, с помощью которых можно выбирать инструменты и элементы для размещения на страницах. В нижней части экрана справа и слева будут появляться различные вспомогательные окна. При желании вы можете изменить взаимное расположение окон, но сейчас лучше оставить их в прежних позициях, так вам будет легче следовать приведенным здесь указаниям. Если вы случайно закроете окно, вы можете вернуть его назад, используя пункт View главного меню.

## Соединение с данными и создание рабочей области

Для установки соединения с данными необходимо знать некоторые подробности о том, как была создана база. В большинстве случаев можно использовать значения по умолчанию, но, чтобы гарантированно получить положительный результат, надо проверить конфигурацию базы данных. Если вы сами инсталлировали базу, достаточно просмотреть записи, сделанные в процессе установки, в противном слу-

чае надо запустить Oracle Net Manager и просмотреть с помощью этой программы значения некоторых параметров.

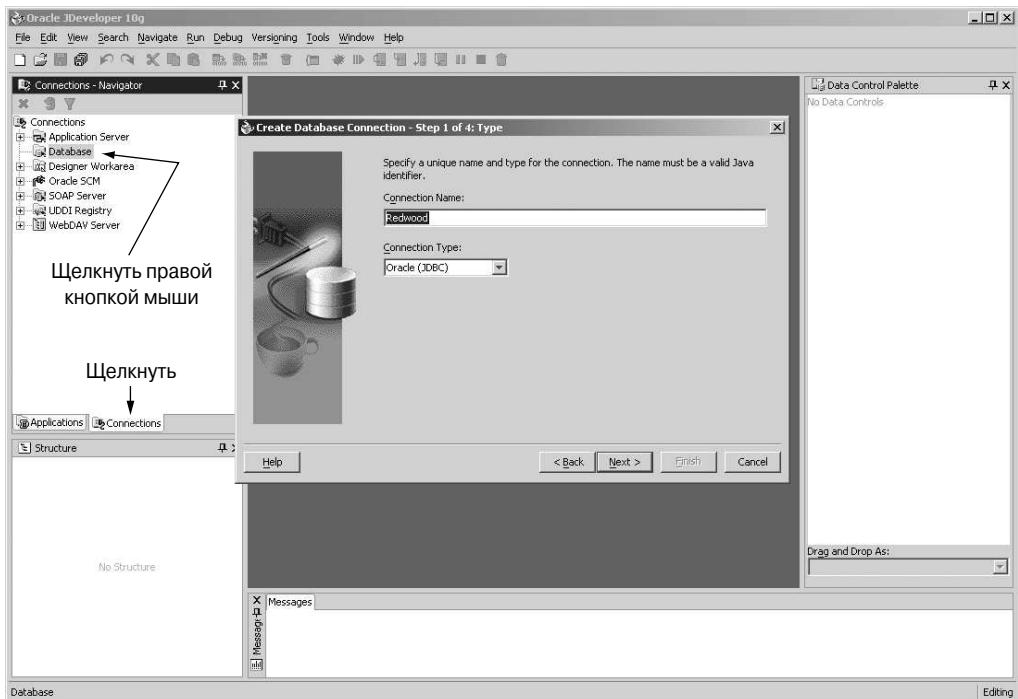
1. Работая с JDeveloper, щелкните на вкладке Connections в Application Navigator.
2. Щелкните правой кнопкой мыши на пункте Database и выберите New Database Connection.
3. Если работа мастера начинается с окна, отображающего только приветствие, щелкните в этом окне на кнопке Next.
4. На рис. 9.24 показан внешний вид окна JDeveloper на данном этапе работы. Введите в поле Connection Name значение Redwood.
5. Щелкните на кнопке Next и введите имя пользователя и пароль для работы с приложением.
6. Установите флажок Deploy Password и щелкните на кнопке Next.

В процессе работы вам надо следовать указаниям мастера, но в некоторых случаях, чтобы принять обоснованное решение, приходится обращаться к справочной системе. Например, если вы зададите доставку пароля с приложением, каждый, кто запустит форму, получит доступ к приложению и предоставляемым им данным. Если у вас нет возможности назначить пароли потенциальным потребителям, вам надо воспользоваться флажком опции, о котором шла речь выше. Возможно также создание специальной учетной записи с ограниченными правами.

1. В окне, соответствующем соединению с базой, укажите имя узла.
2. При желании вы можете изменить SID и даже номер порта, но желательно на первом этапе работать со значениями по умолчанию.
3. Щелкните на кнопке Next, а затем на кнопке Test Connection.
4. Если проверка закончилась успешно, щелкните на кнопке Next, а затем на кнопке Finish, завершив тем самым работу с мастером. В противном случае выясните корректные значения.

Если соединение установлено, вам вряд ли понадобится изменять его. Когда вы начнете новый проект, надо создать рабочую область для поддержки всех файлов. По сути, рабочая область — это папка на вашем компьютере. Обычно она находится в папке `jdev\mywork`, которая расположена в том каталоге, в котором был инсталлирован JDeveloper.

5. Щелкните на вкладке Applications в Application Navigator.
6. Щелкните правой кнопкой мыши на узле Applications и выберите New Application Workspace.
7. Введите имя рабочей области RedwoodRealty.



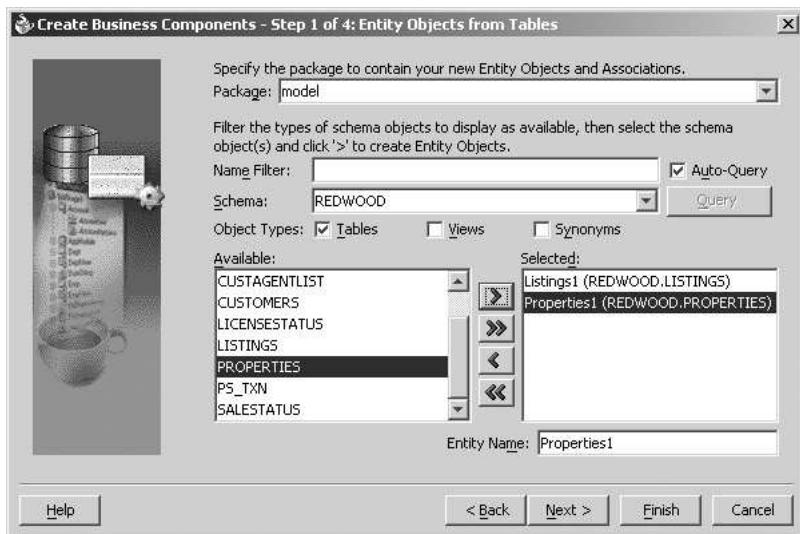
**Рис. 9.24.** Установка нового соединения с данными с помощью JDeveloper

8. Выберите `Web Application [Default]` в качестве значения `Application Template` и щелкните на кнопке `OK`.

Система добавит новую рабочую область, а в окно Application Navigator включит разделы `Model` и `ViewController`. Они будут использованы в дальнейшей работе.

## Создание модели данных бизнес-компонентов

Для поддержки большинства задач, выполняющихся на сервере, Oracle использует набор базовых средств Apache Jakarta. В основу их работы положена архитектура “модель–представление–контроллер” (`MVC – model-view-controller`). Этот подход соответствует общим принципам работы с базами данных. Вначале формируется модель данных. Эта модель создается посредством Java-кода, выполняющегося на сервере, но поскольку таблицы в базе данных уже определены, мастер лишь перенесет эти определения в проект. Наличие представления позволяет создавать страницы с отображаемой информацией независимо от модели данных. Контроллер также отделен от остальных двух компонентов, но его поддержка практически полностью реализована в наборе базовых средств. В основном вам надо выбрать необходимые элементы



**Рис. 9.25.** Выбор таблицы

данных и создать страницы для их отображения.

1. Щелкните правой кнопкой мыши на узле Model в окне навигатора приложений и выберите пункт New.
2. В диалоговом окне разверните узел Business Tier и выберите элемент Business Components. В окне Items выберите Business Components from Tables. Щелкните на кнопке OK для перехода к следующему окну.
3. Убедитесь в том, что выбрано соединение RedwoodRealty и щелкните на кнопке OK. Если на экране появится окно с приветственным сообщением, щелкните на кнопке Next.

Следующие три окна похожи друг на друга, но отличаются своим назначением. Первое из них показано на рис. 9.25. Обращайте внимание на заголовок каждого окна, чтобы представлять себе его назначение.

1. При необходимости выберите схему базы данных, затем выберите таблицу Listings и щелкните на кнопке >, чтобы переместить таблицу в окно Selected, расположенное справа. Сделайте то же для таблицы Properties и щелкните на кнопке Next.
2. Приложение не модифицирует данные, поэтому вам не нужны обновляемые таблицы. Щелкните на кнопке Next.
3. Снова выберите и переместите в правое окно таблицы Listings и Properties,

чтобы использовать их в качестве источников данных, предназначенных только для чтения. Щелкните на кнопке **Next**.

4. В качестве имени для нового модуля приложения задайте **RedwoodRealtyAM** и щелкните на кнопке **Next**. В последнем окне мастера щелкните на кнопке **Finish**.

Для создания объектов, необходимых для внутреннего применения, мастер использует определение таблицы. Объект примитива определяет базовый набор данных. При наличии нескольких примитивов они объединяются посредством связей, основой для которых служат внешние ключи, определенные в таблицах. Еще в большей степени система зависит от объектов представления, которые выполняют те же функции, что и представления базы данных, и могут объединять данные из нескольких примитивов. И примитивы, и представления имеют атрибуты, или столбцы. При необходимости вы можете устанавливать значения по умолчанию, задавать правила проверки и формировать атрибуты на базе вычислений как для примитива, так и для представления. Для этого достаточно дважды щелкнуть на требуемом пункте в окне навигатора приложений и с помощью редактора свойств установить нужные значения. Но пока мы не будем уделять внимание этим подробностям.

Несмотря на то что система создает связь между примитивами **Listings** и **Properties**, она пока не позволяет выбирать атрибуты обоих примитивов и помещать в форму. Для рассматриваемого здесь примера нам понадобятся столбцы **Bedrooms**, **Bathrooms** и **SQFT** таблицы **Properties**. Модифицируйте **ListingsView** для решения этой задачи.

1. В окне навигатора дважды щелкните на пункте **ListingsView**.
2. В дереве выберите узел **Entity Objects**, затем в среднем списке выделите примитив **Listings**. Щелкните на кнопке **>**, чтобы переместить его в окно **Selected**. Повторите те же действия для примитива **Properties**.
3. Разверните раздел **Attributes** и выберите атрибут **Housephoto**. Щелкните на вкладке **Control Hints**. Измените значение **Display Hint** на **Hide**.
4. Выберите узел **Query**, чтобы просмотреть SQL-выражение для данного представления. Отредактируйте представление и удалите столбец **Housephoto**.
5. Добавьте четыре (или более) столбца из таблицы **Properties** и условия объединения таблиц **Listings** и **Properties**. Обычно мастер использует старые синтаксические правила для объединения, но при желании вы можете задать **INNER JOIN**.

```
SELECT Listings.LISTINGID,  
       Listings.PROPERTYID,  
       Listings.LISTINGAGENTID,  
       Listings.SALESTATUSID,  
       Listings.BEGINLISTDATE,  
       Listings.ENDLISTDATE,  
       Listings.ASKINGPRICE,
```

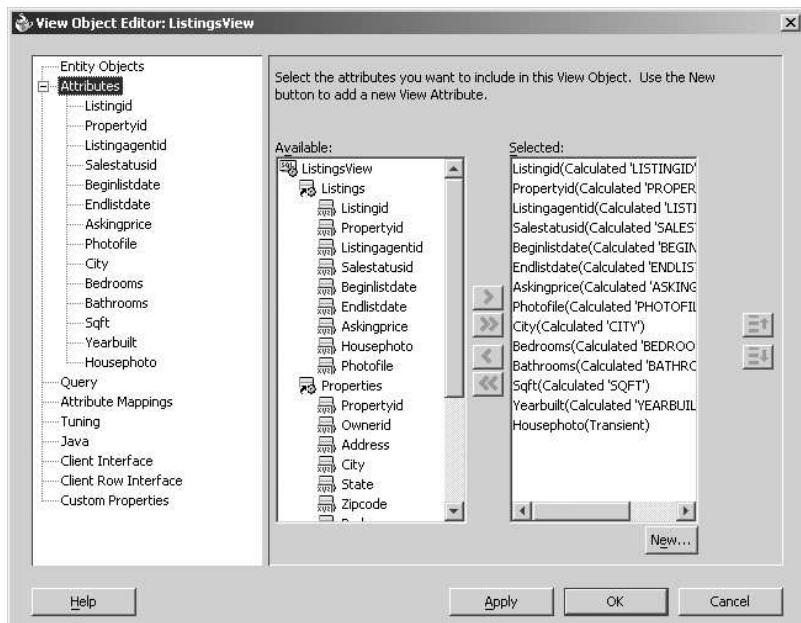


Рис. 9.26. Выбор атрибутов из таблицы Properties

```

Listings.PHOTOFI,
Properties.BEDROOMS,
Properties.BATHROOMS,
Properties.SQFT,
Properties.YEARBUILT
FROM LISTINGS Listings, PROPERTIES Properties
WHERE Listings.PROPERTYID = Properties.PROPERTYID

```

6. В строке Order By введите LISTINGID для сортировки результатов. Активизируйте Test, чтобы проверить синтаксис и устранить проблемы.
7. Щелкните на узле Attributes. Выберите в разделе Properties, отображаемом в средней части окна, пункты Bedrooms, Bathrooms, SQFT и YearBuilt. Переместите их в окно Selected, расположенное справа. Щелкните на кнопке OK, чтобы закрыть редактор.

На рис. 9.26 показано окно Attribute, в котором вы выбирали атрибуты из таблицы Properties, чтобы сделать их доступными приложению. Если вы планируете дальнейшее развитие продукта, вы можете выбрать все атрибуты на случай, если они потребуются вам впоследствии. Напомним, что для объединения данных из различных примитивов надо выполнить следующие действия: выбрать Entity Objects и добавить примитивы к списку; выбрать Query, написать выражение объединения и добавить новые столбцы к выражению SELECT; выбрать Attributes и добавить требуемые атрибуты к выбранному списку.

## Создание документа JSP для отображения данных

На следующем этапе работы надо создать Web-страницу для отображения данных из созданного вами представления. Существует инструмент типа “мастер”, который создает HTML-страницу и предоставляет средства для переноса в нее выбранных элементов. Если вы знакомы с языком HTML, можете изменить компоновку и общий стиль отображения данных, включив в состав страницы дополнительные дескрипторы. Создаваемый здесь вариант страницы достаточно прост.

1. Щелкните правой кнопкой мыши на проекте `ViewController` и выберите `New`.
2. В диалоговом окне разверните `Web Tier` и выберите `JavaServer Pages (JSP)`, также выберите `JSP Page` в окне `Items`. Щелкните на кнопке `OK`.
3. Введите в качестве имени новой страницы `ListingProperty.jsp` и щелкните на кнопке `OK`.
4. Щелкните в верхней части новой страницы и введите заголовок `Property Listing`. Выделите этот элемент и в раскрывающемся списке, посредством которого задаются стили, выберите пункт `Heading 1`.
5. Выберите вкладку `Components` в нижней части палитры, расположенной в правой части окна. В верхнем раскрывающемся списке выберите пункт `CSS`. Перетащите пункт `JDeveloper` и поместите его в форму.

Обратите внимание на изменение стиля заголовка при добавлении `CSS`. Таблицы стилей можно изменить впоследствии для того, чтобы настроить нужным образом внешний вид страницы. Теперь добавим простую форму и некоторые элементы данных.

6. Щелкните в форме, чтобы поместить курсор под заголовком. В раскрывающемся списке выберите пункт `HTML` и щелкните на элементе `Form`.
7. Щелкните в новой области формы и введите текстовые метки для данных: `Listing ID`, `Asking Price`, `Bedrooms`, `Bathrooms`, `SQFT` и `Year Built`. В нижней части формы предусмотрите пустое место для кнопок.
8. В правой палитре выберите вкладку `Data Controls`. Разверните поддерево `RedwoodRealtyAMDataControl`, затем выберите раздел `ListingsView1`. Перетащите пункт `Listingid` в форму и разместите рядом с соответствующей меткой. Повторите ту же процедуру для других элементов данных. Перетащите в форму элемент `Photofile`.
9. Разверните узел `Operations`, который находится в разделе `ListingsView1`. Перетащите кнопку `Previous` в нижнюю часть формы. Сделайте то же самое, чтобы разместить рядом кнопку `Next`.

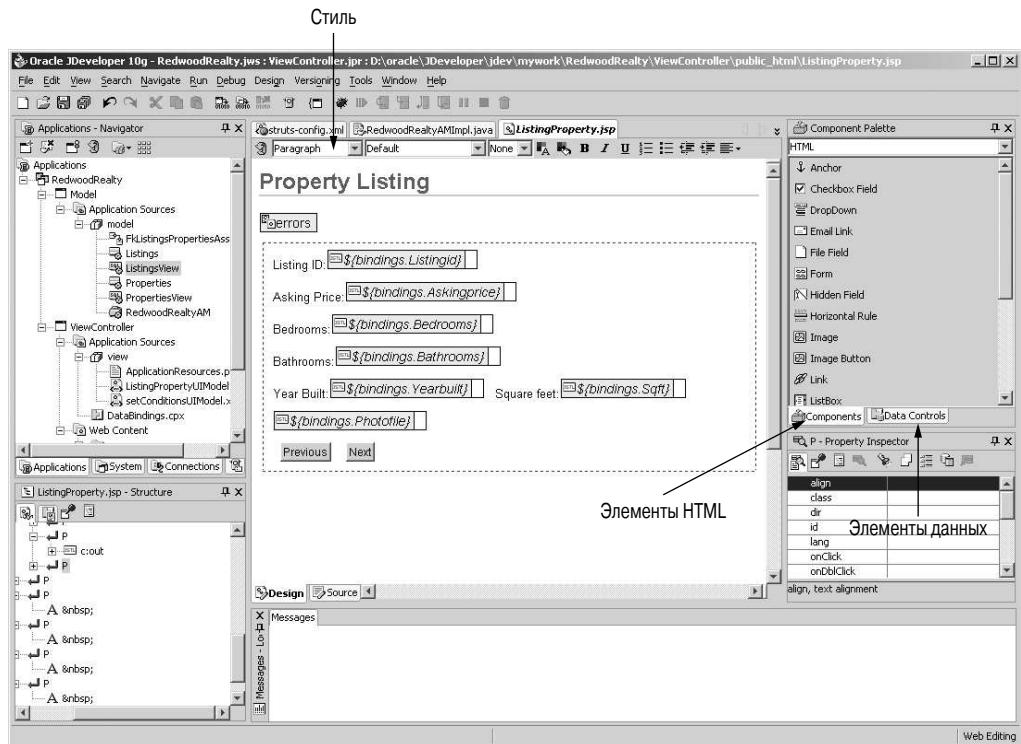


Рис. 9.27. Размещение элементов в форме

10. Переместите дескриптор, предназначенный для отображения ошибок, в верхнюю часть формы и разместите его ниже заголовка. Убедитесь, что стиль элемента Paragraph, а не Heading 1.

На рис. 9.27 показан проект формы. При желании вы можете изменить компоновку элементов.

11. Чтобы проверить форму, щелкните правой кнопкой мыши на узле ViewController и выберите пункт Run. Щелкнув несколько раз на кнопке Next, вы увидите новые записи.

Обратите внимание, что если файл с изображением существует, то в текущем варианте формы лишь отображается его имя. Пользователю же надо видеть само изображение. Включите дескриптор изображения, в котором имя файла будет указано в качестве источника данных. Кроме того, если файл пуст, надо пропустить изображение. Для этого можно использовать дескриптор `<if>`, определенный в JSTL. С помощью HTML-дескриптора `<img>` изображение будет выводиться на экран.



**Рис. 9.28.** Внешний вид формы

1. Закройте окно браузера. Щелкните на вкладке Source в нижней части той области, в которой вы компоновали элементы. Найдите дескриптор, содержащий Photofilename. Замените эту строку следующим образом:

```
<c:if test="${bindings.Photofilename != ''}">
<img src='c:out value="${bindings.Photofilename}" />' height="100px">
</c:if>
```

2. Сохраните форму и запустите ее. На рис. 9.28 показана страница, разработанная здесь в качестве примера. Если вы получите сообщение об ошибке, еще раз проверьте последние три строки кода. Обратите внимание на то, что ссылка на Photofilename помещена не в круглые скобки, а в фигурные.

Просмотрите исходный код формы, и вы увидите, что это обычный HTML-документ. Таким образом, с данной формой может работать любой потребитель. Подобный подход позволяет создавать форму практически любого типа. Несложно также объединить основную форму с подчиненной.

## Добавление поисковой формы

Для того чтобы со страницей было удобнее работать, надо предоставить потребителям возможность задавать критерии поиска, например, диапазон цен или число спальных комнат. В данный момент у нас нет возможности подробно прорабатывать страницу поиска, поэтому мы ограничимся простой формой, которая демонстрирует возможности системы Oracle.

Для реализации поиска надо решить четыре основные задачи. Самый важный этап работы — модификация запроса, чтобы в выражении `Where` использовались значения, заданные пользователем. Затем надо создать небольшую форму, позволяющую пользователю сделать выбор. После этого надо создать канал взаимодействия между формой с параметрами и формой с результатами и связать отклик пользователя с каналом взаимодействия.

Изменение запроса — самый ответственный этап. Oracle позволяет сделать это различными способами. Проще всего, наверное, создать отдельную процедуру и добавить к запросу выражение `WHERE`, используя строковые переменные. Однако этот подход связан с определенным риском, так как пользователь получает возможность вводить произвольный текст в поле редактирования. В результате ваша система может подвергнуться атаке путем включения SQL-выражения. Избежать ее можно, запретив пользователю задавать текст в качестве значения. Предпочтительнее использовать раскрывающиеся списки или переключатели. Если вы допустите ввод текста, вам обязательно понадобится программа, анализирующая текст и выявляющая специальные символы (например, удаляющая комментарии) и ограничивающая общую длину данных. Для того чтобы данный пример не был слишком сложным, мы разрешим ввод текстовых значений для минимальной и максимальной цены. Однако не следует писать подобный код в реальных проектах, иначе атака путем включения SQL-выражений станет если не неизбежной, то, по крайней мере, вполне возможной.

1. В Application Navigator выберите узел `RedwoodRealtyAM`, затем разверните раздел `Structure` и дважды щелкните на файле `RedwoodRealtyAMImpl.java`, чтобы начать его редактирование.
2. Перейдите в конец файла и непосредственно перед последней фигурной скобкой введите следующий код:

```
public void setConditions (String sLow, String sHigh,
    String nBedrooms, String nBathrooms)
{
    oracle.jbo.ViewObject LVO = findViewObject("ListingsView1");
    String sWhereClause = "SALESTATUSID=101"
        + " AND (BATHROOMS >= " + nBathrooms + ")"
        + " AND (BEDROOMS >= " + nBedrooms + ")";

    sWhereClause += " AND (ASKINGPRICE >= " + sLow + ")"
        + " AND (ASKINGPRICE < " + sHigh + ")";
```

```
LVO.setWhereClause(sWhereClause);
LVO.executeQuery();
}
```

3. Дважды щелкните на узле RedwoodRealtyAM в окне навигатора, чтобы открыть диалоговое окно редактирования. Выберите узел Client Interface. Затем выберите пункт setConditions и щелкните на клавише >, чтобы переместить его в окно Selected. Щелкните на кнопке OK.

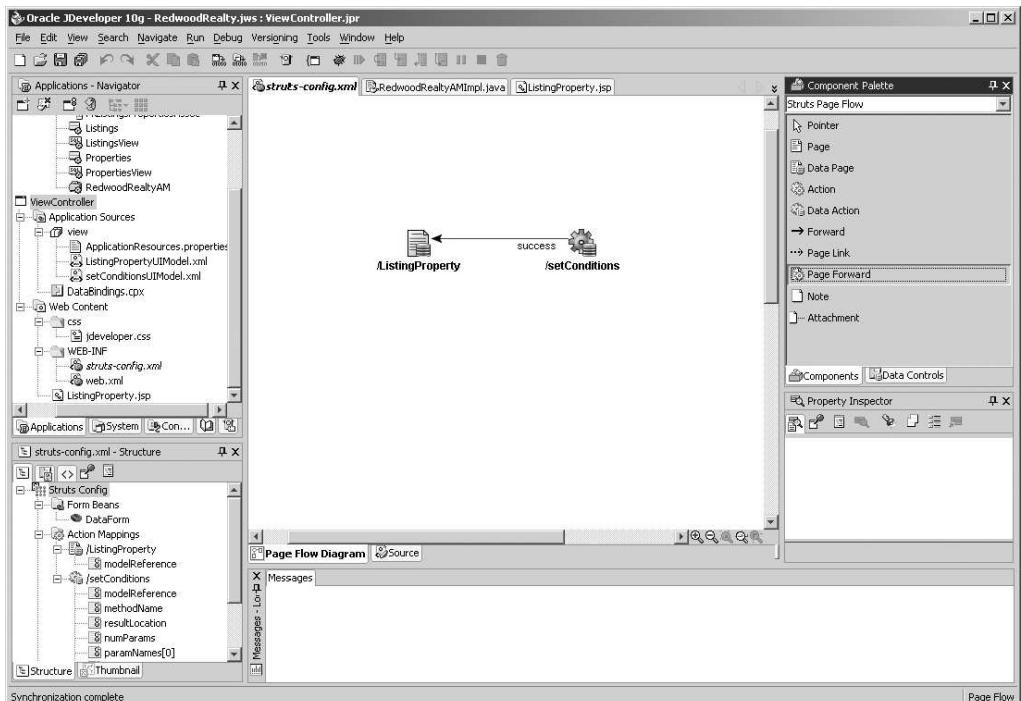
После того как вы создали и зарегистрировали функцию, ее надо связать с приложением. JDeveloper предоставляет редактор, обеспечивающий визуальные средства объединения страниц с данными и действий над данными. Контроллер Struts отвечает за координацию всех действий; определить последовательность страниц можно, используя Struts Page Flow Diagram.

1. Щелкните правой кнопкой мыши на узле ViewController и выберите Open Struts Page Flow Diagram. Перетащите Data Action из палитры Component на страницу диаграммы и разместите ее около существующей пиктограммы /ListingProperty. Щелкните на имени пиктограммы или дважды щелкните на ней самой и переименуйте ее в /setConditions.
2. В палитре, расположенной справа, щелкните на вкладке Data Controls. Разверните узел RedwoodRealtyAMDataControl, а затем узел Operations. Перетащите элемент setConditions на диаграмму и опустите его на новый элемент Data Action.
3. Щелкните на вкладке Components, изменив тем самым вид палитры. Щелкните на элементе Forward. Щелкните на пиктограмме /setConditions, а затем на /ListingProperty.

На рис. 9.29 показано, как действие /setConditions будет взаимодействовать со страницей /ListingProperty. Когда выражение WHERE применяется в новой функции, запрос выполняется и результаты передаются странице ListingProperty для отображения.

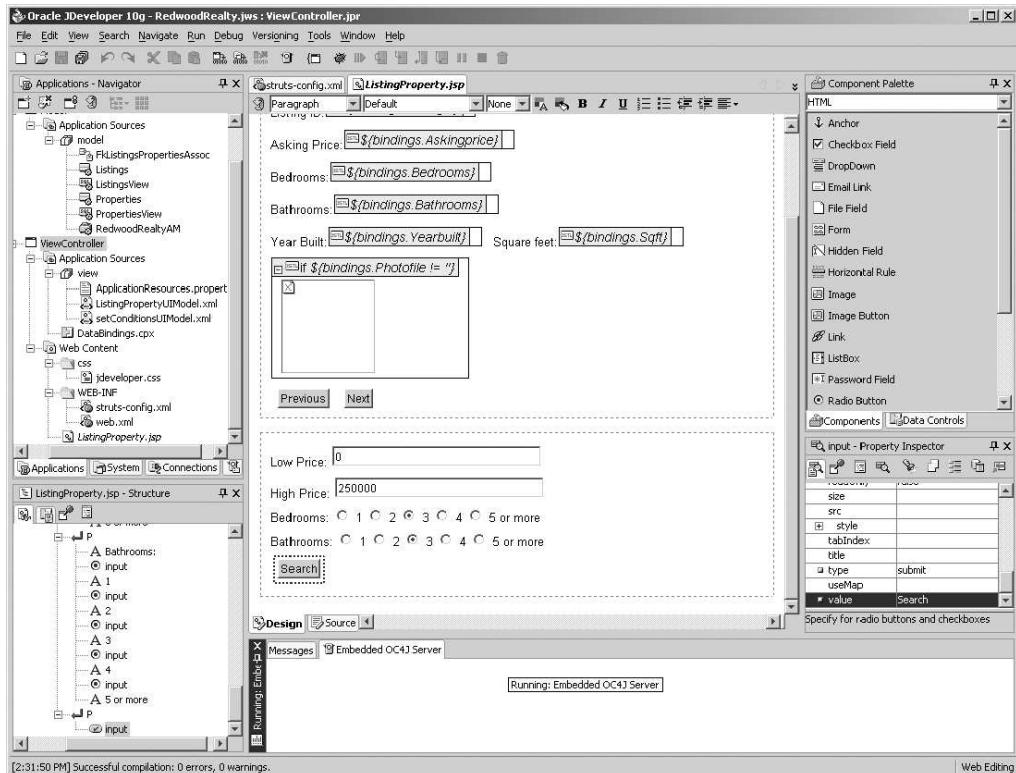
Конечно же, вам надо каким-то образом собрать данные от пользователя и передать их функции setConditions. Форму запроса можно сформировать на отдельной странице, но для пользователя будет несложно проще, если вы разместите элементы для ввода данных в составе исходной формы.

1. На диаграмме struts дважды щелкните на пиктограмме /ListingProperty, чтобы открыть форму.
2. Щелкните на странице, чтобы поместить курсор под существующей формой. Перейдите на вкладку Components. Выберите в раскрывающемся списке Struts HTML (не путать с пунктом HTML!).



**Рис. 9.29.** Struts Page Flow Diagram

3. Перетащите форму в нижнюю часть JSP-страницы и выберите `setConditions.do` в качестве действия при отображении диалогового окна. Щелкните на кнопке OK.
4. В новой форме введите метки для Low Price, High Price, Bedrooms и Bathrooms.
5. С помощью раскрывающегося списка переключитесь на работу с обычным HTML-кодом. Выберите элемент, реализующий поле редактирования, и перетащите его в форму, разместив рядом с меткой Low Price. С помощью Property Inspector установите для него имя `LowPrice` и значение, равное 0. Повторите процесс для `HighPrice` и значения 250000.
6. Перетащите элемент Radio Button, поместите его рядом с меткой Bedrooms и поставьте справа от него цифру 1. Задайте для данного элемента имя `nBeds` и значение 1. Перетащите остальные кнопки переключателей так, чтобы у вас сформировалась последовательность 1, 2, 3, 4 и 5 or more. Убедитесь, что все они имеют одно и то же имя и, следовательно, выбор одного из них автоматически означает отмену выбора остальных. Для третьей кнопки установите свойство Checked равным `true`.



**Рис. 9.30.** Поисковая форма

7. Повторите для Bathrooms действия, выполненные на предыдущем этапе; на этот раз используйте имя nBaths. Перетащите в форму кнопку Submit и задайте для нее значение Search свойства Value.
8. В центральной области перейдите на вкладку Source и найдите первый элемент <form>. Он должен выполнять действия, поэтому измените код следующим образом:

```
<form action="ListingProperty.do">
```

На рис. 9.30 показана структура новой формы. Вы можете расположить элементы по-другому и даже заменить поля редактирования раскрывающимися списками. Как уже говорилось ранее, позволяя пользователям вводить данные непосредственно в полях редактирования, вы снижаете уровень безопасности системы. Вполне вероятно, что они зададут недопустимые значения. Но для первоначальной проверки формы текстовые окна вполне подходят. Включить же раскрывающийся список несложно. Надо лишь перетащить соответствующий элемент в форму, дважды щелкнуть на нем и установить требуемые свойства.

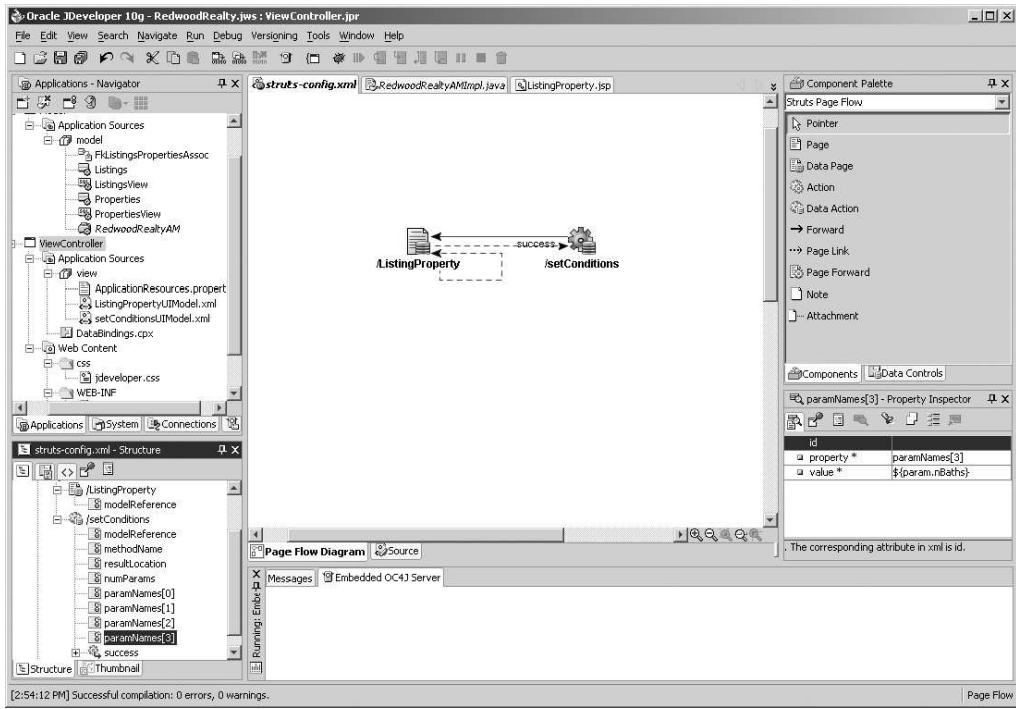
Следующий важный шаг — это соединение запроса значений формы с процедурой действия над данными, которую вы написали.

1. В главном окне вернитесь к вкладке, представляющей последовательность страниц.
2. В окне **Structure**, расположенном в нижнем левом краю экрана, разверните узел **Action Mappings**, а затем разверните пункт **/setConditions**. Щелкните на пункте, соответствующем первому параметру: `paramNames[0]`.
3. В окне инспектора свойств (оно расположено в правом нижнем углу) измените значение пункта `value` на  `${param.LowPrice}`. Помните, что надо использовать не круглые, а фигурные скобки.
4. Установите второй параметр равным  `${param.HighPrice}`, третий —  `${param.nBeds}` и четвертый —  `${param.nBaths}`.
5. Щелкните правой кнопкой мыши в центральном окне и выберите **Diagram⇒Refresh Diagram from All Pages**.

Как показано на рис. 9.31, диаграмма демонстрирует, что форма запроса на странице `/ListingProperty` передает данные (штриховая линия) функции `/setConditions`. Если запрос обрабатывается успешно, результаты передаются назад странице `/ListingProperty` и отображаются. Если пользователь щелкает на кнопке **Next** или **Previous**, страница `ListingProperty` вызывает сама себя и отображает следующий элемент. Теперь можно запустить приложение и проверить форму.

1. Щелкните правой кнопкой мыши на пиктограмме `ListingProperty` и выберите пункт `Run`.
2. Введите некоторые значения в поисковой форме, например цены 100000 и 130000, и щелкните на кнопке `Query`.
3. Щелкните на кнопке `Next`, чтобы прокрутить список и убедиться в том, что все свойства удовлетворяют критериям поиска.

На рис. 9.32 показана форма с данными. Перед тем как предоставлять форму пользователям, над ней надо дополнительного поработать. Попытайтесь ввести цену, содержащую запятую или знак доллара, и посмотрите, что произойдет. Теперь вы видите одну из причин, по которой раскрывающийся список предпочтительнее поля редактирования. Вы можете также организовать выбор по площади или названию города. Возможно, имеет смысл отформатировать выходные данные, чтобы улучшить внешний вид страницы и добавить контактную информацию потребителя. Но это детали. Главное на данный момент то, что вы знаете, как сформировать страницу.



**Рис. 9.31.** Обмен данными с JSP-документом

## Резюме

Мастера форм — удобные инструменты, но в некоторых случаях необходимо понимать структуру форм и иметь возможность специально настраивать их. Основная цель, преследуемая при создании формы, — упростить работу пользователя при выполнении стоящих перед ним задач. Для этого приходится реализовывать специальные возможности формы, создавать обработчики событий, использовать несколько холстов и т.д. Работая с формами, надо помнить следующее. Во-первых, холст содержит элементы, отображаемые для пользователя, во-вторых, блок данных поддерживает взаимодействие форм и базы данных, и, в-третьих, разработчик может написать код, выполняющий требуемые действия в ответ на некоторое событие.

Если вам надо создать элементы формы, которые непосредственно не связаны с базой данных, проще всего сделать это вручную, сформировав холст и блок данных. Вы также можете вручную создавать блоки данных, предназначенные для извлечения и хранения информации, но мастера позволят решить эту задачу более эффективно.

Рис. 9.32. Окончательный вариант формы

Oracle поддерживает большое количество событий, связанных с формами. Полный их набор никогда не используется, но необходимо представлять себе временные зависимости между событиями, чтобы иметь возможность выбрать самое подходящее из них. Наиболее часто используются события Pre-Form, When-Button-Pressed и When-Validate-Item. Другие события также могут пригодиться для отслеживания изменений данных. Ограничения на область видимости и время жизни приводят к тому, что переменные, созданные в одном обработчике, недоступны из других. Для того чтобы обеспечить обмен данными между обработчиками, приходится применять глобальные переменные. Работая над формой, надо следить за тем, чтобы в нее были включены средства обработки ошибок. Ошибки, возникающие в процессе выполнения, перехватываются посредством раздела EXCEPTION. Обработчик формы On-Error позволяет реагировать на ошибки, часто встречающиеся при работе с базой данных. Отладчик помогает выявить источник ошибок. Он позволяет выполнять работу в пошаговом режиме и просматривать значения переменных.

Использовать обработчики можно для решения некоторых часто встречающихся задач, например, для организации перехода к конкретному элементу формы или блоку данных. Дополнительных усилий по написанию кода обычно требуют следующие за-

дачи: генерация последовательных ключевых значений, проверка данных, введенных пользователем, и вычисление промежуточных результатов. Перечисленные задачи часто встречаются при работе над бизнес-приложениями. Решить их можно посредством обработчиков, причем правильно написанный код относительно прост. Умев решить простые задачи, можно эффективно работать над более сложными.

Чтобы упростить работу пользователя, приходится применять различные программные решения. Дополнительные возможности открывает применение нескольких холстов. Существенной особенностью реляционной базы данных является тот факт, что информация распределена по отдельным таблицам. Однако мало кто из пользователей способен мыслить в терминах таблиц. Поэтому в формах приходится объединять связанные между собой данные. В некоторых случаях всю информацию невозможно или нецелесообразно поместить в одно окно. Приходится использовать несколько холстов. Различные типы холстов предоставляют пользователю разные способы навигации между ними. При работе с отдельными холстами, представляющими собой независимые области отображения, необходимо помещать на каждый из них кнопку, позволяющую обращаться к другому холсту. Холсты в виде слоев позволяют размещать их один поверх другого так, что данные на обоих холстах видны одновременно. Холсты с вкладками дают возможность пользователю быстро перемещаться от одного раздела данных к другому по щелчку на ярлыке вкладки. Последовательности действий для создания каждого из трех типов холстов мало отличаются друг от друга. Выбор того или иного типа холста диктуется конкретной задачей, стоящей перед пользователем.

## Основные термины

- Форма alert
- Выражение EXCEPTION
- Отладчик форм
- Фрейм
- Префикс GLOBAL
- Жесткое кодирование
- Перегрузка данными
- Время жизни переменной
- Ошибка в процессе выполнения
- Область видимости переменной
- Последовательность
- Последовательность событий
- Холсты в виде слоев

- Стандартные расширения (встроенные функции)
- Холсты с вкладками
- Область просмотра
- Выражение WHERE

## **Повторение пройденного материала**

### **Истина или ложь?**

1. Формы alert – удобное средство, так как они прерывают последовательность действий пользователя.
2. Отладчик форм используется для поиска синтаксических ошибок в программном коде.
3. Глобальная переменная, созданная обработчиком When-Button-Pressed, доступна из обработчика Pre-Form.
4. Холсты в виде слоев позволяют пользователям сравнивать данные из двух различных таблиц.
5. Холсты с вкладками невозможно использовать совместно с холстами в виде слоев.

### **Заполнить пропущенное**

1. Для того чтобы создать поисковую форму, которая извлекает данные, соответствующие критериям поиска, надо установить в блоке данных свойство \_\_\_\_\_, которое обеспечит связь с элементами формы.
2. Встроенная функция \_\_\_\_\_ может использоваться для отображения различных холстов, передавая фокус полю редактирования на конкретном холсте.
3. Чтобы получить новое значение последовательности, используется ее свойство \_\_\_\_\_.
4. При работе с холстами в виде слоев устанавливаются свойства \_\_\_\_\_, которые определяют, где на новом холсте будут размещены данные.
5. Чтобы проверить корректность данных, введенных пользователем, и гарантировать их соответствие определенным условиям, надо написать код обработчика \_\_\_\_\_.

### **Варианты ответов**

1. Для выявления ошибок, возникающих в процессе выполнения формы, для создания которой не применялся код PL/SQL, используется:
  - a) Выражение EXCEPTION.

- б) Обработчик On-Error.  
в) Выражение DECLARE.  
г) Префикс GLOBAL.  
д) Обработчик Pre-Form.
2. Для анализа значения в поле редактирования, перед тем как пользователь изменил его, используется:  
а) GLOBAL.oldValue в обработчике Pre-Text-Item.  
б) .oldValue в обработчике Post-Text-Item.  
в) Sequence.NEXTVAL в обработчике Pre-Text-Item.  
г) GO\_ITEM('GLOBAL') в обработчике Pre-Text-Item.  
д) Локальная переменная в обработчике Post-Text-Item.
3. Какая из следующих встроенных функций используется для добавления элементов списка?  
а) GO\_ITEM  
б) SHOW\_ALERT  
в) ADD\_MONTHS  
г) ADD\_LIST\_ELEMENT  
д) SHOW\_VIEW
4. При запуске формы события возникают в такой последовательности:  
а) Pre-Form, On-Logon, When-New-Item-Instance, Pre-Record.  
б) When-New-Form-Instance, Pre-Block, When-New-Block-Instance, Post-Logon.  
в) Pre-Logon, Pre-Form, Pre-Block, When-New-Block-Instance.  
г) Pre-Form, Pre-Record, Pre-Text-Item, Pre-Block, Post-Form.  
д) When-New-Item-Instance, When-New-Record-Instance, When-New-Block-Instance.
5. Для поиска каких ошибок отладчик форм подходит больше всего?  
а) Пользователь получает сообщение о том, что он не имеет права изменять данные.  
б) При запуске формы генерируются два последовательных номера вместо одного.  
в) В имени встроенной функции был пропущен знак подчеркивания.  
г) Пользователь сообщает о том, что холсты в виде вкладок отображаются некорректно.  
д) При попытке сохранить данные форма генерирует сообщение об ошибке.

## Упражнения

### 1. Readwood Realty

Менеджеру часто нужен общий список продаж для каждого агента. Вам надо создать форму, подобную поисковой, но более простую. Менеджер хочет иметь возможность выбрать агента, ввести начальную и конечную даты, выбрать состояние объекта (для продажи, в процессе оформления, продан) и видеть список записей, соответствующих этим условиям. Для списка в форме должно отображаться суммарное значение. Основные этапы работы — такие же, как над формой Search, поэтому подробные инструкции вы найдете в тексте главы.

1. Создайте новую форму, не прибегая к помощи мастеров, и добавьте пустой холст (Search\_Canvas) и пустой блок данных (Search\_Block). Добавьте поля редактирования Beginning Date, Ending Date и Listing Agent. Задайте исходные значения начальной и конечной дат, 01-Jan-2006 и 31-Dec-2006. Добавьте список Sale Status с тремя элементами (For Sale = 101, Pending = 102, Sold = 103). С помощью мастера LOV создайте список значений для агентов.

```
SELECT AGENTS.AGENTID, AGENTS.LASTNAME,
AGENTS.FIRSTNAME FROM AGENTS ORDER BY
AGENTS.LASTNAME, AGENTS.FIRSTNAME
```

2. Используйте SQL PLUS для создания нового представления.

```
CREATE VIEW AgentListings AS SELECT
ListingID, Listings.PropertyID,
ListingAgentID, SaleStatusID,
BeginListDate, EndListDate, AskingPrice,
City FROM Properties INNER JOIN Listings ON
Properties.PropertyID =
Listings.PropertyID;
```

3. Создайте блок данных (Listings), соответствующий новому представлению. Убедитесь в том, что установлены основные свойства: Query All Records=Yes, Query Data Name=

```
(SELECT ListingID, PropertyID,
ListingAgentID, SaleStatusID,
BeginListDate, EndListDate, City,
AskingPrice FROM AgentListings)
WHERE clause =
(BeginListDate <= :SEARCH_BLOCK.TXT_ENDDATE
AND EndListDate >= :SEARCH_BLOCK.TXT_
BEGINDATE) AND (ListingAgentID =
:SEARCH_BLOCK.TXT_AGENTID) AND
(SaleStatusID =
:SEARCH_BLOCK.LIST_SALESTATUS)
ORDER By = BeginListDate
```

и включите все столбцы в список Query Data Source Columns.

4. Добавьте к холсту фрейм таблицы, в котором отображались бы данные из блока Listings. Включите поля редактирования ListingID, PropertyID, BeginListDate, EndListDate и AskingPrice.
5. Добавьте к Search\_Block кнопку Search и свяжите с ней код обработчика GO\_BLOCK('LISTINGS'); EXECUTE\_QUERY;.
6. Менеджер не любит работать с клавиатурой и предпочитает там, где возможно, использовать мышь. Зная этот факт, поместите рядом с полем редактирования Agent кнопку, посредством которой будет отображаться список значений. Напишите код обработчика GO\_ITEM('TXT\_AGENTID'); LIST\_VALUES;.
7. Для вычисления суммарной цены на отображаемые объекты добавьте поле редактирования к блоку данных LISTINGS и установите свойства, аналогичные тем, которые были показаны на рис. 9.23, но установите свойства Prompt и Name=Subtotal, Summary Function=Sum, Summarized Block=LISTINGS и Summarized Item=ASKINGPRICE.
8. Добавьте к Search\_Block поле редактирования, поместив его над столбцом Asking Price. Установите свойства Name=Subtotal, Justification=Right, Data Type=Number, Format Mask=\$99,999,990, Calculation Mode=Formula, Formula=:LISTINGS.Subtotal, Database Item=No, Prompt=Subtotal.
9. Выведите форму на печать и предъявите преподавателю. Возможно, преподаватель потребует копии файлов.

## 2. Coffee Merchant

В приложении Coffee Merchant требуется усовершенствовать форму Order, так как пользователям необходимо быстро получать информацию о заказанных товарах. Если вы помните, в приложении используются основная и подчиненная формы, в которых отображаются данные о заказах и потребителе. В подчиненной форме содержится список заказанных товаров. Для InventoryID уже создан список значений, позволяющий сотруднику выбирать значение из таблицы Inventory. Однако желательно предоставить холст в виде слоя, чтобы сотрудник при необходимости мог видеть более подробные данные из описи.

1. Вначале скопируйте файл Orders.fmb, созданный в главе 8, в новую папку. Скомпилируйте и проверьте форму, чтобы убедиться, что она работает. При редактировании формы установите для даты свойство Initial Value=\$\$DATE\$\$.
2. Добавьте холст в виде слоя и создайте область просмотра в правой части подчиненной формы. Переименуйте исходный холст (Main\_Canvas) и назовите его Inventory.
3. С помощью мастера блока данных добавьте новый блок, выбрав все столбцы из таблицы Inventory. Убедитесь, что создана связь Inventory.InventoryID

=OrderLines.InventoryID. С помощью мастера компоновки поместите все элементы на новом холсте Inventory. Добейтесь, чтобы все элементы поместились в области просмотра. Используя раскрывающийся список в верхней части формы, перейдите на Inventory\_Canvas, чтобы видеть и фрейм, и область просмотра.

4. Создайте обработчик Pre-Form и используйте его, чтобы первоначально скрыть новый холст. Добавьте в основную форму флажок опции, с помощью которого пользователь будет отображать или скрывать холст, выполненный в виде слоя.

```
IF (:CHK_INVENTORY > 0) THEN
    SHOW_VIEW('INVENTORY');
ELSE
    HIDE_VIEW('INVENTORY');
END IF;
```

5. Вам необходимо автоматически генерировать OrderID, поэтому добавьте к таблице Orders последовательность (seq\_Orders, отсчет начинается со значения 300000) и напишите код обработчика, который при добавлении новой записи в форму Orders вставлял бы очередное значение последовательности. Используйте пример, рассмотренный в данной главе в качестве руководства. В особенности он потребуется вам тогда, когда придется задавать глобальную переменную в качестве индикатора для обработчиков Pre-Form и When-New-Form-Instance.
6. Вам также надо вычислить суммарное значение цены для заказанных товаров. При этом необходимо умножать цену товара на количество приобретенных изделий. Для того чтобы отобразить итоговое значение в форме, вам потребуется удалить столбец Name. Создайте новое поле редактирования с именем ItemValue, установите для него свойства Calculation Mode=Formula и Formula = ROUND ((:ORDERLINES.QUANTITY\*: ORDERLINES.PRICE) \* (1-:ORDERLINES. DISCOUNT), 2). Вам также необходимо установить основные свойства, включая метку и маску формата.
7. Добавьте новое поле редактирования в таблицу, отображающую заказы. Оно нужно для вычисления итогового значения. Поступайте так же, как и в примере, показанном на рис. 9.23, но установите свойства Name=Subtotal, Summary Function=Sum, Summarized Block=ORDERLINES и Summarized Item=ITEMVALUE.
8. Добавьте к блоку Orders поле редактирования, поместив его над столбцом ItemValue. Установите свойства Name=Subtotal, Justification=Right, Data Type=Number, Format Mask=FML9G999D00, Calculation Mode=Formula, Formula=:ORDERLINES.Subtotal, Database Item=No, Prompt=Subtotal. Убедитесь, что для свойства Query All Records блока данных ORDERLINES задано значение Yes.
9. Выведите форму на печать и предъявите преподавателю. Возможно, преподаватель потребует копии файлов.

### 3. Rowing Ventures

Число таблиц в приложении Rowing Ventures относительно мало. В то же время они тесно связаны друг с другом, и очень часто пользователям требуется обращаться к данным из различных таблиц. Таким образом, имеет смысл создать табличную форму, в которой содержались бы связанные между собой данные. В этом случае пользователи смогут переходить от одной формы к другой и видеть именно ту информацию, которая им необходима.

Начнем с нуля. Сформируем новую табличную форму и назовем ее *RaceDay*. Создайте страницы *Race*, *RaceTimes*, *Boat + BoatCrew*, *Person* и *Organization*. Используя мастер блоков данных и мастер компоновки, создайте формы для каждой страницы. Для таблиц *Race* и *Times* данные будут размещены в виде таблицы. Задайте порядок сортировки для блока данных *Times*. Для страницы *Boat + BoatCrew* нам понадобится основная и подчиненная формы, аналогичные тем, которые были созданы в главе 8. Формы *Person* и *Organization* будут отображать по одной строке данных. При создании новой формы убедитесь в том, что условия объединения заданы корректно. Там, где необходимо, используйте списки значений.

Поскольку форма *Race* включает информацию о всех гонках, вам следует автоматически загружать данные при запуске формы. Включите выражение `EXECUTE_QUERY;` в обработчик события *When-New-Block-Instance* для блока данных *Race*. Вам также следует добавить последовательность и обеспечить автоматическую генерацию ключей *RaceID*.

При тестировании формы учитывайте, что она не позволяет вводить новые значения для лодок, участников или организаций. Поскольку эти данные связаны с самой гонкой, соответствующие страницы могут быть использованы только для отображения уже зарегистрированных участников и организаций. Данная форма хорошо подходит для просмотра информации о гонках. Для ввода данных о новых лодках и участниках можно использовать уже имеющиеся формы. При желании вы можете создать дополнительные таблицы для включения лодок, участников и организаций. При этом следите за тем, чтобы не создать связь с гонкой.

### 4. Broadcloth Clothing

Продолжая работу над приложением компании *Broadcloth Clothing*, создадим форму, отображающую сведения о продажах. Поскольку в этой форме будет содержаться большой объем данных, используем в ее составе холст в виде вкладок. На первой вкладке будут отображаться данные из таблиц *Shipment* и *ShipmentItem*. Они будут оформлены как основная и подчиненная формы. Обеспечьте вычисление количества проданных товаров. На одной из вкладок холста следует реализовать отображение данных из таблицы *Item*. Данные о потребителях должны быть отображены на отдельной вкладке, связанной с формой, соответствующей продажам. С помощью этой

формы не надо вводить данные о новых потребителях, так как на момент продажи потребитель уже должен быть определен. Сотруднику нужна возможность лишь скорректировать номер телефона или адрес. Создайте необходимые списки значений. Сформируйте последовательность, которая автоматически генерировала бы значения *ShipmentID*. Для формы *Item* создайте представление, которое включало бы все столбцы таблицы *Item*, а также *ModelDescription* из таблицы *Model*. Для страницы *Customer* используйте представление, включающее имя и номер телефона потребителя. Они нужны для того, чтобы у сотрудника компании была под рукой контактная информация.

Нетрудно заметить, что таблица *CustomerOrder* уже содержит адрес доставки. Было бы неплохо, если бы форма продаж могла при выборе потребителя автоматически извлекать этот адрес и использовать его в качестве адреса по умолчанию. В подчиненной форме *Shipment Item* вам потребуется создать код обработчика, проверяющего корректность *OrderID*. Единственная сложность состоит в том, что этот код будет выполняться при каждом вводе *OrderID*, т.е для каждой строки табличной формы. Убедитесь, что для поля *OrderID* создан список значений, в котором отображаются заказы только от выбранных потребителей. Создайте следующий запрос:

```
SELECT ALL CUSTOMERORDER.ORDERID,
CUSTOMEORDER.CUSTOMERID,
CUSTOMEORDER.ORDERDATE
FROM CUSTOMEORDER
WHERE CUSTOMERID=:SHIPMENT.CUSTOMERID
ORDER BY CUSTOMEORDER.ORDERDATE
```

Затем используйте обработчик, проверяющий корректность *OrderID*.

```
BEGIN
  SELECT DeliveryAddress,
         DeliveryCity,
         DeliveryState,
         DeliveryPostal,
         DeliveryNation
    INTO :SHIPMENT.ShipAddress,
          :SHIPMENT.ShipCity,
          :SHIPMENT.ShipState,
          :SHIPMENT.ShipPostal,
          :SHIPMENT.ShipNation
   FROM CustomerOrder WHERE
OrderID=:SHIPMENTITEM.OrderID;
END;
```

Для каждого поля редактирования следует решить, можно ли составить какие-нибудь правила, позволяющие проверить корректность информации. Например, поскольку форма посвящена продажам, их объем всегда должен быть больше нуля. Поскольку данная форма достаточно сложна, вам возможно, следует обсудить ее структуру с преподавателем.

# Словарь терминов

**Automatic database diagnostic monitor (ADDM).** Инструмент администрирования, выполняющийся в фоновом режиме и выполняющий мониторинг. Анализ производительности можно инициировать из Enterprise Manager; ADDM генерирует рекомендации по изменению параметров для повышения общей производительности системы.

**COMMIT.** Команда, после выполнения которой все изменения, внесенные с момента предыдущей команды COMMIT, становятся постоянными.

**Data Pump.** Утилита Oracle, предназначенная для экспортования и импортирования больших объемов информации.

**Developer Suite.** Интегрированная среда разработки, объединяющая средства для создания приложений и решения бизнес-задач.

**DUAL.** Таблица, определенная в системе Oracle, содержащая одну строку и один столбец. Она используется для отображения значения (например, результата расчетов), не являющегося столбцом конкретной таблицы.

**END.** Ключевое слово, закрывающее блок PL/SQL.

**Enterprise Manager (EM).** Инструмент на базе Web, который помогает начинающим администраторам следить за работой базы данных и управлять ею.

**EXISTS.** Оператор, позволяющий определить наличие условия в подзапросе.

**IN OUT.** Спецификатор параметра, указывающий на то, что параметр может быть передан функции или процедуре, его значение может быть изменено и возвращено вызывающей процедуре.

**IN.** Спецификатор параметра, который позволяет передавать значение функции или процедуре, но не может быть изменен в процессе выполнения функции или процедуры.

**Integer.** Целое число, не имеющее дробной части.

**JInitiator.** Java-приложение, предоставляемое Oracle, которое выполняется в среде Web-браузера и предназначено для обработки форм. Данный продукт должен быть установлен на пользовательском компьютере.

**Lightweight directory application protocol (LDAP).** Стандарт, который определяет принцип хранения данных о пользователях в централизованном каталоге.

**MetaLink.** Группа поддержки в составе Oracle. На сервере MetaLink публикуются обновления и дополнения. Для автоматизации задач поддержки программного обеспечения учетная запись на данном сервере интегрирована в состав Enterprise Manager.

**NULL.** Зарезервированное слово, означающее, что значение неизвестно.

**Oracle Application Server (OracleAS).** Программная система, обрабатывающая

формы и отчеты на сервере. Она также содержит инструменты для выполнения аутентификации и обеспечения безопасности.

**Oracle Containers for Java (OC4J).** Независимый Web-сервер, обрабатывающий элементы форм и отчетов. Для тестирования форм необходимо запустить экземпляр OC4J, в противном случае выполнить проверку можно будет только на сервере приложений.

**Oracle Internet Directory (OID).** Централизованная система, используемая для поддержки единой регистрации.

**OUT.** Спецификатор параметра, позволяющего изменять значение в процессе выполнения процедуры или функции и передавать его вызывающей процедуре.

**P-код.** Переносимый код функции или процедуры. Может выполняться на любом компьютере, на котором установлен интерпретатор PL/SQL.

**Secure sockets layer (SSL).** Протокол, использующий открытый ключ для шифрования при передаче данных между браузером и сервером.

**SQL Access Advisor.** Инструмент, анализирующий группы SQL-выражений и генерирующий предложения по организации хранения данных. Распознает ситуации, в которых целесообразно добавление индексов или материализованных представлений.

**SQL Tuning Advisor.** Инструмент, анализирующий SQL-запрос и генерирующий предложения по повышению производительности.

**SQL\*Loader.** Утилита Oracle, предназначенная для импортирования данных из внешних файлов, в частности файлов .CSV.

**STANDARD Extensions.** Раздел объектного навигатора, который представляет часто используемые функции, объяснения и примеры.

**Unified Modeling Language (UML).** Стандартизованный язык моделирования, предназначенный для проектирования компьютерных и деловых систем.

**Very large-scale database (VLSD).** База данных очень большого размера, содержащая терабайты информации, которая обычно требует специальной настройки системы хранения.

**Агрегирующая функция.** Действует одновременно на несколько строк таблицы и возвращает одну строку.

**Администратор баз данных (Database administrator – DBA).** Сотрудник, отвечающий за бесперебойную работу СУБД и осуществляющий контроль ее производительности.

**Активированная строка.** Единственная строка, доступная программе PL/SQL посредством курсора.

**Анонимный блок.** Анонимный, или неименованный, блок содержит выражения SQL и PL/SQL, пригодные для компиляции и выполнения.

**Ассоциативная ссылка.** Ключевое слово, ссылающееся на список тем, связанных с текущей.

**Атомарные значения.** Значения, которые не подлежат делению на составляющие.

**Атрибуты.** Характеристики объектов.

**База данных.** Набор данных, хранящихся в стандартных форматах, предназначенный для работы с ними нескольких пользователей. Базой данных также называют набор таблиц, предназначенный для конкретной бизнес-задачи.

**Базисная таблица.** Таблица, содержащая информацию о единичном основном объекте. Обычно не содержит внешних ключей.

**Бесконечный цикл.** Цикл, выполнение которого не прекращается.

**Блок данных.** Внутренний элемент формы, используемый для поддержки передачи форме информации из базы данных.

**Блок.** Содержит выражения SQL и PL/SQL.

**Вертикальное объединение.** Сложный запрос называется вертикальным объединением, поскольку извлекаемые строки формируются на основе столбцов, а не строк.

**Верхний колонтитул.** Часть отчета, отображаемая вверху каждой страницы.

**Виртуальная частная база данных (virtual private database – VPD).** Инструмент Oracle, который использует представления для определения подмножества данных для любой конкретной роли. В частности, каждый пользователь видит только те строки, которые имеют отношение к стоящей перед ним задаче.

**Внешнее объединение.** Возвращает строки не только тогда, когда удовлетворяется условие объединения, но и когда один из столбцов содержит значение NULL.

**Внешние таблицы.** Файл, не входящий в систему Oracle, определяемый SQL\*Loader и представляющий псевдотаблицу. Часто используется с файлами .CSV. Единожды определив внешние таблицы, их можно использовать для передачи данных.

**Внешний ключ.** Столбец таблицы, который является первичным ключом другой таблицы.

**Внутреннее объединение.** Объединение двух или более таблиц, которое возвращает только строки, соответствующие условию объединения.

**Внутреннее представление.** Подзапрос, следующий за выражением FROM и помещенный в кавычки.

**Время жизни.** Время между созданием и удалением переменной. Данное понятие связано с областью видимости. Переменная существует только во фрейме, в котором она была создана. Значения, присваиваемые при обработке события, удаляются по завершении работы обработчика. Если тот же обработчик будет вызван повторно, предыдущие значения будут удалены.

**Вторая нормальная форма.** Таблица находится во второй нормальной форме (2NF), если столбцы, не относящиеся к ключу, зависят от всего ключа.

**Вызов.** Обращение к процедуре или функции из другой процедуры или из SQL\*Plus.

**Выражение EXCEPTION.** Структура, используемая в составе кода обработчика для перехвата ошибок, возникающих в процессе выполнения. В случае возникновения ошибки процессор форм переходит к разделу обработки исключений.

**Выражение GROUP BY.** Необязательное выражение в составе оператора SELECT, которое разбивает полученный набор результатов на группы.

**Выражение HAVING.** Необязательное выражение в составе оператора SELECT, которое задает дополнительный фильтр для сгруппированных строк.

**Выражение ORDER BY.** Фрагмент выражения SELECT, выполняющий сортировку возвращаемых строк в порядке, заданном посредством имен столбцов.

**Выражение ROLLBACK.** SQL-выражение, отменяющее одну или несколько команд, составляющих транзакцию.

**Выражение WHERE.** Необязательное выражение оператора SELECT, определяющее фильтр или условия поиска, которым должны соответствовать все возвращаемые строки таблицы.

**Выражение WITH ADMIN OPTION.** Выражение, которое используется в SQL-команде GRANT и дает пользователю право назначать определенные привилегии другим пользователям.

**Глобальная переменная.** Переменная, к которой можно обращаться из-за пределов блока PL/SQL.

**Грифы секретности.** Сложная система защиты, предполагающая присвоение специальных меток пользователям и данным. Применяется в дополнение к обычной системе привилегий. Например, данные, помеченные как совершенно секретные, доступны только тем пользователям, которые имеют права на работу с информацией этого уровня секретности.

**Группа записей.** Внутренняя структура формы, которая содержит выражение SELECT, применяемое для извлечения данных. Часто используется для заполнения списка значений.

**Групповая функция** (см. Агрегирующая функция).

**Групповой отчет.** Отчет, в котором данные разделены на группы. Обычно для каждой группы формируется раздел заголовка и раздел окончания.

**Групповые символы.** Специальные символы, замещающие один или несколько символов в строке поиска.

**Детализация.** Детальная установка прав доступа. Например, вместо того, чтобы предоставлять доступ ко всей таблице, можно предоставить права для работы только с конкретными столбцами.

**Диспетчер восстановления (recovery manager – RMAN).** Инструмент Oracle, запускаемый из командной строки и предназначенный для настройки операций создания резервных копий и восстановления системы при сбое.

**Дополнения.** Код, изменяющий СУБД. Для установки дополнений обычно требуется новая лицензия и иногда приходится реструктуризовать существующие файлы базы данных.

**Доступность.** Характеристика системы, отражающая тот факт, что авторизованные пользователи в случае необходимости могут получить доступ к ней.

**Дочерние пункты меню.** Пункты второго уровня, которые становятся доступными при выборе пункта первого уровня.

**Единая регистрация.** Средства, позволяющие пользователю единожды зарегистрироваться в сети и автоматически получать права для работы со всем службами и приложениями.

**Естественное объединение.** Объединение по эквивалентности, при котором имена столбцов в обеих таблицах совпадают.

**Жесткое кодирование.** Включение данных непосредственно в код формы. Такой подход затрудняет последующее внесение изменений. Желательно использовать для определения информации, которая может впоследствии изменяться, глобальные переменные, чтобы данные хранились в одном месте.

**Заголовок группы.** Фрагмент отчета, отображаемый в начале раздела. Обычно используется для отображения заголовков столбцов.

**Заголовок отчета.** Раздел, который выводится в начале отчета.

**Закрытая переменная.** Переменная, недоступная из-за пределов блока PL/SQL, в котором она определена.

**Запись.** Составная структура, которая построена так же, как строка, извлеченная из явного курсора.

**Запрос действия.** Команда INSERT, UPDATE или DELETE, изменяющая данные.

**Запрос.** Средство, позволяющее задавать вопросы базе данных и получать от нее ответы.

**Звездочка (\*)**. В выражении SELECT указывает на то, что должны быть извлечены все столбцы таблицы.

**Имена с разделителями.** Имена объектов (таблицы, столбцы и т.п.), помещенные в двойные кавычки, например "CreditLimit".

**Именованный блок.** Блок PL/SQL, имеющий имя и хранящийся в базе. Обращение к такому блоку осуществляется по его имени.

**Именованный визуальный атрибут.** Определение стиля, воздействующего на свойства объекта и определяющие его визуальное представление (шрифт, цвет, фоновый узор).

**Индексная страница.** Отсортированный список ключевых слов, используемый в справочных системах.

**Инструменты администрирования.** Средства, помогающие администратору отслеживать состояние СУБД и настраивать ее для повышения производительности.

**Инфраструктура открытого ключа (public-key infrastructure – PKI).** Кодирование с помощью открытого ключа требует поддержки средств генерации и хранения ключей. Когда кодирование используется для аутентификации серверов, подлинность сервера подтверждает специальная организация, выдавшая цифровой сертификат.

**Исключение.** Ошибка или недопустимая ситуация, возникающая при выполнении блока PL/SQL.

**Каскадные таблицы стилей (Cascading style sheet – CSS).** Набор свойств, определяющих внешний вид Web-страницы.

**Клавиша быстрого доступа.** Клавиша, нажимаемая в комбинации с клавишей <Alt>, для выбора пункта меню.

**Класс свойств.** Набор значений свойств, используемый для применения стандартных стилей к различным объектам.

**Класс.** Описание набора объектов, имеющих общую структуру, поведение и отношения. Класс представляет сущность и ее свойства.

**Кластер.** Группа таблиц, разделяющая один блок данных. Используется для объединения связанных между собой таблиц и снижения общего количества операций чтения и записи.

**Команды SQL\*Plus.** Расширение Oracle команд SQL, предоставляющее различные средства для форматирования результатов, отображения определений таблиц, редактирования и сохранения файлов.

**Команды форматирования отчета.** Команды, предназначенные для создания заголовков и окончаний, нижних и верхних колонтитулов, подавления вывода дублирующихся данных, форматирования столбцов символов и чисел и выполнения вычислений.

**Компиляция.** Преобразование высокочувственных команд PL/SQL в низкоуровневый формат, который быстрее обрабатывается на компьютере.

**Конкатенация.** Объединение последовательностей символов, при котором первый символ очередной последовательности располагается за последним символом предыдущей.

**Константа.** Значение, занимающее определенную часть области хранения. Оно существует лишь то время, в течение которого доступен блок PL/SQL, и не изменяется в процессе выполнения кода.

**Контекстно-зависимая подсказка.** Фрагмент справочной информации, связанный с определенной задачей или фрагментом приложения.

**Конфиденциальность.** Характеристика системы защиты, означающая, что доступ к данным могут получать только авторизованные пользователи. Для неавторизованных пользователей не допускается даже чтение информации или выполнение выражений SELECT.

**Коррелированный подзапрос.** Подзапрос, ссылающийся на один или несколько столбцов внешнего запроса.

**Корреляционные имена.** В процессе выполнения обработчика они содержат старое и новое значение модифицируемой строки.

**Курсор.** Область памяти, выделенная процессором PL/SQL для хранения строк, возвращаемых в результате выражения SELECT.

**Левостороннее внешнее объединение.** Объединение, которое возвращает строки из левой таблицы, даже если они возвращают значение NULL.

**Массив независимых дисковых устройств (Redundant array of independent drives – RAID).** Система хранения данных, состоящая из нескольких независимых устройств. Данные записываются на разные диски с некоторой избыточностью. При этом увеличивается производительность и надежность данной системы.

**Мастер блоков данных.** Инструмент, используемый в начале работы над формой. По окончании выполнения он обычно вызывает мастер компоновки.

**Мастер компоновки.** Основной инструмент для размещения на холсте элементов из блока данных. Он уменьшает объем работы по вводу кода. Данный инструмент можно использовать даже после завершения работы над формой.

**Материализованное представление.** “Мгновенный снимок” таблицы в конкретный момент времени. Используется для ускорения выполнения запросов при наличии объединений. Копии данных находятся во временном хранилище, поэтому система должна выполнить объединение только один раз при обновлении информации.

**Матричный отчет.** Отчет, содержащий данные из различных таблиц и отображающий объединенные значения двух управляющих переменных. Одна переменная определяет строки, а другая — столбцы.

**Метрика.** Мера активности и производительности базы данных.

**“Многие ко многим”.** Отношение, согласно которому одна строка таблицы может соответствовать нескольким строкам второй таблицы. В то же время строка второй таблицы может соответствовать нескольким строкам первой таблицы.

**Многостолбцовый подзапрос.** Возвращает внешнему выражению SQL больше одного столбца.

**Многострочный подзапрос.** Возвращает внешнему выражению SQL две или несколько строк.

**Модель данных.** Внутреннее описание данных, требующихся для отчета. Используется для передачи информации из базы данных.

**Модель форматирования.** Набор символов, описывающий правила отображения элемента и ввода данных пользователем. Примеры приведены в табл. 8.3. Подробная информация о допустимых символах содержится в документации по Oracle.

**Модуль меню.** Инструмент, предназначенный для хранения подменю в процессе создания дерева меню.

**Наблюдаемая таблица.** Именованная производная таблица.

**Набор символов национального языка.** Позволяет записывать информацию из языков, отдельные символы которых нельзя представить в восьмибитовой схеме кодирования.

**Наследование.** Способность создавать новые классы, порожденные от класса более высокого уровня. Новый класс наследует все свойства и методы своего родителя.

**Неявный курсор.** Курсор, который автоматически управляется процессором PL/SQL. Он используется тогда, когда выражение SELECT в блоке PL/SQL возвращает не больше одной строки.

**Нижний колонтитул.** Часть отчета, отображаемая внизу каждой страницы.

**Нормализация.** Процесс создания набора таблиц, эффективно хранящих данные, направленный на минимизацию избыточности и обеспечение целостности информации.

**Область видимости.** Определяет, когда доступна переменная. Глобальная переменная доступна в пределах всего приложения. Переменная, объявленная в процедуре, доступна только из этой процедуры.

**Область просмотра.** Область, используемая для отображения части холста, расположенного поверх другого холста.

**Обновление.** Код, предназначенный для устранения недостатков в работе СУБД. Управлять обновлениями удобно с помощью инструмента Enterprise Manager.

**Обработчик проверки корректности.** Используется для тестирования параметров, введенных пользователем в форме.

**Обработчик событий форматирования.** Код, выполняющийся каждый раз, когда целевой объект отображается в составе отчета.

**Обработчик уровня выражений.** Обработчик, получающий управление один раз при возникновении события DML, независимо от числа затронутых строк.

**Обработчик уровня строки.** Обработчик, который вызывается при воздействии на строку.

**Обработчик.** Фрагмент кода, получающий управление при выполнении выражений INSERT, UPDATE или DELETE над определенной таблицей базы данных.

**Обработчики отчета.** Обработчики событий, не связанные с конкретными данными, а соответствующие различным этапам обработки отчета.

**Объединение по эквивалентности.** Наиболее часто используемый тип условия объединения, основанный на соответствии значений в двух таблицах.

**Объединение, не требующее эквивалентности.** Условие объединения, которое вместо оператора равенства использует оператор отношения между двумя столбцами.

**Объединение.** Временная связь между двумя таблицами. Часто эта связь создается путем формирования соответствия строк одной таблицы строкам другой таблицы и базируется на столбцах, имеющих в каждой таблице одинаковое значение.

**Объектные привилегии.** Разрешают операции над существующими таблицами базы данных, представлениями, процедурами и другими подобными объектами. Объектные привилегии можно назначать ролям.

**Объектный навигатор.** Список элементов формы с иерархической организацией. В древовидной структуре объектного навигатора может быть отражено несколько форм.

**Объекты схемы.** Имена, присваиваемые всем объектам, которые хранятся в пользовательской схеме.

**Ограничение CHECK.** Инициирует применение к столбцу логического выражения, результат которого должен быть равен `true` для всех значений столбца.

**Ограничение NOT NULL.** Запрещает существование столбцов, в которых отсутствуют данные или содержится значение `NULL`.

**Ограничение внешнего ключа.** Если ограничения первичного ключа обеспечивают целостность данных в таблице и однозначную идентификацию каждой строки, ограничение внешнего ключа указывает на то, что поле должно выполнять роль первичного ключа в другой таблице.

**Ограничение на первичный ключ.** Гарантирует, что все значения столбца, в котором оно объявлено, не пустые и уникальные.

**Ограничения домена.** Определяют конкретные значения данных или диапазоны значений в тех столбцах, с которыми связаны ограничения.

**Ограничения целостности.** Правила, применяющиеся к таблицам и их столбцам и ограничивающие значения, которые могут быть помещены в таблицы.

**“Один ко многим”.** Отношение, согласно которому одна и только одна строка первой таблицы может соответствовать нескольким строкам второй таблицы. Однако каждая строка второй таблицы может соответствовать лишь одной строке первой таблицы.

**Однострочная функция.** Воздействует на одну строку таблицы и возвращает одну строку для каждой входной строки.

**Однострочный подзапрос.** Возвращает внешнему SQL-выражению одну строку или ни одной строки.

**Окончание группы.** Фрагмент отчета, завершающий группу; обычно в нем отображаются итоговые значения.

**Окончание отчета.** Раздел, который выводится в конце отчета.

**Оператор IN.** Задает условие, истинное в том случае, если данные в столбце таблицы соответствуют одному из элементов списка допустимых значений.

**Оператор LIKE.** Использует групповые символы для поиска в строке заданных символьных последовательностей.

**Операторы действий над множествами.** Операторы, которые объединяют результаты нескольких выражений `SELECT` в один запрос.

**“Основа–детали”.** Отчет, содержащий два раздела и поддерживающий отношение “один ко многим” между разделом основы и разделом детальной информации.

**Откат.** Отмена незавершенных изменений базы данных.

**Отключенное ограничение.** Ограничение, получившее имя, но не введенное в действие.

**Отладчик форм.** Инструмент, обеспечивающий пошаговое выполнение обработчиков и контроль за выполнением кода. Отладчик также позволяет просматривать переменные и данные формы.

**Ошибка выполнения.** Проблема, возникающая при работе приложения, например, деление на нуль. Подобные ошибки проявляются только на этапе обработки форм. Их можно перехватить с помощью выражения EXCEPTION.

**Палитра свойств.** Небольшое окно, используемое для просмотра и модификации свойств формы и ее элементов.

**Параметр DESFORMAT.** Параметр службы отчетов, определяющий формат для генерации отчетов.

**Параметр DESTYPE.** Параметр службы генерации отчетов, который задает тип устройства для вывода.

**Параметр PCTFREE.** Параметр, управляющий хранением информации. Используется в составе CREATE TABLE и определяет, когда к табличному пространству должен добавляться новый блок данных.

**Параметр PCTUSED.** Параметр, управляющий хранением информации. Используется в составе CREATE TABLE и определяет момент, с которого Oracle снова сможет использовать блок данных для записи информации.

**Параметр UserID.** Стандартный идентификатор <имя\_пользователя>/<пароль> @<имя\_сервера>, сообщающий службам отчетов о том, как установить соединение с базой данных.

**Параметр.** Список значений, разделенных запятыми и помещенных в круглые скобки. Указывается после имени функции или процедуры и используется для передачи значений.

**Первая нормальная форма.** Таблица находится в первой нормальной форме в том случае, если она не содержит повторяющихся групп.

**Первичный ключ.** Один или несколько столбцов, содержащих значения, которые однозначно идентифицируют каждую строку таблицы.

**Перегруженность информацией.** Ситуация, при которой предпринимается попытка одновременно отобразить в форме слишком много сведений.

**Переключатель.** Элемент формы, используемый для установки значения опции. Принято отображать переключатели круглой формы и реализовывать в них зависимую фиксацию.

**Переменная подстановки.** Используется в выражении SQL. При выполнении выражения вместо нее подставляется реальное значение.

**Переменная среды.** Системная переменная SQL\*Plus, определяющая, как должны отображаться различные элементы.

**Переменная.** Область для временного хранения значений, полностью содержащихся в блоке PL/SQL.

**План выполнения запроса.** Объясняет, как система Oracle должна обрабатывать запрос. Полезен в тех случаях, когда надо выяснить причину низкого быстродействия.

**Платформенно-ориентированный код.** Выполняется на компьютере определенной архитектуры и не требует интерпретации.

**Повторяющийся фрейм.** Используется для отображения нескольких строк данных. Все данные в составе отчета в конечном итоге должны быть помещены в повторяющийся фрейм.

**Подзапрос.** Запрос, содержащийся в составе другого SQL-выражения, как правило, запроса.

**Подпрограмма.** Компьютерная программа, содержащаяся в составе другой программы и выполняющаяся почти независимо от последней. Подпрограмма вызывается включающей программой.

**Подсистема защиты.** Часть СУБД, ответственная за идентификацию пользователей и использование механизма полномочий для контроля за их деятельностью.

**Поисковая страница.** Инструмент в составе справочной системы, который позволяет пользователям выполнять поиск по ключевым словам.

**Полное внешнее объединение.** Возвращает не только строки, для которых было найдено соответствие, но и строки, не имеющие аналогов в других таблицах.

**Пользовательская роль.** Набор полномочий, которые получают пользователи, принадлежащие определенной группе. Права желательно назначать ролям, а не отдельным пользователям.

**Пользовательские параметры.** Переменные, определенные и используемые в запросах отчетов. При выполнении отчета пользователь вводит значение каждого параметра, и запрос отчета изменяется соответствующим образом.

**Последовательность первичных событий.** Порядок генерации событий при работе пользователя с формой. Одна из основных задач — выбор события, обеспечивающего вызов обработчика в нужный момент.

**Последовательность.** Объект базы данных, который генерирует набор уникальных целочисленных значений.

**Почтовое уведомление.** N-кратный отчет, отображающий данные на странице в виде групп записей.

**Правостороннее внешнее объединение.** Объединение, которое возвращает строки из правой таблицы, даже если они содержат значение NULL.

**Представление подмножества столбцов.** Скрывает некоторые столбцы от указанных пользователей.

**Представление подмножества строк.** Скрывает выбранные строки от некоторых пользователей.

**Представление.** Объект, поддерживающий подмножество строк и столбцов из базовой таблицы.

**Предупреждающее сообщение.** Отображается в специальном окне. Используя подобные сообщения, необходимо следить за тем, чтобы приложение не было перегружено ими.

**Префикс GLOBAL.** Синтаксическая конструкция для объявления глобальной переменной.

**Приложение** Полнофункциональная система, выполняющая конкретный набор задач. Обычно приложение состоит из интегрированных форм и отчетов и может включать меню и справочную систему.

**Приоритет операторов.** Порядок, в котором Oracle вычисляет подвыражения.

**Производная таблица.** Результаты, возвращаемые системой в ответ на запрос (SELECT).

**Простое представление.** Представление, которое определено на одной базовой таблице.

**Простой цикл.** Цикл, который выполняется до тех пор, пока не будет явным образом завершен.

**Профиль SQL.** Набор запросов и предложений по их улучшению. Анализатор запросов использует данные профиля для повышения производительности.

**Процедура.** Именованный блок PL/SQL, который хранится на сервере Oracle и выполняет некоторые действия.

**Процессор данных.** Часть СУБД, поддерживающая большую часть реальных операций с компьютерной системой.

**Процессор запросов.** Часть СУБД, которая с помощью языка PL/SQL определяет, как хранить и извлекать данные в ответ на запросы пользователей и приложений.

**Псевдоним столбца.** Альтернативное имя столбца, указываемое в выражении SELECT.

**Псевдоним таблицы.** Короткое имя, используемое в запросе вместо реального имени таблицы.

**Псевдоним.** Альтернативное имя таблицы или столбца.

**Пункт меню.** Позволяет выполнить конкретное действие.

**Пункт.** Единица измерения размера шрифтов. Если размер шрифта установлен равным 72 пунктам, высота прописных букв составляет около одного дюйма.

**Разграничительный контроль доступа.** Предоставление разным пользователям доступа к различным подмножествам строк и столбцов одной и той же базовой таблицы.

**Раздел деклараций.** Необязательный раздел блока PL/SQL, который начинается с ключевого слова DECLARE. В нем определяются переменные и константы, требуемые для работы блока.

**Раздел заголовка.** Раздел именованного блока, содержащего имя блока и информацию о том, является ли он функцией или процедурой.

**Раздел исполняемого кода.** Исполняемые выражения, содержащиеся между ключевыми словами BEGIN и END.

**Раздел обработки исключений.** Необязательный раздел блока PL/SQL, который начинается с ключевого слова EXCEPTION и обеспечивает обработку ошибок.

**Разделение обязанностей.** Принцип защиты, согласно которому в выполнение финансовой операции должны быть вовлечены несколько сотрудников. Это затрудняет хищение денег одним непорядочным служащим.

**Разделение по диапазонам.** Способ хранения таблиц, при котором таблица разделяется на части, хранящиеся на разных носителях. Решение о принадлежности конкретному разделу принимается в зависимости от того, в какой диапазон попадает значение. Например, можно разделить таблицу на основе значений даты так, чтобы более старые данные хранились на медленных и недорогих носителях, а новые — на дисках с большим быстродействием.

**Разделение по списку.** Способ хранения таблицы, предполагающий разбиение таблицы на разделы, которое основывается на группах данных в указанном столбце. Часто используется для разделения данных по коду штата.

**Разработчик.** Специалист, который проектирует базу данных, составляет запросы и создает формы и отчеты.

**Распределение.** Частота попадания значений в каждую категорию.

**Результаты мониторинга.** Файлы протоколов, отражающие действия с базой данных и позволяющие выявлять ошибки или недопустимые операции.

**Реляционная база данных.** Набор таблиц, логически связанных между собой посредством хранящихся в них данных.

**Родительский пункт меню.** Пункт меню первого уровня.

**Руководство по стилям.** Задает общую структуру форм и основные элементы, содержащиеся в них.

**Самообъединение.** Возвращает строки из одной таблицы, объединяя таблицу саму с собой и рассчитывая условие объединения.

**Сеанс.** Отдельное соединение с сервером баз данных Oracle.

**Сеть.** Обеспечивает взаимодействие пользователя с другими компьютерами.

**Символьная функция.** Функция, которая обрабатывает строки символов и возвращает результат, также представленный в виде строки символов.

**Символьные данные переменной длины.** Данные, длина которых может изменяться от одной записи к другой.

**Символьные данные фиксированной длины.** При их использовании поля различных записей имеют одинаковую длину.

**Система управления базой данных (СУБД).** Программное обеспечение, используемое для определения базы, размещения в ней данных, поддержки запросов, генерации отчетов и создания окон для ввода информации.

**Системная привилегия.** Привилегия, позволяющая пользователю выполнять определенные действия в системе Oracle, например, создавать таблицы и представления.

**Системные параметры.** Предопределенные переменные, используемые для контроля среды выполнения отчета. С их помощью можно, например, задать число копий или имя принтера.

**Системные привилегии.** Права пользователей на изменение структуры базы, например, создание таблиц, пользователей или представлений.

**Скалярный подзапрос.** Подзапрос, возвращающий одну строку и один столбец.

**Словарь Oracle.** Содержит таблицы с информацией обо всех таблицах и объектах.

**Словарь данных.** Таблицы, хранящие определение всех таблиц данных и описывающие типы хранящихся данных.

**Сложное представление.** Включает подзапрос, извлекающий данные из нескольких таблиц, группирующий строки с использованием оператора GROUP BY или DISTINCT либо содержащий вызов функции.

**Сложный запрос.** Выражение SQL, содержащее операторы действий над множествами.

**Службы форм и отчетов.** Отдельная часть системы Oracle, поддерживающая создание, отладку и выполнение форм и отчетов.

**Содержание.** Входная страница справочной системы, которая дает представление о темах, по которым можно получить информацию.

**Составляющий запрос.** Каждое выражение SELECT в сложном запросе.

**Составной первичный ключ.** Первичный ключ, состоящий из нескольких столбцов.

**Список значений (List of Values – LOV).** Список всех возможных значений, которые могут быть выбраны для конкретного поля. Список может подвергаться фильтрации и предоставлять пользователю возможность поиска.

**Список таблиц.** Список, следующий за выражением FROM и содержащий одну или несколько таблиц, из которых должны быть извлечены данные.

**Столбец объединения.** Столбец первичного или внешнего ключа, используемый для объединения таблиц.

**Строка.** Содержит данные, соответствующие одному экземпляру объекта.

**Суперкласс.** Родительский класс.

**Схема пользователя.** Oracle позволяет многим пользователям одновременно работать с базой данных, которая обычно размещается на отдельном сервере. Файлы баз данных хранятся и поддерживаются путем размещения пользовательских таблиц и других объектов в отдельной защищенной области, которая называется схемой пользователя.

**Сценарий.** Любые SQL-команды, помещенные в файл.

**Таблица.** Структура, состоящая из строк и столбцов и содержащая данные, которые представляют один элемент бизнес-модели.

**Табличная форма.** Форма, в которой данные отображаются в виде строк и столбцов. Такая форма применима, если число столбцов ограничено, а пользователям надо одновременно видеть и сравнивать несколько строк.

**Табличное пространство Undo.** Специальное табличное пространство для временного хранения изменений в базе данных. Пользователи могут использовать команду Rollback для отмены изменений и восстановления предыдущего состояния данных и таблиц.

**Табличное пространство.** Логическое объединение элементов базы данных. Табличное пространство отображается в физические файлы данных.

**Табличный отчет.** Простой отчет, организованный в виде строк и столбцов.

**Тип Magic.** Пункты меню, выполняющие предопределенные действия, например копирование и вставку.

**Тип данных.** Определяет или ограничивает набор значений, которые могут храниться в столбце.

**Точность.** Общее число цифр, хранимых в числе; масштаб — число цифр справа от десятичной точки.

**Транзакция.** Группа выражений SQL, представляющих собой отдельный элемент задачи. Элементы одной транзакции должны быть либо все выполнены, либо отвергнуты.

**Третья нормальная форма.** Таблица находится в третьей нормальной форме (3NF), если она находится в 2NF и каждый неключевой столбец зависит от всего ключа и ни от чего более.

**Усечение.** В применении к таблице это действие, при котором из таблицы удаляются все данные и не сохраняется информация, позволяющая отменить его.

**Условие объединения.** Условие, которое проверяет Oracle, чтобы определить, удовлетворяют ли два значения критерию объединения двух таблиц.

**Условие поиска.** Имена столбцов, указанные в выражении WHERE.

**Условие поиска.** Следует за выражением WHERE и идентифицирует строки таблиц, которые система Oracle должна извлечь.

**Условие уникальности.** Означает, что все значения в некотором столбце должны встречаться только по одному разу; повтор значений не допускается.

**Уточненное имя.** Состоит из двух частей. Одна часть задает таблицу, а другая — столбец, который в ней содержится.

**Файл Redo.** Содержит записи, применяемые для восстановления данных в случае отказа базы. Данный файл желательно хранить на отдельном устройстве.

**Флажок опции.** Элемент формы, используемый для установки значения опции. По соглашениям флажок опции отображается в виде небольшого квадратного поля. Флажки могут устанавливаться и сбрасываться независимо друг от друга.

**Форма пользовательского параметра.** Предопределенная форма, упрощающая ввод пользователями значений параметров для отчетов.

**Форма.** Интерактивное приложение, используемое для ввода информации и редактирования данных, хранящихся в таблицах.

**Формат CSV.** Формат, используемый для передачи данных между различными системами. Содержимое столбцов разделяется запятыми, строки обычно помещаются в кавычки, а каждая строка файла представляет одну строку таблицы базы данных.

**Фрейм (в составе отчета).** Средство группировки элементов. Используется для установки общих свойств элементов, содержащихся во фрейме.

**Фрейм (в составе формы).** Средство группировки элементов в составе формы. Группировка выполняется как для улучшения внешнего вида, так и для объединения связанных между собой элементов, например переключателей.

**Функция даты.** Функция, возвращаемым значением которой является дата или время.

**Функция преобразования.** Функция, которая конвертирует один тип данных в другой.

**Функция регулярного выражения.** Функция, использующая регулярные выражения для поиска символьных данных.

**Функция.** Заранее заданный набор выражений, которому может передаваться любое (в том числе нулевое) количество входных значений или параметров и которое возвращает выходное значение.

**Холст в виде вкладок.** Холст с несколькими разделами. Каждый раздел выбирается щелчком на ярлыке вкладки. Обычно разделы представляют собой связанные между собой формы, например форма *Customer* для соответствующей формы *Order*.

**Холст.** Составная часть формы, предназначенная для отображения элементов. В форме может находиться несколько холстов.

**Холсты в виде слоев.** Холсты, расположенные один над другим. Область просмотра определяет видимую часть верхнего холста.

**Хэш-разделы.** Способ хранения таблиц, при котором для записи и извлечения данных используется ключевое значение. При этом данные, базирующиеся на отдельных значениях, оказываются изолированными, и обеспечивается их быстрое извлечение.

**Целостность на уровне ссылок.** Ограничение целостности данных, означающее, что данные можно включать в столбец внешнего ключа, только если в базовой таблице уже существует соответствующее значение первичного ключа.

**Целостность.** Характеристика системы защиты, означающая, что данные хранятся корректно и соответствуют тому понятию, для представления которого они были записаны. Для достижения целостности принимаются меры по восстановлению базы данных и запрещается выполнение неавторизованными пользователями операций, предполагающих изменение информации.

**Цикл FOR курсора.** Курсор может автоматически обрабатываться посредством специального цикла FOR. При этом нет необходимости открывать, закрывать курсор или выполнять команду FETCH.

**Цикл FOR.** Цикл, повторяющийся заранее заданное число раз.

**Цикл WHILE.** Цикл, который выполняется до тех пор, пока истинно указанное условие.

**Цифровой сертификат.** Структура, использующая шифрование с открытым ключом для аутентификации Web-сервера и генерирующая ключи для кодирования данных, которыми браузер обменивается с сервером.

**Число с плавающей точкой.** Содержит переменное количество знаков после десятичной точки.

**Число фиксированной точности.** Содержит заданное число десятичных знаков.

**Числовая функция.** Позволяет выполнять расчеты; после обработки возвращает числовой результат.

**Шаблон формы.** Стандартный набор стилей, используемый как основа для всех форм в приложении.

**Шаблон.** Набор атрибутов, которые влияют на расположение и внешний вид элементов и обеспечивают согласованное представление отчетов.

**Шифрование.** Процедура, в результате которой обычный текст преобразуется с помощью ключевого значения в кодированную последовательность, скрывающую в себе исходную информацию.

**Элементы.** Различные объекты, помещаемые в формы, в том числе текстовые метки и поля.

**Явный курсор.** Курсор, действия с которым (определение, открытие, загрузка и закрытие) необходимо выполнять явно. Используется в сочетании с блоком PL/SQL, который возвращает больше одной строки.



# Предметный указатель

## A

ADDM Advisor, *CD214*  
ADF, *650*  
Application development framework, *650*

## B

Buffer Cache Advisor, *CD217*

## C

CSS, *657*

## D

Database administrator, *CD171*  
DBA, *CD171*  
Developer Suite, *549; 675*  
DML-выражение, *503*

## E

Enterprise Manager, *CD173*

## F

FGA, *CD163*  
Fine-Grained Auditing, *CD163*  
Forms Builder, *549; 551*

## J

J2EE, *650*  
Java 2 Enterprise Edition, *650*  
JDeveloper, *650*  
JInitiator, *549; 560; 649*  
JSP, *657*

## L

LOV, *575*

## M

Memory Advisor, *CD217*  
MetaLink, *CD200*  
Microsoft Active Directory, *CD126*  
Model-view-controller, *653*  
MVC, *653*

## N

Net Manager, *550*

## O

Object Navigator, *552*  
OC4J, *550*  
OID, *CD197; CD125*  
Oracle Application Server, *CD92*  
Oracle Containers for Java, *550*  
Oracle Internet Directory, *CD97*  
OracleAS, *CD92*  
OTN, *650*

## P

Р-код, *520*  
PGA Advisor, *CD217*  
PKI, *CD151*  
PL/SQL, *471*  
Public-key infrastructure, *CD151*

## Q

Query Access Advisor, *CD217*

## R

RAID, *CD151*  
RAISE, *478*  
Recovery manager, *CD209*  
Report Wizard, *CD28*  
Reports Builder, *CD5; CD9*

RMAN, *CD209*

## S

Secure sockets layer, *CD151*  
 Segment Advisor, *CD217*  
 Shared Pool Advisor, *CD217*  
 SSL, *CD151*  
 SQL Access Advisor, *CD217*  
 SQL Tuning Advisor, *CD217*  
 SQL\*Loader, *CD195*  
 SQL\*Plus, *483*  
 SunONE Directory Server, *CD126*

## U

Undo Advisor, *CD217*

## V

Very large-scale database, *CD191*  
 Virtual private database, *CD133; CD146*  
 VLSD, *CD191*  
 VPD, *CD133; CD146*

## A

АБД, *32*  
 Автоматическое выполнение запроса, *577*  
 Администратор базы данных, *32, CD271*  
 Анонимный блок, *473*  
 Архитектура модель–представление–контроллер, *653*  
 Ассоциативная ссылка, *CD112*  
 Аутентификация пользователей, *CD125*

## Б

База данных, *31*  
 Блок PL/SQL, *473*  
 Блок данных, *552; 601; CD181*  
 Большой объект, *476*

## В

Верхний колонтитул, *CD7*  
 Виртуальная приватная база данных, *CD133*  
 Виртуальная частная база данных, *CD146*  
 Вкладка, *644*  
 Включение двоичного объекта в базу данных, *571*  
 Включение изображений в отчет, *CD41*  
 Включение фотоснимка в базу данных, *571*  
 Внешняя таблица, *CD195*  
 Временное табличное пространство, *CD182*  
 Время жизни переменной, *627*  
 Входной параметр, *508*  
 Входной/выходной параметр, *508*  
 Вызов процедуры, *525*  
 Вызов функции, *513*  
 Выравнивание столбцов отчета, *CD22*  
 Выражение CASE, *501*  
 Выражение IF, *501*  
 Выходной параметр, *508*

## Г

Глобальная переменная, *476; CD89*  
 Гриф секретности, *CD147*  
 Групповой отчет, *CD2*  
 Горячий режим резервирования, *CD209*

## Д

Детальный мониторинг, *CD163*  
 Диаграмма, *CD62*  
 Динамический отчет, *CD67*  
 Дополнение, *CD172; CD199*  
 Доставка форм и отчетов, *CD97*  
 Доступность, *CD124*  
 Дочерний пункт меню, *CD102*

**Е**

Единая регистрация, *CD97*

**Ж**

Жесткое кодирование, *CD89*

**З**

Заголовок группы, *CD7*

Заголовок окна, *557*

Заголовок отчета, *CD7*

Закрытая переменная, *474*

Закрытый ключ, *CD151*

Защита, *CD123*

**И**

Именованный блок, *507*

Именованный визуальный атрибут, *CD79*

Импортирование данных, *CD193*

Индексный файл, *CD114*

Инфраструктура открытого ключа, *CD151*

Исходная форма, *CD87*

Итоговые данные, *CD7*

Итоговые значения, *635*

**К**

Ключ шифрования, *CD150*

Класс свойств, *CD79*

Кластер, *CD189*

Ключевое слово BEGIN, *474; 476*

Ключевое слово DECLARE, *474*

Ключевое слово EXCEPTION, *474; 478; 625*

Ключевое слово END, *474*

Кнопки с зависимой фиксацией, *579*

Комбинированная форма, *546*

Комментарии, *475*

Компиляция, *471*

Константа, *474*

Конфиденциальность, *CD124*

Курсор, *477; 490*

**М**

Маска формата, *590*

Маска формата, *697*

Массив независимых

дисковых устройств, *CD178*

Мастер LOV, *575*

Мастер блоков данных, *553*

Мастер компоновки, *553*

Мастер отчетов, *CD7*

Материализованное представление, *CD226*

Матричный отчет, *CD2*

Межтабличный отчет, *CD2*

Меню, *CD100*

Модель данных, *653; CD29*

Модификация процедуры, *526*

Модификация функции, *515*

Мониторинг, *CD155*

Мониторинг, *CD199*

Мутировавшая таблица, *287*

**Н**

Неименованный блок, *473*

Неявный курсор, *477*

Неявный курсор, *490*

Нижний колонтитул, *CD7*

**О**

Область видимости переменной, *627*

Область просмотра, *642*

Обновление, *CD190*

Обновление, *CD199*

Обработка исключений, *487*

Обработчик исключений, *624*

Обработчик события, *552*

Обработчик, *610; 612*

Обработчики проверки

корректности, *CD48*

- Обработчики событий  
отчетов, *CD48*  
форматирования, *CD48*
- Объединение, *394*
- Объектные привилегии, *CD129*
- Объектный навигатор, *552*
- Объектный навигатор, *565*
- Окончание группы, *CD7*
- Оператор выбора, *501*
- Основная форма, *546; 547*
- Откат, *269*
- Открытый ключ, *CD151*
- Отладчик, *620*
- Отношение один ко многим, *547; 682*
- Отчет основа–детали, *CD37*
- Отчет, *CD1*
- Ошибка выполнения, *623*
- П**
- Палитра свойств, *551*
- Параметр IN OUT, *508*
- Параметр IN, *508*
- Параметр OUT, *508*
- Параметр desformat, *CD94*
- Параметр destype, *CD94*
- Параметр PCTFREE, *CD187*
- Параметр PCTUSED, *CD188*
- Параметр отчета, *CD43*
- Параметр, *508*
- Первичный ключ, *38*
- Переключатель, *579*
- Переопределяющий стиль, *CD50*
- Платформенно-ориентированный код, *520*
- Платформенно-ориентированная компиляция, *473*
- Повторяющиеся фреймы, *CD38*
- Подпрограмма, *473*
- Подстановочные переменные, *254*
- Подчиненная форма, *547; 584*
- Полоса прокрутки, *557*
- Пользовательская роль, *CD128*
- Пользовательский параметр, *CD44*
- Пользовательский шаблон, *CD49*
- Последовательность первичных событий, *613*
- Последовательность, *240; 629*
- Постоянное табличное пространство, *CD182*
- Почтовое уведомление, *CD2*
- Пошаговое выполнение кода, *620*
- Представление отчета в виде Web-страницы, *CD55*
- Приложение, *CD73*
- Проверка вводимых данных, *632*
- Простой список, *CD2*
- Простой фрейм, *CD38*
- Простой цикл, *490*
- Протокол LDAP, *CD126*
- Процедура, *473; 508*
- Р**
- Раздел группы, *CD7*
- Раздел деклараций, *474*
- Раздел заголовка, *507*
- Раздел исполняемого кода, *476*
- Раздел обработки исключений, *474*
- Раздел, *CD191*
- Разделение обязанностей, *CD131*
- Разделение по диапазонам, *CD192*
- Расширение, *CD181*
- Регистрация шаблонов, *CD153*
- Редактор компоновки, *564*
- Резервная копия, *CD204*
- Родительский пункт меню, *CD102*
- Роль, *CD128*
- Руководство по стилям, *CD76*

**С**

Самообъединение, 405  
 Сегмент, *CD181*  
 Сервер приложений Oracle, *CD92*  
 Симметричное шифрование, *CD150*  
 Системное представление, *CD227*  
 Системные привилегии, *CD129*  
 Системный параметр, *CD43*  
 Скалярная переменная, 476  
 Скалярный тип, 476  
 Служба отчетов, *CD5*  
 Событие, 612  
 Согласованный внешний вид, *CD76*  
 Создание отчета вручную, *CD36*  
 Создание процедуры, 521  
 Создание функции, 509  
 Составная переменная, 476  
 Специальная настройка отчетов, *CD28*  
 Список значений, 575  
 Списочный раздел, *CD193*  
 Справочная система  
     на базе Java, *CD110*  
 Справочная система  
     на базе Web, *CD110*  
 Справочная система, *CD108*  
 Ссылка, 476  
 Страница Page Layout, *CD15*  
 Страница Paper Design, *CD13*  
 Стока меню, *CD100*  
 СУБД, 31

**Т**

Таблица мутировавшая, 287  
 Таблицы стилей, *CD61*  
 Табличная форма, 546; 582  
 Табличное пространство Undo, *CD182*  
 Табличное пространство, *CD180*  
 Табличный отчет, *CD2*  
 Точка останова, 620  
 Транзакция

откат, 269

фиксация, 268

**У**

Удаление процедуры, 528  
 Удаление функции, 517; 519  
 Управляющий файл, *CD179*  
 Утилита *wrap*, *CD155*  
 Учетная запись, *CD126*

**Ф**

Файл .CSV, *CD195*  
 Файл данных, *CD181*  
 Файл протокола Redo, *CD186*  
 Файл протокола Undo, *CD186*  
 Фиксация, 268  
 Фильтр, *CD47*  
 Фильтрация, *CD46*  
 Флажок опции, 579  
 Форма на основе Web, 549  
 Форма пользовательского  
     параметра, *CD44*  
 Форма, 546  
 Фрейм, 601; *CD19*  
 Функция CALL\_FORM, *CD88*  
 Функция OPEN\_FORM, *CD88*  
 Функция, 473; 508

**Х**

Холодный режим резервирования, *CD207*  
 Холст, 551; 600  
     в виде вкладок, 639; 644  
     в виде слоев, 639; 641  
     в виде отдельных областей  
         отображения, 639  
 Хэш-раздел, *CD193*

**Ц**

Целостность на уровне ссылок, 263  
 Целостность, *CD124*

Цикл FOR, 490  
Цикл WHILE, 490  
Цикл, 490  
Цифровой сертификат, CD151

### **III**

Шаблон Default, CD50  
Шаблон, CD49; CD77  
Шифрование, CD150  
    с помощью двух ключей, CD150  
    с помощью открытого ключа, CD150

### **Э**

Экспортирование данных, CD193  
Элемент формы, 551

### **Я**

Явный курсор, 477; 490

*Научно-популярное издание*

**Джеймс Перри, Джеральд Пост**

# **Введение в Oracle 10g**

Литературный редактор *И.А. Попова*

Верстка *О.С. Назаренко*

Художественные редакторы *О.А. Василенко, Е.П. Дынник*

Корректоры *О.В. Мишутина, Л.В. Чернокозинская*

Издательский дом “Вильямс”  
101509, г. Москва, ул. Лесная, д. 43, стр. 1  
Подписано в печать 15.08.2006. Формат 70x100/16.

Гарнитура Times. Печать офсетная.

Усл. печ. л. 56,76. Уч.-изд. л. 38,68.

Тираж 3000 экз. Заказ № .

Отпечатано по технологии CtP  
в ОАО “Печатный двор” им. А. М. Горького  
197110, Санкт-Петербург, Чкаловский пр., 15.