

**SAMS**  
**ОСВОЙ**  
**САМОСТОЯТЕЛЬНО**



**3-е  
издание**

Включая MySQL  
и PostgreSQL

# SQL

Бен Форта

**10** минут  
на урок

**SAMS**  
**ОСВОЙ**  
**САМОСТОЯТЕЛЬНО**

# SQL

**10** **МИНУТ**  
**на урок**

Ben Forta

800 E. 1st Street, Indianapolis, Indiana 46240 USA

SAMS



**SAMS**  
**Teach**  
**Yourself**

# SQL

Ben Forta

*in* **10**  
**Minutes**

**THIRD EDITION**

**SAMS**

800 East 96th Street, Indianapolis, Indiana, 46240 USA

**SAMS**  
**Освой**  
**самостоятельно**

# SQL

Бен Форта

**10 минут**  
**на урок**

3-е издание



Издательский дом "Вильямс"  
Москва • Санкт-Петербург • Киев  
2005

ББК 32.973.26-018.2.75

Ф80

УДК 681.3.07

Издательский дом "Вильямс"

Главный редактор *С.Н. Тригуб*

Зав. редакцией *В.Р. Гинзбург*

Перевод с английского и редакция *В.С. Гусева*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:  
info@williamspublishing.com, <http://www.williamspublishing.com>  
115419, Москва, а/я 783, 03150, Киев, а/я 152.

**Форга, Бен.**

**Ф80** Освой самостоятельно SQL. 10 минут на урок, 3-е издание. :  
Пер. с англ. — М. : Издательский дом "Вильямс", 2005. —  
288 с. : ил. — Парал. тит. англ.

ISBN 5-8459-0827-2 (рус.)

Данная книга поможет вам в кратчайшие сроки освоить SQL — самый популярный язык баз данных. Начиная с простых запросов на выборку данных, автор урок за уроком рассматривает все более сложные темы, такие как использование операций объединения, подзапросы, хранимые процедуры, индексы, триггеры и ограничения. На изучение материала каждого урока вам потребуется не более 10 минут. Благодаря этой книге вы быстро научитесь самостоятельно составлять запросы к базам данных на языке SQL без чьей-либо помощи.

Примеры, приведенные в книге, будут работать во всех наиболее популярных СУБД — IBM DB2, Microsoft Access, Microsoft SQL Server, MySQL, Oracle, PostgreSQL и Sybase Adaptive Server.

**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 Sams Publishing.

Authorized translation from the English language edition published by Sams Publishing Copyright © 2004

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Russian language edition is published by Williams Publishing House according to the Agreement with R&I Enterprises International, Copyright © 2005

ISBN 5-8459-0827-2 (рус.)

ISBN 0-672-32567-5 (англ.)

© Издательский дом "Вильямс", 2005

© Sams Publishing, 2004

# Оглавление

|   |     |
|---|-----|
| Введение  | 15  |
| Урок 1. Что такое SQL                                 | 19  |
| Урок 2. Выборка данных                                | 29  |
| Урок 3. Сортировка выбранных данных                   | 35  |
| Урок 4. Фильтрация данных                             | 43  |
| Урок 5. Расширенная фильтрация данных                 | 51  |
| Урок 6. Использование метасимволов для фильтрации     | 61  |
| Урок 7. Создание вычисляемых полей                    | 69  |
| Урок 8. Использование функций манипулирования данными | 79  |
| Урок 9. Суммирование данных                           | 89  |
| Урок 10. Итоговые данные                              | 101 |
| Урок 11. Использование подзапросов                    | 113 |
| Урок 12. Объединение таблиц                           | 121 |
| Урок 13. Создание расширенных объединений             | 133 |
| Урок 14. Комбинированные запросы                      | 145 |
| Урок 15. Добавление данных                            | 153 |
| Урок 16. Обновление и удаление данных                 | 163 |
| Урок 17. Создание таблиц и работа с ними              | 169 |
| Урок 18. Использование представлений                  | 181 |
| Урок 19. Работа с хранимыми процедурами               | 193 |
| Урок 20. Обработка транзакций                         | 203 |
| Урок 21. Использование курсоров                       | 211 |
| Урок 22. Расширенные возможности SQL                  | 219 |
| Приложение А. Сценарии демонстрационных таблиц        | 235 |
| Приложение Б. Работа с популярными приложениями       | 243 |
| Приложение В. Синтаксис операторов SQL                | 259 |
| Приложение Г. Использование типов данных SQL          | 265 |
| Приложение Д. Резервированные слова SQL               | 273 |
| Предметный указатель                                  | 278 |

# Содержание

|  |           |
|--|-----------|
| Об авторе                                  | 12        |
| Благодарности                              | 13        |
| Ждем ваших отзывов!                        | 14        |
| <b>Введение</b>                            | <b>15</b> |
| Для кого эта книга?                        | 15        |
| СУБД, используемые в этой книге            | 16        |
| Условные обозначения                       | 16        |
| <b>Урок 1. Что такое SQL</b>               | <b>19</b> |
| Основы баз данных                          | 19        |
| Что такое база данных                      | 20        |
| Таблицы                                    | 21        |
| Столбцы и типы данных                      | 22        |
| Строки                                     | 23        |
| Первичные ключи                            | 24        |
| Что такое SQL?                             | 25        |
| Попробуйте сами                            | 27        |
| Резюме                                     | 27        |
| <b>Урок 2. Выборка данных</b>              | <b>29</b> |
| Оператор SELECT                            | 29        |
| Выборка отдельных столбцов                 | 30        |
| Выборка нескольких столбцов                | 32        |
| Выборка всех столбцов                      | 33        |
| Резюме                                     | 34        |
| <b>Урок 3. Сортировка выбранных данных</b> | <b>35</b> |
| Сортировка данных                          | 35        |
| Сортировка по нескольким столбцам          | 37        |
| Сортировка по положению столбца            | 38        |
| Указание направления сортировки            | 39        |
| Резюме                                     | 42        |
| <b>Урок 4. Фильтрация данных</b>           | <b>43</b> |
| Использование предложения WHERE            | 43        |
| Операции в предложении WHERE               | 45        |
| Проверка одного значения                   | 46        |
| Проверка на несовпадения                   | 46        |
| Проверка на диапазон значений              | 48        |
| Проверка на отсутствие значения            | 48        |
| Резюме                                     | 50        |

|  |    |
|--|----|
| <b>Урок 5. Расширенная фильтрация данных</b>                 | 51 |
| Комбинирование предложений WHERE                             | 51 |
| Использование ключевого слова AND                            | 52 |
| Использование ключевого слова OR                             | 53 |
| Порядок обработки  | 54 |
| Использование ключевого слова IN                             | 56 |
| Использование ключевого слова NOT                            | 57 |
| Резюме   | 59 |
| <b>Урок 6. Использование метасимволов для фильтрации</b>     | 61 |
| Использование логического оператора LIKE                     | 61 |
| Метасимвол “знак процента” (%)                               | 62 |
| Метасимвол “символ подчеркивания” ( _ )                      | 65 |
| Метасимвол “квадратные скобки” ( [ ] )                       | 66 |
| Советы по использованию метасимволов                         | 68 |
| Резюме   | 68 |
| <b>Урок 7. Создание вычисляемых полей</b>                    | 69 |
| Что такое вычисляемые поля                                   | 69 |
| Конкатенация полей   | 70 |
| Использование псевдонимов                                    | 74 |
| Выполнение математических вычислений                         | 76 |
| Резюме   | 78 |
| <b>Урок 8. Использование функций манипулирования данными</b> | 79 |
| Что такое функция  | 79 |
| Проблемы с функциями   | 79 |
| Использование функций  | 81 |
| Функции манипулирования текстом                              | 82 |
| Функции манипулирования датой и временем                     | 84 |
| Функции для манипулирования числами                          | 87 |
| Резюме   | 88 |
| <b>Урок 9. Суммирование данных</b>                           | 89 |
| Использование статистических функций                         | 89 |
| Функция AVG ( )  | 90 |
| Функция COUNT ( )  | 92 |
| Функция MAX ( )  | 93 |
| Функция MIN ( )  | 94 |
| Функция SUM ( )  | 95 |
| Статистические вычисления для отдельных значений             | 96 |
| Комбинирование статистических функций                        | 98 |
| Резюме   | 99 |

|   |     |
|---|-----|
| <b>Урок 10. Итоговые данные</b>                           | 101 |
| Получение итоговых данных                                 | 101 |
| Создание групп  | 102 |
| Фильтрующие группы  | 104 |
| Группирование и сортировка                                | 108 |
| Упорядочение предложения SELECT                           | 110 |
| Резюме  | 111 |
| <b>Урок 11. Использование подзапросов</b>                 | 113 |
| Что такое подзапросы                                      | 113 |
| Фильтрация посредством подзапросов                        | 114 |
| Использование подзапросов в качестве<br>вычисляемых полей | 118 |
| Резюме  | 120 |
| <b>Урок 12. Объединение таблиц</b>                        | 121 |
| Что такое объединения                                     | 121 |
| Что такое реляционные таблицы                             | 121 |
| Для чего используют объединения                           | 123 |
| Создание объединения                                      | 124 |
| Важность предложения WHERE                                | 126 |
| Внутренние объединения                                    | 128 |
| Объединение многих таблиц                                 | 129 |
| Резюме  | 132 |
| <b>Урок 13. Создание расширенных объединений</b>          | 133 |
| Использование псевдонимов таблиц                          | 133 |
| Использование объединений других типов                    | 134 |
| Самообъединения   | 135 |
| Естественные объединения                                  | 137 |
| Внешние объединения                                       | 138 |
| Использование объединений со статистическими<br>функциями | 142 |
| Использование объединений и условий<br>объединения        | 143 |
| Резюме  | 144 |
| <b>Урок 14. Комбинированные запросы</b>                   | 145 |
| Что такое комбинированные запросы                         | 145 |
| Создание комбинированных запросов                         | 146 |
| Использование оператора UNION                             | 146 |
| Правила применения запросов UNION                         | 149 |
| Включение или исключение повторяющихся<br>строк           | 150 |

|   |            |
|---|------------|
| Сортировка результатов комбинированных запросов                     | 151        |
| Резюме  | 152        |
| <b>Урок 15. Добавление данных</b>                                   | <b>153</b> |
| Что такое добавление данных   | 153        |
| Добавление полных строк   | 153        |
| Добавление части строки   | 157        |
| Добавление выбранных данных   | 158        |
| Копирование данных из одной таблицы в другую                        | 160        |
| Резюме  | 162        |
| <b>Урок 16. Обновление и удаление данных</b>                        | <b>163</b> |
| Обновление данных   | 163        |
| Удаление данных   | 165        |
| Советы по обновлению и удалению данных                              | 167        |
| Резюме  | 168        |
| <b>Урок 17. Создание таблиц и работа с ними</b>                     | <b>169</b> |
| Создание таблиц   | 169        |
| Основы создания таблиц  | 170        |
| Работа со значениями NULL   | 172        |
| Определение значений по умолчанию                                   | 174        |
| Обновление таблиц   | 176        |
| Удаление таблиц   | 178        |
| Переименование таблиц   | 179        |
| Резюме  | 179        |
| <b>Урок 18. Использование представлений</b>                         | <b>181</b> |
| Что такое представления   | 181        |
| Для чего используют представления                                   | 182        |
| Представления: правила и ограничения                                | 183        |
| Создание представлений  | 185        |
| Использование представлений для упрощения сложных объединений       | 186        |
| Использование представлений для переформатирования выбранных данных | 187        |
| Использование представлений для фильтрации нежелательных данных     | 190        |
| Использование представлений с вычисляемыми полями                   | 191        |
| Резюме  | 192        |
| <b>Урок 19. Работа с хранимыми процедурами</b>                      | <b>193</b> |
| Что такое хранимые процедуры  | 193        |
| Для чего используют хранимые процедуры                              | 195        |



|  |            |
|--|------------|
| Выполнение хранимых процедур                                 | 197        |
| Создание хранимых процедур                                   | 198        |
| Резюме   | 202        |
| <b>Урок 20. Обработка транзакций</b>                         | <b>203</b> |
| Что такое обработка транзакций                               | 203        |
| Управляемые транзакции                                       | 206        |
| Использование оператора ROLLBACK                             | 207        |
| Использование оператора COMMIT                               | 207        |
| Использование точек сохранения                               | 208        |
| Резюме   | 210        |
| <b>Урок 21. Использование курсоров</b>                       | <b>211</b> |
| Что такое курсоры  | 211        |
| Работа с курсорами   | 213        |
| Создание курсоров  | 214        |
| Использование курсоров                                       | 215        |
| Закрытие курсоров  | 217        |
| Резюме   | 217        |
| <b>Урок 22. Расширенные возможности SQL</b>                  | <b>219</b> |
| Что такое ограничения  | 219        |
| Первичные ключи  | 220        |
| Внешние ключи  | 222        |
| Ограничения уникальности                                     | 224        |
| Ограничения на значения столбца                              | 225        |
| Что такое индексы  | 227        |
| Что такое триггеры   | 229        |
| Безопасность баз данных                                      | 231        |
| Резюме   | 232        |
| <b>Приложение А. Сценарии демонстрационных таблиц</b>        | <b>235</b> |
| Что такое демонстрационные таблицы                           | 235        |
| Описания таблиц  | 236        |
| Таблица Vendors  | 236        |
| Таблица Products   | 237        |
| Таблица Customers  | 237        |
| Таблица Orders   | 238        |
| Таблица OrderItems   | 238        |
| Получение демонстрационных таблиц                            | 239        |
| Загрузка готового к работе MDB-файла для<br>Microsoft Access | 240        |
| Загрузка SQL-сценариев СУБД                                  | 240        |
| <b>Приложение Б. Работа с популярными приложениями</b>       | <b>243</b> |
| Использование Aqua Data Studio                               | 244        |

|   |            |
|---|------------|
| Использование DB2                                   | 245        |
| Использование Macromedia ColdFusion                 | 245        |
| Использование Microsoft Access                      | 246        |
| Использование Microsoft ASP                         | 248        |
| Использование Microsoft ASP.NET                     | 249        |
| Использование Microsoft Query                       | 250        |
| Использование Microsoft SQL Server                  | 251        |
| Использование MySQL                                 | 252        |
| Использование Oracle                                | 252        |
| Использование PHP                                   | 253        |
| Использование PostgreSQL                            | 253        |
| Использование Query Tool                            | 254        |
| Использование Sybase                                | 255        |
| Конфигурирование источников данных ODBC             | 255        |
| <b>Приложение В. Синтаксис операторов SQL</b>       | <b>259</b> |
| ALTER TABLE   | 259        |
| COMMIT  | 260        |
| CREATE INDEX  | 260        |
| CREATE PROCEDURE                                    | 260        |
| CREATE TABLE  | 261        |
| CREATE VIEW   | 261        |
| DELETE  | 261        |
| DROP  | 262        |
| INSERT  | 262        |
| INSERT SELECT                                       | 262        |
| ROLLBACK  | 262        |
| SELECT  | 263        |
| UPDATE  | 263        |
| <b>Приложение Г. Использование типов данных SQL</b> | <b>265</b> |
| Строковые данные                                    | 266        |
| Числовой тип данных                                 | 268        |
| Типы данных даты и времени                          | 269        |
| Двоичные типы данных                                | 270        |
| <b>Приложение Д. Зарезервированные слова SQL</b>    | <b>273</b> |
| Предметный указатель                                | 278        |

## Об авторе

**Бен Форта** — главный технический специалист компании Macromedia, за его плечами 20 лет работы в компьютерной индустрии, включая разработку продуктов, их поддержку и распространение, а также обучение пользованию ими. Бен Форта — автор таких бестселлеров, как *ColdFusion Web Application Construction Kit* и *Advanced ColdFusion Development, Sams Teach Yourself Regular Expressions in 10 Minutes (Освой самостоятельно регулярные выражения. 10 минут на урок*, Издательский дом “Вильямс”), а также книг по Flash, Java, WAP, Windows 2000 и другим технологиям. У него огромный опыт в разработке баз данных, он обеспечил их поддержку в нескольких очень популярных программных пакетах. Он часто читает лекции и пишет статьи для Internet, посвященные технологиям баз данных. Бен родился в Лондоне, там же получил образование, затем учился в Нью-Йорке и Лос-Анджелесе. Сейчас он живет в г. Оук-Парк, штат Мичиган, со своей женой Марси и семьей детьми. Бену можно написать по адресу [ben@forta.com](mailto:ben@forta.com) и посетить его Web-узел по адресу <http://www.forta.com>.

## Благодарности

Спасибо команде из Sams за многолетнюю поддержку. Особую благодарность хочу выразить Майку Стивенсу (Mike Stephens) и Марку Ренфроу (Mark Renfrow) за то, что они провели это новое издание книги от идеи до реальности (при этом иногда им приходилось вести и меня самого).

Спасибо читателям, которые написали отзывы по первым двум изданиям этой книги. К счастью, большинство из этих отзывов были положительными, и все они были оценены. Улучшения и изменения в этой редакции были сделаны именно благодаря вашим замечаниям.

И наконец, благодарю всех, кто приобрел предыдущие издания этой книги. Книга стала не только лично моим бестселлером, но и бестселлером по данной теме. Ваша постоянная поддержка — наилучшая благодарность, которую заслуживает автор.

## **Ждем ваших отзывов!**

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 вам эта книга или нет, 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

# Введение

SQL является самым популярным языком баз данных. Не важно, кто вы — разработчик приложений, администратор баз данных, Web-дизайнер или пользователь пакета Microsoft Office, — хорошее практическое знание SQL поможет вам взаимодействовать с базами данных.

Эта книга была написана из необходимости. Несколько лет я вел курс по разработке Web-приложений, и студенты постоянно просили порекомендовать им книгу по SQL. Существовало много книг, посвященных данной теме, и некоторые из них действительно были очень хороши. Но всем им было присуще одно общее свойство: в них было слишком много информации для большинства пользователей. Вместо того чтобы раскрывать тему SQL, в большинстве книг излагалось все, от разработки баз данных до теории реляционных баз данных и администрирования. Хотя это очень важные темы, они не интересны большинству людей, которые просто хотят изучить SQL.

Итак, не найдя ни одной книги, которую я бы мог порекомендовать, я вложил весь опыт преподавания в книгу, которую вы держите в руках. Данная книга поможет вам быстро освоить SQL. Начнем мы с простой выборки данных, затем перейдем к более сложным темам, таким как использование операций объединения, подзапросы, хранимые процедуры, индексы, триггеры и ограничения. Обучение будет проходить методично, систематично и просто — на каждый урок вам потребуется не более 10 минут.

Третья редакция этой книги уже помогла изучить SQL сотням тысяч пользователей, теперь пришел ваш черед. Переходите к первому уроку и приступайте к работе. Вы быстро научитесь писать первоклассные SQL-запросы.

## Для кого эта книга

Эта книга для вас, если вы

- новичок в SQL;
- хотите быстро научиться использовать SQL;
- хотите научиться использовать SQL в разрабатываемых вами приложениях;
- хотите самостоятельно составлять запросы к базам данных на SQL без чьей-либо помощи.

## СУБД, используемые в этой книге

В большинстве случаев SQL, который описывается в этой книге, можно применять в любой системе управления базой данных (СУБД, Database Management System — DBMS). Однако, так как не все реализации SQL идентичны, в книге особенно внимательно будут рассмотрены следующие СУБД (при необходимости будут даваться специальные инструкции или примечания).

- IBM DB2
- Microsoft Access
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- Sybase Adaptive Server

Примеры баз данных и SQL-сценарии будут работать во всех этих СУБД.

## Условные обозначения

В этой книге используются различные шрифты — во-первых, для того чтобы можно было отличить код от обычного текста, во-вторых, чтобы вы не пропустили важные понятия.

Текст, который вы вводите, и текст, который должен появиться на экране, представлены моноширинным шрифтом.

Он выглядит так, как на вашем экране.

Переменные и выражения-аргументы приведены *моноширинным курсивным* шрифтом. Переменный аргумент необходимо заменять определенным значением, которое он представляет.

Такая стрелка (↵) в начале строки кода означает, что эта строка слишком длинная и не поместилась в одну строку книги. Продолжайте вводить все символы после символа ↵ так, как если бы они были частью предыдущей строки.



В примечаниях находится интересная информация, относящаяся к обсуждаемой теме.



В подсказке вы найдете полезный совет или более быстрый способ что-либо выполнить.



В предупреждении вы узнаете о возможных проблемах и научитесь избегать неприятных ситуаций.



Под данной пиктограммой вы найдете определения новых базовых понятий.

## **ВВОД**

Пиктограммой “Ввод” обозначен код, который вы можете ввести самостоятельно.

## **ВЫВОД**

Пиктограмма “Вывод” указывает на информацию, которая выдается после запуска программы.

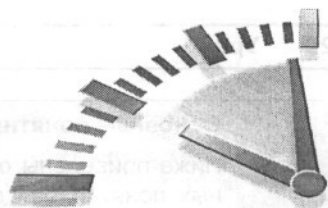
## **Анализ**

Пиктограмма “Анализ” указывает на то, что далее следует пошаговый комментарий к коду.





# Урок 1



## Что такое SQL

*На этом уроке вы узнаете, что такое SQL и что с его помощью можно сделать.*

## Основы баз данных

Тот факт, что вы читаете книгу по SQL, говорит о том, что вам так или иначе необходимо работать с базами данных. Язык SQL предназначен именно для этого, поэтому перед тем, как перейти к его рассмотрению, очень важно, чтобы вы познакомились с некоторыми основными понятиями технологии баз данных.

Хотите вы этого или нет, но вы постоянно пользуетесь базами данных. Каждый раз, когда вы выбираете имя в адресной книге электронной почты, вы используете базу данных. Если вы что-то ищете при помощи поискового сервера в Internet, вы используете базу данных. Когда вы регистрируетесь в локальной сети на работе, вы вводите свое имя и пароль, которые затем сравниваются со значениями, хранящимися в базе данных. И даже когда вы используете свою пластиковую карту в банкомате, вы используете базу данных при проверке PIN-кода и остатка на счету.

Однако несмотря на то, что мы постоянно используем базы данных, для многих остается непонятно, что же это на самом деле такое. И происходит это отчасти потому, что разные люди пользуются одними и теми же терминами, относящимися к базам данных, для определения совершенно разных вещей. Поэтому мы начнем наше обучение со списка определений наиболее важных терминов, относящихся к базам данных.



### Основные понятия

Ниже приведены очень краткие определения основных понятий баз данных. Они предназначены либо для того, чтобы напомнить вам о том, что вы уже знали, либо чтобы дать вам основные представления, если вы новичок в базах данных. Понимание баз данных является очень важной частью при изучении SQL, поэтому рекомендую найти хорошую книгу по основам баз данных и постоянно пополнять свои знания в данном предмете.

## Что такое база данных

Термин *база данных* используется в самых разных аспектах, но мы (а тем более с точки зрения SQL) будем считать базу данных набором сведений, хранящихся некоторым упорядоченным способом. Проще всего рассматривать базу данных как шкаф для хранения документов. Шкаф — это просто физическое местоположение для хранения данных, независимо от того, что это за данные и как они упорядочены.



### База данных

Контейнер (обычно файл или группа файлов) для хранения упорядоченных данных.



### Неправильное использование приводит к путанице

Люди часто используют термин *база данных* для обозначения программного обеспечения базы данных. Это неправильное использование термина часто ведет к путанице. На самом деле программное обеспечение баз данных называется системой управления базами данных (СУБД). База данных — это хранилище, созданное и управляемое посредством СУБД. База данных может быть файлом, хранящимся на жестком диске, а может и не являться таковым. Но чаще всего это несущественно, так как вы все равно никогда не обращаетесь к базе данных напрямую, для доступа к ней вы всегда используете СУБД.

## Таблицы

Когда вы храните информацию в шкафу для документов, вы стараетесь не перемешивать их. Напротив, все документы хранятся в соответствующих папках.

В мире баз данных такая папка называется таблицей. Таблица — это структурированный файл, в котором могут храниться данные определенного типа. В таблице может находиться список клиентов, каталог продукции и любая другая информация.



### Таблица

Структурированный список данных определенного типа.

Ключевой момент заключается в том, что данные, хранимые в таблице, должны быть одного типа или взяты из одного списка. Никогда не храните список клиентов и список заказов в одной таблице базы данных. Это затрудняет поиск и выборку информации. Лучше создать две таблицы для каждого из списков.

Каждая таблица базы данных имеет уникальное имя, ее идентифицирующее, и никакая другая таблица в базе данных не может носить это же имя.



### Имена таблиц

Уникальность имени таблицы достигается комбинацией некоторых вещей, включая имена базы данных и таблицы. В качестве части уникального имени некоторых баз данных используется имя владельца. Это означает, что, хотя нельзя использовать два одинаковых имени таблицы в одной базе, в разных базах данных имена таблиц могут повторяться.

Таблицы имеют характеристики и свойства, определяющие, каким образом в них хранятся данные. Сюда включается информация о том, какие данные могут храниться, как они распределены по таблицам, каким частям информации присвоены имена и многое другое. Такой набор информации, описывающей таблицу, называется *схемой*. Схемы используются для описания как определенных таблиц

в базе данных, так и для базы данных в целом (а также для описания взаимосвязей (отношений) между таблицами, если таковые имеются).



### Схема

Информация о базе данных, компоновке и свойствах таблицы.

## Столбцы и типы данных

Таблицы состоят из столбцов, в которых находятся отдельные фрагменты информации таблицы.



### Столбец

Одно поле таблицы. Все таблицы состоят из одного или нескольких столбцов.

Чтобы лучше понять это, представьте себе таблицы базы данных в виде сетки, наподобие электронных таблиц. В каждом столбце этой сетки находится определенная часть информации. Например, в таблице клиентов в одном столбце находится номер клиента, в другом — его имя. Адрес, город, область, почтовый индекс — все это находится в отдельных столбцах.



### Распределение данных

Очень важно правильно распределить данные по нескольким столбцам. Например, название города, области (штата) и почтовый индекс (для США это ZIP-код) всегда должны быть в отдельных столбцах. Это позволяет отсортировать или отфильтровать данные по определенным столбцам (например, чтобы найти всех клиентов из определенной области или города). Если названия города и области хранятся в одном столбце, будет очень сложно отсортировать или отфильтровать данные по области.

К каждому столбцу базы данных привязан определенный тип данных, который определяет, какие данные могут

содержаться в этом столбце. Например, если в столбце содержится число (скажем, соответствующее количеству продуктов в заказе), то тип данных будет числовой. Если в столбце необходимо хранить даты, текст, заметки, наличные счета и т.д., то для всех этих данных существует определенный тип.



### Тип данных

Тип разрешенных для хранения данных. Каждому столбцу базы данных присваивается тип данных, который запрещает (или разрешает) хранить в нем определенную информацию.

Типы данных ограничивают характер информации, которую можно хранить в столбце (например, предотвращают ввод алфавитных символов в числовое поле). Типы данных также помогают корректно отсортировать информацию и играют важную роль в оптимизации использования места на диске. Таким образом, выбору типа данных для создаваемой таблицы необходимо уделить особое внимание.



### Совместимость типов данных

Типы данных и их названия являются одним из основных источников несовместимости в SQL. Основные типы данных обычно поддерживаются всеми СУБД, в отличие от некоторых расширенных типов. Более того, иногда вы будете сталкиваться с тем фактом, что один и тот же тип данных в разных СУБД называется по-разному. К сожалению, с этим ничего нельзя поделать, но помнить об этом при создании схем таблиц необходимо.

## Строки

Данные в таблице хранятся в строках; каждая запись хранится в своей строке. Возвращаясь к сравнению с сеткой, можно сказать, что ее вертикальные столбцы являются столбцами таблицы, а горизонтальные строки — строками таблицы.

Например, в таблице клиентов информация о каждом клиенте хранится в отдельной строке. Число строк в таблице равно числу записей о клиентах.



### Строка

Запись в таблице.



### Записи или строки?

Часто пользователи баз данных упоминают о *записях*, имея в виду *строки*. Обычно эти два термина взаимозаменяемы, но термин *строка* технически более правилен.

## Первичные ключи

В каждой строке таблицы должно быть несколько столбцов, которые уникальным образом идентифицируют ее. В таблице с клиентами для этого может использоваться столбец с номером клиента, тогда как в таблице, содержащей заказы, таким столбцом может быть идентификатор заказа. В таблице со списком служащих может использоваться номер служащего или столбец с номерами карточек социального страхования.



### Первичный ключ

Столбец (или набор столбцов), значения которого уникально идентифицируют каждую строку таблицы.

Этот столбец (или набор столбцов), уникально идентифицирующий каждую строку таблицы, называется *первичный ключ*. Первичный ключ используется для определения конкретной строки. Без него выполнять обновление или удаление строк таблицы было бы очень затруднительно, так как не было бы никакой гарантии, что мы изменяем нужные строки.



### Всегда определяйте первичные ключи

Несмотря на то что первичные ключи не обязательны, большинство разработчиков баз данных создают их для каждой таблицы, чтобы в будущем манипулирование и управление данными ничем не усложнялось.

Любой столбец таблицы может быть использован в качестве первичного ключа, если выполняются следующие условия.

- Две разные строки не могут иметь одно и то же значение первичного ключа.
- Каждая строка должна иметь определенное значение первичного ключа (столбцы первичного ключа не могут иметь значения NULL).
- Значения в столбце первичного ключа не могут быть изменены.
- Значения первичного ключа нельзя использовать дважды. (Если строка удалена из таблицы, ее первичный ключ нельзя в дальнейшем назначать другим строкам.)

В качестве первичного ключа обычно используется только один столбец таблицы. Но это требование не обязательно и в качестве первичного ключа можно использовать несколько столбцов. При этом правила, приведенные выше, должны выполняться для всех столбцов, используемых в качестве первичного ключа, а все их значения должны быть уникальными (в обычных столбцах значения могут повторяться).

Существует еще один важный тип ключа, который называется *внешний ключ*, но к нему мы вернемся в уроке 12, “Объединение таблиц”.

## Что такое SQL?

SQL — это аббревиатура выражения Structured Query Language (язык структурированных запросов). SQL был специально разработан для взаимодействия с базами данных.

В отличие от других языков (разговорных, таких как английский, или языков программирования, например Java



или Visual Basic), SQL состоит всего из нескольких слов. И сделано это умышленно. SQL был создан для решения одной задачи, с которой он вполне справляется, — предоставлять простой и эффективный способ считывания и записи информации в базу данных.

Каковы же преимущества SQL?

- SQL не относится к числу патентованных языков, используемых разработчиками определенных баз данных. Почти все большие СУБД поддерживают SQL, поэтому знание этого языка позволит вам взаимодействовать практически с любой базой данных.
- SQL легко изучить. Его немногочисленные операторы состоят из простых английских слов.
- Несмотря на кажущуюся простоту, SQL является очень мощным языком; разумно пользуясь его элементами, можно выполнять очень сложные операции с базами данных.

Именно поэтому стоит изучить SQL.



### Расширения SQL

Многие разработчики СУБД расширили возможности SQL, введя в язык дополнительные операторы или инструкции. Эти расширения необходимы для выполнения дополнительных функций или для упрощения выполнения определенных операций. И хотя часто они очень полезны, эти расширения привязаны к определенной СУБД и редко поддерживаются более чем одним разработчиком.

Стандартный SQL поддерживается комитетом стандартов ANSI, и соответственно называется ANSI SQL. Все крупные СУБД и даже те, у которых есть собственные расширения, поддерживают ANSI SQL. Отдельные же реализации носят собственные имена (PL-SQL, Transact-SQL и т.д.).

Чаще всего в этой книге упоминается именно ANSI SQL. В редких случаях, когда используется SQL, относящийся к определенной СУБД, об этом говорится отдельно.

## Попробуйте сами

Подобно изучению любого другого языка, чтобы изучить SQL, лучше всего попробовать его использовать на практике. Для этого вам понадобится база данных и приложение, из которого можно выполнять SQL-запросы.

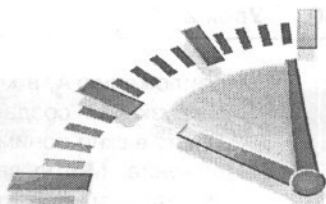
Во всех уроках этой книги используются настоящие SQL-операторы и настоящие таблицы базы данных. В приложении А, “Сценарии демонстрационных таблиц”, описываются примеры таблиц и приводятся советы по их получению (или созданию), чтобы можно было выполнять инструкции каждого урока. В приложении Б, “Работа с популярными приложениями”, описываются действия, необходимые для запуска SQL в различных приложениях. Перед тем как перейти к следующему уроку, прочитайте эти два приложения, чтобы подготовиться к дальнейшим действиям.

## Резюме

Из первого урока вы узнали, что такое SQL и чем он полезен. В связи с тем что SQL используется для взаимодействия с базами данных, мы также рассмотрели некоторые основные термины баз данных.



## Урок 2



# Выборка данных

На этом уроке вы узнаете, как использовать оператор *SELECT* для выборки одного или нескольких столбцов данных из таблицы.

## Оператор *SELECT*

Как уже говорилось в уроке 1, “Что такое SQL”, SQL-операторы являются обычными английскими терминами. Эти термины называются *ключевыми словами*, и каждый SQL-оператор состоит из одного или нескольких ключевых слов. Наиболее часто вы будете использовать оператор *SELECT*. Он предназначен для выборки информации из таблиц.



### Ключевое слово

Зарезервированное слово, являющееся частью языка SQL. Никогда не называйте таблицу или столбец таким словом. В приложении Д “Зарезервированные слова SQL” перечислены некоторые из наиболее часто используемых ключевых слов.

Чтобы при помощи оператора *SELECT* извлечь данные из таблицы, нужно указать как минимум две вещи — что вы хотите выбрать и откуда.



### Рассматриваемые примеры

В примерах SQL-операторов (а также полученных с их помощью результатов) в этой книге используются файлы данных, описанные в приложении А, “Сценарии демонстрационных таблиц”. Если вы хотите самостоятельно выполнить действия, указанные в примерах (очень рекомендуем это делать), обратитесь к

приложению А, в котором вы найдете инструкции по загрузке или созданию этих файлов.

Очень важно понимать, что SQL — это язык, а не приложение. Метод ввода SQL-операторов и вывод результатов их выполнения различен для разных приложений. Чтобы помочь вам приспособить примеры к вашей СУБД, в приложении Б, “Работа с популярными приложениями” объясняется, как выполнять команды, приведенные в этой книге, в нескольких популярных программах и средах разработки. А если вам нужно приложение, которое поможет вам выполнять примеры, там же вы найдете рекомендации по его выбору.

## Выборка отдельных столбцов

Начнем с простого SQL-оператора SELECT:

### ВВОД

```
SELECT prod_name
FROM Products;
```

### Анализ

В приведенном выше операторе используется оператор SELECT для выборки одного столбца под названием prod\_name из таблицы Products. Искомое имя столбца указывается сразу после ключевого слова SELECT, а ключевое слово FROM указывает на имя таблицы, из которой выбираются данные. Результат выполнения этого оператора будет следующий:

### ВЫВОД

```
prod_name
-----
Fish bean bag toy
Bird bean bag toy
Rabbit bean bag toy
8 inch teddy bear
12 inch teddy bear
18 inch teddy bear
Raggedy Ann
King doll
Queen doll
```



### Неотсортированные данные

Если вы попробовали выполнить этот запрос самостоятельно, то заметили, что данные были отображены в ином порядке. В этом случае не нужно волноваться — так и должно быть. Если результаты запроса не отсортированы явным образом (это мы обсудим в следующем уроке), то данные будут возвращены без особого порядка. Это может быть порядок, в котором данные были занесены в таблицу, или какой-либо другой порядок. Главное, чтобы ваш запрос возвращал одно и то же число строк.

Простой оператор `SELECT`, который использовался в предыдущем примере, возвращает все строки таблицы. Данные не фильтруются (как это делается при возвращении подмножества данных) и не сортируются. Эту тему мы обсудим в следующих нескольких уроках.



### Используйте пробелы

Все лишние пробелы в SQL-операторе при обработке пропускаются. Поэтому SQL-оператор может быть записан как в одной длинной строке, так и разбит на несколько строк. Большинство SQL-разработчиков разбивают операторы на несколько строк, чтобы их было легче читать и отлаживать.



### Завершение операторов

Несколько SQL-операторов должны быть разделены точкой с запятой (символом `;`). В большинстве СУБД не требуется вставлять точку с запятой после единственного оператора, но если в вашем конкретном случае СУБД выдает ошибку, вам придется это делать. Несомненно, при желании можно всегда добавлять точку с запятой, она никому не будет мешать, даже если этот символ не обязателен. Исключением является СУБД Sybase Adaptive Server, которая “не любит” SQL-операторы, заканчивающиеся символом `;`.

### SQL-операторы и регистр

Важно отметить, что SQL-операторы нечувствительны к регистру, поэтому операторы `SELECT`, `select` и `Select` эквивалентны. Многие SQL-разработчики используют верхний регистр для всех ключевых слов SQL и нижний регистр для имен столбцов и таблиц, чтобы код легче читался. Однако будьте внимательны: SQL-операторы не зависят от регистра, в отличие от имен таблиц, столбцов и значений (которые зависят от СУБД и ее конфигурации).

## Выборка нескольких столбцов

Для выборки из таблицы нескольких столбцов используется тот же оператор `SELECT`. Отличие состоит в том, что после ключевого слова `SELECT` необходимо через запятую указать несколько имен столбцов.



### Будьте внимательны с запятыми

При перечислении нескольких столбцов вставляйте между ними запятые, но не после последнего столбца в списке. Это приведет к ошибке.

В следующем операторе `SELECT` из таблицы `Products` выбираются три столбца:

### ВВОД

```
SELECT prod_id, prod_name, prod_price  
FROM Products;
```

### Анализ

Как и в предыдущем примере, в этом операторе для выборки данных из таблицы `Products` используется оператор `SELECT`. В этом примере перечислены три имени столбца, разделенные запятыми. Результат обработки этого оператора показан ниже:

**Вывод**

| prod_id | prod_name           | prod_price |
|---------|---------------------|------------|
| BNMG01  | Fish bean bag toy   | 3.4900     |
| BNMG02  | Bird bean bag toy   | 3.4900     |
| BNMG03  | Rabbit bean bag toy | 3.4900     |
| BR01    | 8 inch teddy bear   | 5.9900     |
| BR02    | 12 inch teddy bear  | 8.9900     |
| BR03    | 18 inch teddy bear  | 11.9900    |
| RGAN01  | Raggedy Ann         | 4.9900     |
| RYL01   | King doll           | 9.4900     |
| RYL02   | Queen doll          | 9.4900     |

**Представление данных**

Как видно из предыдущего результата, SQL-операторы обычно возвращают “сырые”, неотформатированные данные. Форматирование данных является проблемой представления, а не выборки. Поэтому представление (например, отображение приведенных выше цен в виде определенной суммы с правильно расставленными десятичными запятыми) обычно зависит от приложения, посредством которого отображаются данные. Просто выбранные данные (без форматирования) используются редко.

**Выборка всех столбцов**

Помимо возможности осуществлять выборку определенных столбцов (одного или нескольких), при помощи оператора `SELECT` можно запросить все столбцы, не перечисляя каждый из них. Для этого вместо имен столбцов вставляется групповой символ “звездочка” (\*). Это делается следующим образом.

**Ввод**

```
SELECT *
FROM Products;
```



## Анализ

При указании группового символа (\*) возвращаются все столбцы. Столбцы обычно (но не всегда) возвращаются в том порядке, в котором они находились при создании таблицы. Однако SQL-данные редко выводятся в том виде, в каком они хранятся в базе данных. (Обычно они возвращаются в приложение, которое необходимым образом их форматирует.)



### Использование групповых символов

Лучше не использовать групповой символ \* (кроме тех случаев, когда вам действительно необходимы все столбцы таблицы). Хотя групповые символы могут сэкономить вам время и усилия, необходимые для перечисления необходимых столбцов, выборка ненужных столбцов обычно снижает производительность запроса и приложения в целом.



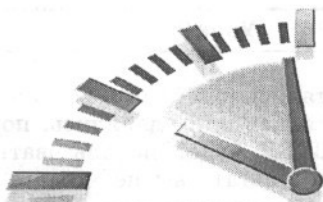
### Выборка неизвестных столбцов

Есть одно большое преимущество в использовании групповых символов. Поскольку вы не указываете точные имена столбцов (так как при использовании символа "звездочка" возвращаются все столбцы), появляется возможность выбрать столбцы, имена которых неизвестны.

## Резюме

В этом уроке мы рассмотрели порядок использования SQL-оператора SELECT для выборки одного, нескольких и всех столбцов таблицы. Далее мы научимся сортировать данные, полученные в результате выборки.

## Урок 3



# Сортировка выбранных данных

На этом уроке вы узнаете, как использовать предложение *ORDER BY* оператора *SELECT* для сортировки полученных в результате выборки данных.

## Сортировка данных

Из последнего урока вы узнали, что следующий SQL-оператор возвращает один столбец из таблицы базы данных. Но взгляните на результат: данные выводятся в полном беспорядке.

### ВВОД

```
SELECT prod_name  
FROM Products;
```

### ВЫВОД

```
prod_name  
-----  
Fish bean bag toy  
Bird bean bag toy  
Rabbit bean bag toy  
8 inch teddy bear  
12 inch teddy bear  
18 inch teddy bear  
Raggedy Ann  
King doll  
Queen doll
```

Вообще-то выбранные данные отображаются не в полном беспорядке. При отсутствии сортировки данные обычно выводятся в том порядке, в котором они находятся в таблице. Это может быть порядок, в котором они изначально добав-

лялись в таблицу. Однако если данные впоследствии обновлялись или удалялись, порядок будет зависеть от того, как СУБД будет использовать оставшееся свободное место. В результате вы не можете (и не должны) полагаться на порядок сортировки, если вы не контролируете его. В теории реляционных баз данных говорится, что последовательность выбранных данных не имеет смысла, если не был специально указан порядок сортировки.



### Предложение

SQL-операторы завершаются предложениями, одни из которых обязательны, другие — нет. Предложение обычно состоит из ключевого слова и предоставляемых данных. Примером может служить предложение FROM оператора SELECT, которое мы использовали в предыдущем уроке.

Для точной сортировки выбранных при помощи оператора SELECT данных используется предложение ORDER BY. В этом предложении указывается имя одного или нескольких столбцов, по которым и сортируются результаты. Взгляните на следующий пример:

### ВВОД

```
SELECT prod_name
FROM Products
ORDER BY prod_name;
```

### Анализ

Это выражение идентично предыдущему, за исключением предложения ORDER BY, которое указывает системе управления базой данных отсортировать данные в алфавитном порядке по столбцу prod\_name. Результат применения этого выражения будет следующим:

### ВЫВОД

```
prod_name
-----
12 inch teddy bear
18 inch teddy bear
8 inch teddy bear
Bird bean bag toy
```

Fish bean bag toy  
 King doll  
 Queen doll  
 Rabbit bean bag toy  
 Raggedy Ann



### Местоположение предложения ORDER BY

При использовании предложения ORDER BY убедитесь, что оно указано последним в операторе SELECT. Использование предложений в неправильном порядке ведет к появлению сообщений об ошибках.



### Сортировка по невыбранным столбцам

Чаще всего столбцы, используемые в предложении ORDER BY, отображаются на экране. Но это не всегда бывает так, данные могут сортироваться и по столбцу, который не выбирается этим запросом.

## Сортировка по нескольким столбцам

Часто бывает необходимо отсортировать данные по нескольким столбцам. Например, если вы выводите список служащих, вам может понадобиться отсортировать его по имени и фамилии сотрудника (сначала по фамилии, а затем с каждой фамилией по имени). Это может быть полезным, если в компании есть несколько служащих с одинаковыми фамилиями.

Чтобы осуществить сортировку по нескольким столбцам, просто укажите их имена через запятую (так, как вы делали при простом перечислении столбцов).

В следующем коде выбираются три столбца, а результат сортируется по двум из них — сначала по цене, а потом по названию.

### ВВОД

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY prod_price, prod_name;
```

**Вывод**

| prod_id | prod_price | prod_name           |
|---------|------------|---------------------|
| BNBG02  | 3.4900     | Bird bean bag toy   |
| BNBG01  | 3.4900     | Fish bean bag toy   |
| BNBG03  | 3.4900     | Rabbit bean bag toy |
| RGAN01  | 4.9900     | Raggedy Ann         |
| BR01    | 5.9900     | 8 inch teddy bear   |
| BR02    | 8.9900     | 12 inch teddy bear  |
| RYL01   | 9.4900     | King doll           |
| RYL02   | 9.4900     | Queen doll          |
| BR03    | 11.9900    | 18 inch teddy bear  |

Важно понимать, что при сортировке по нескольким столбцам порядок сортировки будет таким, который указан в запросе. Другими словами, в примере, приведенном выше, продукция сортируется по столбцу `prod_name`, только если существует несколько строк с одинаковыми значениями `prod_price`. Если никакие значения столбца `prod_price` не совпадают, данные по столбцу `prod_name` сортироваться не будут.

## Сортировка по положению столбца

Порядок сортировки можно указать не только по именам столбцов, но и по относительному положению столбца (проще говоря — по номеру столбца). Чтобы лучше понять это, рассмотрим пример:

**Ввод**

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY 2, 3;
```

**Вывод**

| prod_id | prod_price | prod_name           |
|---------|------------|---------------------|
| BNBG02  | 3.4900     | Bird bean bag toy   |
| BNBG01  | 3.4900     | Fish bean bag toy   |
| BNBG03  | 3.4900     | Rabbit bean bag toy |
| RGAN01  | 4.9900     | Raggedy Ann         |
| BR01    | 5.9900     | 8 inch teddy bear   |
| BR02    | 8.9900     | 12 inch teddy bear  |

|       |         |                    |
|-------|---------|--------------------|
| RYL01 | 9.4900  | King doll          |
| RYL02 | 9.4900  | Queen doll         |
| BR03  | 11.9900 | 18 inch teddy bear |

## Анализ

Как видите, результат выполнения запроса идентичен предыдущему примеру. Разница только в предложении ORDER BY. Здесь мы не указывали имена столбцов, вместо этого было оговорено их относительное положение в указанном списке SELECT. Предложение ORDER BY 2 означает сортировку по второму столбцу списка SELECT, а именно по столбцу prod\_price. Предложение ORDER BY 2, 3 означает сортировку по столбцу prod\_price, а затем по столбцу prod\_name.

Основное преимущество данного метода заключается в том, что не нужно несколько раз набирать в запросе имена столбцов. Однако имеются и недостатки. Во-первых, неконкретное перечисление столбцов повышает вероятность того, что вы случайно укажете не тот столбец. Во-вторых, можно случайно сменить порядок данных при изменении списка SELECT (при этом забыв внести соответствующие изменения в предложение ORDER BY). И наконец, очевидно, нельзя использовать этот метод для сортировки по столбцам, не указанным в списке SELECT.



### Сортировка по невыбранным столбцам

Очевидно, что этот метод нельзя использовать при сортировке по столбцам, не указанным в списке SELECT. Однако при необходимости можно в одном операторе указывать реальные имена столбцов и их относительные положения.

## Указание направления сортировки

Сортировка данных не ограничена порядком по возрастанию (от А до Я). Несмотря на то что этот порядок является порядком по умолчанию, в предложении ORDER BY также можно использовать порядок по убыванию (от Я до А). Для этого необходимо указать ключевое слово DESC.

В следующем примере продукция сортируется по цене в убывающем порядке (вначале идут самые дорогие товары).

**ВВОД**

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY prod_price DESC;
```

**ВЫВОД**

| prod_id | prod_price | prod_name           |
|---------|------------|---------------------|
| BR03    | 11.9900    | 18 inch teddy bear  |
| RYL01   | 9.4900     | King doll           |
| RYL02   | 9.4900     | Queen doll          |
| BR02    | 8.9900     | 12 inch teddy bear  |
| BR01    | 5.9900     | 8 inch teddy bear   |
| RGAN01  | 4.9900     | Raggedy Ann         |
| BNBG01  | 3.4900     | Fish bean bag toy   |
| BNBG02  | 3.4900     | Bird bean bag toy   |
| BNBG03  | 3.4900     | Rabbit bean bag toy |

Но что, если производится сортировка по нескольким столбцам? В следующем примере продукция сортируется по цене в убывающем порядке (вначале самые дорогие), плюс по названию продукта:

**ВВОД**

```
SELECT prod_id, prod_price, prod_name
FROM Products
ORDER BY prod_price DESC, prod_name;
```

**ВЫВОД**

| prod_id | prod_price | prod_name           |
|---------|------------|---------------------|
| BR03    | 11.9900    | 18 inch teddy bear  |
| RYL01   | 9.4900     | King doll           |
| RYL02   | 9.4900     | Queen doll          |
| BR02    | 8.9900     | 12 inch teddy bear  |
| BR01    | 5.9900     | 8 inch teddy bear   |
| RGAN01  | 4.9900     | Raggedy Ann         |
| BNBG02  | 3.4900     | Bird bean bag toy   |
| BNBG01  | 3.4900     | Fish bean bag toy   |
| BNBG03  | 3.4900     | Rabbit bean bag toy |

## Анализ

Ключевое слово DESC применяется только к тому столбцу, после которого оно указано. В предыдущем примере ключевое слово DESC было указано для столбца `prod_price`, но не для `prod_name`. Таким образом, столбец `prod_price` отсортирован в порядке убывания, а столбец `prod_name` в обычном, возрастающем порядке.



### Сортировка по убыванию по нескольким столбцам

Если вы хотите отсортировать данные в порядке убывания по нескольким столбцам, укажите для каждого из них ключевое слово DESC.

Следует упомянуть, что DESC — это сокращение от DESCENDING, можно использовать оба ключевых слова. Противоположным словом для DESC является ASC (ASCENDING), которое можно указывать для сортировки по возрастанию. Однако на практике слово ASC обычно не применяется, поскольку такой порядок используется по умолчанию (он предполагается, если не указано ни ASC, ни DESC).



### Чувствительность к регистру и порядок сортировки

При сортировке текстовых данных А это то же самое, что и а? И а идет перед Б или после я? Это не теоретические вопросы, ответ на них зависит от настройки базы данных.

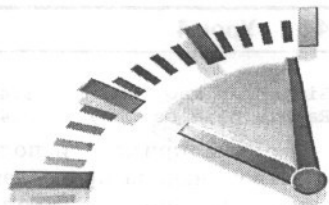
При *лексикографическом* порядке сортировки А считается идентичным а, и такое поведение является обычным для большинства систем управления базами данных. Однако в некоторых СУБД администратор может при необходимости это поведение изменить. (Это может оказаться полезным, если в вашей базе данных содержится много символов из другого языка.) Суть в том, что если вам понадобится альтернативный порядок сортировки, его нельзя будет достичь посредством обычного предложения ORDER BY. Вам придется обратиться к администратору базы данных.



## Резюме

Этот урок был посвящен сортировке выбранных данных при помощи предложения ORDER BY оператора SELECT. Это предложение, которое должно быть последним в операторе SELECT, можно использовать для сортировки данных по одному или нескольким столбцам.

## Урок 4



# Фильтрация данных

На этом уроке вы узнаете, как использовать предложение `WHERE` оператора `SELECT` для указания предложений поиска.

## Использование предложения `WHERE`

В таблицах баз данных обычно содержится очень много информации и довольно редко возникает необходимость выбирать все строки таблицы. Гораздо чаще бывает нужно извлечь какую-то часть данных таблицы для каких-либо действий или отчетов. Выборка только необходимых данных включает в себя *критерий поиска*, также известный под названием *предложение фильтрации*.

В операторе `SELECT` данные фильтруются путем указания критерия поиска в предложении `WHERE`. Предложение `WHERE` указывается сразу после названия таблицы (предложения `FROM`) следующим образом:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price = 3.49;
```

### Анализ

Этот оператор извлекает два столбца из таблицы товаров, но показывает не все строки, а только те, значение в столбце `prod_price` которых равно 3.49:

### ВЫВОД

| <code>prod_name</code> | <code>prod_price</code> |
|------------------------|-------------------------|
| Fish bean bag toy      | 3.4900                  |

|                     |        |
|---------------------|--------|
| Bird bean bag toy   | 3.4900 |
| Rabbit bean bag toy | 3.4900 |

В этом примере используется простая проверка на равенство: сначала проверяется, существует ли в столбце указанное значение, а затем данные фильтруются соответствующим образом. Однако SQL позволяет использовать не только проверку на равенство.



### Требовательная PostgreSQL

СУБД PostgreSQL имеет строгие правила, управляющие значениями, передающимися в SQL-операторы, особенно это касается чисел с десятичными дробями. Таким образом, предыдущий пример может и не работать в PostgreSQL. Чтобы он заработал, необходимо точно указать, что 3.49 — это «правильное» число, включив в предложение WHERE его тип. Для этого замените `= 3.49` на `= decimal '3.49'`.



### Фильтрация в SQL и в приложении

Данные также могут быть отфильтрованы на уровне приложения. Для этого посредством оператора SELECT осуществляется выборка большего количества данных, чем на самом деле необходимо для клиентского приложения, а затем клиентский код обрабатывает полученные данные для извлечения только нужных строк.

Как правило, этот метод не приветствуется. Базы данных оптимизированы для быстрой и эффективной фильтрации. Заставляя клиентское приложение выполнять работу базы данных, вы значительно ухудшаете его производительность, а также затрудняете его корректное масштабирование. Кроме того, если данные фильтруются у клиента, сервер отправляет ненужные данные по сети, тем самым занимая лишнюю полосу канала.

**Положение предложения WHERE**

При использовании обоих предложений, ORDER BY и WHERE, убедитесь, что предложение ORDER BY следует за предложением WHERE, иначе возникнет ошибка. (Более подробно предложение ORDER BY описывается в уроке 3.)

## Операции в предложении WHERE

В первом предложении WHERE, которое мы рассмотрели, проводилась проверка на равенство, т.е. определялось, содержится ли в столбце указанное значение. SQL поддерживает весь спектр условных (логических) операций, которые приведены в табл. 4.1.

**Таблица 4.1. Операции в предложении WHERE**

| <i>Операция</i> | <i>Описание</i>                   |
|-----------------|-----------------------------------|
| =               | Равенство                         |
| <>              | Неравенство                       |
| !=              | Неравенство                       |
| <               | Меньше                            |
| <=              | Меньше или равно                  |
| !<              | Не меньше                         |
| >               | Больше                            |
| >=              | Больше или равно                  |
| !>              | Не больше                         |
| BETWEEN         | Между двумя указанными значениями |
| IS NULL         | Значение NULL                     |

**Совместимость операций**

Некоторые из операций, приведенных в табл. 4.1, повторяются (например, <> — это то же самое, что и !=). Выполнение операции !< (не меньше чем) дает такой

же результат, что и  $\geq$  (больше или равно). Однако заметьте: не все из этих операций поддерживаются всеми СУБД. Обратитесь к документации вашей СУБД, чтобы точно знать, какие логические операции она поддерживает.

## Проверка одного значения

Мы рассмотрели пример проверки на равенство. Теперь рассмотрим примеры использования других операций.

В первом примере выводятся названия товаров, стоимость которых не превышает \$10:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price < 10;
```

### ВЫВОД

| prod_name           | prod_price |
|---------------------|------------|
| Fish bean bag toy   | 3.4900     |
| Bird bean bag toy   | 3.4900     |
| Rabbit bean bag toy | 3.4900     |
| 8 inch teddy bear   | 5.9900     |
| 12 inch teddy bear  | 8.9900     |
| Raggedy Ann         | 4.9900     |
| King doll           | 9.4900     |
| Queen doll          | 9.4900     |

В следующем выражении выбираются все товары, которые стоят \$10 и меньше (результат будет такой же, как и в первом примере, так как в базе данных нет товаров, которые бы стоили ровно \$10):

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price <= 10;
```

## Проверка на несовпадения

В этом примере выводятся товары, не изготовленные фирмой DLL01.

**ВВОД**

```
SELECT vend_id, prod_price
FROM Products
WHERE vend_id <> 'DLL01';
```

**ВЫВОД**

| vend_id | prod_name          |
|---------|--------------------|
| BRS01   | 8 inch teddy bear  |
| BRS01   | 12 inch teddy bear |
| BRS01   | 18 inch teddy bear |
| FNG01   | King doll          |
| FNG01   | Queen doll         |

**Когда использовать кавычки**

Если вы внимательно рассмотрите выражения в предыдущих предложениях WHERE, то заметите, что некоторые значения заключены в одинарные кавычки, а некоторые — нет. Одинарные кавычки используются для определения границ строки. При сравнении значения со столбцом, содержащим строковые данные, необходимы отделяющие строку кавычки. При использовании числовых столбцов кавычки не используются.

Ниже приведен тот же пример, только здесь уже используется операция !=, вместо <>:

**ВВОД**

```
SELECT vend_id, prod_price
FROM Products
WHERE vend_id != 'DLL01';
```

**Операция != или <>**

Операции != и <> обычно взаимозаменяемы. Однако не во всех СУБД поддерживаются обе формы операции неравенства. Например, в Microsoft Access поддерживается операция <> и не поддерживается !=. Если у вас возникли сомнения по поводу своей СУБД, обратитесь к ее документации.

## Проверка на диапазон значений

Для поиска диапазона значений можно использовать операцию BETWEEN. Ее синтаксис немного отличается от других операций предложения WHERE, так как для нее требуются два значения: начальное и конечное. Например, операцию BETWEEN можно использовать для поиска товаров, цена которых находится в промежутке между \$5 и \$10, или всех дней, которые попадают в диапазон между указанными начальным и конечным числами.

В следующем примере демонстрируется использование операции BETWEEN для выборки всех товаров, цена которых выше \$5 и ниже \$10:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE prod_price BETWEEN 5 AND 10;
```

### ВЫВОД

| prod_name          | prod_price |
|--------------------|------------|
| -----              | -----      |
| 8 inch teddy bear  | 5.9900     |
| 12 inch teddy bear | 8.9900     |
| King doll          | 9.4900     |
| Queen doll         | 9.4900     |

### Анализ

Как видно из этого примера, при использовании операции BETWEEN нужно указывать два значения — меньшее и большее из выбранного диапазона. Эти два значения должны быть разделены ключевым словом AND. При этом выбираются все значения из диапазона, включая указанные начальное и конечное значения.

## Проверка на отсутствие значения

После создания таблицы разработчик может указать, допустимо ли, чтобы в отдельных ее столбцах не содержались никакие значения. Когда в столбце не содержится никакого значения, это значит, что в нем содержится значение NULL.

**NULL**

Отсутствие какого-либо значения, в отличие от поля, содержащего или 0, или пустую строку, или просто несколько пробелов.

Для оператора `SELECT` предусмотрена специальная форма предложения `WHERE`, которая используется для проверки значений `NULL` в столбцах и содержит проверку `IS NULL`. Синтаксис выглядит следующим образом:

**ВВОД**

```
SELECT prod_name
FROM Products
WHERE prod_price IS NULL;
```

Это выражение возвращает список товаров без цены (поле `prod_price` пустое, а не с ценой 0), а поскольку таких нет, никаких данных мы не получим. Однако в таблице `Vendors` есть столбцы со значениями `NULL` — в столбце `vend_state` будет содержаться `NULL`, если не указан никакой штат (в случае, когда адресат находится за пределами Соединенных Штатов):

**ВВОД**

```
SELECT vend_id
FROM Vendors
WHERE vend_state IS NULL;
```

**ВЫВОД**

```
vend_id
-----
FNG1
JTS01
```

**Особые операции СУБД**

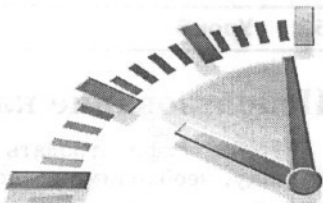
Во многих СУБД набор операций расширен дополнительными фильтрами. Обратитесь к документации вашей СУБД за дополнительной информацией.



## Резюме

В этом уроке рассказывалось о том, как отфильтровывать возвращаемые данные при помощи предложения WHERE оператора SELECT. Теперь вы знаете, как можно проверить данные на равенство, неравенство, наличие значений больше чем и меньше чем, диапазон значений, а также на значение NULL.

## Урок 5



# Расширенная фильтрация данных

В этом уроке вы узнаете, как можно комбинировать предложения *WHERE* для создания мощных и сложных условий поиска. Вы также узнаете, как следует использовать ключевые слова *NOT* и *IN*.

## Комбинирование предложений *WHERE*

Все предложения *WHERE*, представленные в уроке 4, “Фильтрация данных”, отфильтровывают данные с использованием одного критерия. Чтобы увеличить уровень контроля над фильтром в SQL, можно использовать несколько предложений *WHERE*. Эти предложения допустимо использовать двумя способами: в виде предложений *AND* или *OR*.



### Оператор

Специальное ключевое слово, используемое для объединения или изменения предложений внутри предложения *WHERE*. Также известны под названием *логические операторы*<sup>1</sup>.

<sup>1</sup> Термин *statement* в русскоязычной литературе по языку SQL принято переводить как *оператор* (и мы следовали этой традиции во всех предыдущих уроках), хотя точнее его следовало бы переводить как *инструкция*. В данном случае речь идет именно об операторах (*operator*). — Прим. ред.

## Использование ключевого слова AND

Чтобы отфильтровать данные по более чем одному столбцу, необходимо воспользоваться ключевым словом AND для добавления предложений в предложение WHERE. Вот как это делается:

### ВВОД

```
SELECT prod_id, prod_price, prod_name
FROM Products
WHERE vend_id = 'DLL01' AND prod_price <=4;
```

### Анализ

Посредством данного оператора извлекается название продукции и цена для всех товаров, изготовленных производителем DLL01, с ценой \$4 и меньше. Предложение WHERE в операторе SELECT состоит из двух предложений, а ключевое слово AND используется для их объединения. Ключевое слово AND указывает системе управления базой данных возвращать только те строки, которые удовлетворяют всем перечисленным предложениям. Если продукт изготовлен производителем DLL01, но стоит больше \$4, он не попадет в результаты. Аналогично, товары, которые стоят меньше \$4 и изготовлены отличными от указанного производителями, также не будут выведены. Данные, выданные в результате выполнения этой SQL-инструкции, будут выглядеть так:

### ВЫВОД

| Prod_id | prod_price | prod_name           |
|---------|------------|---------------------|
| -----   | -----      | -----               |
| BNBG02  | 3.4900     | Bird bean bag toy   |
| BNBG01  | 3.4900     | Fish bean bag toy   |
| BNBG03  | 3.4900     | Rabbit bean bag toy |



#### AND

Ключевое слово, используемое в предложении WHERE для того, чтобы возвращались только те строки, которые удовлетворяют всем указанным предложениям.

## Использование ключевого слова OR

Действие ключевого слова OR противоположно действию ключевого слова AND. Ключевое слово OR указывает системе управления базой данных выбирать только те строки, которые удовлетворяют хотя бы одному предложению. На самом деле в большинстве лучших СУБД второе предложение даже не рассматривается в предложении OR WHERE, если удовлетворено первое предложение. (Если первое предложение выполнено, строка будет выведена независимо от второго предложения.)

Взгляните на следующий оператор SELECT:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';
```

### Анализ

Посредством этого SQL-оператора выбираются названия товаров и их цены для всех продуктов, изготовленных одним из указанных производителей. Ключевое слово OR указывает СУБД использовать какое-то одно предложение, а не сразу два. Если бы здесь использовалось ключевое слово AND, мы бы не получили никаких данных. После выполнения этого SQL-запроса мы получим следующие данные:

### ВЫВОД

| prod_name           | prod_price |
|---------------------|------------|
| -----               | -----      |
| Fish bean bag toy   | 3.4900     |
| Bird bean bag toy   | 3.4900     |
| Rabbit bean bag toy | 3.4900     |
| 8 inch teddy bear   | 5.9900     |
| 12 inch teddy bear  | 8.9900     |
| 18 inch teddy bear  | 11.9900    |
| Raggedy Ann         | 4.9900     |



### OR

Ключевое слово, применяемое в предложении WHERE для того, чтобы возвращались все строки, удовлетворяющие любому из указанных предложений.

## Порядок обработки

Предложения WHERE могут содержать любое количество логических операторов AND и OR. Комбинируя их, можно создавать сложные фильтры.

Однако при комбинировании ключевых слов AND и OR возникает одна проблема. Рассмотрим следующий пример. Необходимо вывести список всех изготовленных производителями DLL01 и BRS01 товаров, цена которых \$10 и выше. В следующей инструкции SELECT используется комбинация ключевых слов AND и OR для формулирования предложения WHERE:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01'
      AND prod_price >= 10;
```

### ВЫВОД

| prod_name           | prod_price |
|---------------------|------------|
| Fish bean bag toy   | 3.4900     |
| Bird bean bag toy   | 3.4900     |
| Rabbit bean bag toy | 3.4900     |
| 18 inch teddy bear  | 11.9900    |
| Raggedy Ann         | 4.9900     |

### Анализ

Взгляните на результат. В четырех возвращенных строках значатся цены ниже \$10 — очевидно, строки не были отфильтрованы так, как надо. Что же произошло? Причина в порядке обработки. SQL (как и большинство других языков) вначале обрабатывает логические операторы AND, а потом уже логические операторы OR. Когда SQL “видит” такое предложение WHERE, он его считывает так: *выбрать все продукты, которые стоят \$10 и больше, изготовленные производителем BRS01, и все продукты, изготовленные производителем DLL01 независимо от их цены*. Другими словами, так как приоритет у логического оператора AND выше, были объединены “не те” операторы.

Решение этой проблемы состоит в использовании скобок для точного группирования необходимых логических опе-

раторов. Взгляните на следующий оператор SELECT и его выходные данные:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE (vend_id = 'DLL01' OR vend_id = 'BRS01')
      AND prod_price >= 10;
```

### ВЫВОД

| prod_name          | prod_price |
|--------------------|------------|
| -----              | -----      |
| 18 inch teddy bear | 11.9900    |

### Анализ

Единственным отличием между предыдущим выражением и этим являются скобки, в которые заключены первые два предложения оператора WHERE. Поскольку скобки имеют еще больший приоритет, чем логические операторы AND и OR, СУБД вначале обрабатывает условие OR внутри скобок. Соответственно, SQL-оператор будет пониматься так: *выбрать все продукты, изготовленные либо производителем DLL01, либо производителем BRS01, которые стоят \$10 и больше*, а это именно то, что нужно.



#### Использование скобок в предложениях WHERE

Когда бы вы ни использовали предложения WHERE с ключевыми словами AND и OR, всегда вставляйте скобки, чтобы точно сгруппировать логические операторы. Не полагайтесь на порядок обработки по умолчанию, даже если он подразумевает необходимый вам результат. Нет никаких недостатков в использовании скобок, кроме того, вы всегда будете застрахованы от неопределенностей.

## Использование ключевого слова IN

Ключевое слово IN используется для указания диапазона условий, любое из которых может быть выполнено. При этом значения, заключенные в скобки, перечисляются через запятую. Рассмотрим следующий пример:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id IN ('DLL01', 'BRS01')
ORDER BY prod_name
```

### ВЫВОД

| prod_name           | prod_price |
|---------------------|------------|
| -----               | -----      |
| 12 inch teddy bear  | 8.9900     |
| 18 inch teddy bear  | 11.9900    |
| 8 inch teddy bear   | 5.9900     |
| Bird bean bag toy   | 3.4900     |
| Fish bean bag toy   | 3.4900     |
| Rabbit bean bag toy | 3.4900     |
| Raggedy Ann         | 4.9900     |

### Анализ

Инструкция SELECT осуществляет выборку всех товаров, изготовленных производителями DLL01 и BRS01. После ключевого слова IN следует список значений через запятую, а весь список заключен в скобки.

Если вы подумаете, что ключевое слово IN выполняет ту же функцию, что и OR, то будете совершенно правы. Следующий SQL-запрос выполняет ту же функцию, что и предыдущий:

### ВВОД

```
SELECT prod_name, prod_price
FROM Products
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01'
ORDER BY prod_name;
```

**Вывод**

| prod_name           | prod_price |
|---------------------|------------|
| 12 inch teddy bear  | 8.9900     |
| 18 inch teddy bear  | 11.9900    |
| 8 inch teddy bear   | 5.9900     |
| Bird bean bag toy   | 3.4900     |
| Fish bean bag toy   | 3.4900     |
| Rabbit bean bag toy | 3.4900     |
| Raggedy Ann         | 4.9900     |

Зачем же нужно ключевое слово `IN`? Его преимущества следующие.

- При работе с длинными списками необходимых значений синтаксис логического оператора `IN` гораздо легче читать.
- При использовании ключевого слова `IN` гораздо легче управлять порядком обработки (так как используется меньшее количество операторов).
- Логические операторы `IN` почти всегда быстрее обрабатываются, чем списки логических операторов `OR`.
- Самое большое преимущество логического оператора `IN` в том, что в данном операторе может содержаться еще одна инструкция `SELECT`, а это позволяет создавать очень динамичные предложения `WHERE`. Более подробно вы об этом узнаете в уроке 11, “Использование подзапросов”.

**IN**

Ключевое слово, используемое в предложении `WHERE` для указания списка значений, обрабатываемых так же, как это делается в случае применения ключевого слова `OR`.

## Использование ключевого слова `NOT`

Логический оператор `NOT` предложения `WHERE` служит для выполнения только одной функции — отрицать все



предложения, следующие за ним. Поскольку NOT никогда не используется сам по себе (а только вместе с другими логическими операторами), его синтаксис немного отличается от синтаксиса остальных операторов. В отличие от них, NOT вставляется перед названием столбца, значения которого нужно отфильтровать, а не после.

**NOT**

Ключевое слово, применяемое в предложении WHERE для отрицания какого-то условия.

В следующем примере демонстрируется использование логического оператора NOT. Чтобы извлечь список продуктов, изготовленных всеми производителями, кроме DLL01, можно потребовать выполнить следующее:

**ВВОД**

```
SELECT prod_name
FROM Products
WHERE NOT vend_id = 'DLL01'
ORDER BY prod_name;
```

**ВЫВОД**

```
prod_name
-----
12 inch teddy bear
18 inch teddy bear
8 inch teddy bear
King doll
Queen doll
```

**Анализ**

Здесь логический оператор NOT отрицает предложение, следующее за ним. Поэтому СУБД извлекает не те значения vend\_id, которые совпадают с DLL01, а все остальные.

Предыдущий запрос можно было также выполнить при помощи операции <>:

**ВВОД**

```
SELECT prod_name
FROM Products
WHERE vend_id <> 'DLL01'
ORDER BY prod_name;
```

## ВЫВОД

```
prod_name
-----
12 inch teddy bear
18 inch teddy bear
8 inch teddy bear
King doll
Queen doll
```

## Анализ

Зачем же использовать логический оператор NOT? Конечно, для таких простых предложений WHERE, какие мы здесь рассматриваем, этот оператор не обязателен. Он полезен в более сложных предложениях. Например, для нахождения всех строк, которые не совпадают со списком критериев, можно использовать логический оператор NOT в паре с ключевым словом IN.



### NOT в MySQL

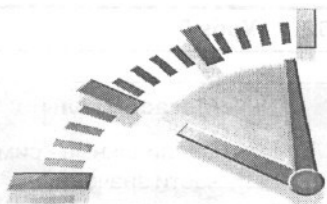
Форма логического оператора NOT, который здесь описывается, не поддерживается в СУБД MySQL. В MySQL NOT используется только для отрицания вхождений EXISTS (т.е. как NOT EXISTS).

## Резюме

В этом уроке вы узнали, как нужно комбинировать предложения WHERE с логическими операторами AND и OR. Вы также узнали, как следует управлять порядком обработки и как использовать ключевые слова IN и NOT.



## Урок 6



# Использование метасимволов для фильтрации

В этом уроке вы узнаете, что такое метасимволы, как их использовать и как выполнять поиск с применением метасимволов и логического оператора *LIKE* для фильтрации выводимых данных.

## Использование логического оператора *LIKE*

Все предыдущие операторы, которые мы рассмотрели, производили фильтрацию по известным значениям. Они искали совпадения по одному или нескольким значениям, более чем и менее чем известное значение или диапазон значений. При этом везде искалось известное значение. Однако фильтрация данных таким способом не всегда работает. Например, как бы вы искали продукты, в названии которых содержатся слова *bean bag*? Этого нельзя сделать при помощи простых операций сравнения, здесь на помощь приходит поиск с использованием метасимволов. При помощи метасимволов можно создавать условия поиска данных. В этом примере, для того чтобы найти все продукты, в названии которых содержатся слова *bean bag*, необходимо составить шаблон поиска, позволяющий найти текст *bean bag* в любом месте названия продукта.

**Метасимволы**

Специальные символы, применяемые для поиска части значения.

**Шаблон поиска**

Условие поиска, состоящее из текста, метасимволов и любой их комбинации.

Метасимволы сами по себе являются символами, которые имеют в условии WHERE специальное значение. В SQL поддерживаются метасимволы нескольких типов. Чтобы применять метасимволы в условиях поиска, необходимо использовать ключевое слово LIKE. Оно сообщает СУБД, что следующий шаблон для поиска необходимо сравнивать с использованием метасимволов, а не искать точные совпадения.

**Предикат**

Когда оператор не является оператором? Тогда, когда он является *предикатом*. Технически, LIKE — это предикат, а не оператор. Конечный результат остается тем же, просто не пугайтесь этого термина, если вы встретите его в документации по SQL.

Поиск с использованием метасимволов может осуществляться только в текстовых полях (строках), нельзя использовать метасимволы при поиске полей с нетекстовым типом данных.

**Метасимвол “знак процента” (%)**

Наиболее часто используемый метасимвол — знак процента (%). В строке поиска % означает *найти все вхождения любого символа*. Например, чтобы найти все продукты, названия которых начинаются со слова Fish, можно выполнить следующий запрос:

**ВВОД**

```
SELECT prod_id, prod_name
FROM Products
WHERE prod_name LIKE 'Fish%';
```

**ВЫВОД**

| prod_id | prod_name         |
|---------|-------------------|
| BNBG01  | Fish bean bag toy |

**Анализ**

В этом примере используется шаблон поиска 'Fish%'. При выполнении этого условия возвращаются все значения, которые начинаются с символов Fish. Знак % указывает СУБД принимать все символы после слова Fish независимо от их количества.

**Метасимволы Microsoft Access**

Если вы работаете в Microsoft Access, необходимо использовать символ \* вместо символа %.

**Зависимость от регистра**

Ваша СУБД и ее конфигурация могут влиять на то, что поиск будет зависеть от регистра. В этом случае по строке 'fish%' значение Fish bean bag toy не будет найдено.

Метасимволы можно использовать в любом месте шаблона поиска, причем в неограниченном количестве. В следующем примере используются два метасимвола, по одному на каждом конце шаблона.

**ВВОД**

```
SELECT prod_id, prod_name
FROM Products
WHERE prod_name LIKE '%bean bag%';
```

**ВЫВОД**

| prod_id | prod_name           |
|---------|---------------------|
| -----   | -----               |
| BNBG01  | Fish bean bag toy   |
| BNBG02  | Bird bean bag toy   |
| BNBG03  | Rabbit bean bag toy |

**Анализ**

Шаблон поиска '%bean bag%' означает *найти все значения, содержащие bean bag в любом месте названия, независимо от количества символов перед или после указанного текста.*

Метасимвол можно также использовать внутри шаблона поиска, хотя это редко бывает полезным. В следующем примере производится поиск всех продуктов, которые начинаются на F и заканчиваются на y:

**ВВОД**

```
SELECT prod_name
FROM Products
WHERE prod_name LIKE 'F%y';
```

Важно отметить, что помимо поиска одного или нескольких символов, знак % также означает и отсутствие символов в указанном месте шаблона поиска.

**Следите за замыкающими пробелами**

Многие СУБД, включая Microsoft Access, заполняют содержимое поля пробелами. Например, если столбец рассчитан на 50 символов, а в нем вставлен текст Fish bean bag toy (17 символов), то, чтобы заполнить столбец, в него может быть добавлено еще 33 пробела. Обычно это не влияет на данные или их использование, но может негативно отразиться на предыдущем SQL-выражении. По условию WHERE prod\_name LIKE 'F%y' будут найдены только те значения prod\_name, которые начинаются на F и заканчиваются на y. Если значение заполнено пробелами, оно не будет заканчиваться на y, и значение Fish bean bag toy не будет извлечено. Одним из простых решений может быть добавление второго

символа % в шаблон поиска: 'F%y%', после чего будут учитываться символы (пробелы) после буквы y. Но лучше "отрезать" пробелы при помощи функций, которые обсуждаются в уроке 8, "Использование функций манипулирования данными".

## Метасимвол "символ подчеркивания" ( \_ )

Еще одним полезным метасимволом является символ подчеркивания ( \_ ). Символ подчеркивания используется так же, как и %, но при этом учитывается не много символов, а только один.



### Метасимволы в Microsoft Access

Если вы работаете в Microsoft Access, вам нужно использовать знак ? вместо символа \_.

— Взгляните на этот пример.

### ВВОД

```
SELECT prod_id, prod_name
FROM Products
WHERE prod_name LIKE '__ inch teddy bear';
```



### Следите за замыкающими пробелами

Как и в предыдущем примере, возможно, понадобится добавить метасимвол % в шаблон, чтобы пример работал.

### ВЫВОД

| prod_id | prod_name          |
|---------|--------------------|
| BNBG02  | 12 inch teddy bear |
| BNBG03  | 18 inch teddy bear |

### Анализ

В шаблоне поиска этого предложения WHERE использованы два метасимвола, затем следует текст. В результате были выбраны только те строки, которые удовлетворяли шаб-



лону поиска: по двум символам подчеркивания было найдено число 12 в первой строке и 18 во второй. Продукт 8 inch teddy bear не был найден, так как в шаблоне поиска требуется два совпадения, а не одно. Для сравнения, в следующем выражении SELECT используется метасимвол %, вследствие чего извлекаются три названия товара:

### ВВОД

```
SELECT prod_id, prod_name
FROM Products
WHERE prod_name LIKE '% inch teddy bear';
```

### ВЫВОД

| prod_id | prod_name          |
|---------|--------------------|
| BNBG01  | 8 inch teddy bear  |
| BNBG02  | 12 inch teddy bear |
| BNBG03  | 18 inch teddy bear |

В отличие от знака %, который подразумевает также отсутствие символов, знак \_ всегда означает один символ — не более и не менее.

## Метасимвол “квадратные скобки” ([ ])

Метасимвол “квадратные скобки” ([ ]) используется для указания набора символов, каждый из которых должен совпадать со значением, причем точно в указанном месте (в местоположении метасимвола).



### Наборы не всегда поддерживаются

В отличие от метасимволов, описанных ранее, использование квадратных скобок для создания наборов многими СУБД не поддерживается. Наборы поддерживаются в СУБД Microsoft Access, Microsoft SQL Server и Sybase Adaptive Server. Обратитесь к документации по вашей СУБД, чтобы определить, поддерживаются ли в ней наборы.

Например, чтобы найти все контакты людей, имена которых начинаются на букву J или M, необходимо сделать следующее:

**ВВОД**

```
SELECT prod_id, prod_name
FROM Customers
WHERE cust_contact LIKE '[JM]%'
ORDER BY cust_contact
```

**ВЫВОД**

```
cust_contact
-----
```

```
Jim Jones
John Smith
Michelle Green
```


**Анализ**

Условие WHERE в этом выражении выглядит как '[JM]%' . В этом шаблоне поиска используются два разных метасимвола. По метасимволам [JM] производится поиск всех контактных лиц, имена которых начинаются на одну из указанных в скобках букв, но при этом учитывается только один символ. Поэтому все имена длиннее одного символа не будут извлечены. По метасимволу %, следующему после [JM], производится поиск любого количества символов после первой буквы, что и приводит к требуемому результату.

Можно использовать метасимвол, выполняющий противоположное действие, добавив перед ним символ ^ . Например, в следующем примере выбираются все имена, которые *не* начинаются с буквы J или M (в отличие от предыдущего примера):

**ВВОД**

```
SELECT prod_id, prod_name
FROM Customers
WHERE cust_contact LIKE '[^JM]%'
ORDER BY cust_contact
```

**Противоположные наборы в Microsoft Access**


Если вы работаете в Microsoft Access и требуется создать противоположный набор, необходимо использовать символ ! вместо ^, поэтому указывайте [!JM], а не [^JM].

Конечно, можно достичь того же результата, воспользовавшись логическим оператором NOT. Единственным преимуществом символа ^ является более простой синтаксис при выполнении нескольких предложений WHERE.



### Внимание!

Метасимвол ([ ]) поддерживается не всеми СУБД. Обратитесь к документации по вашей СУБД, чтобы определить, поддерживается ли этот метасимвол.

## Советы по использованию метасимволов

Как видите, метасимволы в SQL — это очень мощный механизм. Но за эту мощь приходится платить: поиск с использованием метасимволов требует больше времени на обработку, чем любые другие виды поиска, которые мы обсуждали ранее. Ниже приведены несколько советов по использованию метасимволов.

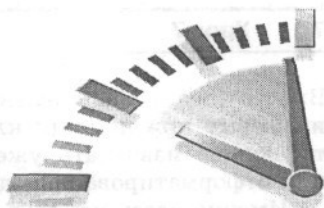
- Не злоупотребляйте метасимволами. Если можно использовать другой оператор поиска, воспользуйтесь им.
- При использовании метасимволов старайтесь по возможности не вставлять их в начало шаблона поиска. Шаблоны поиска, начинающиеся с метасимволов, обрабатываются медленнее всего.
- Внимательно следите за местоположением метасимволов. Если они находятся не на своем месте, будут извлечены не те данные.

Исходя из всего вышесказанного, можно заключить, что метасимволы очень важны и очень полезны при поиске — вы часто будете ими пользоваться.

## Резюме

В этом уроке рассказывалось о том, что такое метасимволы и как их использовать в условиях WHERE. Теперь вы знаете, что метасимволы нужно использовать осторожно, не следует злоупотреблять ими.

## Урок 7



# Создание вычисляемых полей

В этом уроке вы узнаете, что такое вычисляемые поля, как их создавать и как использовать псевдонимы для ссылки на такие поля из вашего приложения.

## Что такое вычисляемые поля

Данные, хранимые в таблицах базы данных, обычно выглядят представлены не в таком виде, который необходим для ваших приложений. Вот несколько примеров.

- Вам необходимо отобразить поле, содержащее имя компании с ее адресом, но эта информация расположена в разных столбцах таблицы.
- Город, штат и ZIP-код хранятся в отдельных столбцах (как и должно быть), но для программы печати почтовых наклеек необходима эта информация в одном, корректно сформированном поле.
- Данные в столбце введены с заглавными и строчными буквами, но в вашем отчете необходимо использовать только заглавные буквы.
- В таблице с предметами заказа хранятся цены продуктов и их количество, но не полная цена (цена одного продукта, умноженная на его количество) каждого продукта. Чтобы распечатать счет, необходимы полные цены.
- Вам необходимы общая сумма, среднее значение или результаты других расчетов, основанные на данных, имеющихся в таблице.

В каждом из этих примеров данные хранятся не в том виде, в котором их необходимо предоставить приложению.

Вместо того чтобы извлекать эти данные, а затем изменять их форму при помощи клиентского приложения или отчета, лучше извлекать уже преобразованные, подсчитанные или отформатированные данные прямо из базы данных.

Именно здесь помогут вычисляемые поля. В отличие от всех выбранных нами ранее столбцов, вычисляемых полей на самом деле в таблице базы данных нет. Они создаются “на лету” SQL-оператором SELECT.



### Поле

Изначально термин *поле* означал то же самое, что и *столбец*, и в основном эти понятия взаимозаменяемы, хотя столбцы базы данных обычно называют *столбцами*, а термин *поля* обычно используется по отношению к вычисляемым полям.

Важно отметить, что только база данных “знает”, какие столбцы в операторе SELECT являются реальными столбцами таблицы, а какие — вычисляемыми полями. С точки зрения клиента (например, вашего приложения), данные вычисляемого поля возвращаются точно так же, как и данные из любого другого столбца.



### Клиентское или серверное форматирование?

Многие преобразования и изменения форматов, которые могут быть выполнены посредством SQL-операторов, могут быть также выполнены и клиентским приложением. Однако, как правило, эти операции гораздо быстрее выполняются на сервере базы данных, чем у клиента, так как СУБД предназначена, кроме всего, для быстрого и эффективного выполнения операций такого типа.

## Конкатенация полей

Чтобы продемонстрировать работу вычисляемых полей, рассмотрим простой пример — создание заголовка, состоящего из двух столбцов.

В таблице `Vendors` содержится название поставщика и его адрес. Предположим, что вам необходимо создать отчет по поставщику и указать его адрес как часть его имени в виде *имя (адрес)*.

В отчете должно быть одно значение, а данные в таблице хранятся в двух столбцах: `vend_name` и `vend_country`. Кроме того, значение `vend_country` необходимо заключить в скобки, которых нет в таблице базы данных. Выражение `SELECT`, которое возвращает имена поставщиков и адреса, довольно простое, но как создать комбинированное значение?



### Конкатенация

Комбинирование значений (путем присоединения их друг к другу) для получения одного “длинного” значения.

Для этого необходимо соединить два значения. В SQL-выражении `SELECT` можно выполнить конкатенацию двух столбцов при помощи специального оператора. В зависимости от СУБД это может быть знак “плюс” (+) или две вертикальные черточки (||).



### Оператор + или ||

В СУБД Access, SQL Server и Sybase для конкатенации используется знак +. В СУБД DB2, Oracle, PostgreSQL и Sybase используется знак ||. Более подробную информацию ищите в документации по вашей СУБД. Вообще-то || — более предпочтительный оператор конкатенации, так что он поддерживается все большим и большим количеством СУБД.

Ниже приведен пример использования знака “плюс” (применяется синтаксис, принятый в большинстве СУБД):

### ВВОД

```
SELECT vend_name + ' (' + vend_country + ')'  
FROM Vendors  
ORDER BY vend_name;
```

### ВЫВОД

```
-----  
Bear Emporium                (USA                )
```

|                 |          |   |
|-----------------|----------|---|
| Beras R Us      | (USA     | ) |
| Doll House Inc. | (USA     | ) |
| Fun and Games   | (England | ) |
| Furball Inc.    | (USA     | ) |
| Jouets et ours  | (France  | ) |

Ниже приведена та же инструкция, но с использованием оператора ||:

### ВВОД

```
SELECT vend_name || ' (' || vend_country || ')'  
FROM Vendors  
ORDER BY vend_name;
```

### ВЫВОД

```
-----  
Bear Emporium          (USA      )  
Beras R Us             (USA      )  
Doll House Inc.        (USA      )  
Fun and Games          (England  )  
Furball Inc.           (USA      )  
Jouets et ours         (France   )
```

### Анализ

В предыдущих операторах SELECT была выполнена конкатенация следующих элементов:

- имя, хранящееся в столбце vend\_name;
- строка, содержащая пробел и открывающую круглую скобку;
- название штата, хранящееся в столбце vend\_country;
- строка, содержащая закрывающую круглую скобку.

Как видно из приведенного выше результата, выражение SELECT возвращает один столбец (вычисляемое поле), содержащий все четыре элемента как одно целое.



#### Конкатенация в MySQL

В MySQL не поддерживается конкатенация при помощи оператора + или ||. Здесь необходимо использовать функцию CONCAT(), в которой указывается список элементов, по отношению к которым необходимо выполнить конкатенацию. При использовании функции CONCAT() первая строка примера выглядела бы так:

```
SELECT CONCAT(vend_name, ' (' ,
vend_country, ')')
```

В MySQL поддерживается использование оператора `||`, но не для конкатенации. В MySQL `||` является эквивалентом логического оператора `OR`, а `&&` — эквивалентом логического оператора `AND`.

Взгляните еще раз на результат, полученный после применения оператора `SELECT`. Два столбца, объединенные в вычисляемое поле, заполнены пробелами. Во многих базах данных (но не во всех) сохраненный текст дополняется пробелами до ширины столбца. Чтобы выбрать правильно отформатированные данные, необходимо убрать добавленные пробелы. Это можно сделать при помощи SQL-функции `RTRIM()` следующим образом:

### ВВОД

```
SELECT RTRIM(vend_name) + ' (' + RTRIM(vend_country) + ') '
FROM Vendors
ORDER BY vend_name;
```

### ВЫВОД

```
-----
Bear Emporium (USA)
Beras R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)
```

Ниже приведено это же выражение, но с использованием оператора `||`:

### ВВОД

```
SELECT RTRIM(vend_name) || ' (' || RTRIM(vend_country) || ') '
FROM Vendors
ORDER BY vend_name;
```

### ВЫВОД

```
-----
Bear Emporium (USA)
Beras R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)
```



**Анализ**

Функция `RTRIM()` отбрасывает все пробелы справа от указанного значения. При использовании функции `RTRIM()` каждый отдельный столбец обрабатывается корректно. Город, штат указываются через запятую и пробел, а штат и ZIP-код — через пробел.

**Функции TRIM**

В большинстве СУБД поддерживаются как функция `RTRIM()` (которая, как мы увидели, “обрезает” правую часть строки), так и `LTRIM()` (которая удаляет левую часть строки), а также `TRIM()` (которая “обрезает” строку слева и справа).

**Использование псевдонимов**

Оператор `SELECT`, который использовался для конкатенации полей имени и адреса, как видите, справился со своей задачей. Но как же называется новый вычисляемый столбец? По правде говоря — никак, это просто значение. Этого может быть достаточно, если вы просматриваете результаты в программе тестирования SQL-запросов, однако столбец без названия нельзя использовать в клиентском приложении, так как клиент не сможет к нему обратиться.

Для решения этой проблемы в SQL была включена поддержка псевдонимов. Псевдоним — это альтернативное имя для поля или значения. Псевдонимы присваиваются при помощи ключевого слова `AS`. Взгляните на следующий оператор `SELECT`:

**ВВОД**

```
SELECT RTRIM(vend_name) + ' (' + RTRIM(vend_country) + ')'  
  AS vend_title  
FROM Vendors  
ORDER BY vend_name;
```

**ВЫВОД**

```
vend_title  
-----  
Bear Emporium (USA)
```

```
Beras R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)
```

Ниже приведена эта же инструкция, но с использованием оператора `||`:

## ВВОД

```
SELECT RTRIM(vend_name) || ' (' || RTRIM(vend_country) || ') '
  ⇨AS vend_title
FROM Vendors
ORDER BY vend_name;
```

## ВЫВОД

```
vend_title
-----
Bear Emporium (USA)
Beras R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)
```

## Анализ

Сам по себе этот оператор `SELECT` ничем не отличается от предыдущего, за исключением того, что вычисляемое поле указывается после текста `AS vend_title`. Таким образом, `SQL` создает вычисляемое поле, содержащее результат вычислений, под названием `vend_title`. Как видите, результат остается тем же, но столбец теперь носит имя `vend_title` и любое клиентское приложение может обращаться к нему по имени, как если бы это был реальный столбец таблицы.



### Другое использование псевдонимов

Псевдонимы можно использовать и по-другому. Часто псевдонимы используются для переименования столбца, если в реальном названии присутствуют недопустимые символы (например, пробелы) или если название сложное и трудночитаемое.



### Имена псевдонимов

Псевдонимом может служить как одно слово, так и целая строка. Если используется строка, она должна быть заключена в кавычки. В принципе, так делать можно, хотя и не рекомендуется. Многословные имена, несомненно, удобнее читать, но они создают множество проблем для многих клиентских приложений. Таким образом, наиболее часто псевдонимы используются для переименования многословных названий столбцов в однословные.



### Производные столбцы

Псевдонимы иногда называют “производные столбцы”, но, независимо от того, какой термин вы будете использовать, означают они одно и то же.

## Выполнение математических вычислений

Еще одним способом использования вычисляемых полей является выполнение математических операций над выбранными данными. Рассмотрим пример. В таблице `Orders` хранятся все полученные заказы, а в таблице `OrderItems` содержатся наименования продуктов для каждого заказа. Следующий SQL-оператор осуществляет выборку всех продуктов в заказе номер 20008:

### ВВОД

```
SELECT prod_id, quantity, item_price
FROM OrderItems
WHERE order_nam = 20008;
```

### ВЫВОД

| prod_id | quantity | item_price |
|---------|----------|------------|
| RGAN01  | 5        | 4.9900     |
| BR03    | 5        | 11.9900    |
| BNBG01  | 10       | 3.4900     |

|        |    |        |
|--------|----|--------|
| BNBG02 | 10 | 3.4900 |
| BNBG03 | 10 | 3.4900 |

В столбце `item_price` содержится цена на продукт для каждой записи, имеющейся в заказе. Чтобы узнать полную цену (цена за один продукт, умноженная на количество продуктов в заказе), необходимо сделать следующее:

### ВВОД

```
SELECT prod_id,
       quantity,
       item_price
       quantity*item_price AS expanded_price
FROM OrderItems
WHERE order_nam = 20008;
```

### ВЫВОД

| prod_id | quantity | item_price | expanded_price |
|---------|----------|------------|----------------|
| RGAN01  | 5        | 4.9900     | 24.9500        |
| BR03    | 5        | 11.9900    | 59.9500        |
| BNBG01  | 10       | 3.4900     | 34.9000        |
| BNBG02  | 10       | 3.4900     | 34.9000        |
| BNBG03  | 10       | 3.4900     | 34.9000        |

### Анализ

Столбец `expanded_price`, показанный в предыдущем результате, является вычисляемым полем; вычисление было простым: `quantity*item_price`. Теперь клиентское приложение может использовать этот новый вычисляемый столбец, как и любой другой в таблице.

В SQL поддерживаются основные математические операции, перечисленные в табл. 7.1. Кроме того, для управления порядком обработки можно использовать круглые скобки. В уроке 5, "Расширенная фильтрация данных", рассказывается о порядке обработки.

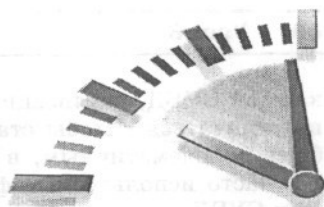
Таблица 7.1. Математические операции в SQL

| Операция | Описание  |
|----------|-----------|
| +        | Сложение  |
| -        | Вычитание |
| *        | Умножение |
| /        | Деление   |

## Резюме

В этом уроке вы узнали, что такое вычисляемые поля и как их можно создавать. Были рассмотрены примеры использования вычисляемых полей для конкатенации строк и выполнения математических операций. Кроме того, вы узнали, как следует создавать и использовать псевдонимы так, чтобы ваше приложение могло обращаться к вычисляемым полям.

## Урок 8



# Использование функций манипулирования данными

В этом уроке вы узнаете, что такое функции, какие типы функций поддерживаются в СУБД и как их можно применять. Вы также узнаете, почему использование SQL-функций может быть проблематичным.

## Что такое функция

Как и в большинстве других языков программирования, в SQL поддерживается использование функций для манипулирования данными. Функции — это операции, которые обычно производятся над данными, чаще всего для облегчения преобразований и манипулирования.

Примером может служить функция `RTRIM()`, которую мы использовали в предыдущем уроке для удаления пробелов в конце строки.

## Проблемы с функциями

Перед тем как начать урок и рассмотреть примеры, обращаю ваше внимание на то, что использование SQL-функций может быть проблематичным.

В отличие от SQL-операторов (например, `SELECT`), которые в основном поддерживаются всеми СУБД одинаково, в разных СУБД могут применяться различные функции. Только некоторые функции в различных СУБД выполняются одинаково. И хотя все типы функций обычно доступны в

каждой СУБД, реализация этих функций может значительно отличаться. Чтобы стало понятно, насколько это может быть проблематичным, в табл. 8.1 перечислены три наиболее часто используемые функции и их синтаксис в различных СУБД.

**Таблица 8.1. Различия в функциях СУБД**

| <i>Функция</i>             | <i>Синтаксис</i>   |
|----------------------------|--|
| Выборка части строки       | В Access используется функция MID(). В DB2, Oracle и PostgreSQL используется функция SUBSTR(). В MySQL, SQL Server и Sybase — SUBSTRING()  |
| Преобразование типа данных | В Access и Oracle используются несколько функций, по одной на каждый тип преобразования. В DB2 и PostgreSQL используется функция CAST(). В MySQL, SQL Server и Sybase используется функция CONVERT() |
| Получение текущей даты     | В Access используется функция NOW(). В DB2 и PostgreSQL — CURRENT_DATE. В MySQL используется функция CURDATE(). В Oracle — SYSDATE. В SQL Server и Sybase — GETDATE()                                |

Как видите, в отличие от SQL-операторов, SQL-функции не относятся к числу переносимых. Это означает, что код, который вы напишете для одной реализации SQL, может не работать в другой.



#### **Переносимый код**

Код, который может работать в разных системах.

Учитывая переносимость кода, многие SQL-программисты стараются не использовать зависящие от реализации функции. Несмотря на то что это довольно благородная и в чем-то идеальная позиция, она не всегда вписывается в интересы приложения с точки зрения производительности. Ему приходится использовать другие методы выполнения того, что СУБД сделала бы более эффективно.



### Стоит ли использовать функции?

Итак, вы пытаетесь решить, использовать функции или нет. Это решение зависит от вас, и здесь нет правильного или неправильного выбора. Если вы решили использовать функции, дописывайте подробные комментарии к коду, чтобы в будущем вы (или другой разработчик) могли узнать, для какой реализации SQL писался данный код.

## Использование функций

В большинстве реализаций SQL поддерживаются следующие типы функций.

- **Текстовые функции;** используются для управления текстовыми строками (например, для обрезания или заполнения значений и преобразования значений в верхний или нижний регистр).
- **Числовые функции;** используются для выполнения математических операций над числовыми данными (например, для вычисления абсолютных значений и выполнения алгебраических вычислений).
- **Функции даты и времени;** используются для управления значениями даты и времени и для выборки отдельных частей этих значений (например, для возвращения разницы между датами и проверки даты на корректность).
- **Системные функции;** возвращают информацию, специфичную для используемой СУБД (например, возвращают регистрационную информацию пользователя).

В предыдущем уроке встречалась функция, которая использовалась в списке столбцов выражения `SELECT`, но это допустимо не для всех функций. Функции можно использовать как в других частях оператора `SELECT` (например, в условии `WHERE`), так и в других SQL-операторах (об этом вы узнаете в дальнейших уроках).



## Функции манипулирования текстом

В примере функций манипулирования текстом в седьмом уроке функция `RTRIM()` использовалась для удаления пробелов в конце значения столбца. Ниже приведен еще один пример, в котором используется функция `UPPER()`:

### ВВОД

```
SELECT vend_name UPPER(vend_name) AS vend_name_upcase
FROM Vendors
ORDER BY vend_name;
```

### ВЫВОД

| <code>vend_name</code> | <code>vend_name_upcase</code> |
|------------------------|-------------------------------|
| Bear Emporium          | BEAR EMPORIUM                 |
| Beras R Us             | BERAS R US                    |
| Doll House Inc.        | DOLL HOUSE INC.               |
| Fun and Games          | FUN AND GAMES                 |
| Furball Inc.           | FURBALL INC.                  |
| Jouets et ours         | JOUETS ET OURS                |

### Анализ

Функция `UPPER()` преобразует текст в верхний регистр и, таким образом, в этом примере имя каждого изготовителя перечислено дважды: первый раз в таком виде, в каком оно хранится в таблице `Vendors`, а второй раз — будучи преобразованным в верхний регистр, в виде столбца `vend_name_upcase`.

В табл. 8.2 перечислены наиболее часто используемые функции манипулирования текстом.

Один элемент из табл. 8.2 требует более подробного объяснения. `SOUNDEX` — это алгоритм, преобразующий текстовую строку в буквенно-цифровой шаблон, описывающий фонетическое представление данного текста. Функция `SOUNDEX` берет в расчет похожие по звучанию буквы и слоги, позволяя сравнивать строки не по тому, как они пишутся, а по тому, как они звучат. Хотя `SOUNDEX` не подпадает под основные концепции SQL, большинство СУБД осуществляют поддержку этой функции.

Таблица 8.2. Наиболее часто используемые функции манипулирования текстом

| Функция                                | Описание                                   |
|--|--|
| LEFT() (или функция подстроки)         | Возвращает символы из левой части строки   |
| LENGTH(a также DATALENGTH() или LEN()) | Возвращает длину строки                    |
| LOWER()                                | Преобразует строку в нижний регистр        |
| LTRIM() (LCASE() в Access)             | Удаляет пробелы в левой части строки       |
| RIGHT() (или функция подстроки)        | Возвращает символы из правой части строки  |
| RTRIM()                                | Удаляет пробелы в правой части строки      |
| SOUNDEX()                              | Возвращает значение SOUNDEX строки         |
| UPPER() (UCASE в Access)               | Преобразует текст строки в верхний регистр |



#### Поддержка SOUNDEX

Функция SOUNDEX() не поддерживается Microsoft Access или PostgreSQL, поэтому следующий пример не будет работать в этих СУБД.

Ниже приведен пример использования функции SOUNDEX(). Клиент Kids Place находится в таблице Customers и имеет контактное лицо Michelle Green. Но что, если это опечатка и на самом деле контактное лицо пишется как Michael Green? Очевидно, поиск по корректному имени ничего не даст, это показано ниже:

#### ВВОД

```
SELECT cust_name cust_contract
FROM Customers
WHERE cust_contract = 'Michael Green';
```

**Вывод**

| cust_name | cust_contract |
|-----------|---------------|
| -----     | -----         |

А теперь попробуйте выполнить поиск при помощи функции `SOUNDEX()`, чтобы найти все имена контактных лиц, которые звучат как Michael Green:

**Ввод**

```
SELECT cust_name cust_contract
FROM Customers
WHERE SOUNDEX(cust_contract) = SOUNDEX('Michael
↵Green');
```

**Вывод**

| cust_name  | cust_contract  |
|------------|----------------|
| -----      | -----          |
| Kids Place | Michelle Green |

**Анализ**

В этом примере в предложении `WHERE` используется функция `SOUNDEX()` для преобразования значения столбца `cust_contact` и искомой строки в их `SOUNDEX`-значения. Так как Michael Green и Michelle Green звучат одинаково, их `SOUNDEX`-значения совпадут и предложение `WHERE` корректно отфильтрует необходимые данные.

## Функции манипулирования датой и временем

Дата и время хранятся в таблицах с использованием соответствующих типов данных, каждая СУБД использует свои собственные типы. Значения даты и времени хранятся в специальном формате, поэтому их можно быстро и эффективно сохранить или отфильтровать, а также сохранить физическое пространство на диске.

Формат, в котором хранятся дата и время, обычно нельзя использовать в приложениях, поэтому почти всегда используются функции даты и времени для чтения, расширения и манипулирования этими значениями. Функции манипулирования датой и временем являются одними из наиболее важных функций в SQL. К сожалению, они мень-

ше всего поддаются переносу на другие платформы и реализации SQL.

Чтобы продемонстрировать процедуру использования функции манипулирования датой и временем, приведем простой пример. В таблице Orders все заказы хранятся с датой заказа. Чтобы извлечь список всех заказов, сделанных в 2004 году, в SQL Server и Sybase необходимо выполнить следующее:

### ВВОД

```
SELECT order_num
FROM Orders
WHERE DATEPART(yy, order_date) = 2004;
```

### ВЫВОД

```
order_num
-----
20005
20006
20007
20008
20009
```

В Access используйте следующую версию примера:

### ВВОД

```
SELECT order_num
FROM Orders
WHERE DATEPART('yyyy', order_date) = 2004;
```

### Анализ

В этом примере (в версиях для SQL Server и Sybase и в Access) используется функция DATEPART(), которая, как видно из названия, возвращает только часть даты. В функции DATEPART() используются два параметра: часть, подлежащая возвращению, и дата, из которой эта часть возвращается. В рассматриваемом примере функция DATEPART() из столбца order\_column возвращает только год. Путем сравнения полученного значения со значением 2004 предложение WHERE выбирает только те заказы, которые были сделаны в этом году.

Ниже приведена версия данного примера для PostgreSQL, в которой используется похожая функция DATE\_PART():

**ВВОД**

```
SELECT order_num
FROM Orders
WHERE DATE_PART('year', order_date) = 2004;
```

В MySQL, помимо DATEPART(), есть множество других функций, предназначенных для манипулирования значениями дат. Пользователи MySQL могут использовать функцию YEAR() для выборки из даты значения года:

**ВВОД**

```
SELECT order_num
FROM Orders
WHERE YEAR(order_date) = 2004;
```

В Oracle также нет функции DATEPART(), но существуют несколько других функций манипулирования датой, которые можно использовать с этой же целью. Рассмотрим пример:

**ВВОД**

```
SELECT order_num
FROM Orders
WHERE to_number(to_char(order_date, 'YY')) = 2004;
```

**Анализ**

В этом примере функция to\_char() используется для извлечения части даты, а функция to\_number() — для преобразования этой части в числовое значение, чтобы его можно было сравнить со значением 2004.

Тех же результатов можно добиться при помощи оператора BETWEEN:

**ВВОД**

```
SELECT order_num
FROM Orders
WHERE order_date BETWEEN to_date('01-JAN-2004')
AND to_date('31-DEC-2004');
```

**Анализ**

В этом примере функция Oracle to\_date() используется для преобразования двух строк в даты. В одной содержится дата 1 января 2004, а в другой — 31 декабря 2004. Стан-

дартный оператор BETWEEN используется для поиска всех заказов, сделанных в период между этими двумя датами. Этот код не будет работать в SQL Server, так как в этой СУБД не поддерживается функция to\_date(). Однако если заменить функцию to\_date() функцией DATAPART(), этот оператор можно будет использовать.



### Даты в Oracle

Даты в формате ДД-МММ-ГГГГ (как в предыдущих примерах) системой Oracle обычно обрабатываются, даже если они не приведены к тому виду, как при использовании функции to\_date(). Однако для надежности лучше всегда использовать эту функцию.

В приведенных примерах выбиралась и использовалась только часть даты (год). Чтобы выбрать заказы по месяцу, необходимо сделать то же самое, указав ключевое слово AND для сравнения месяца и года.

СУБД обычно могут выполнять гораздо больше действий, чем просто выборка части даты. В большинстве из них присутствуют функции для сравнения дат, выполнения простых арифметических операций с датами, опции форматирования дат и многое другое. Но, как вы уже заметили, функции манипулирования датой и временем различны для разных СУБД. Обратитесь к документации по своей СУБД и уточните, какие функции манипулирования датой и временем в ней поддерживаются.

## Функции для манипулирования числами

Числовые функции предназначены для манипулирования числовыми данными. Эти функции используются только для алгебраических, тригонометрических и геометрических вычислений, поэтому они используются не так часто, как функции манипулирования датой и временем.

По иронии судьбы среди всех функций в большинстве СУБД именно числовые функции наиболее стандартизированы. В табл. 8.3 перечислены наиболее часто используемые функции манипулирования числовыми данными.

**Таблица 8.3. Наиболее часто используемые функции манипулирования числами**

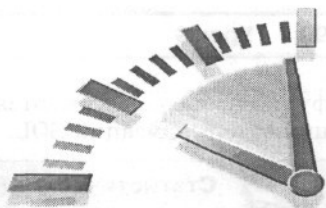
| <i>Функция</i> | <i>Описание</i>                               |
|----------------|---|
| ABS ( )        | Возвращает абсолютное значение числа          |
| COS ( )        | Возвращает косинус указанного угла            |
| EXP ( )        | Возвращает экспоненту указанного числа        |
| PI ( )         | Возвращает значение числа пи                  |
| SIN ( )        | Возвращает синус указанного угла              |
| SQRT ( )       | Возвращает квадратный корень указанного числа |
| TAN ( )        | Возвращает тангенс указанного угла            |

Обратитесь к документации по вашей СУБД, чтобы определить, какие функции манипулирования числовыми данными она поддерживает.

## Резюме

В этом уроке объяснялось, как можно использовать SQL-функции манипулирования данными. Несмотря на то, что эти функции могут быть очень полезными при форматировании, манипулировании и фильтрации данных, они весьма различны для разных реализаций SQL.

## Урок 9



# Суммирование данных

В этом уроке вы узнаете, что такое статистические SQL-функции и как их можно использовать для суммирования данных таблицы

## Использование статистических функций

Часто бывает необходимо просуммировать данные без их выборки, и в SQL предусмотрены для этого специальные функции. SQL-запросы с этими функциями часто используются с целью выборки данных для анализа и создания отчетов. Примерами таких выборок могут послужить:

- определение числа строк в таблице (либо числа строк, которые удовлетворяют какому-то условию или содержат определенное значение);
- получение суммы по набору строк в таблице;
- поиск наибольшего, наименьшего и среднего значений из столбца таблицы (из всех или из каких-то конкретных строк).

В каждом из этих примеров необходимы какие-то итоговые данные по таблице, а не сами данные. Поэтому возвращение реальных данных таблицы было бы пустой тратой времени и ресурсов (не говоря о пропускной способности сети). Итак, все, что вам нужно, — это только итоговая информация.

Чтобы облегчить такой способ извлечения информации, в SQL предусмотрен набор из пяти статистических функций, которые перечислены в табл. 9.1. Эти функции позволяют выполнять все варианты выборки, которые были перечислены выше. В отличие от функций манипулирования данными из предыдущего урока, статистические SQL-



функции поддерживаются без особых изменений в большинстве реализаций SQL.



### Статистические (итоговые) функции

Функции, обрабатывающие набор строк для подсчета и возвращения одного значения.

**Таблица 9.1. Статистические SQL-функции**

| Функция  | Описание                                      |
|----------|---|
| AVG ()   | Возвращает среднее значение столбца           |
| COUNT () | Возвращает число строк в столбце              |
| MAX ()   | Возвращает самое большое значение в столбце   |
| MIN ()   | Возвращает самое маленькое значение в столбце |
| SUM ()   | Возвращает сумму значений столбца             |

Способы использования каждой из этих функций рассматриваются в следующих разделах.

## Функция AVG ()

Функция AVG () используется для возвращения среднего значения определенного столбца путем подсчета числа строк в таблице и суммирования их значений. Эту функцию можно использовать для возвращения среднего значения всех столбцов или определенных столбцов или строк.

В первом примере функция AVG () используется для возвращения средней цены для всех продуктов таблицы Products:

### ВВОД

```
SELECT AVG(prod_price) AS avg_price
FROM Products;
```

### ВЫВОД

```
avg_price
-----
6.823333
```

**Анализ**

Выражение `SELECT`, приведенное выше, возвращает одно значение, `avg_price`, в котором содержится средняя цена всех продуктов таблицы `Products`. Здесь `avg_price` — это псевдоним, описанный в уроке 7, “Создание вычисляемых полей.”

Функцию `AVG()` можно также использовать для нахождения среднего значения определенных столбцов или строк. В следующем примере возвращается средняя цена продуктов, предлагаемых определенным поставщиком:

**ВВОД**

```
SELECT AVG(prod_price) AS avg_price
FROM Products
WHERE vend_id = 'DLL01';
```

**ВЫВОД**

```
avg_price
```

```
-----
6.8650
```

**Анализ**

Этот оператор `SELECT` отличается от предыдущего только тем, что в нем содержится предложение `WHERE`. В соответствии с предложением `WHERE` выбираются только те наименования продуктов, значение `vend_id` для которых равно `DLL01`, поэтому значение, возвращенное в столбце с псевдонимом `avg_price`, является средним только для продуктов этого изготовителя.

**Только отдельные столбцы**

Функцию `AVG()` можно использовать только для вычисления среднего значения определенного числового столбца, имя этого столбца должно быть указано в качестве параметра функции. Чтобы получить среднее значение нескольких столбцов, необходимо использовать несколько функций `AVG()`.



### Значения NULL

Строки столбца, содержащие значения NULL, игнорируются функцией AVG().

## Функция COUNT()

Функция COUNT() подсчитывает число строк. При помощи функции COUNT() можно узнать общее число строк в таблице или число строк, удовлетворяющих определенному критерию.

Эту функцию можно использовать двумя способами:

- В виде COUNT(\*) для подсчета числа строк в таблице независимо от того, содержат столбцы значения NULL или нет.
- В виде COUNT(column) для подсчета числа строк, которые имеют значения в указанных столбцах, причем значения NULL игнорируются.

В первом примере возвращается общее число имен клиентов, содержащихся в таблице Customers:

### ВВОД

```
SELECT COUNT(*) AS num_cust
FROM Customers;
```

### ВЫВОД

```
num_cust
```

```
-----
5
```

### Анализ

В этом примере функция COUNT(\*) используется для подсчета всех строк независимо от их значений. Сумма возвращается в переменную num\_cust.

В следующем примере подсчитываются только клиенты, имеющие адреса электронной почты:

### ВВОД

```
SELECT COUNT(cust_email) AS num_cust
FROM Customers;
```

**ВЫВОД**

```
num_cust
-----
3
```

**Анализ**

В этом выражении SELECT используется функция COUNT(cust\_email) для подсчета только строк, имеющих ненулевое значение в столбце cust\_email. В этом примере значение cust\_email равно 3 (это означает, что только 3 из 5 клиентов имеют адрес электронной почты).

**Значения NULL**

Строки столбцов со значениями NULL игнорируются функцией COUNT(), если указано имя столбца, и учитываются, если используется звездочка (\*).

**Функция MAX()**

Функция MAX() возвращает самое большое значение из указанного столбца. Для этой функции необходимо указывать имя столбца, как это показано ниже:

**ВВОД**

```
SELECT MAX(prod_price) AS max_price
FROM Products;
```

**ВЫВОД**

```
max_price
-----
11.9900
```

**Анализ**

Здесь функция MAX() возвращает цену наиболее дорогого продукта в таблице Products.

**Использование функции MAX() с нечисловыми данными**

Несмотря на то что функция MAX() обычно используется для поиска наибольшего числового значения или даты, многие (но не все) СУБД позволяют использовать ее для возвращения наибольшего значения из всех столбцов, включая текстовые. При использовании с текстовыми данными функция MAX() возвращает строку, которая была бы последней, если бы данные были отсортированы по этой строке.

**Значения NULL**

Строки столбцов со значениями NULL игнорируются функцией MAX().

**Функция MIN()**

Функция MIN() производит противоположное по отношению к MAX() действие — она возвращает наименьшее значение в указанном столбце. Так же, как и для функции MAX(), для MIN() требуется указать имя столбца, как показано ниже:

**ВВОД**

```
SELECT MIN(prod_price) AS min_price
FROM Products;
```

**ВЫВОД**

```
min_price
-----
3.4900
```

**Анализ**

Здесь MIN() возвращает цену самого дешевого продукта в таблице Products.

**Использование функции MIN() с нечисловыми данными**

Несмотря на то что функция MIN() обычно используется для поиска наименьшего числового значения или даты, многие (но не все) СУБД позволяют использовать ее для возвращения наименьшего значения из всех столбцов, включая текстовые. При использовании с текстовыми данными функция MIN() возвращает строку, которая была бы первой, если бы данные были отсортированы по этой строке.

**Значения NULL**

Строки столбцов со значениями NULL игнорируются функцией MIN().

**Функция SUM()**

Функция SUM() возвращает сумму (общую) значений в определенном столбце.

Ниже приведен пример, демонстрирующий это действие. В таблице OrderItems содержатся предметы заказа, причем каждому предмету соответствует определенное количество заказов. Общее число заказанных продуктов (сумма всех значений переменной quantity) может быть выбрана следующим образом:

**ВВОД**

```
SELECT SUM(quantity) AS item_ordered
FROM OrderItems
WHERE order_item = 20005;
```

**ВЫВОД**

```
item_ordered
-----
200
```

**Анализ**

Функция `SUM(quantity)` возвращает сумму всех предметов заказа, а предложение `WHERE` гарантирует, что учитываться будут только необходимые продукты.

**ВВОД**

```
SELECT SUM(item_price*quantity) AS total_price
FROM OrderItems
WHERE order_item = 20005;
```

**ВЫВОД**

```
total_price
-----
1648.0000
```

Функция `SUM(item_price*quantity)` возвращает сумму всех цен в заказе, а предложение `WHERE` гарантирует, что учитываться будут только необходимые продукты.

**Вычисления с несколькими столбцами**

Все статистические функции можно использовать для выполнения вычислений над несколькими столбцами при помощи стандартных математических операторов, как показано в примере.

**Значения NULL**

Строки столбцов со значениями `NULL` игнорируются функцией `SUM()`.

## Статистические вычисления для отдельных значений

Все пять статистических функций могут быть использованы двумя способами:

- для вычисления во всех строках при указании аргумента `ALL` или без указания какого-либо аргумента (так как `ALL` является аргументом по умолчанию);

- для указания отдельных значений при помощи аргумента `DISTINCT`.



#### ALL по умолчанию

Аргумент `ALL` не обязательно указывать, так как он является аргументом по умолчанию. Если не указан аргумент `DISTINCT`, то подразумевается аргумент `ALL`.



#### Не в Access

Microsoft Access не поддерживает использование аргумента `DISTINCT` в статистических функциях, поэтому следующий пример не будет работать в Access.

В следующем примере используется функция `AVG()` для возвращения средней цены продуктов, предлагаемых определенным поставщиком. Это такой же оператор `SELECT`, как и предыдущий, но с использованием ключевого слова `DISTINCT` — при вычислении среднего значения учитываются только определенные цены.

#### ВВОД

```
SELECT AVG(DISTINCT prod_price) AS avg_price
FROM Products
WHERE vend_id = 'LL01';
```

#### ВЫВОД

```
avg_price
```

```
-----
4.2400
```

#### Анализ

В этом примере вследствие использования ключевого слова `DISTINCT` значение `avg_price` получается более высоким, так как в таблице есть несколько предметов с одинаково низкой ценой. Не учитывая их, мы получаем более высокую среднюю стоимость.





### Внимание

Ключевое слово `DISTINCT` можно использовать с функцией `COUNT()` только в том случае, если указано имя столбца. Его нельзя использовать с функцией `COUNT(*)`. Аналогично, `DISTINCT` нужно использовать с именем столбца, но не с выражением.



### Использование ключевого слова `DISTINCT` с функциями `MIN()` и `MAX()`

Несмотря на то что ключевое слово `DISTINCT` технически можно использовать с функциями `MIN()` и `MAX()`, реальной необходимости в этом нет. Минимальные и максимальные значения в столбце будут теми же, независимо от того, указаны определенные значения или нет.



### Предикаты

Помимо ключевых слов `DISTINCT` и `ALL`, некоторые СУБД поддерживают предикаты, такие как `TOP` и `TOP PERCENT`, позволяющие выполнять действия над подмножествами результатов запроса. Обратитесь к документации вашей СУБД, чтобы точно узнать, какие предикаты вы можете использовать.

## Комбинирование статистических функций

Во всех примерах применения статистических функций, приведенных до сих пор, указывалась только одна функция. Но на самом деле операторы `SELECT` могут содержать столько статистических функций, сколько нужно. Рассмотрим пример:

**ВВОД**

```
SELECT COUNT(*) AS num_items,
       MIN(prod_price) AS price_min,
       MAX(prod_price) AS price_max,
       AVG(prod_price) AS price_avg
FROM Products;
```

**ВЫВОД**

| num_items | price_min | price_max | price_avg |
|-----------|-----------|-----------|-----------|
| -----     | -----     | -----     | -----     |
| 9         | 3.4900    | 11.9900   | 6.823333  |

**Анализ**

В одном операторе SELECT используются сразу четыре статистические функции и возвращаются четыре значения (число элементов в таблице Products, самая высокая, самая низкая и средняя их стоимость).

**Псевдонимы**

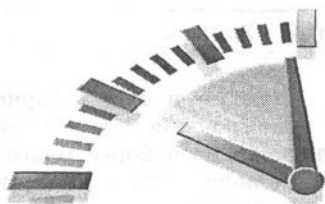
При указании псевдонимов для хранения результатов статистической функции старайтесь не использовать реальных названий столбцов в таблице, поскольку во многих реализациях SQL такое поведение не приветствуется — вы получите сообщение об ошибке.

**Резюме**

Статистические функции используются для получения итоговых данных. В SQL поддерживается пять статистических функций, каждая из которых может использоваться несколькими способами для возвращения только необходимых в данный момент результатов. Эти функции разработаны для повышения эффективности работы, обычно они возвращают результат гораздо быстрее, чем если бы вы производили вычисления в своем клиентском приложении.



## Урок 10



# Итоговые данные

В этом уроке вы узнаете, как получать итоговые данные таким образом, чтобы можно было суммировать подмножества из содержимого таблицы. Для этого используются два новых предложения оператора *SELECT*: предложение *GROUP BY* и предложение *HAVING*.

## Получение итоговых данных

Из предыдущего урока вы узнали, что статистические функции SQL можно использовать для суммирования данных. Это позволяет подсчитывать число строк, вычислять суммы и средние значения, а также получать наибольшее и наименьшее значения, не прибегая к выборке всех данных.

Все эти вычисления до сих пор выполнялись над всеми данными таблицы или над данными, которые соответствовали указанному предложению *WHERE*. В качестве напоминания приведем пример, в котором возвращается количество продуктов, предлагаемых поставщиком *DLL01*:

### ВВОД

```
SELECT COUNT(*) AS num_prods
FROM Products
WHERE vend_id = 'DLL01';
```

### ВЫВОД

```
num_prods
```

```
-----
4
```

Но что, если вы хотите узнать количество продуктов, предлагаемых каждым поставщиком? Или выяснить, какие поставщики предлагают только один продукт, или, наоборот, несколько продуктов?

Именно в таких случаях нужно использовать *группы*. Группирование дает возможность разделить все данные на логические наборы, благодаря чему становится возможным выполнение статистических вычислений отдельно по каждой группе.

## Создание групп

Группы создаются с помощью предложения GROUP BY оператора SELECT.

Лучше всего это можно объяснить на примере:

### ВВОД

```
SELECT vend_id, COUNT(*) AS num_prods
FROM Products
GROUP BY vend_id;
```

### ВЫВОД

| vend_id | num_prods |
|---------|-----------|
| BRS01   | 3         |
| DLL01   | 4         |
| FNG01   | 2         |

### Анализ

Вышеприведенный оператор SELECT предписывает вывести два столбца — vend\_id, содержащий идентификатор поставщика продукта, и num\_prods, содержащий вычисляемые поля (он создается с помощью функции COUNT(\*)). Предложение GROUP BY указывает СУБД сортировать данные и группировать их по столбцу vend\_id. В результате значение num\_prods будет вычисляться по одному разу для каждой группы записей vend\_id, а не один раз для всей таблицы products. Как видите, в результатах указывается, что поставщик BRS01 предлагает три продукта, поставщик DLL01 — четыре продукта, а поставщик FNG01 — 2 продукта.

Поскольку было использовано предложение GROUP BY, не потребовалось указывать каждую группу, для которой должны быть произведены вычисления. Это было сделано автоматически. Предложение GROUP BY указывает СУБД группировать данные и затем выполнять вычисление по каждой группе, а не по всему набору результатов.

Прежде чем использовать предложение GROUP BY, ознакомьтесь с важными правилами, которыми необходимо руководствоваться.

- В предложениях GROUP BY можно указывать столько столбцов, сколько вам необходимо. Это позволяет вкладывать группы одна в другую, благодаря чему обеспечивается тщательный контроль за тем, какие данные подлежат группированию.
- Если вы используете вложенные группы в предложении GROUP BY, данные суммируются для последней указанной вами группы. Другими словами, если введено группирование, вычисления осуществляются для всех указанных столбцов (вы не сможете вернуть данные для каждого отдельного столбца).
- Каждый столбец, указанный в предложении GROUP BY, должен быть столбцом выборки или выражением (но не функцией группирования). Если в операторе SELECT используется какое-то выражение, то же самое выражение должно быть указано в предложении GROUP BY. Псевдонимы применять нельзя.
- В большинстве реализаций SQL нельзя указывать в предложении GROUP BY столбцы, в которых содержатся данные переменной длины (т.е. столбцы, содержащие текстовые поля или поля комментариев).
- За исключением операторов статистических вычислений, каждый столбец, упомянутый в операторе SELECT, должен быть представлен в предложении GROUP BY.
- Если столбец, подлежащий группированию, содержит строку со значением NULL, оно будет возвращено в качестве группы. Если имеется несколько строк со значениями NULL, они будут сгруппированы вместе.
- Предложение GROUP BY должно следовать после предложения WHERE и до какого-либо предложения ORDER BY.



#### Предложение ALL

В некоторых реализациях SQL (например, в Microsoft SQL Server) опционально поддерживается предложе-

ние ALL в предложении GROUP BY. Это предложение можно использовать для возвращения всех групп, даже тех, которые не имеют соответствующих строк (в этом случае функция группирования возвращает значение NULL). Обратитесь к документации своей СУБД, чтобы узнать, поддерживает ли она предложение ALL.



#### Указание столбцов по их относительному положению

Некоторые реализации SQL позволяют указывать столбцы в предложении GROUP BY по их положению в списке SELECT. Например, выражение GROUP BY 2, 1 может означать группирование по выбранному второму столбцу и затем по первому. Хотя этот “стенографический” синтаксис удобен, он поддерживается не всеми реализациями SQL. Его применение также является рискованным в том смысле, что весьма высока вероятность появления ошибок при редактировании операторов SQL.

## Фильтрующие группы

В дополнение к способности группировать данные с помощью предложения GROUP BY, SQL также позволяет осуществлять фильтрацию — указывать, какие группы должны быть включены в результат, а какие исключены из него. Например, вам может понадобиться список клиентов, которые сделали хотя бы два заказа. Чтобы получить такие данные, необходим фильтр, относящийся к целой группе, а не к отдельным строкам.

Вы уже знаете, как действует предложение WHERE (его мы рассматривали ранее, в уроке 4, “Фильтрация данных”). Однако в данном случае предложение WHERE использовать нельзя, поскольку фильтры WHERE указывают строки, а не группы. Собственно говоря, WHERE “не знает”, что такое группы.

Но тогда что можно использовать вместо предложения WHERE? SQL предлагает другое предложение, подходящее для этих целей: предложение HAVING. Предложение HAVING очень похоже на предложение WHERE. И действительно, все типы выражений в предложении WHERE, с которыми вы уже знакомы, могут быть также использованы с предложением HAVING. Единственная разница состоит в том, что WHERE фильтрует строки, а HAVING — группы.



### Предложение HAVING можно использовать со всеми операторами

Из уроков 4 и 5 вы знаете, как можно применять предложение WHERE (включая использование метасимволов и логических операций). Все эти методы и опции могут быть применены и по отношению к HAVING. Синтаксис такой же, отличаются только ключевые слова.

Вспомним, как фильтруются строки. Посмотрите на следующий пример.

### ВВОД

```
SELECT cust_id, COUNT(*) AS orders
FROM Orders
GROUP BY cust_id
HAVING COUNT(*) >= 2;
```

### ВЫВОД

| cust_id    | orders |
|------------|--------|
| 1000000001 | 2      |

### Анализ

Первые три строки этого оператора SELECT аналогичны оператору, рассмотренному ранее. Последняя строка добавляет к нему предложение HAVING, которое фильтрует эти группы с помощью функции COUNT(\*) >= 2 — два или больше заказов.

Как видите, предложение WHERE здесь не работает, поскольку фильтрация основана на итоговом значении группы, а не на значениях указанных строк.





### Разница между HAVING и WHERE

Вот как это можно рассматривать: WHERE фильтрует до того, как данные будут сгруппированы, а HAVING фильтрует после того, как данные были сгруппированы. Это — важное различие; строки, которые были выброшены по предложению WHERE, не будут включены в группу, иначе это могло бы изменить вычисляемые значения, которые, в свою очередь, могли бы повлиять на фильтрацию групп в предложении HAVING.

А теперь подумаем: возникает ли необходимость в использовании как предложения WHERE, так и предложения HAVING в одном операторе?

Конечно, возникает. Предположим, вы хотите усовершенствовать фильтр предыдущего оператора таким образом, чтобы он возвращал имена всех клиентов, которые сделали два или больше заказов за последние 12 месяцев. Чтобы добиться этого, вы можете добавить предложение WHERE, которое берет во внимание только заказы, сделанные в последние 12 месяцев. Затем вы добавляете предложение HAVING, чтобы отфильтровать только те группы, в которых имеются две или больше строк.

Чтобы лучше разобраться в этом, рассмотрим следующий пример, где перечисляются все поставщики, которые предлагают несколько продуктов по цене 4 и более за единицу:

#### ВВОД

```
SELECT vend_id, COUNT(*) AS num_prods
FROM Products
WHERE prod_price >= 4
GROUP BY vend_id
HAVING COUNT(*) >= 2;
```

#### ВЫВОД

| vend_id | num_prods |
|---------|-----------|
| BRS01   | 3         |
| FNG01   | 2         |

**Анализ**

Этот пример нуждается в пояснении. Первая строка представляет собой основной оператор SELECT, использующий статистическую функцию, — точно так же, как в предыдущих примерах. Предложение WHERE фильтрует все строки со значениями в столбце prod\_price не менее 4. Затем данные группируются по столбцу vend\_id, а потом предложение HAVING фильтрует только группы, содержащие не менее двух членов. При отсутствии предложения WHERE была бы получена лишняя строка (поставщик, предлагающий 4 продукта, каждый из которых дешевле 4), как показано ниже.

**ВВОД**

```
SELECT vend_id, COUNT(*) AS num_prods
FROM Products
GROUP BY vend_id
HAVING COUNT(*) >= 2;
```

**ВЫВОД**

| vend_id | num_prods |
|---------|-----------|
| BRS01   | 3         |
| DLL01   | 4         |
| FNG01   | 2         |

**Использование предложений HAVING и WHERE**

Предложение HAVING так похоже на предложение WHERE, что в большинстве СУБД оно трактуется точно так же, если только не указано никакое предложение GROUP BY. И все же вы должны знать, что между ними существует разница. Используйте предложение HAVING только вместе с предложениями GROUP BY, а предложение WHERE — для стандартной фильтрации на уровне строк.

## Группирование и сортировка

Важно понимать, что предложения GROUP BY и ORDER BY весьма различны, хотя с их помощью иногда можно добиться одинаковых результатов. Разобраться в этом вам поможет табл. 10.1.

**Таблица 10.1. Сравнение предложений ORDER BY и GROUP BY**

| <i>ORDER BY</i>  | <i>GROUP BY</i>   |
|--|---|
| Сортирует полученные результаты  | Группирует строки. Однако отображаемый результат может не соответствовать порядку группирования                                   |
| Могут быть использованы любые столбцы (даже не выбранные в предложении SELECT) | Могут быть использованы только выбранные столбцы или выражения; должно быть использовано выражение для каждого выбранного столбца |
| Не является необходимым  | Требуется, если используются столбцы (или выражения) со статистическими функциями   |

Первое из отличий, перечисленных в табл. 10.1, является очень важным. Чаще всего вы обнаружите, что данные, сгруппированные с помощью предложения GROUP BY, будут отображаться в порядке группирования. Но так будет не всегда, и в действительности это не требуется в спецификациях SQL. Более того, даже если ваша СУБД сортирует данные так, как указано в предложении GROUP BY, вам вдруг может понадобиться отсортировать их по-другому. То, что вы группируете данные одним способом (чтобы получить для группы указанные итоговые значения), не означает, что желанный для вас результат должен быть отсортирован именно так. Следует использовать явным образом предложение ORDER BY, даже если результат его применения будет совпадать с результатом использования предложения GROUP BY.



### Не забывайте использовать предложение ORDER BY

Как правило, каждый раз, когда вы используете предложение GROUP BY, приходится указывать и предложение ORDER BY. Это — единственный способ, гарантирующий, что данные будут отсортированы правильно. Не следует надеяться на то, что ваши данные отсортирует предложение GROUP BY.

Чтобы вы могли понять, как следует использовать совместно предложения GROUP BY и ORDER BY, рассмотрим пример. Следующий оператор SELECT аналогичен тем, которые использовались ранее: он выводит номер заказа и количество предметов, упорядоченных по всем заказам, которые содержат три или больше предметов.

### ВВОД

```
SELECT order_num, COUNT(*) AS items
FROM OrderItems
GROUP BY order_num
HAVING COUNT(*) >= 3;
```

### ВЫВОД

| order_num | items |
|-----------|-------|
| -----     |       |
| 20006     | 3     |
| 20007     | 5     |
| 20008     | 5     |
| 20009     | 3     |

Чтобы отсортировать результат по количеству заказанных предметов, все, что вам необходимо сделать, — это добавить предложение ORDER BY, как показано ниже:

### ВВОД

```
SELECT order_num, COUNT(*) AS items
FROM OrderItems
GROUP BY order_num
HAVING COUNT(*) >= 3;
ORDER BY items, order_num;
```



### Несовместимость с Access

СУБД Microsoft Access не позволяет осуществлять сортировку по псевдонимам, и для нее этот пример неприменим. Выход состоит в замене столбца `items` (в предложении `ORDER BY`) вычисляемым выражением или номером поля. По существу, будут работать оба предложения, `ORDER BY COUNT(*)`, `order_num` и `ORDER BY 1, order_num`.

### Вывод

| <code>order_num</code> | <code>items</code> |
|------------------------|--------------------|
| 20006                  | 3                  |
| 20009                  | 3                  |
| 20007                  | 5                  |
| 20008                  | 5                  |

### Анализ

В этом примере предложение `GROUP BY` используется для группирования данных по номеру заказа (столбец `order_num`), так что функция `COUNT(*)` может вернуть количество предметов в каждом заказе. Предложение `HAVING` фильтрует данные таким образом, что возвращаются только заказы с тремя и более предметами. Наконец, результат сортируется за счет использования предложения `ORDER BY`.

## Упорядочение предложения SELECT

Предложения оператора `SELECT` указываются в определенном порядке. В табл. 10.2 перечислены все предложения, которые мы изучили о сих пор, в порядке, в котором они должны использоваться.

**Таблица 10.2. Предложения оператора SELECT и последовательность их использования**

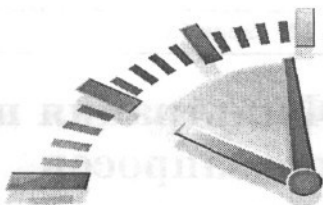
| <i>Предложение</i> | <i>Описание</i>                                       | <i>Необходимость</i>   |
|--------------------|---|--|
| SELECT             | Столбцы или выражения, которые должны быть возвращены | Да   |
| FROM               | Таблица для возвращения данных из...                  | Только если выбираются данные из таблицы                       |
| WHERE              | Фильтрация на уровне строк                            | Нет  |
| GROUP BY           | Определение группы                                    | Только если осуществляются статистические вычисления по группе |
| HAVING             | Фильтрация на уровне групп                            | Нет  |
| ORDER BY           | Упорядочивание результатов сортировки                 | Нет  |

## Резюме

В уроке 9, “Суммирование данных,” вы узнали, как использовать статистические функции SQL для выполнения операций суммирования над данными. В этом уроке рассказывалось о том, как нужно использовать предложение GROUP BY для выполнения вычислений по отношению к группам данных, возвращения результатов для каждой группы. Было описано, как можно использовать предложение HAVING для фильтрации указанных групп, в уроке также пояснялось, какова разница между ORDER BY и GROUP BY и между предложениями WHERE и HAVING.



## Урок 11



# Использование подзапросов

В этом уроке рассказывается о том, что такое подзапросы и как их можно использовать.

## Что такое подзапросы

Операторы `SELECT` представляют собой запросы SQL. Все операторы `SELECT`, с которыми мы имели дело до сих пор, представляли собой простые запросы: посредством отдельных операторов извлекались данные из отдельных таблиц базы данных.



### Запрос

Какой-либо оператор SQL. Однако этот термин обычно используется по отношению к операторам `SELECT`.

Язык SQL позволяет также создавать *подзапросы*: запросы, которые вложены в другие запросы. Их также называют *вложенные запросы*, или *подчиненные запросы*. Почему возникает необходимость в подзапросах? Лучший способ объяснить эту концепцию — рассмотреть несколько примеров.



### Поддержка в MySQL

Если вы используете MySQL, знайте, что подзапросы поддерживаются этой СУБД начиная с версии 4.1. Более ранние версии MySQL запросы не поддерживают.



## Фильтрация посредством подзапросов

Таблицы баз данных, используемые во всех уроках этой книги, являются реляционными таблицами (см. приложение А, в котором описана каждая из таблиц и отношения между ними). Заказы хранятся в двух таблицах. Таблица `Orders` (заказы) содержит по одной строке для каждого заказа; в ней указываются номер заказа, идентификатор клиента и дата заказа. Предметы отдельного заказа хранятся в соответствующей таблице `OrderItems`. Таблица `Orders` не содержит информацию о клиентах. Она хранит только идентификатор клиента. Информация о клиентах хранится в таблице `Customers` (клиенты).

Теперь предположим, что вы хотите получить перечень всех клиентов, которые заказали продукт `RGAN01`. Что нужно сделать, чтобы получить эту информацию? Для этого нужно сделать следующее.

1. Выбрать номера всех заказов, в которых содержится продукт `RGAN01`.
2. Выбрать идентификатор клиента для всех клиентов, которые имеют заказы, перечисленные среди номеров заказов, возвращенных на предыдущем шаге.
3. Выбрать информацию о клиенте для всех клиентов, идентификаторы которых были возвращены на предыдущем шаге.

Каждый из этих шагов можно выполнить в виде отдельного запроса. Делая это, вы используете результаты, возвращенные одним оператором `SELECT`, чтобы заполнить предложение `WHERE` для следующего оператора `SELECT`.

Вы можете также использовать подзапросы для того, чтобы объединить все три запроса в один-единственный оператор.

Первый оператор `SELECT` выбирает столбец `order_num` для всех продуктов заказа, у которых в столбце `prod_id` значится `RGAN01`. Результат представляет собой номера двух заказов, содержащих этот предмет:

**ВВОД**

```
SELECT order_num
FROM OrderItems
WHERE prod_id = 'RGAN01';
```

**ВЫВОД**

```
order_num
-----
20007
20008
```

Следующий шаг состоит в выборке идентификаторов клиентов, связанных с заказами 20007 и 20008. Используя предложение IN, о котором говорилось в уроке 5, “Расширенная фильтрация данных”, вы можете создать такой оператор SELECT:

**ВВОД**

```
SELECT cust_id
FROM Orders
WHERE order_num IN (20007,20008);
```

**ВЫВОД**

```
cust_id
-----
1000000004
1000000005
```

Теперь объединим эти два запроса путем превращения первого из них (того, который возвращает номера заказов) в подзапрос. Посмотрите на следующий оператор SELECT:

**ВВОД**

```
SELECT cust_id
FROM Orders
WHERE order_num IN (SELECT order_num
FROM OrderItems
WHERE prod_id = 'RGAN01');
```

**ВЫВОД**

```
cust_id
-----
1000000004
1000000005
```

## Анализ

Подзапросы всегда обрабатываются, начиная с самого внутреннего оператора SELECT в направлении “изнутри наружу”. При обработке предыдущего оператора СУБД в действительности выполняет две операции.

Вначале она выполняет подзапрос:

```
SELECT order_num FROM orderitems WHERE prod_id='RGAN01'
```

В результате выполнения этого запроса возвращается два номера заказа, 20007 и 20008. Эти два значения затем передаются в предложение WHERE внешнего запроса в формате с разделителем в виде запятой, необходимом для оператора IN. Теперь внешний запрос становится таким:

```
SELECT cust_id FROM orders WHERE order_num IN (20007,20008)
```

Как видите, результат корректен и точно такой же, как полученный путем жесткого кодирования предложения WHERE в предыдущем примере.



### Форматируйте ваши SQL-запросы

Операторы SELECT, содержащие подзапросы, могут оказаться трудными для чтения и устранения ошибок, особенно если их сложность возрастает. Разбиение запросов на многие строки и соответствующее обозначение строк, как показано ниже, значительно облегчит вашу работу с подзапросами.

Теперь у нас есть идентификаторы всех клиентов, заказавших продукт RGAN01. Следующий шаг состоит в получении клиентской информации для каждого из этих идентификаторов клиентов. Оператор SQL, осуществляющий выборку двух столбцов, таков:

## ВВОД

```
SELECT cust_name, cust_contact
FROM Customers
WHERE cust_id IN ('1000000004','1000000005');
```

Вместо жесткого указания идентификаторов клиентов вы можете превратить данное предложение WHERE в подзапрос:

**ВВОД**

```
SELECT cust_name, cust_contact
FROM Customers
WHERE cust_id IN (SELECT cust_id
FROM Orders
WHERE order_num IN (SELECT order_num
FROM OrderItems
WHERE prod_id = 'RGAN01'));
```

**ВЫВОД**

| cust_name     | cust_contact       |
|---------------|--------------------|
| -----         | -----              |
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |

Чтобы выполнить вышеприведенное выражение SELECT, СУБД должна в действительности выполнить три оператора SELECT. Самый внутренний подзапрос возвращает перечень номеров заказов, который затем используется как предложение WHERE для подзапроса, внешнего по отношению к данному. Этот подзапрос возвращает перечень идентификаторов клиентов, которые используются в предложении WHERE запроса более высокого уровня. Запрос верхнего уровня возвращает искомые данные.

Как видите, за счет использования подзапросов можно создавать очень мощные и гибкие SQL-операторы. Не существует ограничений на число подчиненных запросов, хотя на практике вы можете столкнуться с тем, что снижение производительности подскажет вам, что было использовано слишком много уровней подзапросов.

**Только один столбец**

Операторы SELECT подзапроса могут выбирать только один столбец. Попытка произвести выборку нескольких столбцов приведет к появлению сообщения об ошибке.

**Подзапросы и производительность**

Представленные нами коды работают и приводят к получению необходимых результатов. Однако подза-

просы — не всегда наиболее эффективный способ выборки данных такого типа. Более подробно об этом рассказано в уроке 12, "Объединение таблиц", в котором повторно будет рассмотрен этот же самый пример.

## Использование подзапросов в качестве вычисляемых полей

Другой способ использования подзапросов состоит в использовании вычисляемых полей. Предположим, вы хотите вывести общее количество заказов, размещенных каждым клиентом в таблице Customers (клиенты). Заказы хранятся в таблице Orders вместе с соответствующими идентификаторами клиентов.

Чтобы выполнить эту операцию, необходимо сделать следующее.

1. Выбрать перечень клиентов из таблицы Customers.
2. Для каждого выбранного клиента посчитать число его заказов в таблице Orders.

Как следует из предыдущих двух уроков, вы можете использовать оператор `SELECT COUNT(*)` для подсчета строк в таблице, а используя предложение `WHERE` для фильтрации идентификатора конкретного клиента, вы можете подсчитать заказы только этого клиента. Например, посредством следующего кода можно подсчитать количество заказов, сделанных клиентом 1000000001:

### ВВОД

```
SELECT COUNT(*) AS orders
FROM Orders
WHERE cust_id = '1000000001';
```

Чтобы получить итоговую информацию посредством функции `COUNT(*)` для каждого клиента, используйте `COUNT*` как подзапрос. Посмотрите на следующий код:

### ВВОД

```
SELECT cust_name,
       cust_state,
```

```
(SELECT COUNT(*)
FROM Orders
WHERE Orders.cust_id = Customers.cust_id) AS orders
FROM Customers
ORDER BY cust_name;
```

## Вывод

| cust_name     | cust_state | orders |
|---------------|------------|--------|
| Fun4All       | IN         | 1      |
| Fun4All       | AZ         | 1      |
| Kids Place    | OH         | 0      |
| The Toy Store | IL         | 1      |
| Village Toys  | MI         | 2      |

## Анализ

Этот оператор SELECT возвращает три столбца для каждого клиента из таблицы Customers: cust\_name, cust\_state и orders. Поле Orders является вычисляемым; оно формируется в результате выполнения подзапроса, который заключен в круглые скобки. Этот подзапрос выполняется один раз для каждого выбранного клиента. В приведенном примере подзапрос выполняется пять раз, потому что были выбраны имена пяти клиентов.

Предложение WHERE в подзапросе несколько отличается от предложений WHERE, с которыми мы работали ранее, потому что в нем используются полные имена столбцов. Следующее предложение требует от SQL, чтобы было проведено сравнение значения cust\_id в таблице Orders с тем, которое в данный момент выбирается из таблицы Customers:

```
WHERE Orders.cust_id = Customers.cust_id
```

Этот синтаксис — имя таблицы и имя столбца разделяются точкой — должен использоваться всякий раз, когда может возникнуть неопределенность в именах столбцов. В данном примере имеется два столбца cust\_id, один в таблице Customers и один в таблице Orders. Без использования полностью определенных имен столбцов СУБД будет считать, что вы сравниваете cust\_id в таблице Orders с самим собой. Поэтому запрос

```
SELECT COUNT(*) FROM Orders WHERE cust_id = cust_id
```

будет всегда возвращать общее число заказов в таблице Orders, но это не тот результат, который вам нужен:

**ВВОД**

```
SELECT cust_name,
       cust_state,
       (SELECT COUNT(*)
        FROM Orders
        WHERE cust_id = cust_id) AS orders
FROM Customers
ORDER BY cust_name;
```

**ВЫВОД**

| cust_name     | cust_state | orders |
|---------------|------------|--------|
| -----         | -----      | -----  |
| Fun4All       | IN         | 5      |
| Fun4All       | AZ         | 5      |
| Kids Place    | OH         | 5      |
| The Toy Store | IL         | 5      |
| Village Toys  | MI         | 5      |

Подзапросы чрезвычайно полезны при подготовке оператора SELECT такого типа, однако внимательно следите за тем, чтобы были правильно указаны неоднозначные имена столбцов.

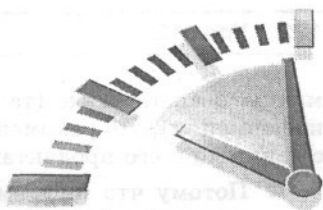
**Всегда есть несколько решений**

Хотя простой код, представленный в этом уроке, и работоспособен, зачастую он оказывается не самым эффективным способом выборки данных такого типа. Мы еще раз рассмотрим этот пример в одном из следующих уроков.

**Резюме**

В этом уроке вы узнали, что такое подзапросы и как их можно использовать. Чаще всего подзапросы используют в операторах IN предложения WHERE и для заполнения вычисляемых столбцов. Были представлены примеры операций обоих названных типов.

## Урок 12



# Объединение таблиц

В этом уроке вы узнаете, что такое объединения, для чего они применяются и как следует оформлять операторы `SELECT`, использующие объединения.

## Что такое объединения

Одной из самых мощных особенностей реализаций языка SQL является возможность “на лету” объединять таблицы при выполнении запросов на выборку данных. Объединения — это самые мощные операции, которые можно выполнить с использованием оператора `SELECT` языка SQL, поэтому тщательное изучение объединений и их синтаксиса является чрезвычайно важной частью процесса освоения SQL.

Прежде чем вы сможете эффективно использовать объединения, вам следует уяснить, что такое реляционные таблицы, и ознакомиться с основами построения реляционных баз данных. В этой книге полностью осветить эту тему не удастся, но сказанного будет достаточно для того, чтобы вы могли получить полное представление об этом предмете и двигаться дальше.

## Что такое реляционные таблицы

Понять, что представляют собой реляционные таблицы, поможет пример из реальной жизни.

Предположим, что некоторая таблица базы данных содержит каталог продуктов, в котором для каждого предмета, включенного в каталог, выделена одна строка. Информация, которая хранится о каждом предмете, должна включать описание продукта и его цену, а также сведения о поставщике и компании, выпустившей данный продукт. Теперь предположим, что вы получили обширный каталог от одного из поставщиков. Где вы должны хранить инфор-



мацию о поставщике (такую как имя, адрес и контактная информация)? Не рекомендуется хранить эти данные вместе с данными о его продуктах по нескольким причинам.

- Потому что информация о поставщике одна и та же для всех его продуктов; повторение этой информации для каждого продукта приведет к напрасной потере времени и места на диске.
- Если информация о поставщике изменяется (например, если он переезжает или изменяется его почтовый код), вам придется обновлять информацию каждый раз, когда вы будете вносить данные об этом поставщике.
- Когда данные повторяются (а так происходит, когда информация о поставщике указывается для каждого продукта), высока вероятность того, что данные не будут каждый раз вводиться одним и тем же образом. Несовместимые данные очень трудно использовать при создании отчетов.

Отсюда можно сделать вывод, что хранить множество экземпляров одних и тех же данных крайне нежелательно, именно этот принцип и лежит в основе создания реляционных баз данных.

Реляционные таблицы разрабатываются таким образом, что вся информация распределяется по множеству таблиц, причем для данных каждого типа создается отдельная таблица. Эти таблицы соотносятся (связываются) между собой через общие значения (и таким образом являются *реляционными* (относительными) в реляционной конструкции).

В нашем примере вы можете создать две таблицы: одну для хранения информации о поставщике, вторую — о его продуктах. Таблица *Vendors* содержит информацию о поставщиках, по одной строке для каждого поставщика, где обязательно значится его уникальный идентификатор. Это значение, называемое *первичный ключ*, может быть или идентификатором поставщика, или каким-то другим уникальным (неповторяющимся) значением.

В таблице *Products* хранится только информация о продуктах, и никакой конкретной информации о поставщиках, за исключением их идентификаторов (первичного ключа таблицы *Vendors*). Этот ключ связывает таблицу *Vendors* с таблицей *Products*. Благодаря применению это-

го идентификатора поставщика вы можете использовать таблицу Vendors для поиска информации о соответствующем поставщике.

Что это дает? Давайте рассмотрим следующие моменты.

- Информация о поставщике никогда не повторяется, благодаря чему экономится время и место на диске.
- Если информация о поставщике изменяется, бывает достаточно обновить только одну запись о нем, единственную в таблице Vendors. Данные в связанных с ней таблицах изменять не нужно.
- Поскольку никакие данные не повторяются, они, очевидно, оказываются непротиворечивыми, благодаря чему составление отчетов и манипулирование данными значительно упрощаются.

Таким образом, реляционные данные можно эффективно хранить и ими можно легко манипулировать. Благодаря этому реляционные базы данных используются намного чаще, чем другие.



### Масштабирование

Позаботьтесь о том, чтобы можно было беспрепятственно расширять базу данных. О хорошо сконструированной базе данных или приложении говорят, что она (оно) хорошо масштабируется.

## Для чего используют объединения

Распределение данных по многим таблицам обеспечивает их более эффективное хранение, упрощает манипулирование данными и повышает масштабируемость. Однако эти преимущества не получаются даром — за все нужно платить.

Если данные хранятся во многих таблицах, как их можно извлечь с помощью одного оператора SELECT? Ответ таков: посредством объединения данных. Проще говоря, объединение представляет собой механизм, используемый для объединения таблиц внутри оператора (отсюда термин “объединение”). Используя особый синтаксис, можно объединить несколько таблиц таким образом, что будет возвращаться один результат, и это объединение будет “на лету” связывать нужные строки из каждой таблицы.



### Использование интерактивных инструментов СУБД

Важно понимать, что объединение не является “физическим объектом” — другими словами, оно не существует как реальная таблица в базе данных. Объединение создается СУБД в случае необходимости и сохраняется только на время выполнения запроса.

Многие СУБД предлагают графический интерфейс, который может быть использован для интерактивного определения отношений таблицы. Эти инструменты могут оказаться чрезвычайно полезными для поддержания целостности ссылочных данных. При использовании реляционных таблиц важно, чтобы только достоверные данные содержались в реляционных столбцах. Вернемся к нашему примеру: если в таблице *Products* хранится недостоверный идентификатор поставщика, его продукты окажутся недоступными, поскольку они не будут относиться ни к одному поставщику. Во избежание этого база данных должна разрешать пользователю вводить только достоверные значения (т.е. такие, которые представлены в таблице *Vendors*) в столбце идентификаторов поставщика таблицы *Products*. Целостность на уровне ссылок (целостность ссылочных данных) означает, что СУБД обязывает пользователя соблюдать правила, обеспечивающие целостность данных. И соблюдение этих правил часто обеспечивается при посредстве интерфейсов СУБД.

## Создание объединения

Создание объединения — очень простая процедура. Нужно указать все таблицы, которые должны быть включены в объединение, а также “объяснить” СУБД, как они должны быть соотнесены между собой. Посмотрите на следующий пример.

**ВВОД**

```
SELECT vend_name, prod_name, prod_price
FROM Vendors, Products
WHERE Vendors.vend_id = Products.vend_id;
```

**ВЫВОД**

| vend_name       | prod_name           | prod_price |
|-----------------|---------------------|------------|
| Doll House Inc. | Fish bean bag toy   | 3.4900     |
| Doll House Inc. | Bird bean bag toy   | 3.4900     |
| Doll House Inc. | Rabbit bean bag toy | 3.4900     |
| Bears R Us      | 8 inch teddy bear   | 5.9900     |
| Bears R Us      | 12 inch teddy bear  | 8.9900     |
| Bears R Us      | 18 inch teddy bear  | 11.9900    |
| Doll House Inc. | Raggedy Ann         | 4.9900     |
| Fun and Games   | King doll           | 9.4900     |
| Fun and Games   | Queen doll          | 9.4900     |

**Анализ**

Рассмотрим представленный выше код. Оператор SELECT начинается точно так же, как все операторы, которые мы до сих пор рассматривали, — с указания столбцов, которые должны быть выбраны. Существенная разница состоит в том, что два из указанных столбцов (prod\_name и prod\_price) находятся в одной таблице, а третий (vend\_name) — в другой.

Теперь посмотрим на предложение FROM. В отличие от предыдущих операторов SELECT, этот содержит две таблицы, указанные в предложении FROM, Vendors и Products. Это имена двух таблиц, которые должны быть объединены в данном операторе SELECT.

Таблицы корректно объединяются в предложении WHERE, которое указывает СУБД связывать идентификатор поставщика vend\_id из таблицы Vendors со значением vend\_id таблицы Products.

Обратите внимание на то, что эти столбцы указаны как Vendors.vend\_id и Products.vend\_id. Такие полностью определенные имена необходимы здесь потому, что, если вы укажете только vend\_id, СУБД не сможет понять, на какие именно столбцы vend\_id вы ссылаетесь. (Их два, по одному в каждой таблице). Как можно видеть из представленного результата, один оператор SELECT возвращает данные из двух разных таблиц.



### Полностью определенные имена столбцов

Используйте полностью определенные имена столбцов (таблицы и столбцы разделяются точкой) всякий раз, когда может возникнуть неоднозначность относительно того, на какой столбец вы ссылаетесь. В большинстве СУБД будет возвращено сообщение об ошибке, если вы укажете неоднозначное имя столбца, не определив его полностью путем указания имени таблицы.

## Важность предложения WHERE

Может показаться странным использование предложения WHERE для установления отношения в объединении, но на это есть существенная причина. Помните: когда таблицы объединяются в операторе SELECT, это отношение создается “на лету”.

В определениях таблиц базы данных ничего не говорится о том, как СУБД должна объединять таблицы. Вы должны указать это сами. Когда вы объединяете две таблицы, то, что вы в действительности делаете, — это создаете пары, состоящие из каждой строки первой таблицы и каждой строки второй таблицы. Предложение WHERE действует как фильтр, позволяющий включать в результат только строки, которые соответствуют указанному предложению фильтрации — в данном случае предложению объединения. Без предложения WHERE каждая строка в первой таблице будет образовывать пару с каждой строкой второй таблицы независимо от того, есть логика в их объединении или нет.



### Декартово произведение

Результаты, возвращаемые при умножении таблиц без указания условия объединения. Количество выбранных строк будет равно числу строк в первой таблице, умноженному на число строк во второй таблице.

Для того чтобы разобраться в этом, посмотрите на следующий оператор SELECT и результат его применения.

## ВВОД

```
SELECT vend_name, prod_name, prod_price
FROM Vendors, Products;
```

## ВЫВОД

| <u>vend_name</u> | <u>prod_name</u>    | <u>prod_price</u> |
|------------------|---------------------|-------------------|
| Bears R Us       | 8 inch teddy bear   | 5.99              |
| Bears R Us       | 12 inch teddy bear  | 8.99              |
| Bears R Us       | 18 inch teddy bear  | 11.99             |
| Bears R Us       | Fish bean bag toy   | 3.49              |
| Bears R Us       | Bird bean bag toy   | 3.49              |
| Bears R Us       | Rabbit bean bag toy | 3.49              |
| Bears R Us       | Raggedy Ann         | 4.99              |
| Bears R Us       | King doll           | 9.49              |
| Bears R Us       | Queen doll          | 9.49              |
| Bear Emporium    | 8 inch teddy bear   | 5.99              |
| Bear Emporium    | 12 inch teddy bear  | 8.99              |
| Bear Emporium    | 18 inch teddy bear  | 11.99             |
| Bear Emporium    | Fish bean bag toy   | 3.49              |
| Bear Emporium    | Bird bean bag toy   | 3.49              |
| Bear Emporium    | Rabbit bean bag toy | 3.49              |
| Bear Emporium    | Raggedy Ann         | 4.99              |
| Bear Emporium    | King doll           | 9.49              |
| Bear Emporium    | Queen doll          | 9.49              |
| Doll House Inc.  | 8 inch teddy bear   | 5.99              |
| Doll House Inc.  | 12 inch teddy bear  | 8.99              |
| Doll House Inc.  | 18 inch teddy bear  | 11.99             |
| Doll House Inc.  | Fish bean bag toy   | 3.49              |
| Doll House Inc.  | Bird bean bag toy   | 3.49              |
| Doll House Inc.  | Rabbit bean bag toy | 3.49              |
| Doll House Inc.  | Raggedy Ann         | 4.99              |
| Doll House Inc.  | King doll           | 9.49              |
| Doll House Inc.  | Queen doll          | 9.49              |
| Furball Inc.     | 8 inch teddy bear   | 5.99              |
| Furball Inc.     | 12 inch teddy bear  | 8.99              |
| Furball Inc.     | 18 inch teddy bear  | 11.99             |
| Furball Inc.     | Fish bean bag toy   | 3.49              |
| Furball Inc.     | Bird bean bag toy   | 3.49              |
| Furball Inc.     | Rabbit bean bag toy | 3.49              |
| Furball Inc.     | Raggedy Ann         | 4.99              |
| Furball Inc.     | King doll           | 9.49              |
| Furball Inc.     | Queen doll          | 9.49              |
| Fun and Games    | 8 inch teddy bear   | 5.99              |
| Fun and Games    | 12 inch teddy bear  | 8.99              |
| Fun and Games    | 18 inch teddy bear  | 11.99             |
| Fun and Games    | Fish bean bag toy   | 3.49              |
| Fun and Games    | Bird bean bag toy   | 3.49              |
| Fun and Games    | Rabbit bean bag toy | 3.49              |
| Fun and Games    | Raggedy Ann         | 4.99              |

|                |                     |       |
|----------------|---------------------|-------|
| Fun and Games  | King doll           | 9.49  |
| Fun and Games  | Queen doll          | 9.49  |
| Jouets et ours | 8 inch teddy bear   | 5.99  |
| Jouets et ours | 12 inch teddy bear  | 8.99  |
| Jouets et ours | 18 inch teddy bear  | 11.99 |
| Jouets et ours | Fish bean bag toy   | 3.49  |
| Jouets et ours | Bird bean bag toy   | 3.49  |
| Jouets et ours | Rabbit bean bag toy | 3.49  |
| Jouets et ours | Raggedy Ann         | 4.99  |
| Jouets et ours | King doll           | 9.49  |
| Jouets et ours | Queen doll          | 9.49  |

### Анализ

Как видно из представленного результата, декартово произведение вы, скорее всего, будете использовать очень редко. Данные, возвращенные таким образом, ставят в соответствие каждому продукту каждого поставщика, включая продукты с указанием “не того” поставщика (и даже поставщиков, которые вообще не предлагают продуктов).



#### Не забудьте указать предложение WHERE

Проверьте, включили ли вы в оператор предложение WHERE, иначе СУБД возвратит намного больше данных, чем вам нужно. Кроме того, убедитесь в том, что предложение WHERE сформулировано правильно. Некорректное предложение фильтрации приведет к тому, что СУБД выдаст вам неверные данные.



#### Перекрестное объединение

Иногда объединение, которое возвращает декартово произведение, называют перекрестным объединением.

## Внутренние объединения

Объединение, которое мы до сих пор использовали, называется объединение по эквивалентности — оно основано на проверке эквивалентности двух таблиц. Объединение такого типа называют также *внутреннее объединение*. Для этих объединений можно использовать несколько иной синтаксис, явно указывающий на тип объединения. Следующий оператор SELECT возвращает в точности такие же данные, как и в предыдущем примере.

**ВВОД**

```
SELECT vend_name, prod_name, prod_price
FROM Vendors INNER JOIN Products
ON Vendors.vend_id = Products.vend_id;
```

**Анализ**

Оператор SELECT здесь точно такой же, как и предыдущий, но предложение FROM другое. Здесь отношение между двумя таблицами является частью предложения FROM, указанного как INNER JOIN. При использовании такого синтаксиса предложение объединения указывается с использованием специального предложения ON вместо предложения WHERE. Фактическое предложение, передаваемое в ON, то же самое, которое передавалось бы в предложение WHERE.

Обратитесь к документации своей СУБД чтобы узнать, какой синтаксис предпочтительнее использовать.

**“Правильный” синтаксис**

Согласно спецификации ANSI на SQL, предпочтительнее использование синтаксиса INNER JOIN.

**Объединение многих таблиц**

SQL не ограничивает число таблиц, которые могут быть объединены посредством оператора SELECT. Основные правила для создания объединения остаются теми же. Вначале перечисляются все таблицы, затем определяются отношения между ними. Вот пример.

**ВВОД**

```
SELECT prod_name, vend_name, prod_price, quantity
FROM OrderItems, Products, Vendors
WHERE Products.vend_id = Vendors.vend_id
AND OrderItems.prod_id = Products.prod_id
AND order_num = 20007;
```

**ВЫВОД**

| prod_name          | vend_name       | prod_price | quantity |
|--------------------|-----------------|------------|----------|
| -----              | -----           | -----      | -----    |
| 18 inch teddy bear | Bears R Us      | 11.9900    | 50       |
| Fish bean bag toy  | Doll House Inc. | 3.4900     | 100      |



|                     |                        |     |
|---------------------|------------------------|-----|
| Bird bean bag toy   | Doll House Inc. 3.4900 | 100 |
| Rabbit bean bag toy | Doll House Inc. 3.4900 | 100 |
| Raggedy Ann         | Doll House Inc. 4.9900 | 50  |

### Анализ

В этом примере выводятся предметы заказа номер 20007. Предметы заказа хранятся в таблице OrderItems. Каждый продукт хранится в соответствии с идентификатором продукта, который ссылается на продукт в таблице Products. Эти продукты связаны с соответствующими поставщиками в таблице Vendors по идентификатору поставщика, который хранится вместе с каждой записью о продукте. В предложении FROM этого примера перечисляются три таблицы, а предложение WHERE определяет оба названных предложения объединения. Дополнительное предложение WHERE используется затем для фильтрации только предметов заказа 20007.



#### К вопросу о производительности

СУБД обрабатывают объединения, тратя время на обработку каждой указанной таблицы. Этот процесс может оказаться очень ресурсоемким, поэтому не следует использовать объединения таблиц без особой на то необходимости. Чем больше таблиц вы объединяете, тем ниже производительность.



#### Максимальное число таблиц в объединении

Хотя SQL не накладывает каких-либо ограничений на число таблиц в объединении, многие СУБД на самом деле имеют такие ограничения. Обратитесь к документации своей СУБД чтобы узнать, какие ограничения такого рода она налагает (если они есть).

Теперь самое время пересмотреть следующий пример из урока 11, “Использование подзапросов”, где оператор SELECT возвращает список клиентов, заказавших продукт RGN01:

**ВВОД**

```
SELECT cust_name, cust_contact
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Orders.cust_id
  AND OrderItems.order_num = Orders.order_num
  AND prod_id = 'RGAN01';
```

**ВЫВОД**

| cust_name     | cust_contact       |
|---------------|--------------------|
| -----         | -----              |
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |

**Анализ**

Как уже говорилось в уроке 11, возвращение необходимых для этого запроса данных требует использования трех таблиц. Однако вместо использования подчиненных подзапросов здесь были применены два объединения для связи таблиц. Здесь были также указаны три предложения WHERE. Первые два связывают таблицы в объединение, последнее фильтрует данные по продукту RGAN01.

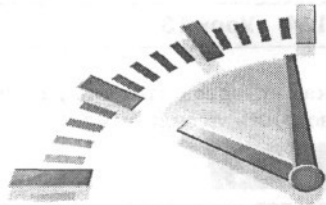
**Только экспериментально**

Как видите, часто существует несколько способов для выполнения одной и той же операции SQL. И редко удается определить, какой способ правильный, а какой нет. Производительность может зависеть от типа выполняемой операции, используемой СУБД, количества данных в таблицах, присутствия либо отсутствия индексов и ключей, а также целого ряда других критериев. Следовательно, зачастую бывает целесообразно поэкспериментировать с различными механизмами выборки для выяснения того, какой из них работает быстрее.

## Резюме

Объединения — одна из самых важных и востребованных особенностей SQL, их эффективное использование возможно только на основе знаний о “конструкции” реляционной базы данных. В этом уроке вы познакомились с основами построения баз данных, а также узнали, как следует создавать объединение по эквивалентности (называемое также внутреннее объединение), которое чаще всего используют при создании объединения. В следующем уроке вы научитесь создавать объединения других типов.

## Урок 13



# Создание расширенных объединений

В этом уроке вы узнаете все необходимое о дополнительных типах объединений — что они собой представляют и как их можно использовать. Вы также узнаете, как следует применять псевдонимы таблиц и как можно использовать статистические функции по отношению к объединенным таблицам.

## Использование псевдонимов таблиц

Ранее, в уроке 7, “Создание вычисляемых полей” вы узнали, как можно использовать псевдонимы в качестве ссылок на выбираемые столбцы таблицы. Синтаксис псевдонимов столбцов выглядит следующим образом:

### ВВОД

```
SELECT RTRIM(vend_name) + ' (' + RTRIM(vend_country)
&+ ')' AS vend_title
FROM Vendors
ORDER BY vend_name;
```

Помимо возможности применения псевдонимов для имен столбцов и вычисляемых полей, SQL позволяет также использовать псевдонимы вместо имен таблиц. На то есть две основных причины:

- более короткий синтаксис SQL;
- это позволяет много раз использовать одну и ту же таблицу в операторе SELECT.

Обратите внимание на следующий оператор SELECT. В основном он такой же, как в примерах предыдущего уро-

ка, но здесь этот оператор был модифицирован для использования псевдонимов:

### ВВОД

```
SELECT cust_name, cust_contact
FROM Customers AS C, Orders AS O, OrderItems AS OI
WHERE C.cust_id = O.cust_id
      AND OI.order_num = O.order_num
      AND prod_id = 'RGAN01';
```

### Анализ

Заметьте, что все три таблицы в предложениях FROM имеют псевдонимы. Выражение Customers AS C задает C в качестве псевдонима для таблицы Customers и т.д., что позволяет использовать сокращение C вместо полного слова Customers. В этом примере псевдонимы таблиц были использованы только в предложении WHERE, но псевдонимы можно применять и в других случаях. Их можно использовать в списке SELECT, предложении ORDER BY, а также в любой другой части этого оператора.



#### Никаких AS в Oracle

СУБД Oracle не поддерживает ключевое слово AS.

Для того чтобы использовать псевдонимы в СУБД Oracle, просто укажите их без ключевого слова AS (т.е. укажите Customers C вместо Customers AS C).

Нет также ничего плохого в том, что псевдонимы таблиц используются только во время выполнения запроса. В отличие от псевдонимов столбцов, псевдонимы таблиц никогда не возвращаются клиенту.

## Использование объединений других типов

До сих пор вы использовали только простые объединения, которые называют внутренние объединения или объединения по эквивалентности. Теперь мы рассмотрим три дополнительных типа объединения: самообъединение, естественное объединение и внешнее объединение.

## Самообъединения

Одна из основных причин для использования псевдонимов таблиц состоит в возможности обращения к одной и той же таблице несколько раз в одном операторе SELECT. Покажем это на примере.

Предположим, вы хотите послать письма по всем контактным адресам клиентов, которые работают с той же компанией, с которой работает Джим Джонс. Такой запрос требует, чтобы вначале вы выяснили, с какой компанией работает Джим Джонс, а затем — какие клиенты работают с этой же компанией. Вот один из способов решения этой задачи.

### ВВОД

```
SELECT cust_id, cust_name, cust_contact
FROM Customers
WHERE cust_name = (SELECT cust_name
FROM Customers
WHERE cust_contact = 'Jim Jones');
```

### ВЫВОД

| cust_id    | cust_name | cust_contact       |
|------------|-----------|--------------------|
| -----      | -----     | -----              |
| 1000000003 | Fun4All   | Jim Jones          |
| 1000000004 | Fun4All   | Denise L. Stephens |

### Анализ

В первом решении используются подзапросы. Внутренний оператор SELECT выполняет простую выборку, чтобы возвратить имя компании (cust\_name), с которой работает Джим Джонс. Только это имя используется в предложении WHERE внешнего запроса, так что выбираются имена всех служащих, работающих с этой компанией. (Все о подзапросах читайте в уроке 11, “Использование подзапросов”).

Теперь рассмотрим тот же самый запрос, в котором используется объединение.

### ВВОД

```
SELECT c1.cust_id, c1.cust_name, c1.cust_contact
FROM Customers AS c1, Customers AS c2
WHERE c1.cust_name = c2.cust_name
AND c2.cust_contact = 'Jim Jones';
```

**Вывод**

| cust_id    | cust_name | cust_contact       |
|------------|-----------|--------------------|
| -----      | -----     | -----              |
| 1000000003 | Fun4All   | Jim Jones          |
| 1000000004 | Fun4All   | Denise L. Stephens |

**Никаких AS в СУБД Oracle**

Пользователи Oracle, не забывайте убирать из своих операторов ключевое слово AS.

**Анализ**

Две таблицы, необходимые для выполнения запроса, на самом деле — одна и та же таблица, поэтому таблица Customers появляется в предложении FROM дважды. Хотя это совершенно законно, некоторые ссылки на таблицу Customers могли бы оказаться неоднозначными, потому что СУБД “не знает”, на какую именно таблицу Customers вы ссылаетесь.

Для решения этой проблемы используются псевдонимы. Первый раз для таблицы Customers назначается псевдоним C1, второй — псевдоним C2. Теперь эти псевдонимы можно применять в качестве имен таблиц. Например, оператор SELECT использует префикс C1 для однозначного указания полного имени нужного столбца. Если этого не сделать, СУБД возвратит сообщение об ошибке, потому что имеется по два столбца с именами cust\_id, cust\_name и cust\_contact. СУБД не может знать, какой именно столбец вы имеете в виду (даже если в действительности это один и тот же столбец). Первое предложение WHERE объединяет эти таблицы, а затем оно фильтрует данные второй таблицы по столбцу cust\_contact, чтобы вернуть только нужные данные.

**Самообъединения вместо подзапросов**

Самообъединения часто используют для замены операторов, применяющих подзапросы, которые выбирают данные из той же таблицы, что и внешний оператор. Хотя конечный результат получается тем же самым, многие СУБД обрабатывают объединения на-

много быстрее, чем подзапросы. Стоит поэкспериментировать с тем и другим, чтобы определить, какой запрос работает быстрее.

## Естественные объединения

Всякий раз, когда объединяются таблицы, по крайней мере один столбец будет появляться более чем в одной таблице (т.е. столбцы, которые объединялись). Обычные объединения (внутренние объединения, которые мы рассмотрели в предыдущем уроке) возвращают все данные, даже многократные вхождения одного и того же столбца. Естественное объединение просто уничтожает эти многократные вхождения, так что в результате возвращается только один столбец.

Естественное объединение — это объединение, в котором вы выбираете только не повторяющиеся столбцы. Обычно это делается при помощи метасимвола (`SELECT *`) для одной таблицы и указания явного подмножества столбцов для всех остальных таблиц. Вот пример:

### ВВОД

```
SELECT C.*, O.order_num, O.order_date, OI.prod_id,
       OI.quantity, OI.item_price
FROM Customers AS C, Orders AS O, OrderItems AS OI
WHERE C.cust_id = O.cust_id
       AND OI.order_num = O.order_num
       AND prod_id = 'RGAN01';
```



### Никаких AS в Oracle

Пользователи Oracle, не забывайте выбрасывать из кода ключевое слово AS.

### Анализ

В этом примере метасимвол используется только для первой таблицы. Все остальные столбцы указаны явно, поэтому никакие дубликаты столбцов не выбираются.

Несомненно, каждое внутреннее объединение, которое вы использовали до сих пор, представляло собой в действительности естественное объединение и, возможно, вам ни-



когда не понадобится внутреннее объединение, не являющееся естественным.

## Внешние объединения

Большинство объединений связывают строки одной таблицы со строками другой, но в некоторых случаях вам может понадобиться включать в результат строки, не имеющие связанных. Например, вы можете использовать объединения для решения следующих задач:

- подсчета количества заказов каждого клиента, включая клиентов, которые еще не сделали заказ;
- составления перечня продуктов с указанием количества заказов на них, включая продукты, которые никто из клиентов не захотел заказывать;
- вычисления средних объемов продаж с учетом клиентов, которые еще не сделали заказ.

В каждом из этих случаев объединение должно включать строки, не имеющие ассоциирующихся с ними строк в связанной таблице. Объединение такого типа называется внешним.



### Разница в синтаксисе

Важно отметить, что синтаксис, используемый при создании внешнего объединения, может несколько отличаться для различных реализаций SQL. Различные формы синтаксиса, описанные в следующем разделе, помогут вам работать с большинством реализаций, но все же обратитесь к документации своей СУБД и уточните, какой синтаксис следует использовать, прежде чем начинать работу.

Следующий оператор `SELECT` позволяет выполнить простое внутреннее объединение. С его помощью выбирается список всех клиентов и их заказы:

### ВВОД

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers INNER JOIN Orders
ON Customers.cust_id = Orders.cust_id;
```

Синтаксис внешнего объединения похож на этот. Для выборки имен всех клиентов, включая тех, которые еще не сделали заказов, можно сделать следующее.

### ВВОД

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers LEFT OUTER JOIN Orders
ON Customers.cust_id = Orders.cust_id;
```

### ВЫВОД

| cust_id    | order_num |
|------------|-----------|
| -----      | -----     |
| 1000000001 | 20005     |
| 1000000002 | NULL      |
| 1000000003 | 20006     |
| 1000000004 | 20007     |
| 1000000005 | 20008     |

### Анализ

Аналогично внутреннему объединению, которое мы рассматривали на прошлом уроке, в этом операторе SELECT используются ключевые слова OUTER JOIN для указания типа объединения (вместо указания его в предложении WHERE). Но, в отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие связанных с ними строк. При использовании синтаксиса OUTER JOIN вы должны использовать ключевое слово RIGHT или LEFT, чтобы указать таблицу, все строки которой будут включены в результат (RIGHT для таблицы, имя которой стоит справа от OUTER JOIN, LEFT — для той, имя которой значится слева).

В предыдущем примере используются ключевые слова LEFT OUTER JOIN для выборки всех строк таблицы, указанной в левой части предложения FROM (таблицы Customers). Чтобы выбрать все строки из таблицы, указанной справа, используйте правое внешнее объединение (RIGHT OUTER JOIN), как показано в следующем примере.

### ВВОД

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers RIGHT OUTER JOIN Orders
ON Orders.cust_id = Customers.cust_id;
```

SQL Server дополнительно поддерживает упрощенный синтаксис внешнего объединения. Чтобы выбрать перечень всех клиентов, включая тех, которые не разместили ни одного заказа, можно сделать следующее.

### ВВОД

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers, Orders
WHERE Customers.cust_id *= Orders.cust_id;
```

### ВЫВОД

| cust_id    | order_num |
|------------|-----------|
| 1000000001 | 20005     |
| 1000000001 | 20009     |
| 1000000002 | NULL      |
| 1000000003 | 20006     |
| 1000000004 | 20007     |
| 1000000005 | 20008     |

### Анализ

Здесь предложение объединения указано в предложении WHERE. Вместо проверки на равенство с помощью оператора = используется оператор \*= для указания того, что в результат должна быть включена каждая строка таблицы Customers. Оператор \*= представляет собой оператор левого внешнего объединения. С его помощью выбираются все строки левой таблицы.

Противоположностью описанного левого внешнего объединения является правое внешнее объединение, его оператор таков: =\*. Это объединение можно использовать для возвращения всех строк таблицы, имя которой находится справа от данного оператора, как показано в следующем примере.

### ВВОД

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers, Orders
WHERE Orders.cust_id =* Customers.cust_id;
```

Еще одна форма внешнего объединения (используемая только в СУБД Oracle) требует использования оператора (+) после имени таблицы, как показано ниже.

**ВВОД**

```
SELECT Customers.cust_id, Orders.order_num
FROM Customers, Orders
WHERE Customers.cust_id (+) = Orders.cust_id
```

**Типы внешнего объединения**

Независимо от формы используемого внешнего объединения всегда существуют только две его основные формы — левое внешнее объединение и правое внешнее объединение. Единственная разница между ними состоит в порядке указания таблиц, которые связываются. Другими словами, левое внешнее объединение может быть превращено в правое внешнее объединение просто за счет изменения порядка указания имен таблиц в предложении FROM или WHERE. А раз так, то эти два типа внешнего объединения могут заменять друг друга, и решение о том, какое именно из них нужно использовать, определяется субъекто удобством выполнения операции.

Существует и другой вариант внешнего объединения — это полное внешнее объединение, которое извлекает все строки из обеих таблиц и связывает между собой те, которые могут быть связаны. В отличие от левого внешнего и правого внешнего объединений, которые включают в результат несвязанные строки только из одной таблицы, полное внешнее объединение включает в результат несвязанные строки из обеих таблиц. Синтаксис полного внешнего объединения таков:

**ВВОД**

```
SELECT Customers.cust_id, Orders.order_num
FROM Orders FULL OUTER JOIN Customers
ON Orders.cust_id = Customers.cust_id;
```

**Поддержка полного внешнего объединения**

Синтаксис полного внешнего объединения не поддерживается в СУБД Access, MySQL, SQL Server и Sybase.

## Использование объединений со статистическими функциями

Как вы узнали из урока 9, “Суммирование данных”, статистические функции применяются для получения статистических сведений о данных. Хотя во всех до сих пор рассмотренных примерах посредством статистических функций получались данные только для одной таблицы, эти функции можно использовать также по отношению к объединениям.

Рассмотрим пример. Допустим, вы хотите выбрать список всех клиентов и число сделанных ими заказов. Чтобы получить искомое, в следующем коде применяется функция COUNT().

### ВВОД

```
SELECT Customers.cust_id, COUNT(Orders.order_num) AS
num_ord
FROM Customers INNER JOIN Orders
ON Customers.cust_id = Orders.cust_id
GROUP BY Customers.cust_id;
```

### ВЫВОД

| cust_id    | num_ord |
|------------|---------|
| -----      | -----   |
| 1000000001 | 2       |
| 1000000003 | 1       |
| 1000000004 | 1       |
| 1000000005 | 1       |

### Анализ

В этом операторе SELECT используются ключевые слова INNER JOIN для связи таблиц Customers и Orders между собой. Предложение GROUP BY служит для получения итоговых данных по клиентам, и, таким образом, обращение к функции COUNT(Orders.order\_num) позволяет подсчитать количество заказов каждого клиента и вернуть результат в виде столбца num\_ord.

Статистические функции можно использовать с объединениями других типов:

**ВВОД**

```
SELECT Customers.cust_id, COUNT(Orders.order_num) AS
    num_ord
FROM Customers LEFT OUTER JOIN Orders
    ON Customers.cust_id = Orders.cust_id
GROUP BY Customers.cust_id;
```

**Никаких AS в Oracle**

Еще раз напоминаем пользователям Oracle о необходимости удаления AS из кода запроса.

**ВЫВОД**

| cust_id    | num_ord |
|------------|---------|
| -----      | -----   |
| 1000000001 | 2       |
| 1000000002 | 0       |
| 1000000003 | 1       |
| 1000000004 | 1       |
| 1000000005 | 1       |

**Анализ**

В этом примере используется левое внешнее объединение для включения в результат всех клиентов, даже тех, которые не сделали ни одного заказа. Как видите, клиент 1000000002 также включен в результат, хотя на данный момент у него было 0 заказов.

## Использование объединений и условий объединения

Прежде чем завершить обсуждение объединений, которое заняло два урока, есть смысл напомнить о некоторых ключевых моментах, относящихся к объединениям и их использованию.

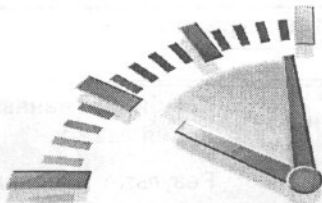
- Будьте внимательны при выборе типа объединения, которое собираетесь использовать. Возможно, что чаще вы будете использовать внутреннее объединение, хотя в зависимости от ситуации это можно также сказать и о внешнем объединении.

- Посмотрите в документации СУБД, какой именно синтаксис объединений она поддерживает. (Многие СУБД поддерживают одну из форм синтаксиса, описанных в этих двух уроках).
- Проверьте, правильно ли вы указали условие объединения (независимо от используемого синтаксиса), иначе будут возвращены неверные данные.
- Не забывайте указывать предложение объединения, в противном случае вы получите декартово произведение.
- Можно включать в объединение много таблиц и даже применять для каждой из них свой тип объединения. Хотя это допустимо и часто оказывается полезным, рекомендуем проверить каждое объединение отдельно, прежде чем применять их вместе. Это намного упростит поиск ошибок.

## Резюме

Этот урок был продолжением предыдущего урока, посвященного объединениям. Начали мы с того, что рассказали, как и для чего используют псевдонимы, а затем продолжили рассмотрение объединений различных типов и различных вариантов синтаксиса, применяемых для каждого из них. Теперь вы знаете, как с объединениями можно использовать статистические функции, а также какие правила важно соблюдать при использовании объединений.

## Урок 14



# Комбинированные запросы

В этом уроке вы узнаете, как использовать оператор *UNION* для комбинирования многих операторов *SELECT* с целью получения одного набора результатов.

## Что такое комбинированные запросы

В большинстве SQL-запросов используется один оператор, посредством которого возвращаются данные из одной или нескольких таблиц. SQL позволяет также выполнять множественные запросы (за счет многократного использования оператора *SELECT*) и возвращать результаты в виде одного набора результатов запроса. Эти комбинированные запросы обычно называют *соединениями* или *сложными запросами*.

Можно назвать два основных сценария, для выполнения которых вам понадобятся сложные запросы:

- для возвращения одинаковым образом структурированных данных из различных таблиц посредством одного запроса;
- для выполнения многократных запросов к одной таблице и возвращения данных в виде результата одного запроса.





### Комбинированные запросы и многократные условия WHERE

Результат комбинирования двух запросов к одной и той же таблице в основном аналогичен результату, полученному при выполнении одного запроса с несколькими требованиями в предложении WHERE. Иначе говоря, как будет показано в следующем разделе, любой оператор SELECT с многократно используемым условием WHERE можно также рассматривать как сложный запрос.

## Создание комбинированных запросов

Запросы в языке SQL комбинируются с помощью оператора UNION. Оператор UNION позволяет многократно указывать оператор SELECT, и по завершении их работы может быть выведен один набор результатов.

### Использование оператора UNION

Использовать оператор UNION довольно просто. Все, что вы должны сделать, — это указать каждый необходимый вам оператор SELECT и разместить ключевое слово UNION между ними.

Рассмотрим пример. Допустим, вам необходим отчет, содержащий сведения обо всех клиентах из штатов Иллинойс, Индиана и Мичиган. Вы также хотите включить в него данные о клиенте Fun4All независимо от штата. Конечно, можно создать условие WHERE, благодаря которому будет выполнено требуемое, но в данном случае гораздо удобнее использовать оператор UNION.

Как уже говорилось, применение оператора UNION подразумевает многократное использование операторов SELECT. Вначале рассмотрим отдельные операторы:

**ВВОД**

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI');
```

**ВЫВОД**

| cust_name     | cust_contact | cust_email           |
|---------------|--------------|----------------------|
| Village Toys  | John Smith   | sales@villagetoy.com |
| Fun4All       | Jim Jones    | jjones@fun4all.com   |
| The Toy Store | Kim Howard   | NULL                 |

**ВВОД**

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All';
```

**ВЫВОД**

| cust_name | cust_contact       | cust_email            |
|-----------|--------------------|-----------------------|
| Fun4All   | Jim Jones          | jjones@fun4all.com    |
| Fun4All   | Denise L. Stephens | dstephens@fun4all.com |

**Анализ**

Первый оператор SELECT выбирает все строки, относящиеся к штатам Иллинойс, Индиана и Мичиган, передавая аббревиатуры этих штатов в условие IN. Второй оператор SELECT использует простую проверку на равенство, чтобы найти все местонахождения в таблицах клиента Fun4All.

Чтобы скомбинировать эти два запроса, выполните следующее.

**ВВОД**

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All';
```

**ВЫВОД**

| cust_name     | cust_contact      | cust_email            |
|---------------|-------------------|-----------------------|
| -----         | -----             | -----                 |
| Fun4All       | Denise L.Stephens | dstephens@fun4all.com |
| Fun4All       | Jim Jones         | jjones@fun4all.com    |
| Village       | Toys John Smith   | sales@villagetoy.com  |
| The Toy Store | Kim Howard        | NULL                  |

**Анализ**

Операторы предыдущего примера состоят из обоих предшествующих операторов SELECT, разделенных ключевым словом UNION. Оператор UNION указывает СУБД выполнить оба оператора SELECT и вывести результаты в виде одного набора результатов запроса.

Для сравнения приводим тот же самый запрос, использующий не оператор UNION, а несколько предложений WHERE:

**ВВОД**

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
OR cust_name = 'Fun4All';
```

В нашем простом примере применение оператора UNION может оказаться более сложным, чем использование предложения WHERE. Однако если условие фильтрации окажется более сложным или если понадобится выбирать данные из многих таблиц (а не только из одной), то оператор UNION может значительно упростить процесс.

**Ограничения оператора UNION**

В стандартном SQL не существует ограничений на число операторов SELECT, которые могут быть скомбинированы посредством операторов UNION. Однако лучше все же обратиться к документации СУБД и убедиться в том, что она не накладывает каких-либо ограничений на максимально допустимое число операторов.



### Проблемы, связанные с производительностью

В большинстве хороших СУБД используется внутренний оптимизатор запросов, комбинирующий операторы SELECT, прежде чем СУБД начинает их обработку. Теоретически это означает, что, с точки зрения производительности, нет реальной разницы между использованием многих предложений WHERE и оператора UNION. Мы говорим "теоретически", потому что на практике многие оптимизаторы запросов не всегда выполняют свою работу так хорошо, как следовало бы. Лучшим вариантом было бы протестировать оба метода и посмотреть, какой из них вам лучше подходит.

## Правила применения запросов UNION

Как видите, запросы UNION очень просты в использовании. Но существует несколько правил, четко указывающих, что именно может быть объединено.

- Запрос UNION должен включать два или более операторов SELECT, отделенных один от другого ключевым словом UNION (таким образом, если в запросе используется четыре оператора SELECT, должно быть использовано три ключевых слова UNION).
- Каждый запрос в операторе UNION должен содержать одни и те же столбцы, выражения или статистические функции (кроме того, столбцы должны быть перечислены в одном и том же порядке).
- Типы данных столбцов должны быть совместимыми. Они не обязательно должны быть одного типа, но они должны быть того типа, который СУБД сможет однозначно преобразовать (например, это могут быть различные числовые типы данных или различные типы даты).

При соблюдении этих основных правил и ограничений, запросы на соединение можно использовать для решения любых задач по возвращению данных.

## Включение или исключение повторяющихся строк

Возвратимся к одному из предыдущих разделов “Использование оператора UNION” и рассмотрим использованные в нем простые операторы SELECT. Вы можете заметить, что, когда они выполняются отдельно, первый оператор SELECT возвращает три строки, второй — две. Однако когда эти два оператора SELECT комбинируются с UNION, возвращаются только четыре строки, а не пять.

Запрос UNION автоматически удаляет все повторяющиеся строки из набора результатов запроса (иными словами, он ведет себя точно так же, как вели бы себя несколько предложений WHERE в одном операторе SELECT). Поэтому здесь присутствует запись о клиенте Fun4All из штата Индиана — эта строка была возвращена обоими операторами SELECT. Когда же использовался запрос UNION, повторяющаяся строка была удалена.

Таково поведение запроса UNION по умолчанию, но при желании вы можете изменить его. Если бы требовалось, чтобы возвращались все вхождения соответствий, вам следовало бы использовать UNION ALL вместо оператора UNION.

Рассмотрим следующий пример:

### ВВОД

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION ALL
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All';
```

### ВЫВОД

| cust_name    | cust_contact       | cust_email            |
|--------------|--------------------|-----------------------|
| Village Toys | John Smith         | sales@villagetoy.com  |
| Fun4All      | Jim Jones          | jjones@fun4all.com    |
| The Toy      | Store Kim Howard   | NULL                  |
| Fun4All      | Jim Jones          | jjones@fun4all.com    |
| Fun4All      | Denise L. Stephens | dstephens@fun4all.com |

**Анализ**

При использовании запроса UNION ALL СУБД не удаляет дубликаты. Поэтому в предыдущем примере возвращено пять строк, одна из них повторяется дважды.

**UNION или WHERE**

В начале этого урока мы говорили, что оператор UNION выполняет то же самое, что и несколько условий WHERE. Оператор UNION ALL является формой запроса UNION, которая делает то, что не способны выполнить предложения WHERE. Если вы хотите получить все вхождения соответствий для каждого условия (включая дубликаты), вам следует использовать оператор UNION ALL, а не WHERE.

## Сортировка результатов комбинированных запросов

Результат применения оператора SELECT сортируется с помощью предложения ORDER BY. При комбинировании запросов посредством UNION только одно предложение ORDER может быть использовано, и оно должно появиться после заключительного оператора SELECT. Практически не имеет смысла сортировать часть набора результатов одним способом, а часть — другим, поэтому несколько предложений ORDER BY применять не разрешается.

В следующем примере сортируются результаты, полученные предыдущим запросом UNION:

**ВВОД**

```
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All'
ORDER BY cust_name, cust_contact;
```

**Вывод**

| cust_name     | cust_contact      | cust_email            |
|---------------|-------------------|-----------------------|
| -----         | -----             | -----                 |
| Fun4All       | Denise L.Stephens | dstephens@fun4all.com |
| Fun4All       | Jim Jones         | jjones@fun4all.com    |
| The Toy Store | Kim Howard        | NULL                  |
| Village Toys  | John Smith        | sales@villagetoys.com |

**Анализ**

Этот запрос UNION использует одно предложение ORDER BY после заключительного оператора SELECT.

Несмотря на то что ORDER BY является частью только последнего оператора SELECT, на самом деле СУБД будет использовать его для сортировки всех результатов, возвращенных всеми операторами SELECT.

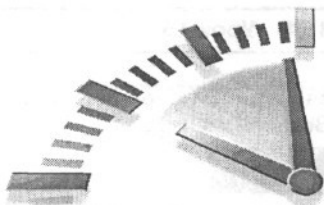
**Другие типы запроса на соединение**

Некоторые СУБД поддерживают два дополнительных запроса типа UNION. Оператор EXCEPT (иногда называемый MINUS) может быть использован только для чтения строк, которые существуют в первой таблице, но не во второй, а оператор INTERSECT можно использовать для чтения строк, которые имеются в обеих таблицах. Однако на практике такие запросы UNION используются редко, поскольку те же самые результаты могут быть получены посредством объединений.

**Резюме**

В этом уроке вы узнали, как можно комбинировать запросы SELECT посредством оператора UNION. Используя оператор UNION, вы можете вернуть результаты нескольких запросов в виде одного комбинированного запроса, включающего или исключающего дубликаты. За счет использования оператора UNION можно значительно упростить сложные предложения WHERE и одновременно выбирать данные из многих таблиц.

## Урок 15



# Добавление данных

В этом уроке вы узнаете, как можно добавлять данные в таблицы, используя оператор `INSERT` языка SQL.

## Что такое добавление данных

Несомненно, `SELECT` является наиболее часто используемым оператором языка SQL (именно поэтому мы посвятили его рассмотрению 14 уроков). Но помимо него в SQL часто применяются еще три оператора, которыми вам необходимо уметь пользоваться. Первый из них — оператор `INSERT`. (О двух других мы расскажем в следующем уроке.)

Как следует из названия, оператор `INSERT` используется для ввода (добавления) строк в таблицу базы данных. Добавление можно осуществить несколькими способами:

- добавить одну полную строку;
- добавить часть одной строки;
- добавить результаты запроса.

Далее мы рассмотрим все эти варианты.



### Оператор `INSERT` и безопасность системы

Для использования оператора `INSERT` могут потребоваться особые права на доступ в СУБД со структурой клиент-сервер. Прежде чем применять оператор `INSERT`, убедитесь в том, что у вас есть на это право.

## Добавление полных строк

Простейший способ добавления данных в таблицу может быть реализован при использовании основного синтаксиса оператора `INSERT`. Для этого нужно указать имя таблицы и



значения, которые должны быть введены в новую строку. Вот пример:

### ВВОД

```
INSERT INTO Customers
VALUES ('1000000006',
'Toy Land',
'123 Any Street',
'New York',
'NY',
'11111',
'USA',
NULL,
NULL);
```

В этом примере в таблицу добавляются сведения о новом клиенте. Данные, которые должны быть сохранены в каждом столбце таблицы, указываются в условии VALUES, значения должны быть приведены для каждого столбца. Если для какого-то столбца не имеется соответствующего значения (например, как это произошло для столбцов cust\_contact и cust\_email в данном примере), следует использовать значение NULL (предполагается, что для данной таблицы разрешено не указывать значения в этих столбцах). Столбцы должны заполняться в порядке, в котором они появились в определении таблицы.



#### Ключевое слово INTO

В некоторых реализациях SQL вслед за оператором INSERT опционально указывается ключевое слово INTO. Однако хорошим тоном считается указание этого ключевого слова даже в случаях, когда это не является необходимым. Поступая таким образом, вы обеспечите переносимость своего кода между СУБД.

Этот синтаксис довольно прост, но он не вполне безопасен, поэтому его применения следует всячески избегать. Результаты применения вышеприведенного оператора SQL весьма чувствительны к порядку, в котором столбцы определены в таблице. Они также зависят от того, соблюдается ли в действительности этот порядок. Однако даже если в данный момент порядок соблюдается, нет гарантий, что столбцы будут расположены в том же самом порядке, когда таблица будет реконструироваться в следующий раз. Следо-

вательно, использовать оператор SQL, результаты применения которого зависят от порядка следования столбцов, весьма небезопасно. Если вы будете пренебрегать этим советом, вас ждут неприятности.

Безопасный (и, к сожалению, более громоздкий) способ записи оператора INSERT таков:

## ВВОД

```
INSERT INTO Customers(cust_id,  
cust_name,  
cust_address,  
cust_city,  
cust_state,  
cust_ZIP,  
cust_country,  
cust_contact,  
cust_email)  
VALUES('1000000006',  
'Toy Land',  
'123 Any Street',  
'New York',  
'NY',  
'11111',  
'USA',  
NULL,  
NULL);
```

## Анализ

В этом примере делается в точности то же самое, что и в предыдущем варианте применения оператора INSERT, но на этот раз имена столбцов явно указаны в круглых скобках, следующих после имени таблицы. Когда строка вводится в таблицу, СУБД устанавливает соответствие каждого предмета в списке столбцов с соответствующим значением в списке VALUES. Первое значение в списке VALUES соответствует первому указанному имени столбца, второе значение соответствует имени второго столбца и т.д.

Поскольку имена столбцов представлены, условие VALUES должно подобрать названные имена столбцов в порядке, в котором указаны столбцы, причем не обязательно в порядке, в каком они следуют в реальной таблице. Преимущество этого способа таково: даже если расположение столбцов в таблице изменяется, оператор INSERT все равно будет работать корректно.

Следующий оператор `INSERT` заполняет все столбцы строки (так же, как и в предыдущем примере), но делает это в другом порядке. Поскольку имена столбцов указываются, добавление будет выполнено правильно:

### ВВОД

```
INSERT INTO Customers (cust_id,
  cust_contact,
  cust_email,
  cust_name,
  cust_address,
  cust_city,
  cust_state,
  cust_ZIP,
VALUES ('1000000006',
  NULL,
  NULL,
  'Toy Land',
  '123 Any Street',
  'New York',
  'NY',
  '11111',
```



#### Всегда используйте список столбцов

Как правило, оператор `INSERT` не используется без явного указания списка столбцов. Благодаря этому значительно возрастает вероятность того, что вы сможете продолжать работу, даже если в таблице произойдут изменения.



#### Аккуратно используйте предложение `VALUES`

Независимо от синтаксиса, используемого для оператора `INSERT`, должны быть правильно указаны значения в предложении `VALUES`. Если имена столбцов не указываются, должно быть указано значение для каждого столбца таблицы. Если имена столбцов указываются, должно наличествовать какое-то значение для каждого столбца, включенного в список. Если что-то не указано, будет сгенерировано сообщение об ошибке, и строка не будет вставлена.

## Добавление части строки

Рекомендуемый в предыдущем разделе способ использования оператора INSERT состоит в явном указании имен столбцов таблицы. Используя такой синтаксис, вы также получаете возможность пропустить некоторые столбцы. Это означает, что вы вводите значения для одних столбцов и не предлагаете для других.

Рассмотрим следующий пример:

### ВВОД

```
INSERT INTO Customers(cust_id,
  cust_name,
  cust_address,
  cust_city,
  cust_state,
  cust_ZIP,
  cust_country)
VALUES('1000000006',
  'Toy Land',
  '123 Any Street',
  'New York',
  'NY',
  '11111',
  'USA');
```

### Анализ

В примере, приведенном ранее в этом уроке, значения не предлагаются для двух столбцов, `cust_contact` и `cust_email`. Это означает, что в данном случае нет причин включать эти столбцы в оператор INSERT. Поэтому данный оператор INSERT не включает эти два столбца и два соответствующих им значения.



#### Попуск столбцов

Вы можете исключать некоторые столбцы из операции INSERT, если это позволяет делать определение таблицы. Должно соблюдаться одно из следующих условий:

- Этот столбец определен как допускающий значения NULL (отсутствие какого-либо значения).

- В определении таблицы указано значение по умолчанию. Это означает, что, если не указано никакое значение, будет использовано значение по умолчанию.

Если вы пропускаете столбец таблицы, которая не допускает появления в своих строках значений NULL и не имеет значения, определенного для использования по умолчанию, СУБД выдаст сообщение об ошибке, и эта строка не будет добавлена.

## Добавление выбранных данных

Обычно оператор INSERT служит для добавления строки в таблицу с использованием указанных значений. Существует и другая форма оператора INSERT, она может быть использована для добавления в таблицу результата применения оператора SELECT. Известна эта форма как оператор INSERT SELECT и, как подсказывает название последнего, данный оператор выполняет то же самое, что делают операторы INSERT и SELECT.

Предположим, вы хотите ввести в таблицу Customers список клиентов из другой таблицы. Вместо того чтобы считывать по одной строке и затем добавлять ее посредством оператора INSERT, вы можете сделать следующее.



### Инструкции для следующего примера

В следующем примере данные импортируются из таблицы CustNew в таблицу Customers. Сначала создайте и наполните таблицу CustNew. Формат таблицы CustNew должен быть таким же, как и таблицы Customers, описанной в приложении А. После заполнения CustNew удостоверьтесь в том, что не были использованы значения cust\_id, которые уже применялись в таблице Customers (последующая операция INSERT потерпит неудачу, если значения первичного ключа будут повторяться).

**ВВОД**

```

INSERT INTO Customers(cust_id, cust_contact,
cust_email,
cust_name,
cust_address,
cust_city,
cust_state,
cust_ZIP,
cust_country)
SELECT cust_id,
cust_contact,
cust_email,
cust_name,
cust_address,
cust_city,
cust_state,
cust_ZIP,
cust_country
FROM CustNew;

```

**Анализ**

В этом примере для импорта всех данных из таблицы CustNew в таблицу Customers используется оператор INSERT SELECT. Вместо того чтобы перечислять значения, которые должны быть добавлены, оператор SELECT выбирает их из таблицы CustNew. Каждый столбец в операторе SELECT соответствует столбцу в списке указанных столбцов. Сколько же строк добавит этот оператор? Это зависит от того, сколько строк содержится в таблице CustNew. Если таблица пуста, никакие строки добавлены не будут (и никакое сообщение об ошибке не будет выдано, поскольку эта операция остается правомерной). Если таблица содержит данные, все они будут добавлены в таблицу Customers.

**Имена столбцов в операторе INSERT SELECT**

В этом примере были использованы одинаковые имена столбцов в операторах INSERT и SELECT. Учтите, что к именам столбцов не предъявляются никакие требования. В действительности СУБД вообще не обращает внимания на имена столбцов, возвращаемых оператором SELECT. Точнее, ею используется положение столбца, так что первый столбец в SELECT (независимо от имени) будет использован для заполнения первого указанного столбца таблицы и т.д.

Оператор `SELECT`, используемый в `INSERT SELECT`, может включать предложение `WHERE` для фильтрации данных, которые должны быть добавлены.



### Добавление нескольких строк

Оператор `INSERT` обычно добавляет только одну строку. Чтобы добавить несколько строк, нужно выполнить несколько операторов `INSERT`. Исключением из этого правила является оператор `INSERT SELECT`, который может быть использован для добавления многих строк посредством одного оператора — какие бы данные ни возвратил оператор `SELECT`, они могут быть добавлены в таблицу посредством оператора `INSERT`.

## Копирование данных из одной таблицы в другую

Это — другая форма добавления данных, при использовании которой оператор `INSERT` вообще не применяется. Чтобы скопировать содержимое какой-то таблицы в новую (которая создается “на лету”), можно использовать оператор `SELECT INTO`.



### Не поддерживается в DB2

СУБД DB2 не поддерживает использование оператора `SELECT INTO` описанным выше способом.

В отличие от оператора `INSERT SELECT`, посредством которого данные добавляются в уже существующую таблицу, `SELECT INTO` копирует данные в новую таблицу (и в зависимости от того, о какой СУБД идет речь, может перезаписать таблицу, если такая уже существует).



### Разница между `INSERT SELECT` и `SELECT INTO`

Одно из отличий между операторами `SELECT INTO` и `INSERT SELECT` состоит в том, что первый оператор экспортирует данные, а второй — импортирует.

Следующий пример демонстрирует способ применения оператора `SELECT INTO`:

### ВВОД

```
SELECT *
INTO CustCopy
FROM Customers;
```

### Анализ

Этот оператор `SELECT` создает новую таблицу с именем `CustCopy` и копирует в нее все содержимое таблицы `Customers`.

Поскольку был использован оператор `SELECT *`, каждый столбец таблицы `Customers` будет создан в таблице `CustCopy` (и соответственно заполнен). Чтобы скопировать только часть доступных столбцов, следует явно указать имена столбцов, а не использовать метасимвол `*` (звездочка).

В СУБД `MySQL` и `Oracle` используется несколько иной синтаксис:

### ВВОД

```
CREATE TABLE CustCopy AS
SELECT *
FROM Customers;
```

### Анализ

При использовании оператора `SELECT INTO` нужно обращать внимание на следующие моменты.

- Можно использовать любые опции и предложения оператора `SELECT`, включая `WHERE` и `GROUP BY`.
- Для добавления данных из нескольких таблиц можно использовать объединения.
- Данные можно добавить только в одну таблицу независимо от того, из скольких таблиц они были извлечены.





### Создание копий таблиц

Оператор `SELECT INTO` является прекрасным средством создания копий таблиц для экспериментов с новыми для вас операторами SQL. Создав копию, вы получите возможность проверить возможности SQL на этой копии, а не на таблицах реальной базы данных.



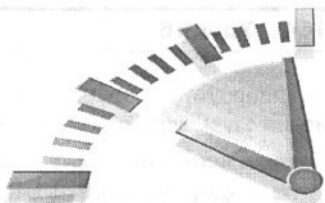
### Больше примеров

Вам нужны еще примеры использования оператора `INSERT`? Ознакомьтесь со сценариями заполнения таблиц, описанными в приложении А.

## Резюме

В этом уроке вы узнали о том, как можно добавлять строки в таблицу базы данных. Вы познакомились с несколькими способами использования оператора `INSERT` и узнали, почему желательно явно указывать имена столбцов. Вы научились использовать оператор `INSERT SELECT` для импорта строк из другой таблицы и применять оператор `SELECT INTO` для экспорта строк в новую таблицу. В следующем уроке мы расскажем о том, как для манипулирования данными таблиц следует использовать операторы `UPDATE` и `DELETE`.

## Урок 16



# Обновление и удаление данных

В этом уроке вы узнаете о том, как нужно использовать операторы `UPDATE` и `DELETE`, позволяющие осуществлять манипуляции с данными вашей таблицы.

## Обновление данных

Для обновления (модификации) данных какой-либо таблицы используется оператор `UPDATE`. Этот оператор можно использовать двумя способами:

- для обновления определенной строки таблицы;
- для обновления всех строк таблицы.

Рассмотрим каждый из названных способов.



### Не пропускайте предложение `WHERE`

Применять оператор `UPDATE` нужно особенно аккуратно, потому что с его помощью можно ошибочно обновить все строки таблицы. Прочитайте весь раздел, посвященный оператору `UPDATE`, прежде чем использовать этот оператор.



### Оператор `UPDATE` и безопасность

Для использования оператора `UPDATE` в СУБД со структурой клиент-сервер могут понадобиться особые права доступа. Прежде чем применять этот оператор, удостоверьтесь в том, что вам предоставлены соответствующие права.

Оператор UPDATE очень прост в использовании, он состоит из трех основных частей:

- имени таблицы, подлежащей обновлению;
- имен столбцов и их новых значений;
- условий фильтрации, определяющих, какие именно строки должны быть обновлены.

Рассмотрим простой пример. Допустим, у клиента 1000000005 появился адрес электронной почты, поэтому его запись нужно обновить. Такое обновление можно выполнить посредством следующего оператора:

### ВВОД

```
UPDATE Customers
SET cust_email = 'kim@thetoystore.com'
WHERE cust_id = '1000000005';
```

Оператор UPDATE всегда начинается с имени таблицы, подлежащей обновлению. В нашем примере это таблица Customers. Затем используется команда SET, чтобы ввести в столбец новое значение. В нашем случае предложение SET устанавливает определенное значение для столбца cust\_email:

```
SET cust_email = 'kim@thetoystore.com'
```

Заканчивается оператор UPDATE предложением WHERE, которое сообщает СУБД, какая строка подлежит обновлению. При отсутствии предложения WHERE СУБД обновила бы все строки таблицы Customers, введя в них новый (причем один и тот же!) адрес электронной почты; это, конечно, не то, что требовалось.

Для обновления нескольких столбцов необходим иной синтаксис:

### ВВОД

```
UPDATE Customers
SET cust_contact = 'Sam Roberts',
    cust_email = 'sam@toyland.com'
WHERE cust_id = '1000000006';
```

Для обновления нескольких столбцов используется только одна команда SET, и каждая пара столбец-значение отделяется от другой запятой (после имени последнего столбца запятая не ставится). В нашем примере оба столбца, cust\_contact и cust\_email, будут обновлены для клиента 1000000006.

**Использование подзапросов в операторе UPDATE**

В операторах UPDATE могут быть использованы подзапросы, что дает возможность обновлять столбцы с данными, выбранными посредством оператора SELECT. Вернитесь к уроку 11, “Использование подзапросов”, за дополнительной информацией о подзапросах и их использовании.

**Ключевое слово FROM**

Некоторые реализации SQL поддерживают предложение FROM в операторе UPDATE, оно может быть использовано для обновления строк одной таблицы данными из другой. Обратитесь к документации своей СУБД и выясните, поддерживает ли она эту особенность.

Чтобы удалить значения столбца, вы можете присвоить его строкам значения NULL (если определение таблицы позволяет вводить в нее значения NULL). Это можно сделать следующим образом:

**ВВОД**

```
UPDATE Customers
SET cust_email = NULL
WHERE cust_id = '1000000005';
```

Здесь ключевое слово NULL используется для хранения “никакого” значения в столбце cust\_email.

## Удаление данных

Для удаления данных из таблицы применяется оператор DELETE.

Его можно использовать двумя способами:

- для удаления из таблицы определенных строк;
- для удаления из таблицы всех ее строк.

**Не забывайте указывать предложение WHERE**

Применять оператор DELETE следует с особой осторожностью, потому что можно ошибочно удалить все строки таблицы. Прочитайте весь раздел, посвященный оператору DELETE, прежде чем его использовать.

**Оператор DELETE и безопасность**

Для использования оператора DELETE в СУБД со структурой клиент-сервер могут понадобиться особые права доступа. Прежде чем применять этот оператор, удостоверьтесь в том, что вам предоставлены соответствующие права.

Рассмотрим каждый из этих способов.

Как уже говорилось, оператор UPDATE очень прост в использовании. Хорошая (и вместе с тем плохая) новость состоит в том, что оператор DELETE еще проще.

Следующий оператор удаляет одну строку из таблицы Customers:

**ВВОД**

```
DELETE FROM Customers
WHERE cust_id = '1000000006';
```

Оператор DELETE FROM требует, чтобы вы указали имя таблицы, из которой должны быть удалены данные. Предложение WHERE фильтрует строки, определяя, какие из них должны быть удалены. В нашем примере должна быть удалена строка, относящаяся к клиенту 1000000006. Если бы предложение WHERE было пропущено, этот оператор удалил бы все столбцы таблицы.

**Ключевое слово FROM**

В некоторых реализациях SQL за DELETE может опционально следовать ключевое слово FROM. Однако хорошим тоном считается всегда указывать это ключевое слово, даже если в нем нет необходимости. Поступая таким образом, вы обеспечите переносимость своего SQL-кода между СУБД.

Оператор `DELETE` не принимает имена столбцов или метасимволы. Он удаляет строки целиком, а не отдельные столбцы. Для удаления определенного столбца следует использовать оператор `UPDATE`.



#### Содержимое таблиц, но не сами таблицы

Оператор `DELETE` удаляет из таблицы отдельные строки или даже все строки за один раз, но он никогда не удаляет саму таблицу.



#### Более быстрое удаление

Если необходимо удалить значения из всех строк таблицы, не используйте оператор `DELETE`. Вместо него нужно применить оператор `TRUNCATE TABLE`, который выполняет то же самое, но делает это намного быстрее (потому что изменения данных не регистрируются).

## Советы по обновлению и удалению данных

Все операторы `UPDATE` `DELETE`, рассмотренные в предыдущем разделе, сопровождалась предложениями `WHERE`, и на это есть очень веская причина. Если вы пропустите предложение `WHERE`, оператор `UPDATE` или `DELETE` будет применен по отношению ко всем строкам таблицы. Другими словами, если вы выполните оператор `UPDATE` без предложения `WHERE`, каждая строка таблицы будет заменена новыми значениями. Аналогичным образом, если вы выполните оператор `DELETE` без предложения `WHERE`, будет удалено все содержимое таблицы.

Далее перечислены правила хорошего тона, которым следуют программисты `SQL`.

- Никогда не выполняйте оператор `UPDATE` или `DELETE` без предложения `WHERE`, если только вы на самом деле не хотите обновить или удалить каждую строку.

- Убедитесь в том, что каждая таблица имеет первичный ключ (вернитесь к уроку 12, “Объединение таблиц”, если забыли, что это такое), и используйте его в предложении WHERE всякий раз, когда это оказывается возможным. (Вы можете указать отдельные первичные ключи, несколько значений или диапазоны значений.)
- Прежде чем использовать предложение WHERE с оператором UPDATE или DELETE, сначала проверьте его с оператором SELECT, чтобы убедиться в том, что оно правильно фильтрует записи, — можно ошибиться и сформулировать неправильное предложение WHERE.
- Используйте средства принудительного обеспечения ссылочной целостности данных (см. урок 12), чтобы СУБД не позволяла удалять строки, для которых в других таблицах имеются связанные с ними данные.
- Некоторые СУБД позволяют администраторам баз данных устанавливать ограничения, препятствующие выполнению операторов UPDATE или DELETE без предложения WHERE. Если ваша СУБД поддерживает эту особенность, рассмотрите возможность ее использования.



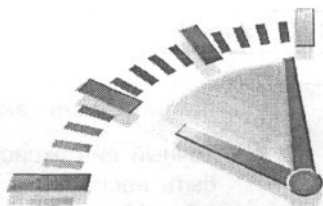
#### Используйте операторы удаления с осторожностью

Основной момент состоит в том, что SQL не имеет кнопки возврата в предыдущее состояние. Будьте очень внимательны, используя операторы UPDATE и DELETE, иначе вы вдруг обнаружите, что удалили или обновили не те данные.

## Резюме

В этом уроке вы узнали, как нужно применять операторы UPDATE и DELETE для манипулирования данными таблиц. Вы познакомились с синтаксисом каждого из этих операторов, а также с опасностями, которыми чревато их применение. Вы также узнали, почему столь важно использовать предложение WHERE в операторах UPDATE и DELETE, и познакомились с основными правилами, которым нужно следовать, чтобы по неосторожности не повредить данные.

## Урок 17



# Создание таблиц и работа с ними

В этом уроке вы познакомитесь с основными правилами создания, перестройки и удаления таблиц.

## Создание таблиц

Язык SQL используется не только для манипуляций с данными таблиц, он предназначен для выполнения всех операций с базами данных и таблицами, включая собственно создание таблиц и работу с ними.

Существует два способа создания таблиц.

- Большинство СУБД сопровождается инструментарием администратора, который можно использовать для интерактивного создания таблиц базы данных и управления ими.
- Таблицами можно также манипулировать посредством операторов языка SQL.

Для создания таблиц программным способом используют оператор SQL `CREATE TABLE`. Стоит отметить, что, когда вы используете интерактивный инструмент, в действительности вся работа выполняется операторами SQL. Однако вам не приходится писать эти операторы; интерфейс создает и выполняет их незаметно для вас (то же самое справедливо и для процедуры изменения существующих таблиц).





### Разница в синтаксисе

Точный синтаксис оператора `CREATE TABLE` может быть несколько различным для разных реализаций SQL. Обязательно обратитесь к документации своей СУБД за дополнительной информацией и выясните, какой в точности синтаксис для нее необходим и какие возможности она поддерживает.

Полное рассмотрение всех опций, применяемых при создании таблиц, не входит в задачи нашего урока, мы рассмотрим только основы. Для получения дополнительной и специфичной для вашей СУБД информации настоятельно рекомендуем обратиться к ее документации.



### Примеры для конкретных СУБД

Операторы `CREATE TABLE` для конкретных СУБД приведены в примерах сценариев создания таблиц (см. приложение А, "Сценарии демонстрационных таблиц").

## Основы создания таблиц

Чтобы создать таблицу с помощью оператора `CREATE TABLE`, нужно указать следующие данные:

- имя новой таблицы; оно вводится после ключевого слова `CREATE TABLE`;
- имена и определения столбцов таблицы, разделенные запятыми;
- в некоторых СУБД также требуется, чтобы было указано место размещения таблицы.

Посредством следующего оператора SQL создается таблица `Products`, часто используемая в нашей книге:

### ВВОД

```
CREATE TABLE Products
(
  prod_id CHAR(10) NOT NULL,
  vend_id CHAR(10) NOT NULL,
  prod_name CHAR(254) NOT NULL,
  prod_price DECIMAL(8,2) NOT NULL,
```

```
prod_desc VARCHAR(1000) NULL  
);
```

## Анализ

Как видите, имя таблицы указывается сразу же после ключевых слов CREATE TABLE.

Определение таблицы (все ее столбцы) заключается в круглые скобки. Имена столбцов разделяются запятыми. Приведенная в примере таблица состоит из пяти столбцов. Определение каждого столбца начинается с имени столбца (которое должно быть уникальным в пределах данной таблицы), за ним указывается тип данных. (Обратитесь к уроку 1, “Что такое SQL”, чтобы вспомнить, что такое типы данных. Кроме того, в приложении Г, “Использование типов данных SQL”, приведен перечень часто используемых типов данных и сведения об их совместимости.) Оператор в целом заканчивается символом “точка с запятой”, следующим после закрывающей круглой скобки.

Ранее уже говорилось, что синтаксис оператора CREATE TABLE весьма различен для разных СУБД, и только что представленный нами простой сценарий доказывает это. В СУБД Oracle, PostgreSQL, SQL Server и Sybase представленный оператор будет работать в той форме, в которой он представлен в примере, а вот в MySQL тип VARCHAR должен быть заменен типом text. В DB2 значение NULL следует удалить из последнего столбца. Именно поэтому нами были предложены различные сценарии создания таблиц SQL для каждой СУБД (см. приложение А).



### Форматирование оператора

Пробелы игнорируются операторами SQL. Оператор можно ввести в одной длинной строке или разбить ее на несколько строк, разницы между ними не будет. Это позволяет форматировать выражения SQL так, как вам удобно. Показанный выше оператор CREATE TABLE — хороший пример форматирования оператора SQL. Код разбит на несколько строк, определения столбцов разнесены для удобства чтения и редактирования. Форматировать выражения SQL подобным образом не обязательно, но все же настоятельно рекомендуется.



### Замена существующих таблиц

Когда вы создаете новую таблицу, указываемое вами имя не должно существовать в СУБД, иначе будет выдано сообщение об ошибке. Чтобы избежать случайной перезаписи, SQL требует, чтобы вы вначале вручную удалили таблицу (подробности освещены в следующем разделе), а затем вновь создали ее, а не просто перезаписали.

## Работа со значениями NULL

В уроке 4, “Фильтрация данных”, рассказывалось о том, что такое значение NULL. Использование этого значения подразумевает, что в столбце не должно содержаться никакое значение или неизвестно значение, которое должно быть в столбце. Столбец, в котором разрешается присутствие значения NULL, позволяет также добавлять в таблицу строки, в которых не предусмотрено значение для данного столбца. Столбец, в котором не разрешается присутствие значения NULL, не принимает строки с отсутствующим значением. Иными словами, для этого столбца всегда потребуется вводить какое-то значение при добавлении или обновлении строк.

Каждый столбец таблицы может быть или пустым (NULL), или не пустым (NOT NULL), и это его состояние оговаривается в определении таблицы во время ее создания. Рассмотрим следующий пример:

### ВВОД

```
CREATE TABLE Orders
(
  order_num INTEGER NOT NULL,
  order_date DATETIME NOT NULL,
  cust_id CHAR(10) NOT NULL
);
```

### Анализ

Посредством этого оператора создается таблица Orders, неоднократно использованная в книге. Таблица Orders состоит из трех столбцов: номер заказа (order number), дата

заказа (order date) и идентификатор клиента (customer ID). Все три столбца являются необходимыми, каждый содержит ключевое слово NOT NULL, которое будет препятствовать добавлению в таблицу столбцов с отсутствующим значением. При попытке добавления такого столбца будет возвращено сообщение об ошибке, и добавить такую запись не удастся.

В следующем примере создается таблица, в которой могут быть столбцы обеих разновидностей, NULL и NOT NULL:

## ВВОД

```
CREATE TABLE Vendors
(
  vend_id CHAR(10) NULL,
  vend_name CHAR(50) NOT NULL,
  vend_address CHAR(50),
  vend_city CHAR(50),
  vend_state CHAR(5),
  vend_ZIP CHAR(10),
  vend_country CHAR(50)
);
```

## Анализ

Посредством этого оператора создается таблица Vendors, неоднократно используемая в книге. Столбцы с идентификатором поставщика и именем поставщика необходимы, поэтому оба определены как NOT NULL (т.е. не допускающие значение NULL). Пять остальных столбцов допускают значения NULL, поэтому для них не указано требование NOT NULL. Значение NULL является значением по умолчанию, поэтому, если не указано требование NOT NULL, предполагается разрешение на использование значения NULL.



### Указание значения NULL

Во многих СУБД отсутствие ключевых слов NOT NULL трактуется как NULL. Однако не во всех. В СУБД DB2 наличие ключевого слова NULL является обязательным; если оно не указано, генерируется сообщение об ошибке. Обратитесь к документации своей СУБД, чтобы получить исчерпывающую информацию о синтаксисе.



### Первичные ключи и значения NULL

В уроке 1 говорилось о том, что первичные ключи представляют собой столбцы, значения которых уникально идентифицируют каждую строку таблицы. Столбцы, которые допускают отсутствие значений, не могут использоваться в качестве уникальных идентификаторов.



### Что такое NULL

Не удивляйтесь, увидев значения NULL в пустых строках. Значение NULL означает отсутствие значения; это не пустая строка. Если вы указали в коде ' ' (две одинарных кавычки, между которыми ничего нет), это "значение" можно было бы ввести в столбец типа NOT NULL. Пустая строка является допустимым значением; это не означает отсутствие значения. Значения NULL указываются посредством ключевого слова NULL, но не пустой строкой.

## Определение значений по умолчанию

Язык SQL позволяет определять значения по умолчанию, которые будут использованы в том случае, если при добавлении строки какое-то ее значение не указано. Значения по умолчанию определяются с помощью ключевого слова DEFAULT в определениях столбца оператора CREATE TABLE.

Рассмотрим следующий пример:

### ВВОД

```
CREATE TABLE OrderItems
(
  order_num INTEGER NOT NULL,
  order_item INTEGER NOT NULL,
  prod_id CHAR(10) NOT NULL,
  quantity INTEGER NOT NULL DEFAULT 1,
  item_price DECIMAL(8,2) NOT NULL
);
```

**Анализ**

Посредством этого оператора создается таблица `OrderItems`, содержащая отдельные предметы, которые могут быть заказаны. (Сам заказ хранится в таблице `Orders`.) Столбец `quantity` (количество) содержит количество каждого предмета в заказе. В данном примере добавление текста `DEFAULT 1` в описание столбца предписывает СУБД указывать количество, равное 1, если не указано иное.

Значения по умолчанию часто используются для хранения в столбцах даты и денежных единиц. К примеру, системная дата может быть использована как дата по умолчанию путем указания функции или переменной, используемой для ссылки на системную дату. Например, пользователи `MySQL` могли бы указать дату как `DEFAULT CURRENT_DATE()`, в то время как пользователям `Oracle` следовало бы вводить дату как `DEFAULT SYSDATE`, а пользователям `SQL Server` — как `DEFAULT GETDATE()`. К сожалению, команда, используемая для получения системной даты, в каждой СУБД своя. В табл. 17.1 приведен синтаксис для нескольких СУБД (если ваша СУБД не представлена в этом списке, обратитесь к ее документации).

Таблица 17.1. Получение значения даты из системы

| СУБД       | Функция/переменная          |
|------------|-----------------------------|
| Access     | <code>NOW()</code>          |
| DB2        | <code>CURRENT_DATE</code>   |
| MySQL      | <code>CURRENT_DATE()</code> |
| Oracle     | <code>SYSDATE</code>        |
| PostgreSQL | <code>CURRENT_DATE</code>   |
| SQL Server | <code>GETDATE()</code>      |
| Sybase     | <code>GETDATE()</code>      |


**Использование значений DEFAULT вместо значений NULL**

Многие разработчики баз данных используют значения `DEFAULT` вместо столбцов `NULL`. Часто значения `DEFAULT` применяются в столбцах, которые будут использованы в вычислениях или при статистической обработке данных.

## Обновление таблиц

Для того чтобы обновить определения таблицы, следует воспользоваться оператором ALTER TABLE. Хотя все СУБД поддерживают этот оператор, то, что они при этом позволяют вам делать, в значительной степени зависит от реализации СУБД. Ниже приведены несколько соображений по поводу применения оператора ALTER TABLE.

- В идеальном случае структура таблицы вообще не должна меняться после того, как в таблицу введены данные. Вам придется потратить немало времени, пытаясь предугадать будущие потребности в процессе разработки таблиц, чтобы позже не потребовалось вносить в их структуру существенные изменения.
- Все СУБД позволяют добавлять в уже существующие таблицы столбцы, но некоторые ограничивают типы данных, которые могут быть добавлены (а заодно и использование значений NULL и DEFAULT).
- Многие СУБД не позволяют удалять или изменять столбцы в таблице.
- Большинство СУБД разрешают переименовывать столбцы.
- Многие СУБД налагают серьезные ограничения на изменения, которые могут быть сделаны по отношению к заполненным столбцам, и несколько меньшие — по отношению к незаполненным.

Как видите, вносить изменения в существующие таблицы ничуть не проще, чем создавать их заново. Обратитесь к документации вашей СУБД, чтобы уточнить, что можно изменять.

Чтобы изменить таблицу посредством оператора ALTER TABLE, нужно ввести следующую информацию.

- Имя таблицы, подлежащей изменению, после ключевых слов ALTER TABLE. (Таблица с таким именем должна существовать, иначе будет выдано сообщение об ошибке.)
- Список изменений, которые должны быть сделаны. Поскольку добавление столбцов в таблицу — единственная операция, поддерживаемая всеми СУБД, именно ее мы рассмотрим в качестве примера.

**ВВОД**

```
ALTER TABLE Vendors  
ADD vend_phone CHAR(20);
```

**Анализ**

Посредством этого оператора в таблицу Vendors добавляется столбец, названный vend\_phone. Должен быть определен тип данных.

Другие операции изменения, например, изменение или удаление столбцов, введение ограничений или ключей, требуют похожего синтаксиса. (Отметим, что следующий пример будет работать уже не во всех СУБД.)

**ВВОД**

```
ALTER TABLE Vendors  
DROP COLUMN vend_phone;
```

Сложные изменения структуры таблицы обычно выполняются вручную и включают следующие шаги.

- Создание новой таблицы с новым расположением столбцов.
- Использование оператора INSERT SELECT (см. урок 15, “Добавление данных,” в котором подробно рассмотрены вопросы применения этого оператора) для копирования данных из старой таблицы в новую. При необходимости используются функции преобразования и вычисляемые поля.
- Проверка того факта, что новая таблица содержит нужные данные.
- Переименование старой таблицы (или удаление ее).
- Присвоение новой таблице имени, которое ранее принадлежало старой таблице.
- Восстановление триггеров, хранимых процедур, индексов и внешних ключей, если это необходимо.





### Аккуратно используйте оператор ALTER TABLE

Оператор ALTER TABLE следует использовать с особой осторожностью. Прежде чем приступить к его использованию, удостоверьтесь в том, что у вас есть полный комплект резервных копий (и схемы, и данных). Внесение изменений в базу данных нельзя оставить незавершенным. Если вы добавляете в нее ненужные вам столбцы, у вас нет возможности их удалить. Аналогично, если вы удаляете столбец, который вам на самом деле нужен, могут быть потеряны все данные, в нем содержавшиеся.

## Удаление таблиц

Удаление таблиц (имеется в виду удаление именно таблиц, а не их содержимого) — очень простой процесс. Таблицы удаляются с помощью оператора DROP TABLE:

### ВВОД

```
DROP TABLE CustCopy;
```

### Анализ

Этот оператор удаляет таблицу CustCopy (которую вы создали в ходе изучения материалов урока 15). В данном случае не требуется никакого подтверждения, невозможно возвратиться к прежнему состоянию — в результате применения этого оператора таблица будет безвозвратно удалена.



### Использование реляционных правил для предотвращения ошибочного удаления

Во многих СУБД применяются правила, препятствующие удалению таблиц, связанных с другими таблицами. Если эти правила действуют и вы применяете оператор DROP TABLE по отношению к таблице, которая связана с другой таблицей, СУБД блокирует проведение этой операции до тех пор, пока не будет удалена данная связь. Применение этих опций приветствуется, поскольку благодаря им можно воспрепятствовать ошибочному удалению нужных таблиц.

## Переименование таблиц

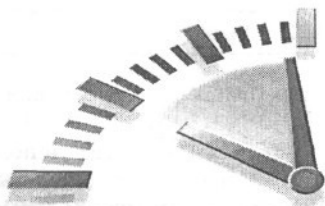
В разных СУБД переименование таблиц осуществляется по-разному. Не существует жестких, устоявшихся стандартов на выполнение этой операции. Пользователи СУБД DB2, MySQL, Oracle и PostgreSQL могут применять для этого оператор `RENAME`. Пользователи SQL Server и Sybase могут использовать хранимую процедуру `sp_rename`. Основной синтаксис для всех операций переименования требует указания старого и нового имен. Однако существуют различия, зависящие от реализации. Обратитесь к документации своей СУБД, чтобы узнать подробности относительно поддерживаемого ею синтаксиса.

## Резюме

В этом уроке вы познакомились с несколькими новыми операторами SQL. Оператор `CREATE TABLE` применяется для создания новых таблиц, `ALTER TABLE` — для изменения столбцов таблицы (или других объектов, таких как ограничения или индексы), а оператор `DROP TABLE` позволяет полностью удалить таблицу. Все эти операторы нужно использовать с особой осторожностью и только после создания резервных копий базы данных. Поскольку точный синтаксис этих операторов варьируется в зависимости от СУБД, вам придется обратиться к документации своей СУБД за дополнительной информацией.



## Урок 18



# Использование представлений

В этом уроке рассказывается о том, что такое представления, как они работают и когда их можно использовать. Вы узнаете также, как представления можно использовать для упрощения некоторых операций SQL, выполненных в прошлых уроках.

## Что такое представления

Представления — это виртуальные таблицы. В отличие от таблиц, содержащих данные, представления содержат запросы, которые динамически выбирают данные, когда это необходимо.

### Поддержка в MySQL

К моменту выхода этой книги СУБД MySQL еще не поддерживала представления (их поддержку планировалось осуществить в версии MySQL 5). Поэтому приведенные нами примеры в настоящее время работать не будут.

Лучший способ объяснить, что такое представления, — рассмотреть конкретный пример. Возвратимся к уроку 12, “Объединение таблиц”, в котором был использован следующий оператор SELECT для выборки данных сразу из трех таблиц:

### ВВОД

```
SELECT cust_name, cust_contact
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Orders.cust_id
```

```
AND OrderItems.order_num = Orders.order_num
AND prod_id = 'RGAN01';
```

Этот запрос был использован для извлечения информации о клиентах, которые заказали указанный продукт. Всякий, кому необходимы эти данные, должен был бы обратиться в структуре таблицы, а также в методике создания запроса и объединения таблиц. Чтобы извлечь аналогичные данные для другого продукта (или для нескольких продуктов), последнее предложение WHERE придется модифицировать.

Теперь предположим, что вы могли бы сохранить весь этот запрос в виртуальной таблице с именем ProductCustomers. Затем для выборки тех же самых данных нужно было бы просто сделать следующее:

### ВВОД

```
SELECT cust_name, cust_contact
FROM ProductCustomers
WHERE prod_id = 'RGAN01';
```

Это как раз тот случай, когда в игру вступают представления. Таблица ProductCustomers является представлением, поэтому она не содержит каких-либо столбцов или данных. Вместо них хранится запрос — тот самый запрос, который был использован выше для объединения таблиц.



#### Постоянство СУБД

Синтаксис создания представлений одинаков для всех основных СУБД.

## Для чего используют представления

Вы только что познакомились с одним случаем использования представления. Довольно часто они применяются для выполнения следующих операций:

- для повторного использования операторов SQL;
- для упрощения выполнения сложных операций. После того как запрос подготовлен, его можно с легкостью использовать повторно, для этого не нужно разбираться в особенностях его работы;

- для вывода частей таблицы вместо вывода ее полностью;
- для защиты данных. Пользователям можно предоставить доступ к определенному поднабору таблиц, а не ко всем таблицам;
- для изменения форматирования и отображения данных. Представления могут возвращать данные, отформатированные и отображенные иначе, чем они хранятся в таблицах.

После того как представления созданы, их можно использовать точно так же, как таблицы. Вы можете выполнять операции SELECT, фильтровать и сортировать данные, объединять представления с другими представлениями или таблицами и, возможно, даже добавлять в них данные либо обновлять их. (На последнюю операцию накладываются некоторые ограничения. Ниже мы расскажем о них.)

Важно не забывать о том, что представления — это только представления, данные которых хранятся в других таблицах. Представления не содержат данных как таковых, поэтому данные, которые они возвращают, извлекаются из других таблиц. Если данные этих таблиц изменяются или происходит добавление в них данных, представления возвратят уже новые, измененные данные.



### Проблемы производительности

Поскольку представления не содержат данных, каждый раз, когда используется представление, для выполнения запроса приходится проводить некоторый поиск. Если вы создали сложное представление с несколькими объединениями и фильтрами или если были использованы вложенные представления, производительность СУБД резко снизится. Рекомендуется провести тестирование, прежде чем использовать приложения, в которых интенсивно используются представления.

## Представления: правила и ограничения

Прежде чем создавать представления, следует ознакомиться с некоторыми накладываемыми на них ограниче-

ниями. К сожалению, представления весьма специфичны для каждой СУБД, поэтому прежде чем приступать к их использованию, рекомендуем обратиться к документации вашей СУБД,

Ниже приведено несколько самых общих правил и ограничений, которыми следует руководствоваться при создании и использовании представлений.

- Представления, так же как и таблицы, должны иметь уникальные имена. (Они не могут быть названы так же, как какая-нибудь другая таблица или представление.)
- Не существует ограничений на количество представлений, которые могут быть созданы.
- Для того чтобы создать представление, вы должны иметь соответствующие права доступа. Обычно их предоставляет администратор базы данных.

Представления могут быть вложенными; это означает, что представление может быть создано посредством запроса, который выбирает данные из другого представления. Точное количество уровней вложения различно для разных СУБД. (Вложенные представления могут серьезно снизить производительность при выполнении запроса, поэтому их нужно основательно протестировать, прежде чем применять в реальных условиях.)

- Во многих СУБД запрещается использование предложения `ORDER BY` в запросах к представлениям.
- В некоторых СУБД требуется, чтобы каждый возвращаемый столбец имел имя — это подразумевает использование псевдонимов, если столбцы представляют собой вычисляемые поля (см. урок 7, “Создание вычисляемых полей,” где представлена дополнительная информация о псевдонимах столбцов).
- Представления не могут быть проиндексированы. Они также не могут иметь триггеров или связанных с ними значений по умолчанию.
- В некоторых СУБД представления трактуются как запросы, предназначенные только для чтения. Это означает, что из представлений можно выбирать данные, но их нельзя вносить в таблицы, на основе которых было создано представление. Обратитесь к до

кументации своей СУБД, если хотите узнать подробности.

- Некоторые СУБД позволяют создавать представления, которые не позволяют добавлять или обновлять строки, если это добавление или обновление может привести к тому, что строки уже не будут являться частью данного представления. Например, если представление возвращает только информацию о клиентах, имеющих адреса электронной почты, обновление информации о клиенте с целью удаления его адреса электронной почты приведет к тому, что данный клиент будет исключен из представления. Такого поведения по умолчанию, и оно допускается, но некоторые СУБД способны препятствовать возникновению подобных случаев.



#### Обратитесь к документации своей СУБД

Список этих правил довольно длинен, и документация вашей СУБД почти наверняка содержит еще какие-то правила. Придется потратить некоторое время на изучение этих ограничений, прежде чем браться за создание представлений.

## Создание представлений

Итак, вы знаете, что такое представления (а также отчасти знакомы с правилами, которыми следует руководствоваться при работе с ними), теперь разберемся, как они создаются.

Представления создаются с помощью оператора `CREATE VIEW`. Аналогично оператору `CREATE TABLE`, оператор `CREATE VIEW` можно использовать только для создания представления, которого до сих пор не существовало.



#### Удаление представлений

Для удаления представления используется оператор `DROP`. Его синтаксис прост: `DROP VIEW имя_представления ;`.



Чтобы перезаписать (или обновить) представление, вначале нужно применить по отношению к нему оператор DROP, а потом заново создать представление.

## Использование представлений для упрощения сложных объединений

Чаще всего представления используются для упрощения работы с SQL, и нередко это относится к объединениям. Посмотрите на следующий оператор:

### ВВОД

```
CREATE VIEW ProductCustomers AS
SELECT cust_name, cust_contact, prod_id
FROM Customers, Orders, OrderItems
WHERE Customers.cust_id = Orders.cust_id
AND OrderItems.order_num = Orders.order_num;
```

### Анализ

Посредством этого оператора создается представление, названное ProductCustomers, которое объединяет три таблицы с целью возвращения списка клиентов, заказавших какой-то продукт. Если бы был использован оператор SELECT \* FROM ProductCustomers, в список был бы включен любой клиент, который сделал заказ.



### Создание представлений и SQL Server

В отличие от большинства операторов SQL, Microsoft SQL Server не поддерживает использование точки с запятой после оператора CREATE VIEW.

Для выборки списка клиентов, заказавших продукт RGAN01, нужно выполнить следующее:

### ВВОД

```
SELECT cust_name, cust_contact
FROM ProductCustomers
WHERE prod_id = 'RGAN01';
```

**Вывод**

| cust_name     | cust_contact       |
|---------------|--------------------|
| Fun4All       | Denise L. Stephens |
| The Toy Store | Kim Howard         |

**Анализ**

Этот оператор возвращает указанные данные из представления в результате применения предложения WHERE. Когда СУБД обрабатывает запрос, она добавляет указанное предложение WHERE к каждому уже существующему предложению WHERE в запросе к представлению, так что данные фильтруются правильно.

Таким образом, представления могут значительно упростить сложные операторы SQL. Используя представления, вы можете один раз записать код SQL и затем повторно использовать его, если возникает такая необходимость.

**Создание повторно используемых представлений**

Хорошей идеей является создание представлений, не привязанных к конкретным данным. Например, представление, созданное в предыдущем примере, возвращает имена клиентов, заказавших все продукты, а не только продукт RGN01 (для которого представление первоначально и создавалось). Расширение возможностей представления позволяет многократно использовать его, тем самым устраняется также необходимость создавать и хранить много похожих представлений, что делает представление еще более эффективным.

## Использование представлений для перематирования выбранных данных

Как уже говорилось выше, другим наиболее частым случаем использования представлений является перематирование выбранных данных. Следующий оператор SELECT (см. урок 7, “Создание вычисляемых полей”) возвращает имя поставщика и его местонахождение в одном комбинированном вычисляемом столбце:

**ВВОД**

```
SELECT RTRIM(vend_name) + ' (' + RTRIM(vend_country)
  + ')' AS vend_title
FROM Vendors
ORDER BY vend_name;
```

**ВЫВОД**

```
vend_title
-----
Bear Emporium (USA)
Bears R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)
```

Следующий оператор такой же, но в нем применяется синтаксис || (его мы рассматривали в уроке 7):

**ВВОД**

```
SELECT RTRIM(vend_name) || ' (' ||
  RTRIM(vend_country) || ')' AS vend_title
FROM Vendors
ORDER BY vend_name;
```

**ВЫВОД**

```
vend_title
-----
Bear Emporium (USA)
Bears R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)
```

Теперь предположим, что результаты регулярно требуются в таком формате. Вместо того чтобы выполнять объединение каждый раз, когда в этом возникает необходимость, вы можете создать представление и использовать его вместо объединения. Для превращения этого оператора в представление нужно сделать следующее:

**ВВОД**

```
CREATE VIEW VendorLocations AS SELECT RTRIM(vend_name) + '
(' + RTRIM(vend_country)
```

```

+ ')' AS vend_title
FROM Vendors;

```

Здесь также применяется оператор, использующий синтаксис `||`:

### ВВОД

```

CREATE VIEW VendorLocations AS
SELECT RTRIM(vend_name) || ' (' ||
+RTRIM(vend_country) || ')' AS vend_title
FROM Vendors;

```

### Анализ

Посредством этого оператора создается представление, использующее в точности тот же самый запрос, как и в предыдущем операторе `SELECT`. Для извлечения данных, необходимых для создания почтовых наклеек, выполните следующее:

### ВВОД

```

SELECT *
FROM VendorLocations;

```

### ВЫВОД

```

vend_title
-----
Bear Emporium (USA)
Bears R Us (USA)
Doll House Inc. (USA)
Fun and Games (England)
Furball Inc. (USA)
Jouets et ours (France)

```



#### Все ограничения оператора `SELECT` сохраняются

Ранее в этом уроке говорилось, что синтаксис, применяемый для создания представлений, различен в разных СУБД. Но почему так много версий синтаксиса операторов? Представление просто скрывает оператор `SELECT`, и синтаксис этого оператора `SELECT` должен четко соответствовать правилам и ограничениям используемой СУБД.

## Использование представлений для фильтрации нежелательных данных

Представления могут также оказаться полезными для применения общих предложений WHERE. Например, вам может понадобиться определить представление CustomerEMailList таким образом, чтобы оно отфильтровывало клиентов, не имеющих адреса электронной почты. Для того чтобы добиться этого, следует применить такой оператор:

### ВВОД

```
CREATE VIEW CustomerEMailList AS
SELECT cust_id, cust_name, cust_email
FROM Customers
WHERE cust_email IS NOT NULL;
```

### Анализ

Очевидно, отправляя сообщение в соответствии со списком адресов e-mail, следовало бы пропустить клиентов, у которых нет адреса электронной почты. Предложение WHERE отфильтровывает здесь строки, имеющие значения NULL в столбцах cust\_email, так что соответствующие записи не будут выбираться.

Теперь представление CustomerEMailList можно использовать подобно любой другой таблице.

### ВВОД

```
SELECT *
FROM CustomerEMailList;
```

### ВЫВОД

| cust_id    | cust_name    | cust_email            |
|------------|--------------|-----------------------|
| -----      | -----        | -----                 |
| 1000000001 | Village Toys | sales@villagetoy.com  |
| 1000000003 | Fun4All      | jjones@fun4all.com    |
| 1000000004 | Fun4All      | dstephens@fun4all.com |



#### Предложения WHERE

Если предложение WHERE используется при выборке данных из представления, эти два набора предложений (одно в представлении и одно передаваемое ему) будут скомбинированы автоматически.

## Использование представлений с вычисляемыми полями

Представления чрезвычайно полезны для упрощения использования вычисляемых полей. Далее приведен оператор SELECT, впервые использованный нами в уроке 7. Он извлекает предметы указанного заказа и вычисляет суммарную стоимость для каждого предмета:

### ВВОД

```
SELECT prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM OrderItems
WHERE order_num = 20008;
```

### ВЫВОД

| prod_id | quantity | item_price | expanded_price |
|---------|----------|------------|----------------|
| RGAN01  | 5        | 4.9900     | 24.9500        |
| BR03    | 5        | 11.9900    | 59.9500        |
| BNBG01  | 10       | 3.4900     | 34.9000        |
| BNBG02  | 10       | 3.4900     | 34.9000        |
| BNBG03  | 10       | 3.4900     | 34.9000        |

Для превращения его в представление необходимо выполнить следующее:

### ВВОД

```
CREATE VIEW OrderItemsExpanded AS
SELECT order_num,
       prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM OrderItems;
```

Чтобы получить информацию относительно заказа 20008 (она была выведена выше), необходимо сделать следующее:

### ВВОД

```
FROM SELECT *
OrderItemsExpanded
WHERE order_num = 20008;
```

**Вывод**

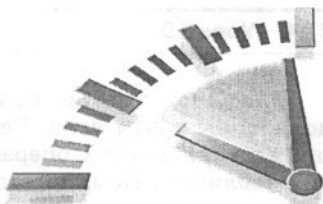
| order_num | prod_id | quantity | item_price | expanded_price |
|-----------|---------|----------|------------|----------------|
| 20008     | RGAN01  | 5        | 4.99       | 24.95          |
| 20008     | BR03    | 5        | 11.99      | 59.95          |
| 20008     | BNBG01  | 10       | 3.49       | 34.90          |
| 20008     | BNBG02  | 10       | 3.49       | 34.90          |
| 20008     | BNBG03  | 10       | 3.49       | 34.90          |

Как видите, представления легко создавать, а использовать еще легче. Будучи использованными корректно, представления могут существенно упростить сложные манипуляции с данными.

## Резюме

Представления — это виртуальные таблицы. Они не содержат данных, вместо данных представления содержат запросы, посредством которых данные выбираются в случае необходимости. Представления обеспечивают должный уровень инкапсуляции SQL-операторов SELECT и могут быть использованы для упрощения манипулирования данными, а также для переформатирования данных и ограничения доступа к ним.

## Урок 19



# Работа с хранимыми процедурами

В этом уроке вы узнаете, что такое хранимые процедуры, для чего и как они используются. Вы также познакомитесь с основами синтаксиса, используемого при создании и применении хранимых процедур.

## Что такое хранимые процедуры

Большинство операций SQL, которые мы до сих пор выполняли, просты в том смысле, что в них применяется только один оператор по отношению к одной или нескольким таблицам. Но не все операции столь просты — зачастую приходится использовать несколько операторов для выполнения сложной операции. Например, рассмотрим следующие сценарии.

- При обработке заказа бывает необходимо удостовериться в том, что соответствующие товары есть на складе.
- Если товары есть на складе, они должны быть зарезервированы, чтобы их не продали кому-нибудь еще, а их количество, доступное другим покупателям, должно быть уменьшено соответственно изменившейся ситуации.
- Товары, отсутствующие на складе, должны быть заказаны, для этого нужно связаться с их поставщиком.
- Клиенту необходимо сообщить, какие товары есть на складе (и могут быть отгружены немедленно) и заказ на какие товары выполнен быть не может.



Очевидно, это не полный список примеров, но суть должна быть вам ясна. Решение подобной задачи потребует применения многих операторов SQL по отношению ко многим таблицам. Помимо того что сами SQL-операторы, подлежащие выполнению в подобных случаях, и их порядок не постоянны, они могут (и будут) изменяться в зависимости от того, какие товары имеются на складе, а каких там нет.

Как бы вы написали этот код? Можно было бы написать каждый из операторов SQL отдельно и выполнить другие операторы в зависимости от полученных результатов. Вам пришлось бы делать это каждый раз, когда возникала бы необходимость подобной обработки данных (и для каждого приложения, которое в ней нуждается).

Вы могли бы также создать хранимую процедуру. Хранимая процедура — это совокупность нескольких операторов (или даже один оператор), сохраненная для последующего использования. Ее можно рассматривать как командный файл, хотя на самом деле это нечто большее.



#### Access и MySQL

Хранимые процедуры не поддерживаются в Access. Кроме того, когда эта книга готовилась к печати, СУБД MySQL v4.x (текущая версия) также не поддерживала хранимые процедуры (их поддержку планировалось осуществить в версии MySQL 5).



#### Гораздо больше информации

Хранимые процедуры — тема довольно сложная, полностью ее рассмотреть можно только в отдельной книге. Этот урок не научит вас всему, что необходимо знать о хранимых процедурах. Скорее это введение в данную тему, призванное познакомить вас с тем, что собой представляют хранимые процедуры и что с их помощью можно делать. По существу, представленные здесь примеры соответствуют только синтаксису Oracle и SQL Server.

## Для чего используют хранимые процедуры

Теперь, когда вы знаете, что такое хранимые процедуры, возникает другой вопрос: для чего их использовать? На это существует множество причин, ниже приведены лишь основные.

- Для упрощения сложных операций (как уже говорилось в предыдущем примере) за счет инкапсуляции процессов в один блок, простой для выполнения.
- Для обеспечения непротиворечивости данных и вместе с тем без необходимости снова и снова воспроизводить одну и ту же последовательность шагов. Если все разработчики и приложения используют одни и те же хранимые процедуры, значит, один и тот же код будет использоваться всеми.
- Побочным эффектом этого является предотвращение ошибок. Чем больше шагов необходимо выполнить, тем выше вероятность появления ошибок. Предотвращение ошибок обеспечивает целостность данных.
- Для упрощения управления изменениями. Если таблицы, имена столбцов, деловые правила (или что-то подобное) изменяются, обновлять приходится только код хранимой процедуры и ничего больше.

Побочным эффектом этого является повышение безопасности. Ограничение доступа к основным данным только через хранимые процедуры снижает вероятность повреждения данных (случайного или преднамеренного).

- Поскольку хранимые процедуры обычно сохраняются в скомпилированном виде, СУБД тратит меньше времени на обработку их команд. Это приводит к повышению производительности.
- Существуют элементы языка SQL и некоторые возможности, реализуемые только в хранимых процедурах. Хранимые процедуры, таким образом, можно использовать для написания более гибкого и мощного кода.

Итак, имеются три основных преимущества: простота, безопасность и производительность. Очевидно, все они чрезвычайно важны. Однако прежде чем вы броситесь превращать весь свой SQL-код в хранимые процедуры, взгляните на другую сторону медали.

- Синтаксис хранимых процедур весьма различен для разных СУБД. Написать по-настоящему переносимый код хранимой процедуры практически невозможно. Следует также упомянуть о том, что метод вызова самих хранимых процедур (их имена и метод передачи им данных) может быть достаточно переносимым, поэтому если вам необходимо перейти на другую СУБД, по крайней мере код вашего клиентского приложения, возможно, не придется изменять.
- Хранимые процедуры сложнее в написании, чем основные операторы SQL, их подготовка требует большей квалификации и опыта. Поэтому многие администраторы баз данных ограничивают права на создание хранимых процедур в качестве меры безопасности.

Несмотря на вышесказанное, хранимые процедуры весьма полезны и непременно должны использоваться. В действительности многие СУБД располагают всевозможными хранимыми процедурами, которые используются для управления базами данных и таблицами. Обратитесь к документации своей СУБД, чтобы получить больше информации по данному вопросу.



### **Не можете написать хранимые процедуры? Тогда просто используйте их**

Во многих СУБД различаются меры безопасности и права доступа, необходимые для написания хранимых процедур, и меры безопасности и права доступа, необходимые для их выполнения. И это хорошо. Если вы не намерены писать свои хранимые процедуры, используйте готовые, если они вам подходят.

## Выполнение хранимых процедур

Хранимые процедуры выполняются намного чаще, чем пишутся, поэтому мы начнем именно с их выполнения. Оператор SQL для выполнения хранимой процедуры — EXECUTE — принимает имя хранимой процедуры и некоторые параметры, необходимые для перехода к ней. Посмотрите на этот пример:

### ВВОД

```
EXECUTE AddNewProduct ('JTS01',  
'Stuffed Eiffel Tower',  
6.49,  
'Plush stuffed toy with the  
text La Tour Eiffel in red white and blue')
```

### Анализ

Здесь выполняется хранимая процедура по имени AddNewProduct; она добавляет новый продукт в таблицу Products. Хранимая процедура AddNewProduct принимает четыре параметра: идентификатор поставщика (первичный ключ таблицы Vendors), название продукта, цена и описание. Эти четыре параметра соответствуют четырем ожидаемым переменным хранимой процедуры (определенным как часть самой хранимой процедуры). Данная хранимая процедура добавляет новую строку в таблицу Products и распределяет эти передаваемые атрибуты по соответствующим столбцам.

В таблице Products есть еще один столбец, нуждающийся в присвоении значения: столбец prod\_id, который является первичным ключом таблицы. Почему это значение не передается в хранимую процедуру в виде атрибута? Для того чтобы идентификаторы генерировались правильно, безопаснее сделать этот процесс автоматизированным (и не полагаться на конечного пользователя). Именно поэтому хранимая процедура используется в этом примере, она выполняет следующие действия.

- Подтверждает правильность передаваемых данных, обеспечивая наличие значений у всех четырех параметров.

- Генерирует уникальный идентификатор, который будет использован в качестве первичного ключа.
- Добавляет данные о новом продукте в таблицу Products, сохраняя созданный первичный ключ и передавая данные в соответствующие столбцы.

Такова основная форма выполнения хранимой процедуры. В зависимости от СУБД могут быть использованы другие опции выполнения, включая следующие.

- Опциональные параметры со значениями по умолчанию, присваиваемыми в случае, если параметр не предложен пользователем.
- Нестандартные параметры, указываемые в виде *параметр=значение*.
- Выходные параметры, позволяющие хранимой процедуре обновлять параметр для использования его в выполняемом приложении.
- Значение даты, выбираемое оператором SELECT.
- Возвращаемые коды, позволяющие хранимой процедуре возвращать значение в выполняемое приложение.

## Создание хранимых процедур

Как уже говорилось, создание хранимой процедуры — задача не из тривиальных. Чтобы вы могли понять, что она собой представляет, рассмотрим простой пример: хранимую процедуру, которая подсчитывает в списке адресатов число клиентов, имеющих адрес электронной почты.

Вот версия для СУБД Oracle:

### ВВОД

```
CREATE PROCEDURE MailingListCount
(ListCount OUT NUMBER)
IS
BEGIN
  SELECT * FROM Customers
  WHERE NOT cust_email IS NULL;
  ListCount := SQL%ROWCOUNT;
END;
```

**Анализ**

Эта хранимая процедура принимает один параметр, называемый `ListCount`. Вместо того чтобы передавать значение в хранимую процедуру, этот параметр передает значение из нее. Ключевое слово `OUT` указывает ей вести себя подобным образом. СУБД Oracle поддерживает параметры типов `IN` (которые передаются в хранимые процедуры), `OUT` (они передаются из хранимых процедур) и `INOUT` (они используются для передачи параметров в хранимые процедуры и из них). Собственно код хранимой процедуры заключен между `BEGIN` и `END`, и здесь для выборки клиентов, имеющих адреса электронной почты, выполняется простой оператор `SELECT`. Затем передаваемому выходному параметру `ListCount` присваивается значение, равное количеству строк в выборке.

А вот версия для Microsoft SQL:

**ВВОД**

```
CREATE PROCEDURE MailingListCount
AS
DECLARE @cnt INTEGER
SELECT @cnt = COUNT(*)
FROM Customers
WHERE NOT cust_email IS NULL;
RETURN @cnt;
```

**Анализ**

Эта хранимая процедура вообще не принимает параметров. Вызываемое приложение выбирает нужное значение, пользуясь тем, что в СУБД SQL Server поддерживается возвращение кода. Здесь посредством оператора `DECLARE` объявлена локальная переменная `@cnt` (имена всех локальных переменных в SQL Server начинаются с символа `@`). Эта переменная затем используется в операторе `SELECT`, так что он содержит значение, возвращаемое функцией `COUNT()`. Наконец, оператор `RETURN` используется для возвращения результатов подсчета в вызывающее приложение — `RETURN @cnt`.

Приведем еще один пример, на этот раз мы будем добавлять новый заказ в таблицу `Orders`. Этот пример подходит только для SQL Server, но он хорошо показывает, как нужно использовать хранимые процедуры:

**ВВОД**

```

CREATE PROCEDURE NewOrder @cust_id CHAR(10)
AS
-- Объявление переменной для номера заказа
DECLARE @order_num INTEGER
-- Получение текущего наибольшего номера заказа
SELECT @order_num=MAX(order_num)
FROM Orders
Determine next order number
SELECT @order_num=@order_num+1
-- Добавление нового заказа
INSERT INTO Orders(order_num, order_date, cust_id)
VALUES(@order_num, GETDATE(), @cust_id)
-- Возвращение номера заказа
RETURN @order_num;

```

**Анализ**

Эта хранимая процедура создает новый заказ в таблице Orders. Она принимает один параметр — идентификатор клиента, сделавшего заказ. Два других столбца таблицы, номер и дата заказа, генерируются автоматически в самой хранимой процедуре. Вначале в коде объявляется локальная переменная для хранения номера заказа. Затем выбирается текущий наибольший номер заказа (посредством функции MAX()) и увеличивается на единицу (с помощью оператора SELECT). После этого добавляется заказ посредством оператора INSERT с использованием только что сгенерированного номера заказа, выбирается текущая системная дата (с помощью функции GETDATE()) и передается идентификатор клиента. Наконец, номер заказа (необходимый для обработки предметов заказа) возвращается как RETURN @order\_num. Отметим, что код снабжен комментариями, это всегда следует делать при написании хранимых процедур.

**Комментируйте ваш код**

Весь ваш код должен быть снабжен комментариями, и хранимая процедура не исключение. Добавление комментариев не окажет никакого влияния на производительность, так что здесь нет «обратной стороны медали» (время тратится только на написание комментариев). Многочисленные преимущества включают, например, облегчение понимания кода другими (и вами тоже), а также удобство его изменения через некоторое время.

Стандартный способ ввода в код комментариев состоит в предварении их символами -- (двумя дефисами). Некоторые СУБД поддерживают и альтернативный синтаксис комментариев, но все поддерживают -- (два дефиса), и поэтому лучше всего использовать их.

Вот несколько различных версий одного и того же кода для SQL Server:

## ВВОД

```
CREATE PROCEDURE NewOrder @cust_id CHAR(10)
AS
-- Добавление нового заказа
INSERT INTO Orders(cust_id)
VALUES(@cust_id)
-- Возвращение номера заказа
SELECT order_num = @@IDENTITY;
```

## Анализ

Эта хранимая процедура также создает новый заказ в таблице Orders. На этот раз СУБД сама генерирует номер заказа. Большинство СУБД поддерживают такой тип функциональности; SQL Server обращается к этим автоинкрементируемым столбцам как к полям Identity (другие СУБД используют такие имена, как Auto Number или Sequences). Опять же, передается только один параметр: идентификатор клиента, сделавшего заказ. Номер и дата заказа не указываются вообще — СУБД использует значение по умолчанию для даты (функция GETDATE()), а номер заказа генерируется автоматически. Как можно узнать, какой идентификатор пользователя был сгенерирован? В СУБД SQL Server для этого используется глобальная переменная @@IDENTITY, возвращаемая в вызывающее приложение (на этот раз с использованием оператора SELECT).

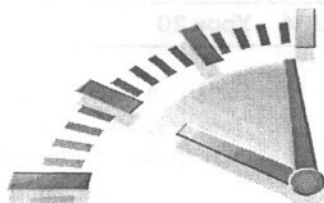
Как видите, хранимые процедуры очень часто позволяют решить одну и ту же задачу разными способами. Метод, который вы выберете, во многом будет зависеть от особенностей СУБД, которую вы используете.



## Резюме

В этом уроке вы узнали, что такое хранимые процедуры и для чего они используются. Вы также познакомились с основами синтаксиса, применяемого для выполнения и создания хранимых процедур, и узнали о некоторых способах их использования. Возможно, в вашей СУБД форма выполнения этих функций будет несколько иной и в ваше распоряжение будут предоставлены возможности, не упомянутые в этой книге. За дополнительной информацией рекомендуем обратиться к документации вашей СУБД.

## Урок 20



# Обработка транзакций

В этом уроке вы узнаете, что такое транзакции и как использовать операторы *COMMIT* и *ROLLBACK* для их обработки.

## Что такое обработка транзакций

Обработка транзакций обеспечивает сохранение целостности базы данных за счет того, что пакеты операций SQL выполняются полностью или не выполняются вовсе.

Как объяснялось в уроке 12, “Объединение таблиц”, реляционные базы данных организованы таким образом, что информация в них хранится во многих таблицах. Благодаря этому облегчается манипулирование, управление данными, а также их повторное использование. Не вдаваясь в подробности, как и почему именно так устроены реляционные базы данных, следует заметить, что схемы всех хорошо спроектированных баз данных можно в какой-то степени отнести к реляционным.

Таблица *Orders*, которую мы использовали в последних 18-ти уроках, — хороший пример. Заказы хранятся в двух таблицах, в таблице *OrderItems* хранится информация об отдельных предметах заказов. Эти две таблицы связаны (соотнесены) между собой с помощью уникального идентификатора, который называется *первичный ключ* (см. урок 1, “Что такое SQL”). Эти таблицы, кроме того, связаны и с другими таблицами, содержащими информацию о клиентах и продуктах.

Процесс добавления нового заказа состоит в выполнении следующих этапов.

1. Проверка, содержится ли информация о клиенте в базе данных. Если нет, такая информация добавляется.

2. Выборка идентификатора клиента.
3. Добавление строки в таблицу `Orders`, связывающую ее (строку) с идентификатором клиента.
4. Выборка идентификатора нового заказа, присвоенного ему в таблице `Orders`.
5. Добавление одной строки в таблицу `OrderItems` для каждого заказанного предмета, соотнесение его с таблицей `Orders` посредством выбранного идентификатора (и с таблицей `Products` посредством идентификатора продукта).

Теперь предположим, что какая-то ошибка в базе данных (например, нехватка места на диске, ограничения, связанные с безопасностью, блокировка таблицы) помешала завершить эту последовательность действий.

Что случится с данными?

Хорошо, если ошибка произойдет после добавления информации о клиенте в таблицу, но до того как она будет добавлена в таблицу `Orders` — в этом случае проблем не будет. Вы можете иметь данные о клиентах без заказов. При повторном выполнении последовательности добавленная запись о клиенте будет возвращена и использована. Вы сможете легко продолжить работу с того места, на котором остановились.

Но что если ошибка произойдет после того, как была добавлена строка в таблицу `Orders`, но до того, как будут добавлены строки в таблицу `OrderItems`? Теперь в вашей базе данных будет присутствовать пустой заказ.

Еще хуже: что если система сделает ошибку в процессе добавления строк в таблицу `OrderItems`? В таком случае в вашу базу данных заказ будет внесен лишь частично, и вы даже не будете знать об этом.

Как можно решить эту проблему? Именно здесь в игру вступает транзактная организация обработки данных, которую мы ради краткости будем называть *обработка транзакций*. Обработка транзакций — это механизм, используемый для управления наборами операций SQL, которые должны быть выполнены в пакете, т.е. таким образом, чтобы в базу данных не могли попасть результаты частичного выполнения этого пакета операций. При обработке транзакций вы можете быть уверенными в том, что выполнение набора операций не было прервано на середине — они или

были выполнены все, или не была выполнена ни одна из них (если только не было явно указано иное). Если ошибки не произошло, результаты работы всего набора операторов фиксируются (записываются) в таблицах базы данных. Если произошла ошибка, должна быть выполнена отмена (аннулирование) всех операций, чтобы вернуть базу данных в известное и безопасное состояние.

Итак, если вернуться к нашему примеру, то вот как должен на самом деле выполняться процесс.

1. Проверка, содержится ли информация о клиенте в базе данных. Если нет, такая информация добавляется.
2. Фиксация информации о клиенте.
3. Выборка идентификатора клиента.
4. Добавление строки в таблицу Orders.
5. Если во время добавления строки в таблицу Orders происходит ошибка, операция отменяется.
6. Выборка идентификатора нового заказа, присвоенного ему в таблице Orders
7. Добавление одной строки в таблицу OrderItems для каждого заказанного предмета.
8. Если в процессе добавления строк в таблицу OrderItems происходит ошибка, добавление всех строк в таблицу OrderItems отменяется.

При работе с транзакциями вы часто будете сталкиваться с одними и теми же терминами:

- **Транзакция (Transaction).** Блок операторов SQL.
- **Отмена (Rollback).** Процесс аннулирования указанных операторов SQL (такой процесс иногда называют “откат”).
- **Фиксация (Commit).** Запись несохраненных операторов SQL в таблицы базы данных.
- **Точка сохранения (Savepoint).** Временное состояние в ходе выполнения транзакции, в которое можно вернуться после отмены части операций пакета (в отличие от отмены всей транзакции). Иногда это состояние называют “точка отката”.

**Действие каких операторов можно отменить?**

Обработка транзакций используется в ходе управления действием операторов INSERT, UPDATE и DELETE. Вы не можете отменить действие оператора SELECT. (В выполнении такой отмены вообще нет смысла.) Вы не можете отменить операции CREATE или DROP. Эти операторы можно использовать в блоке операторов транзакции, но если вам понадобится выполнить отмену (откат), действие этих операторов аннулировано не будет.

## Управляемые транзакции

Теперь, когда вы знаете, что такое обработка транзакций, перейдем к управляемым транзакциям.

**Различия в реализациях**

Точный синтаксис, используемый для обработки транзакций, для разных СУБД различен. Прежде чем заняться такой обработкой, обратитесь к документации своей СУБД.

Чтобы сделать транзакцию управляемой, нужно разбить ее SQL-операторы на логические части и явно указать, когда может быть выполнена отмена, а когда нет.

В некоторых СУБД требуется, чтобы вы явно отметили начало и конец каждого блока операторов транзакции. Например, в SQL Server нужно сделать следующее:

**ВВОД**

```
BEGIN TRANSACTION
...
COMMIT TRANSACTION
```

**Анализ**

В этом примере все операторы, заключенные между BEGIN TRANSACTION и COMMIT TRANSACTION, должны быть или выполнены, или не выполнены.

Эквивалентный код для MySQL таков:

**ВВОД**

```
START TRANSACTION
...
```

**ВВОД**

В СУБД PostgreSQL используется синтаксис ANSI SQL:

**ВВОД**

```
BEGIN;
...
```

В других СУБД используются вариации на указанную тему.

## Использование оператора ROLLBACK

Оператор ROLLBACK используется для отмены (аннулирования) операторов SQL, как показано ниже:

**ВВОД**

```
DELETE FROM Orders;
ROLLBACK;
```

**Анализ**

В этом примере выполняется и сразу же, посредством оператора ROLLBACK, аннулируется операция DELETE. Хотя это и не самый полезный пример, он все равно показывает, что, будучи включенными в блок транзакции, операции DELETE (а также INSERT и UPDATE) не являются окончательными.

## Использование оператора COMMIT

Обычно после выполнения операторов SQL результаты записываются непосредственно к таблицы баз данных. Это называется *неявная фиксация* — операция фиксации (сохранения или записи) выполняется автоматически.

Однако внутри блока транзакции фиксация неявно может и не проводиться. Это зависит от того, с какой СУБД вы работаете. Некоторые СУБД трактуют завершение транзакции как неявную фиксацию.

Для безусловного выполнения неявной фиксации используется оператор COMMIT. Вот соответствующий пример для SQL Server:

### **ВВОД**

```
BEGIN TRANSACTION
DELETE OrderItems WHERE order_num = 12345
DELETE Orders WHERE order_num = 12345
COMMIT TRANSACTION
```

### **Анализ**

В этом примере для SQL Server заказ номер 12345 полностью удаляется из системы. Поскольку это приводит к обновлению двух таблиц базы данных, Orders и OrderItems, блок транзакции применяется для того, чтобы заказ не мог быть удален лишь частично. Конечный оператор COMMIT записывает изменения только в случае, если не произошло ошибки. Если первый оператор будет выполнен, а второй, из-за ошибки, не выполнен, удаление не будет зафиксировано.

Чтобы выполнить то же самое в СУБД Oracle, нужно сделать следующее:

### **ВВОД**

```
DELETE OrderItems WHERE order_num = 12345;
DELETE Orders WHERE order_num = 12345;
COMMIT;
```

## **Использование точек сохранения**

Простые операторы ROLLBACK и COMMIT позволяют записывать или отменять транзакции в целом. Хотя это вполне применимо по отношению к простым транзакциям, для более сложных могут понадобиться частичные фиксации или отмены.

Например, процесс добавления заказа, описанный выше, представляет собой одну транзакцию. Если произойдет ошибка, вы просто вернетесь в состояние, когда строка в таблицу Orders еще не была добавлена. Но вы вряд ли захотите отменить добавление данных в таблицу Customers (если оно было сделано).

Для отмены части транзакции вы должны иметь возможность размещения меток в стратегически важных точ-

как блока транзакции. Потом, если понадобится отмена, вы сможете вернуть базу данных в состояние, соответствующее одной из меток.

В языке SQL эти метки называются *точками сохранения* (savepoints). Для создания такой точки в СУБД MySQL и Oracle применяется оператор SAVEPOINT:

**ВВОД**

```
SAVEPOINT deletel;
```

В SQL Server и Sybase нужно сделать следующее:

**ВВОД**

```
SAVE TRANSACTION deletel;
```

Каждая точка сохранения должна иметь уникальное имя, идентифицирующее ее таким образом, чтобы, когда вы выполняете отмену, СУБД “знала”, в какую точку она должна вернуться. Чтобы выполнить отмену действия всех операторов после этой точки, в СУБД SQL Server нужно выполнить следующее:

**ВВОД**

```
ROLLBACK TRANSACTION deletel;
```

В MySQL и Oracle можно сделать так:

**ВВОД**

```
ROLLBACK TO deletel;
```

А вот полный пример для SQL Server:

**ВВОД**

```
BEGIN TRANSACTION
INSERT INTO Customers(cust_id, cust_name)
VALUES('1000000010', 'Toys Emporium');
SAVE TRANSACTION StartOrder;
INSERT INTO Orders(order_num, order_date, cust_id)
VALUES(20100, '2001/12/1', '1000000010');
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num, order_item,
    prod_id, quantity, item_price)
VALUES(20010, 1, 'BR01', 100, 5.49);
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;
INSERT INTO OrderItems(order_num, order_item,
    prod_id, quantity, item_price)
VALUES(20010, 2, 'BR03', 100, 10.99);
```



```
IF @@ERROR <> 0 ROLLBACK TRANSACTION StartOrder;  
COMMIT TRANSACTION
```

### Анализ

Здесь имеется набор, состоящий из четырех операторов INSERT, включенных в блок транзакции. Точка сохранения определена после первого оператора INSERT, так что если какая-то из последующих операций INSERT закончится неудачей, отмена транзакции произойдет лишь до этой точки. В SQL Server для контроля успешности завершения какой-либо операции может быть использована переменная с именем @@ERROR. (В других СУБД используются иные функции или переменные для возвращения такой информации.) Если переменная @@ERROR возвращает значение, отличное от 0, значит, произошла ошибка и транзакция отменяется до точки сохранения. Если обработка транзакции в целом завершается успешно, выполняется операция COMMIT для сохранения данных.



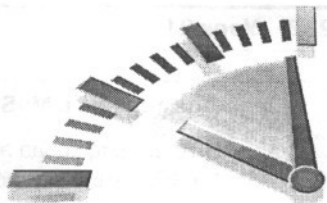
#### Чем больше точек сохранения, тем лучше

Вы можете создать столько точек сохранения в вашем SQL-коде, сколько захотите, и чем больше, тем лучше. Почему? Потому что чем больше у вас точек сохранения, тем большая гибкость вам доступна в управлении отменами.

## Резюме

Транзакции представляют собой блоки из операторов SQL, которые должны выполняться в пакетном режиме (все вместе). Вы познакомились с правилами использования операторов COMMIT и ROLLBACK для явного управления процессами записи и отмены результатов операций. Вы также узнали, как нужно использовать точки сохранения для обеспечения более высокой степени контроля за выполнением операций отмены.

## Урок 21



# Использование курсоров

В этом уроке вы узнаете, что такое курсоры и как ими пользоваться.

## Что такое курсоры

Операции выборки SQL работают с наборами строк, которые называются *результатирующие множества*. Все возвращаемые строки являются строками, соответствующими примененному SQL-оператору, их может быть *нуль* или больше. При использовании простых операторов SELECT невозможно получить первую строку, последнюю строку или предыдущие 10 строк. Это объясняется особенностями функционирования реляционной СУБД.



### Результатирующее множество

Результаты, возвращаемые в ответ на SQL-запрос.

Иногда бывает необходимо просмотреть строки в прямом или обратном направлении один или несколько раз. Именно для этого используются курсоры. Курсор представляет собой запрос к базе данных, хранящийся на сервере СУБД, — это не оператор SELECT, но результирующее множество, выборка, полученная в результате действия оператора SELECT. После того как курсор сохранен, приложения могут “прокручивать” (просматривать) данные в прямом или обратном направлении, как только возникает такая потребность.



### Поддержка в MySQL

В то время, когда эта книга готовилась к печати, СУБД MySQL еще не поддерживала курсоры (поддержку для них планируется ввести в MySQL 5).

Различные СУБД поддерживают разные опции и возможности курсоров. Наиболее часто обеспечиваются следующие из них.

- Возможность пометить курсор как предназначенный только для чтения, в результате чего данные могут считываться, но не могут обновляться или удаляться.
- Возможность управлять направлением выполняемых операций (вперед, назад, первая, последняя, абсолютное положение, относительное положение и т.д.).
- Возможность пометить некоторые столбцы как редактируемые, а другие — как не редактируемые.
- Указание области видимости, благодаря чему курсор может быть доступен или для запроса, посредством которого он был создан (например, для хранимой процедуры), или для всех запросов.
- Указание СУБД скопировать выбранные данные (в противоположность работе с “живыми” данными в таблицах), чтобы они не изменялись в промежуточное время между открытием курсора и обращением к нему.



### Поведение реляционных СУБД становится похожим на поведение нереляционных

В качестве сравнения отметим, что организация доступа и просмотр строк в такой форме в действительности соответствует применению индексно-последовательного метода доступа (Indexed Sequential Access Method, ISAM) в базах данных (таких как Btrieve и dBASE). Курсоры как часть спецификации SQL интересны тем, что с их помощью можно заставить реляционную базу данных вести себя подобно базе данных типа ISAM.

Курсоры используются главным образом интерактивными приложениями, предоставляющими пользователям возможность прокручивать отображаемые на экране данные вперед и назад, просматривать их или изменять.



### Курсоры и Web-приложения

Курсоры практически бесполезны, если их применять к приложениям, основанным на Web-технологиях (например, таких как ASP, ColdFusion, PHP и JSP). Курсоры предназначены для использования в течение сеанса связи между клиентским приложением и сервером, но эта модель "клиент-сервер" не годится для мира Web-приложений, потому что сервер приложений является клиентом базы данных, а не конечным пользователем. А раз так, то большинство разработчиков приложений избегают использования курсоров и добиваются выполнения нужных функций, если это необходимо, своими силами.

## Работа с курсорами

Работу с курсором можно разделить на несколько четко выраженных стадий.

- Прежде чем курсор может быть использован, его следует объявить (определить). В ходе этого процесса выборка данных не производится, просто определяется оператор SELECT, который будет использован, и некоторые опции курсора.
- После объявления курсор может быть открыт для использования. В ходе этого процесса уже производится выборка данных согласно предварительно определенному оператору SELECT.
- После того как курсор заполнен данными, могут быть извлечены (выбраны) отдельные необходимые строки.
- После того как это сделано, курсор должен быть закрыт и, возможно, должны быть освобождены ресурсы, которые он занимал (в зависимости от СУБД).

После того как курсор объявлен, его можно открывать, закрывать столь часто, сколько необходимо. Если курсор открыт, операция выборки может выполняться так часто, как необходимо.

## Создание курсоров

Курсоры создаются с помощью оператора `DECLARE`, синтаксис которого различен для разных СУБД. Оператор `DECLARE` дает курсору имя и принимает оператор `SELECT` дополненный при необходимости предложением `WHERE` и другими. Чтобы показать, как это работает, мы создадим курсор, который будет делать выборку всех клиентов, не имеющих адресов электронной почты, в виде части приложения, позволяющего служащему вводить недостающие адреса.

Приведенная версия подходит для DB2, SQL Server и Sybase:

### ВВОД

```
DECLARE CustCursor CURSOR
FOR
SELECT * FROM Customers
WHERE cust_email IS NULL
```

А вот версия для Oracle и PostgreSQL:

### ВВОД

```
DECLARE CURSOR CustCursor
IS
SELECT * FROM Customers
WHERE cust_email IS NULL
```

### Анализ

В обеих версиях для определения имени курсора используется оператор `DECLARE` — в данном случае это будет имя `CustCursor`. Оператор `SELECT` определяет курсор, содержащий имена всех клиентов, которые не имеют адреса электронной почты (соответствующее значение равно `NULL`).

Теперь, после того как курсор определен, его можно открывать.

## Использование курсоров

Курсоры открываются с помощью оператора OPEN CURSOR, синтаксис которого настолько прост, что его поддерживают большинство СУБД:

```
OPEN CURSOR CustCursor
```

При обработке оператора OPEN CURSOR выполняется запрос, и выборка данных сохраняется для последующих просмотра и прокрутки.

Теперь доступ к данным этого курсора может быть получен с помощью оператора FETCH. Оператор FETCH указывает строки, которые должны быть выбраны, откуда они должны быть выбраны и где их следует сохранить (имя переменной, например). В первом примере используется синтаксис Oracle для выборки одной строки курсора (первой).

### ВВОД

```
DECLARE TYPE CustCursor IS REF CURSOR
  ⊕ RETURN Customers%ROWTYPE;
DECLARE CustRecord Customers%ROWTYPE
BEGIN
  OPEN CustCursor;
  FETCH CustCursor INTO CustRecord;
  CLOSE CustCursor;
END;
```

### Анализ

В данном примере оператор FETCH используется для выборки текущей строки (автоматически он начнет с первой строки) в переменную, объявленную с именем CustRecord. С выбранными данными ничего не делается.

В следующем примере (в нем вновь используется синтаксис Oracle) выбранные данные подвергаются циклической обработке от первой строки до последней:

### ВВОД

```
DECLARE TYPE CustCursor IS REF CURSOR
  ⊕ RETURN Customers%ROWTYPE;
DECLARE CustRecord Customers%ROWTYPE
BEGIN
  OPEN CustCursor;
  LOOP
    FETCH CustCursor INTO CustRecord;
    EXIT WHEN CustCursor%NOTFOUND;
```

```

...
END LOOP;
CLOSE CustCursor;
END;

```

## Анализ

Аналогично предыдущему примеру, здесь используется оператор FETCH для выборки текущей строки в переменную, объявленную с именем CustRecord. Однако в отличие от предыдущего примера, здесь оператор FETCH находится внутри цикла LOOP, так что он выполняется снова и снова. Код EXIT WHEN CustCursor%NOTFOUND указывает, что этот процесс должен быть завершен (выход из цикла), когда больше не останется строк для выборки. В этом примере также не выполняется никакой обработки, тогда как в реальном коде вам следовало бы заменить ... вашим собственным кодом.

Вот другой пример, на этот раз с использованием синтаксиса Microsoft SQL Server:

## ВВОД

```

DECLARE @cust_id CHAR(10),
        @cust_name CHAR(50),
        @cust_address CHAR(50),
        @cust_city CHAR(50),
        @cust_state CHAR(5),
        @cust_ZIP CHAR(10),
        @cust_country CHAR(50),
        @cust_contact CHAR(50),
        @cust_email CHAR(255),
OPEN CustCursor
FETCH NEXT FROM CustCursor
INTO @cust_id, @cust_name, @cust_address,
     @cust_city, @cust_state, @cust_ZIP,
     @cust_country, @cust_contact, @cust_email
WHILE @@FETCH_STATUS = 0
BEGIN
...
FETCH NEXT FROM CustCursor
INTO @cust_id, @cust_name, @cust_address,
     @cust_city, @cust_state, @cust_ZIP,
     @cust_country, @cust_contact, @cust_email
END
CLOSE CustCursor

```

## Анализ

В этом примере переменные объявляются для каждого из выбираемых столбцов, а операторы `FETCH` осуществляют выборку строки и сохранение значений в этих переменных. Цикл `WHILE` используется для организации цикла по строкам, а условие `WHILE @@FETCH_STATUS = 0` обеспечивает завершение обработки (выход из цикла) после того как все строки будут извлечены. И вновь этот пример ничего на самом деле не обрабатывает. В реальном коде нужно заметить . . . вашим собственным кодом.

## Закрытие курсоров

Как следует из предыдущих примеров, после использования курсоров их нужно закрывать. Кроме того, в некоторых СУБД (таких как `SQL Server`) требуется, чтобы ресурсы, занятые курсором, были освобождены явным образом. Вот соответствующий синтаксис для СУБД `DB2`, `Oracle` и `PostgreSQL`:

### ВВОД

```
CLOSE CustCursor
```

А это синтаксис для `Microsoft SQL Server`:

### ВВОД

```
CLOSE CustCursor
DEALLOCATE CURSOR CustCursor
```

Для закрытия курсора используется оператор `CLOSE`; после того как курсор закрыт, его нельзя использовать, не открыв перед этим вновь. Однако его не нужно объявлять заново при повторном использовании, достаточно оператора `OPEN`.

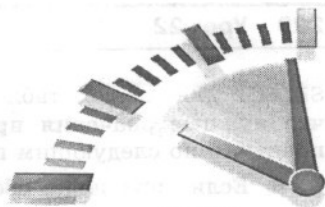
## Резюме

В этом уроке вы узнали, что такое курсоры и как их используют. В вашей СУБД, возможно, эти функции выполняются несколько иначе, а также доступны функции, не упомянутые в книге. За дополнительной информацией вам следует обратиться к документации вашей СУБД.





## Урок 22



# Расширенные возможности SQL

В этом уроке мы рассмотрим несколько расширенных возможностей манипулирования данными: ограничения, индексы и триггеры.

## Что такое ограничения

Было разработано много версий языка SQL, прежде чем он стал столь полноценным и мощным. Многие из наиболее эффективных инструментов манипуляции с данными основаны на таких методах, которые обеспечиваются с помощью ограничений.

И реляционные таблицы, и целостность на уровне ссылок несколько раз упоминались в предыдущих уроках. В них, в частности, говорилось, что реляционные базы данных хранят данные во многих таблицах, каждая из которых содержит данные, связанные с данными из других таблиц.

Для создания ссылок из одной таблицы на другие используются ключи (отсюда термин *целостность на уровне ссылок*).

Чтобы реляционная база данных работала должным образом, необходимо удостовериться в том, что данные в ее таблицы введены правильно. Например, если в таблице `Orders` хранится информация о заказе, а в `OrderItems` — его детальное описание, вы должны быть уверены, что все идентификаторы заказов, упомянутые в таблице `OrderItems`, существуют и в таблице `Orders`. Аналогично, каждый клиент, упомянутый в таблице `Orders`, не должен быть забыт и в таблице `Customers`.

Хотя вы можете проводить соответствующие проверки, прежде чем вводить новые строки (выполняя оператор

SELECT для другой таблицы, дабы удостовериться в том, что нужные значения правильны), лучше избегать такой практики по следующим причинам.

- Если правила, обеспечивающие целостность базы данных, принудительно осуществляются на клиентском уровне, их придется выполнять каждому клиенту (некоторые из клиентов наверняка не захотят этого делать).
- Вам придется принудительно ввести правила для выполнения операций UPDATE и DELETE.
- Выполнение проверок на клиентской стороне — процесс, отнимающий много времени.

Заставить СУБД выполнять эти проверки — метод намного более эффективный.



### Ограничения

Правила, регламентирующие ввод данных в базу данных и манипуляцию ими.

СУБД принудительно обеспечивают целостность на уровне ссылок за счет ограничений, налагаемых на таблицы базы данных. Большинство ограничений вводится в определениях таблиц (с помощью операторов CREATE TABLE или ALTER TABLE, об этом рассказывалось в уроке 17, “Создание таблиц и работа с ними”).



### Предупреждение

Существует несколько типов ограничений, и каждая СУБД обеспечивает свой собственный уровень их поддержки. Следовательно, примеры, приведенные ниже, могут работать не так, как вы предполагаете. Обратитесь к документации своей СУБД, прежде чем выполнять их.

## Первичные ключи

О первичных ключах мы рассказывали в уроке 1, “Что такое SQL”. Первичный ключ — это особое ограничение, применяемое для того, чтобы значения в столбце (или на-

боре столбцов) были уникальными и никогда не изменялись. Другими словами, это столбец (или столбцы) таблицы, значения которого однозначно идентифицируют каждую строку таблицы. Это облегчает непосредственное манипулирование отдельными строками и взаимодействие с ними. Без первичных ключей было бы очень трудно обновлять или удалять определенные строки, не задевая при этом другие.

Любой столбец таблицы может быть назначен на роль первичного ключа, но только если он удовлетворяет следующим условиям.

- Никакие две строки не могут иметь одно и то же значение первичного ключа.
- Каждая строка должна иметь какое-то значение первичного ключа. (В таких столбцах не должно быть разрешено использование значений NULL.)
- Столбец, содержащий значения первичного ключа, не может быть модифицирован или обновлен.
- Значения первичного ключа ни при каких обстоятельствах не могут быть использованы повторно. Если какая-то строка удалена из таблицы, ее первичный ключ не может быть назначен какой-то другой строке.

Одним из способов определения первичных ключей является их создание:

## ВВОД

```
CREATE TABLE Vendors
(
vend_id CHAR(10) NOT NULL PRIMARY KEY,
vend_name CHAR(50) NOT NULL,
vend_address CHAR(50) NULL,
vend_city CHAR(50) NULL,
vend_state CHAR(5) NULL,
vend_ZIP CHAR(10) NULL
vend_country CHAR(50) NULL
);
```

## Анализ

В этом примере в определение таблицы добавлено ключевое слово PRIMARY KEY, так что столбец vend\_id становится первичным ключом.

**ВВОД**

```
ALTER TABLE Vendors  
ADD CONSTRAINT PRIMARY KEY (vend_id);
```

**Анализ**

Здесь в качестве первичного ключа определен тот же самый столбец, но использован синтаксис CONSTRAINT. Этот синтаксис может быть использован в операторах CREATE TABLE и ALTER TABLE.

**Внешние ключи**

Внешний ключ — это столбец одной таблицы, значения которого совпадают со значениями столбца, являющегося первичным ключом другой таблицы. Внешние ключи — очень важная часть механизма обеспечения ссылочной целостности данных. Чтобы разобраться в том, что собой представляют внешние ключи, рассмотрим следующий пример.

Таблица Orders содержит единственную строку для каждого заказа, зафиксированного в базе данных. Информация о клиенте хранится в таблице Customers. Заказы в таблице Orders связаны с определенными строками в таблице Customers за счет идентификатора клиента. Идентификатор клиента является первичным ключом в таблице Customers; каждый клиент имеет уникальный идентификатор. Номер заказа является первичным ключом в таблице Orders; каждый заказ имеет свой уникальный номер.

Значения в столбце таблицы Orders, содержащем идентификаторы клиентов, не обязательно уникальные. Если клиент сделал несколько заказов, может быть несколько строк с тем же самым идентификатором клиента (хотя каждая из них будет иметь свой номер заказа). В то же время единственные значения, которые могут появиться в столбце идентификаторов клиента таблицы Orders, — это идентификаторы клиентов из таблицы Customers.

Именно так и образуются внешние ключи. В нашем примере внешний ключ определен как столбец идентификаторов клиента, содержащихся в первичном ключе таблицы Customers, так что этот столбец может принимать только значения, имеющиеся в первичном ключе таблицы Customers.

Вот один из способов определения внешнего ключа:

### ВВОД

```
CREATE TABLE Orders
(
  order_num INTEGER NOT NULL PRIMARY KEY,
  order_date DATETIME NOT NULL,
  cust_id CHAR(10) NOT NULL REFERENCES
  Customers(cust_id)
);
```

### Анализ

Это определение таблицы, использующее ключевое слово REFERENCES для утверждения того факта, что любое значение в столбце cust\_id должно быть также и в столбце cust\_id таблицы Customers.

Того же результата можно было бы добиться с использованием синтаксиса CONSTRAINT в операторе ALTER TABLE:

### ВВОД

```
ALTER TABLE Customers
ADD CONSTRAINT
FOREIGN KEY (cust_id) REFERENCES Customers (cust_id)
```



#### Внешние ключи могут воспрепятствовать случайному удалению данных

В дополнение к тому, что внешние ключи помогают принудительно сохранять целостность ссылочных данных, они могут выполнять много других важных функций. После того как внешний ключ определен, ваша СУБД не позволит удалять строки, связанные со строками в других таблицах. Например, вы не сможете удалить информацию о клиенте, у которого есть заказы. Единственный способ удалить информацию о таком клиенте состоит в предварительном удалении связанных с ним заказов (для чего, в свою очередь, нужно удалить информацию о предметах этих заказов). Поскольку требуется столь методичное и целенаправленное удаление, внешние ключи могут оказать помощь в предотвращении случайного удаления данных.

Однако в некоторых СУБД поддерживается возможность, получившая название *каскадное удаление*. Если такая функция реализована, можно удалять все связанные с этой строкой данные при удалении ее из таблицы. Например, если возможно каскадное удаление и имя клиента удаляется из таблицы Customers, все связанные с его заказом строки удаляются автоматически.

## Ограничения уникальности

Ограничения уникальности обеспечивают уникальность всех данных в столбце (или в наборе столбцов). Такие столбцы похожи на первичные ключи, но имеются и важные отличия.

- Таблица может содержать множество ограничений уникальности, но у нее должен быть только один первичный ключ.
- Столбцы с ограничением уникальности могут содержать значения NULL.
- Столбцы с ограничением уникальности можно модифицировать и обновлять.
- Значения столбцов с ограничением уникальности можно использовать повторно.
- В отличие от первичных ключей, ограничения уникальности не могут быть использованы для определения внешних ключей.

Примером использования ограничения может служить таблица с данными о служащих. Каждый из них имеет свой уникальный номер карточки социального страхования, но вы вряд ли будете использовать его в качестве первичного ключа, поскольку он слишком длинный (и, кроме того, вы вряд ли захотите сделать эту информацию легко доступной). Поэтому каждому служащему присваивается уникальный идентификатор (первичный ключ) в дополнение к его номеру карточки социального страхования.

Поскольку идентификатор служащего является первичным ключом, вы можете быть уверены в том, что он уникален. К примеру, для того чтобы СУБД проверила уни-

кальность каждого номера карточки социального страхования (дабы вы могли убедиться в том, что не произошла ошибка при вводе и для одного служащего не указали номер карточки другого), нужно определить ограничение UNIQUE для столбца, в котором содержатся номера карточек социального страхования.

Синтаксис ограничения на уникальность похож на синтаксис других ограничений: при определении таблицы указывается ключевое слово UNIQUE или отдельно используется ограничение CONSTRAINT.

## Ограничения на значения столбца

Ограничения на значения столбца используют для того, чтобы данные в столбце (или наборе столбцов) соответствовали ряду определенных вами критериев. Наиболее часто используемыми из них являются следующие:

- Ограничение максимального и минимального значений — например, для предотвращения появления заказов на 0 (нуль) предметов (хотя 0 и является допустимым числом).
- Указание диапазонов — например, ограничение на то, чтобы дата отгрузки наступала позже или соответствовала текущей дате и не отстояла от нее больше, чем на год.
- Разрешение только определенных значений — например, разрешение вводить в поле “пол” только буквы М или F.

Типы данных (см. урок 1) ограничивают типы данных, которые могут храниться в столбце. Ограничения на значения столбца предъявляют дополнительные требования уже к данным определенного типа.

В следующем примере накладывается ограничение на значения столбцов таблицы OrderItems с тем, чтобы для всех предметов указывалось количество, большее 0:

### ВВОД

```
CREATE TABLE OrderItems
(
  order_num INTEGER NOT NULL,
  order_item INTEGER NOT NULL,
  prod_id CHAR(10) NOT NULL,
```



```
quantity INTEGER NOT NULL CHECK
Ψ(quantity > 0),
item_price MONEY NOT NULL
);
```

### Анализ

После применения этого ограничения каждая добавляемая (или обновляемая) строка будет проверяться на предмет того, что количество предметов больше нуля.

Чтобы проконтролировать тот факт, что в столбце с наименованием пола может содержаться только буква M или F, можно сделать следующее в операторе ALTER TABLE:

### ВВОД

```
ADD CONSTRAINT CHECK (gender LIKE '[MF]')
```



#### Пользовательские типы данных

Пользователи некоторых СУБД могут определять собственные типы данных. Обычно это весьма простые типы данных, определенные с контрольными (или другими) ограничениями. Например, вы можете определить свой тип данных, назвав его `gender` (пол); он будет представлять собой тип данных, состоящих из одной буквы с ограничением на значения столбца, допускающим для этих данных только два значения, M или F (и, возможно, NULL, если пол служащего неизвестен). Вы могли бы использовать этот тип данных в определениях таблиц. Преимущество пользовательских типов данных состоит в том, что такие ограничения могут быть определены только один раз (в определении типа данных), а потом они будут автоматически применяться каждый раз, когда будет использован пользовательский тип данных. Посмотрите в документации своей СУБД, поддерживает ли она пользовательские типы данных.

## Что такое индексы

Индексы используются для логической сортировки данных с целью повышения скорости поиска и выполнения в последующем операций сортировки. Лучший способ понять, что такое индексы — взглянуть на предметный указатель (по-английски — “index”) в конце этой книги.

Предположим, что вы хотите найти вхождения слова *индекс* в книге. Простейшим способом выполнения этой задачи было бы вернуться на ее первую страницу и затем просмотреть каждую строку каждой страницы в поисках соответствий. Хотя такой способ и применим, это, очевидно, неработоспособное решение. Просмотреть несколько страниц текста еще можно, но просматривать подобным образом всю книгу — плохая затея. Чем больше объем текста, в котором нужно провести поиск, тем больше времени требуется на выявление мест вхождения нужных данных.

Именно поэтому книги снабжают предметными указателями. Предметный указатель — это список слов, расположенных в алфавитном порядке, со ссылками на страницы, на которых искомые слова упоминаются в книге. Чтобы найти термин *индекс*, в предметном указателе следует определить, на каких страницах он встречается.

Что делает предметный указатель эффективным средством поиска? Попросту говоря, тот факт, что он правильно отсортирован. Трудность поиска слов в книге обусловлена не тем, что ее объем может быть слишком велик, скорее это обусловлено тем, что ее содержимое не отсортировано в алфавитном порядке. Если бы оно было отсортировано подобно тому, как это делается в словарях, в предметном указателе не было бы необходимости (именно поэтому словари не снабжаются предметными указателями).

Индексы баз данных работают весьма похожим образом. Данные первичного ключа всегда отсортированы — СУБД делает это для вас. Выборка указанных строк по первичному ключу, таким образом, всегда гарантирует быстроту и эффективность этой операции.

Поиск значений в других столбцах, однако, выполняется уже не столь эффективно. Что произойдет, например, если вы захотите сделать выборку всех клиентов, проживающих в определенном штате? Поскольку таблица не отсортирована по названиям штатов, СУБД придется читать каждую

строку таблицы (начиная с самой первой), отыскивая соответствия, точно так же как это сделали бы вы в поисках вхождений слов в книге, не имеющей предметного указателя.

Решение этой проблемы состоит в использовании индекса. Вы можете определить в качестве индекса один или несколько столбцов так, чтобы СУБД хранила отсортированный список содержимого удобным для вас образом. После того как индекс определен, СУБД использует его точно так же, как вы используете предметный указатель книги. Она производит поиск в отсортированном индексе, чтобы найти местоположения всех соответствий и затем выбрать эти строки.

Однако прежде чем создавать множество индексов, примите во внимание следующее.

- Индексы повышают производительность операций выборки, но ухудшают производительность при выполнении таких операций, как добавление данных, их модификация и удаление. Вызвано это тем, что при выполнении подобных операций СУБД должна еще и динамически обновлять индекс.
- Для хранения данных индекса требуется много места на жестком диске.
- Не все данные подходят для индексации. Данные, которые не являются по своей сути уникальными (как, например, названия штатов в столбце `cust_state`), не дадут такого выигрыша от индексации, как данные, которые имеют больше возможных значений (как, например, имя и фамилия).
- Индексы используются для фильтрации и сортировки данных. Если вы часто сортируете данные определенным образом, эти данные могут быть кандидатом на индексацию.
- В качестве индекса можно определить несколько столбцов (например, с названием штата и названием города). Такой индекс можно использовать, только если данные будут отсортированы в порядке “штат плюс город”. (Если вы захотите отсортировать данные по названию города, такой индекс использован не будет).

Не существует твердых правил относительно того, что и когда следует индексировать. В большинстве СУБД предлагаются утилиты, которые можно использовать для опреде-

ления эффективности индексов, ими следует регулярно пользоваться.

Индексы создаются с помощью оператора `CREATE INDEX` (синтаксис которого весьма различен для разных СУБД). Посредством следующего оператора создается простой индекс для столбца с наименованием продукта таблицы `Products`:

## ВВОД

```
CREATE INDEX prod_name_ind
ON PRODUCTS (prod_name);
```

## Анализ

Каждый индекс должен иметь уникальное имя. В данном случае оно определено как `prod_name_ind` после ключевых слов `CREATE INDEX`. Ключевое слово `ON` используется для указания таблицы, которая должна быть проиндексирована, столбцы, включаемые в индекс (в нашем примере он один), указываются в круглых скобках после имени таблицы.



### Пересмотр индексов

Эффективность индексов снижается, если в таблицу добавляются данные или происходит их обновление. Многие администраторы баз данных считают так: то, что когда-то было идеальным набором индексов, может перестать быть таковым после нескольких месяцев манипуляции с данными. Целесообразно регулярно пересматривать индексы и при необходимости осуществлять их точную настройку.

## Что такое триггеры

Триггеры — это особые хранимые процедуры, автоматически выполняемые при использовании базы данных определенным образом. С любой операцией, вызывающей изменение содержимого таблицы, можно связать сопутствующее действие (триггер), которое СУБД должна выполнять при выполнении каждой такой операции. Триггеры могут быть связаны с выполнением операций `INSERT`, `UPDATE` и `DELETE` (или какой-то их комбинацией) по отношению к указанным таблицам.



### Поддержка в MySQL

Когда эта книга готовилась к печати, СУБД MySQL еще не поддерживала триггеры (их поддержку планировалось ввести в версии MySQL 5.1).

В отличие от хранимых процедур (которые представляют собой просто хранимые операторы SQL) триггеры привязаны к отдельным таблицам. Триггер, ассоциирующийся с операциями INSERT по отношению к таблице Orders, будет выполняться только в том случае, если строка добавляется в таблицу Orders. Аналогично, триггер, относящийся к операциям INSERT и UPDATE для таблицы Customers, будет выполняться только в случае выполнения названных операций по отношению к указанной таблице.

Будучи примененным в триггере, ваш код может иметь доступ к следующим данным:

- ко всем новым данным в операциях INSERT;
- ко всем новым и старым данным в операциях UPDATE;
- к удаляемым данным в операциях DELETE.

В зависимости от СУБД, которую вы используете, триггер может выполняться до или после связанной с ним операции.

Чаще всего триггеры используются:

- для обеспечения непротиворечивости данных (например, для преобразования всех названий штатов в верхний регистр во время выполнения операций INSERT или UPDATE);
- для выполнения действий по отношению к другим таблицам, основанных на изменениях, которые были сделаны в какой-то таблице (например, для внесения записи в контрольный журнал с целью регистрации каждого случая обновления или удаления строки);
- для выполнения дополнительной проверки и отмены введения данных (например, чтобы удостовериться в том, что разрешенная для клиента сумма кредита не превышена, в противном случае операция блокируется);
- для подсчета значений вычисляемых полей или обновления меток даты/времени.

Как вы, наверное, уже догадываетесь, синтаксис создания триггеров весьма различен для разных СУБД. За подробностями обратитесь к документации своей СУБД.

В следующем примере создается триггер, преобразующий значения столбца `cust_state` в таблице `Customers` в верхний регистр при выполнении любых операций `INSERT` и `UPDATE`.

Вот версия для SQL Server:

### ВВОД

```
CREATE TRIGGER customer_state
ON Customers
FOR INSERT, UPDATE
AS
UPDATE Customers
SET cust_state = Upper(cust_state)
WHERE Customers.cust_id = inserted.cust_id;
```

А вот версия для Oracle и PostgreSQL:

### ВВОД

```
CREATE TRIGGER customer_state
AFTER INSERT OR UPDATE
FOR EACH ROW
BEGIN
UPDATE Customers
SET cust_state = Upper(cust_state)
WHERE Customers.cust_id = :OLD.cust_id
END;
```



#### Ограничения работают быстрее, чем триггеры

Как правило, ограничения обрабатываются быстрее, чем триггеры, поэтому старайтесь использовать именно их, когда это возможно.

## Безопасность баз данных

Нет ничего более ценного для организации, чем ее данные, поэтому они всегда должны быть защищены от кражи или случайного просмотра. Конечно, данные должны быть в то же время доступны для определенных пользователей, поэтому большинство СУБД предоставляет в распоряжение

своих администраторов механизмы, посредством которых они могут разрешить или ограничить доступ к данным.

В основе любой системы безопасности лежат авторизация и аутентификация пользователей. Так называется процесс, в ходе которого пользователь подтверждает, что он — это именно он и что ему разрешено проводить операции, которые он собирается выполнить. Некоторые СУБД используют для этого средства безопасности операционной системы, другие ведут свои собственные списки пользователей и паролей, третьи интегрируются с внешними серверами службы каталогов.

Часто используются следующие механизмы обеспечения безопасности:

- ограничение доступа к механизмам управления базой данных (создание таблиц, изменение или уничтожение существующих таблиц и т.д.);
- ограничение доступа к отдельным базам данных или таблицам;
- ограничение типа доступа (только для чтения, доступ к отдельным столбцам и т.д.);
- организация доступа к таблицам только через представления или хранимые процедуры;
- создание нескольких уровней безопасности, вследствие чего обеспечивается различная степень доступа и контроля на основе регистрационного имени пользователя;
- ограничение возможности управлять учетными записями пользователей.

Управление системой безопасности осуществляется посредством операторов SQL GRANT и REVOKE, хотя большинство СУБД предлагает интерактивные утилиты администрирования, в которых используются те же операторы GRANT и REVOKE.

## Резюме

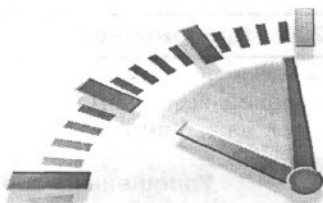
В этом уроке вы узнали, как можно реализовать некоторые расширенные возможности SQL. Ограничения — важная часть системы принудительного обеспечения целостности ссылочных данных, индексы могут улучшить характе-

ристики выборки данных, триггеры можно использовать для обработки данных перед началом или сразу после завершения определенных операций, а опции системы безопасности можно использовать для управления доступом к данным. Возможно, ваша СУБД обеспечивает в той или иной форме эти возможности. Обратитесь к ее документации, чтобы больше узнать об этом.





## Приложение А



# Сценарии демонстрационных таблиц

Процесс написания операторов SQL требует хорошего знания структуры базы данных. Без знания того, какая информация в какой таблице хранится, как таблицы соотносятся одна с другой и как распределены данные в строках, невозможно написать эффективный SQL-код.

Вам настоятельно рекомендуется выполнить в реальных условиях каждый пример каждого урока этой книги. Во всех уроках используется один и тот же набор файлов с данными. Чтобы вам было легче разобраться в этих примерах и выполнять их по мере изучения материала, в этом приложении описываются используемые таблицы, отношения между ними и способы построения таблиц (или их получения).

## Что такое демонстрационные таблицы

Таблицы, используемые на протяжении всей книги, являются частью системы записи заказов воображаемого дистрибьютора игрушек. Эти таблицы служат для решения нескольких задач:

- взаимодействие с поставщиками;
- работа с каталогами продуктов;
- работа со списками клиентов;
- ввод заказов клиентов.

Для выполнения всех этих задач требуется пять таблиц (тесно связанных между собой и являющихся частью реля-

ционной базы данных). В следующих разделах описана каждая из таблиц.



### Упрощенные образцы

Таблицы, используемые в книге, нельзя назвать полными. Реальная система регистрации заказов хранила бы множество других данных, не включенных в представленные таблицы (например, платежи и данные учета, сведения об отгрузке и многие другие). Однако с помощью этих таблиц будет наглядно показано, как организовываются данные и отношения между таблицами в реальных условиях. Вы сможете применить эти методы и технологии по отношению к вашим собственным базам данных.

## Описания таблиц

Далее будут представлены каждая из пяти таблиц с указанием имен столбцов каждой таблицы и их описаниями.

### Таблица Vendors

В таблице `Vendors` хранятся данные о поставщиках, продукты которых продаются. Для каждого поставщика в этой таблице имеется запись, а столбец с идентификатором поставщика (`vend_id`) используется для указания соответствия между продуктами и поставщиками.

**Таблица. А.1. Столбцы таблицы Columns**

| Столбец                   | Описание                                 |
|---------------------------|--|
| <code>vend_id</code>      | Уникальный идентификатор (ID) поставщика |
| <code>vend_name</code>    | Имя поставщика                           |
| <code>vend_address</code> | Адрес поставщика                         |
| <code>vend_city</code>    | Город поставщика                         |
| <code>vend_state</code>   | Штат поставщика                          |
| <code>vend_ZIP</code>     | ZIP-код поставщика                       |
| <code>vend_country</code> | Страна поставщика                        |

- Для всех таблиц должны быть определены первичные ключи. Для данной таблицы в качестве первичного ключа следует использовать ее столбец `vend_id`.

## Таблица Products

Таблица `Products` содержит каталог продуктов, по одному продукту в строке. Каждый продукт имеет уникальный идентификатор (в столбце `prod_id`) и связан с соответствующим поставщиком через `vend_id` (уникальный идентификатор поставщика).

### Таблица А.2. Столбцы таблицы Products

| Столбец                 | Описание   |
|-------------------------|--|
| <code>prod_id</code>    | Уникальный идентификатор (ID) продукта   |
| <code>vend_id</code>    | Идентификатор поставщика продукта (соответствует столбцу <code>vend_id</code> таблицы <code>Vendors</code> ) |
| <code>prod_name</code>  | Название продукта  |
| <code>prod_price</code> | Цена продукта  |
| <code>prod_desc</code>  | Описание продукта  |

- Для всех таблиц нужно определить первичные ключи. Для этой таблицы в качестве первичного ключа следует использовать столбец `prod_id`.
- Для обеспечения целостности ссылочных данных следует определить внешний ключ на основе столбца `vend_id`, связав его с `vend_id` в таблице `Vendors`.

## Таблица Customers

В таблице `Customers` хранится информация обо всех клиентах. Каждый из них имеет уникальный идентификатор (столбец `cust_id`).

### Таблица А.3. Столбцы таблицы Customers

| Столбец                   | Описание                         |
|---------------------------|----------------------------------|
| <code>cust_id</code>      | Уникальный идентификатор клиента |
| <code>cust_name</code>    | Имя клиента                      |
| <code>cust_address</code> | Адрес клиента                    |
| <code>cust_city</code>    | Город клиента                    |

| Столбец      | Описание                                   |
|--------------|--|
| cust_state   | Штат клиента                               |
| cust_ZIP     | ZIP-код клиента                            |
| cust_country | Страна клиента                             |
| cust_contact | Контактное имя клиента                     |
| cust_email   | Контактный адрес электронной почты клиента |

- Для всех таблиц следует определить первичные ключи. Для этой таблицы в качестве первичного ключа следует использовать столбец `cust_id`.

### Таблица Orders

В таблице `Orders` хранится информация о заказах клиентов (без подробностей). Каждый заказ пронумерован с соблюдением правила уникальности (столбец `order_num`). Заказы связаны с соответствующими клиентами через столбец `cust_id` (который связан с уникальным идентификатором клиента в таблице `Customers`).

#### Таблица А.4. Столбцы таблицы Orders

| Столбец    | Описание  |
|------------|---|
| order_num  | Уникальный номер заказа   |
| order_date | Дата заказа   |
| cust_id    | Идентификатор клиента, сделавшего заказ (связан со столбцом <code>cust_id</code> таблицы <code>Customers</code> ) |

- Для всех таблиц следует определить первичные ключи. Для этой таблицы в качестве первичного ключа следует использовать столбец `order_num`.
- Для обеспечения целостности ссылочных данных следует определить внешний ключ на основе столбца `cust_id`, связав его с `cust_id` в таблице `Customers`.

### Таблица OrderItems

В таблице `OrderItems` хранятся предметы каждого заказа, для каждого предмета каждого заказа выделено по одной строке. Каждой строке таблицы `Orders` соответствует

одна или несколько строк в таблице OrderItems. Каждый предмет заказа идентифицирован уникальным образом посредством номера заказа в совокупности с предметом заказа (первый предмет заказа, второй предмет заказа и т.д.).

Предмет заказа связан с соответствующими ему заказами через столбец order\_num (который соотносит его с уникальным идентификатором заказа в таблице Orders). Кроме того, каждая запись о предмете заказа содержит идентификатор продукта (который опять связывает предмет с таблицей Products).

**Таблица А.5. Столбцы таблицы OrderItems**

| <i>Столбец</i> | <i>Описание</i>  |
|----------------|--|
| order_num      | Номер заказа (связан со столбцом order_num в таблице Orders)           |
| order_item     | Номер предмета заказа (последовательно присваиваемый внутри заказа)    |
| prod_id        | Идентификатор продукта (связан со столбцом prod_id в таблице Products) |
| quantity       | Количество предметов   |
| item_price     | Цена предмета  |

- Для всех таблиц следует определить первичные ключи. В качестве первичных ключей для этой таблицы следует использовать столбцы order\_num и order\_item.
- Для обеспечения целостности ссылочных данных следует определить внешние ключи на основе столбца order\_num, связав его с order\_num в таблице Orders, и prod\_id, связав его с prod\_id таблицы Products.

## Получение демонстрационных таблиц

Чтобы попрактиковаться в выполнении представленных в книге примеров, вам нужен будет набор заполненных таблиц. Все, что вам необходимо получить и испытать, можно найти на Web-странице этой книги по адресу <http://www.forta.com/books/0672325675/>.

## Загрузка готового к работе MDB-файла для Microsoft Access

Вы можете загрузить полностью готовый MDB-файл для Microsoft Access с вышеуказанной страницы. Если вы воспользуетесь этим файлом, вам не придется выполнять какие-либо сценарии создания и заполнения.

MDB-файл Access можно использовать с любыми клиентскими утилитами ODBC, а также с интерпретаторами языков сценариев, таких как ASP и ColdFusion.

## Загрузка SQL-сценариев СУБД

Большинство СУБД хранят данные в форматах, которые не позволяют распространять файлы баз данных (в отличие от Access). Для таких СУБД вы можете загрузить SQL-сценарии со страницы <http://www.forta.com/books/0672325675/>.

Для каждой СУБД имеется два файла.

- Файл `create.txt`, содержащий SQL-операторы, необходимые для создания пяти таблиц базы данных (включая определения всех первичных ключей и ограничений внешних ключей).
- Файл `populate.txt`, содержащий SQL-операторы INSERT, используемые для заполнения этих таблиц.

Операторы SQL, содержащиеся в этих файлах, весьма различны для разных СУБД, поэтому следует убедиться, что вы выполняете именно тот, который соответствует вашей СУБД. Эти сценарии предназначены для удобства чтения книги, и никакая ответственность за возможные проблемы при их использовании не предполагается.

К тому времени, когда эта книга уходила в печать, были доступны сценарии для следующих СУБД:

- IBM DB2;
- Microsoft SQL Server;
- MySQL;
- Oracle;
- PostgreSQL;
- Sybase Adaptive Server.

При необходимости этот список может быть пополнен другими СУБД.

В приложении Б, “Работа с популярными приложениями”, приведены инструкции по выполнению сценариев для нескольких популярных сред программирования.

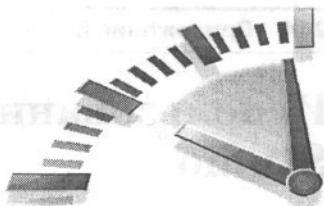
**Вначале создать, потом заполнять**

Следует вначале выполнять сценарии создания таблиц, а потом уже их заполнения. Убедитесь в том, что никаких сообщений об ошибках эти сценарии не возвратили. Если сценарии создания потерпят неудачу, нужно вначале выявить и устранить возникшую проблему, а потом уже заполнять таблицы.





## Приложение Б



# Работа с популярными приложениями

Как говорилось в уроке 1, “Что такое SQL,” SQL — это не приложение, но язык. Для того чтобы выполнить примеры этой книги, вам необходимо приложение, поддерживающее выполнение SQL-операторов.

В приложении Б описываются шаги по выполнению операторов SQL в некоторых наиболее часто используемых приложениях.

Вы можете использовать любое приложение из числа указанных ниже, а также многие другие для проверки SQL-кода и экспериментирования с ним. Итак, какое же приложение использовать?

- Многие СУБД поставляются со своими собственными клиентскими утилитами, с ними вполне можно начинать работу. Однако для них не характерен интуитивно понятный пользовательский интерфейс.
- Пользователи Windows, возможно, установили на своем компьютере утилиту Microsoft Query. Эта простая утилита очень хороша для проверки простых операторов.
- Прекрасная альтернатива для пользователей Windows — Query Tool Джорджа Пулоса (George Poulouse). Ссылку можно найти на Web-странице книги по адресу <http://www.forta.com/books/0672325667/>.
- Aqua Data Studio — чрезвычайно полезная бесплатная Java-утилита, которая работает под управлением Windows, Linux, Unix, Mac OSX и других операционных систем. Ссылку на эту утилиту можно найти на Web-странице книги по адресу <http://www.forta.com/books/0672325667/>.

Любая из названных утилит будет хорошим выбором. За дополнительными рекомендациями обратитесь к Web-странице этой книги.

## Использование Aqua Data Studio

Aqua Data Studio — это бесплатный SQL-клиент, основанный на технологии Java. Он выполняется на всех главных платформах и поддерживает все наиболее распространенные СУБД (а также ODBC<sup>1</sup>). Чтобы выполнить какой-нибудь SQL-оператор в Aqua Data Studio, сделайте следующее.

1. Запустите Aqua Data Studio.
2. Прежде чем СУБД можно будет использовать, ее необходимо зарегистрировать. Выберите в меню Select пункт Register Server.
3. Выберите СУБД, которую вы используете, в отображенном на экране списке (выберите Generic ODBC для использования Microsoft Access или произвольную базу данных ODBC — для этого требуется, чтобы был определен источник данных ODBC, о чем будет сказано в конце данного приложения). На основании выбранной СУБД вам будет предложен путь к файлу или информация относительно регистрации. Заполните форму и щелкните на кнопке ОК. После завершения регистрации этот сервер будет появляться в списке слева.
4. Выберите сервер в списке зарегистрированных серверов.
5. Запустите Query Analyzer, выбрав команду Query Analyzer в меню Server или нажав <Ctrl+Q>.
6. Введите свой SQL-код в окне запроса (верхнее окно).
7. Чтобы выполнить SQL-код, выберите команду Execute в меню Query, или нажмите <Ctrl+E>, или щелкните на кнопке Execute (кнопке с зеленой стрелкой).
8. Результаты появятся в нижнем окне.

---

<sup>1</sup> ODBC (сокр. от *Open DataBase Connectivity*) — открытый интерфейс доступа к базам данных, встроенный в Windows и Windows NT, определяет набор функций, которые можно использовать для доступа к любой реляционной СУБД. — Прим. ред.

## Использование DB2

СУБД DB2 компании IBM — это мощная высокопроизводительная многоплатформенная СУБД. Она поставляется с целым набором клиентских инструментов, которые могут быть использованы для выполнения операторов SQL.

Приведенные ниже инструкции предназначены для Java-утилиты Command Center — одной из самых простых и наиболее универсальной среди всех приложений.

1. Запустите Command Center.
2. Выберите вкладку Script.
3. Введите оператор SQL в поле Script.
4. Выберите команду Execute в меню Script или щелкните на кнопке Execute, чтобы выполнить этот сценарий.
5. Результирующие данные в необработанном виде будут отображены в нижнем окне. Перейдите на вкладку Results, чтобы отобразить результаты в виде таблицы.
6. Command Center предлагает также интерактивный формирователь SQL-операторов, называемый SQL Assist. Его можно запустить из вкладки Interactive.

## Использование Macromedia ColdFusion

ColdFusion компании Macromedia представляет собой платформу для разработки Web-приложений.

ColdFusion использует для создания сценариев язык, основанный на дескрипторах (тегах). Чтобы протестировать ваш SQL-код, создайте простую страницу, которую вы сможете отобразить, вызвав ее посредством своего Web-браузера. Выполните следующие шаги:

1. Прежде чем вы сможете обращаться к каким-либо базам данных из ColdFusion, должен быть определен источник данных (Data Source). Программа ColdFusion Administrator обеспечивает Web-интерфейс для определения источников данных (обратитесь за помощью к документации ColdFusion, если это необходимо).

2. Создайте новую страницу ColdFusion (с расширением CFM).
3. Используйте дескрипторы CFML `<CFQUERY>` и `</CFQUERY>` для создания блока запроса. Назовите его, используя атрибут NAME, и определите источник данных в атрибуте DATASOURCE.
4. Введите свой SQL-оператор между дескрипторами `<CFQUERY>` и `</CFQUERY>`.
5. Используйте цикл `<CFDUMP>` или `<CFOUTPUT>` для отображения результатов запроса.
6. Сохраните страницу в каком-нибудь каталоге исполняемых файлов корневого каталога Web-сервера.
7. Отобразите страницу, вызвав ее из Web-браузера.

## Использование Microsoft Access

Microsoft Access обычно используется интерактивно для создания баз данных, управления, манипулирования и взаимодействия с данными, а Access предлагает еще и конструктор запросов (Query Designer), который можно использовать для интерактивного построения операторов SQL. Зачастую остающаяся невыявленной возможность конструктора запросов состоит в том, что он также позволяет вводить SQL-код для немедленного выполнения. Благодаря этому Access можно использовать для передачи операторов SQL любому источнику данных ODBC, хотя больше всего эта программа подходит для выполнения SQL-кода в уже открытой базе данных. Для того чтобы использовать названную возможность, сделайте следующее.

1. Запустите Microsoft Access. Вам предложат открыть (или создать) базу данных. Откройте базу данных, которую вы собираетесь использовать.
2. Выберите меню Запросы в окне с названием вашей базы данных, а затем дважды щелкните на ссылке Создание запроса в режиме конструктора (либо щелкните на кнопке Создать и выберите в появившемся окне пункт Конструктор).

3. Появится диалоговое окно Добавление таблицы. Закройте его, не выбрав ни одну из таблиц.
4. Выберите команду Режим SQL в меню Вид.
5. Введите ваш оператор SQL в окне запроса.
6. Чтобы выполнить оператор SQL, щелкните на кнопке Запуск (она помечена восклицательным знаком). Результаты будут отображены в этом же окне, но в режиме таблицы.
7. Переходите при необходимости от режима ввода запросов (вам нужно будет повторно выполнить команду Режим SQL для изменения вашего SQL-кода) к режиму отображения их результатов. Вы можете также использовать режим Создание запроса с помощью мастера для интерактивного построения операторов SQL.

Microsoft Access также поддерживает режим запроса к серверу, который позволяет использовать Access для отправки SQL-операторов любому источнику данных ODBC. Эту возможность следует использовать для взаимодействия с внешними базами данных, и никогда — для непосредственного взаимодействия с Access. Чтобы воспользоваться этой возможностью, выполните следующее.

1. Microsoft Access для взаимодействия с базами данных использует ODBC, так что в системе должен присутствовать источник данных ODBC. Только в этом случае можно будет начать обработку запросов (см. предыдущие инструкции).
2. Запустите Microsoft Access. Вам предложат открыть (или создать) базу данных. Откройте какую-нибудь базу данных.
3. Выберите меню Запросы в окне с названием вашей базы данных, а затем дважды щелкните на ссылке Создание запроса в режиме конструктора (либо щелкните на кнопке Создать и выберите в появившемся окне пункт Конструктор).
4. Появится диалоговое окно Добавление таблицы. Закройте его, не выбрав ни одну из таблиц.
5. В меню Запрос выберите подменю Запрос SQL, а затем — команду К серверу.

6. В меню Вид выберите команду Свойства, чтобы отобразить диалоговое окно Свойства запроса.
7. Щелкните в поле Строка подключения ODBC, а затем — на кнопке с многоточием (...), чтобы отобразить диалоговое окно Выбор источника данных, которое вы будете использовать для выбора источника данных ODBC.
8. Выберите ваш источник данных и щелкните на кнопке ОК, чтобы вернуться в диалоговое окно Свойства запроса.
9. Щелкните в поле списка Возврат записей. Если вы выполняете оператор SELECT (или другой оператор, возвращающий результаты), выберите Да. Если же вы выполняете оператор SQL, который не возвращает данные (например, INSERT, UPDATE или DELETE), выберите Нет.
10. Введите свой оператор SQL в окне запроса к серверу.
11. Чтобы выполнить этот оператор SQL, щелкните на кнопке Запуск (на ней изображен восклицательный знак красного цвета).



#### Использование режима запроса к серверу

Режим запроса к серверу лучше работает при подключении к СУБД, отличной от Access. Если устанавливается связь с MDB-файлом Access, лучше использовать какие-то другие клиентские средства, о которых говорилось выше.

## Использование Microsoft ASP

Microsoft ASP — это платформа подготовки сценариев, ориентированная на создание Web-приложений.

Для того чтобы протестировать ваши операторы SQL на странице ASP, вам вначале придется создать страницу, которую вы сможете отобразить на экране, вызвав ее с помощью своего Web-браузера. Ниже перечислены шаги, которые необходимо сделать для выполнения оператора SQL на странице ASP.

1. ASP для взаимодействия с базами данных использует ODBC, поэтому прежде чем начать работу, вам следует побеспокоиться об источнике данных ODBC, дополнительную информацию можно найти в конце этого приложения.
2. Создайте новую страницу ASP (с расширением ASP), используя любой текстовый редактор.
3. Используйте метод `Server.CreateObject` для создания экземпляра объекта `ADODB.Connection`.
4. Используйте метод `Open` для открытия нужного источника данных ODBC.
5. Передайте ваш оператор SQL методу `Execute` в качестве аргумента. Метод возвратит результирующее множество. Используйте команду `Set` для сохранения полученных данных.
6. Чтобы отобразить эти результаты, воспользуйтесь циклом `<% Do While NOT EOF %>`.
7. Сохраните эту страницу в любом каталоге исполняемых файлов корневого каталога Web-сервера.
8. Откройте страницу, вызвав ее посредством Web-браузера.

## Использование Microsoft ASP.NET

Microsoft ASP.NET — это платформа подготовки сценариев для создания Web-приложений с использованием технологии .NET. Чтобы протестировать операторы SQL на странице ASP.NET, создайте страницу, которую можно отобразить, вызвав ее посредством браузера. Это можно сделать разными способами, ниже описан один из них.

1. Создайте новый файл с расширением `.aspx`.
2. Создайте подключение к базе данных, используя функцию `SqlConnection()` или `OleDbConnection()`.
3. Используйте функцию `SqlCommand()` или `OleDbCommand()` для передачи оператора в СУБД.



4. Создайте объект `DataReader`, используя метод `ExecuteReader`.
5. Последовательно обработайте все записи, содержащиеся в объекте, для получения возвращаемых значений.
6. Сохраните эту страницу в любом каталоге исполняемых файлов корневого каталога Web-сервера.
7. Откройте страницу, вызвав ее посредством Web-браузера.

## Использование Microsoft Query

Microsoft Query — это стандартный инструмент подготовки SQL-запросов, он является идеальной утилитой для тестирования операторов SQL с использованием источников данных ODBC. Утилита Microsoft Query опционально устанавливается вместе с другими программами Microsoft, а также с программами других фирм.



### Получение MS-Query

MS-Query часто устанавливается на компьютер вместе с другими программами Microsoft (например, Office), хотя это происходит только при полной установке пакета. Если она не присутствует в меню Пуск, воспользуйтесь командой Пуск⇒Найти⇒Файлы и папки, чтобы найти эту утилиту в своей системе.

(Утилита часто присутствует в системе, однако вы об этом можете не знать.) Файл, который нужно искать, называется MSQRY32.EXE или MSQUERY.EXE.

Для использования Microsoft Query необходимо выполнить следующее.

1. Microsoft Query использует ODBC для взаимодействия с базами данных, поэтому прежде чем вы начнете работу, на компьютере должен быть создан источник данных ODBC (дополнительную информацию можно найти в конце данного приложения).
2. Прежде чем вы сможете использовать утилиту Microsoft Query, она должна быть установлена на ва-

шем компьютере. Просмотрите список программ вашего компьютера, открывающийся после щелчка на кнопке Пуск, и найдите утилиту.

3. В меню Файл утилиты выберите команду Выполнить запрос SQL. Откроется окно Выполнение запроса SQL.
4. Щелкните на кнопке Источники, чтобы выбрать источник данных ODBC. Если нужный вам источник отсутствует в списке, щелкните на кнопке Обзор, чтобы найти его. После того как будет выбран нужный источник данных, щелкните на кнопке ОК.
5. Введите ваш оператор SQL в поле Инструкция SQL.
6. Щелкните на кнопке Выполнить, чтобы выполнить оператор SQL и отобразить полученные данные.

## Использование Microsoft SQL Server

Microsoft SQL Server предлагает основанный на Windows инструментарий анализа запросов, называемый SQL Query Analyzer. Хотя это средство создания запросов в основном предназначено для анализа процесса выполнения операторов SQL и их оптимизации, оно идеально подходит для тестирования SQL-операторов и экспериментирования с ними.

Ниже даны пошаговые инструкции по использованию SQL Query Analyzer.

1. Запустите приложение SQL Query Analyzer (его можно найти в группе программ Microsoft SQL Server).
2. Вам будет представлена информация о сервере и предложено зарегистрироваться. Зарегистрируйтесь в программе SQL Server (запустив сервер, если это еще не сделано).
3. После того как отобразится экран запросов, выберите базу данных в раскрывающемся списке.
4. Введите свой SQL-код в большом текстовом окне и затем щелкните на кнопке Execute Query (с зеленой стрелкой), чтобы выполнить его. (Вы можете также нажать клавишу <F5> или выбрать команду Execute в меню Query.)

5. Результаты будут отображены на отдельной панели под окном SQL.
6. Щелкните на вкладке внизу экрана запроса, чтобы переключиться между режимами просмотра данных и просмотра полученных сообщений.

## Использование MySQL

СУБД MySQL поставляется вместе с утилитой командной строки, называемой `mysql`. Это сугубо текстовое средство создания запросов, которое можно использовать для выполнения любых операторов SQL. Чтобы воспользоваться утилитой `mysql`, выполните следующее.

1. Введите `mysql`, чтобы запустить эту утилиту. В зависимости от ограничений, налагаемых системой безопасности, вам может понадобиться использовать параметры `-u` и `-p`, чтобы ввести регистрационную информацию.
2. В ответ на приглашение `mysql>` введите `USE база_данных`, указывая тем самым имя базы данных, которая будет использоваться.
3. Введите ваш SQL-код после приглашения `mysql>`, проверив, чтобы каждый оператор заканчивался точкой с запятой (;). Результаты будут отображены на экране.
4. Введите `\h` для получения списка команд, которые вы можете использовать, и `\s` для получения информации о статусе (включая информацию о версии MySQL).
5. Введите `\q` для выхода из утилиты `mysql`.

## Использование Oracle

СУБД Oracle поставляется с основанным на Java-технологии средством управления, которое называется Enterprise Manager. На самом деле это набор инструментов, один из которых называется SQL\*Plus Worksheet. Ниже рассказано, как им нужно пользоваться.

1. Запустите SQL\*Plus Worksheet (или напрямую, или из Oracle Enterprise Manager).
2. Вам предложат ввести регистрационные данные. Введите имя пользователя и пароль, чтобы подключиться к серверу базы данных.
3. Экран SQL Worksheet разделен на две панели. Введите ваш SQL-код в верхней.
4. Чтобы выполнить оператор SQL, щелкните на кнопке Execute (на ней изображена молния). Результаты будут отображены в нижней панели.

## Использование PHP

PHP — это популярный язык написания Web-сценариев. PHP предлагает функции и библиотеки, используемые для подключения к различным базам данных, а поэтому код, используемый для выполнения операторов SQL, может меняться в зависимости от того, какая СУБД используется (и как к ней осуществляется доступ). А раз так, то невозможно предложить пошаговые инструкции, которые годились бы для любой ситуации. Обратитесь к документации PHP за инструкциями по подключению именно к вашей СУБД.

## Использование PostgreSQL

PostgreSQL поставляется с утилитой командной строки, которая называется `psql`. Это — сугубо текстовое средство создания запросов, которое можно использовать для выполнения любых операторов SQL. Для того чтобы воспользоваться утилитой `psql`, выполните следующее.

1. Введите `psql`, чтобы запустить утилиту. Чтобы загрузить конкретную базу данных, укажите ее в командной строке как `psql база_данных` (PostgreSQL не поддерживает команду USE).
2. Введите ваш SQL-код в ответ на приглашение `=>`, убедившись в том, что каждый оператор заканчивается точкой с запятой (`;`). Результаты будут отображены на экране.

3. Введите \?, чтобы отобразить список команд, которые вы можете использовать.
4. Введите \h, чтобы получить справку по SQL, \h — чтобы получить справку относительно конкретного оператора SQL (например, \h SELECT).
5. Введите \q, чтобы выйти из утилиты psql.

## Использование Query Tool

Query Tool — это стандартный инструмент запросов, созданный Джорджем Пулосом (George Poulou). Этот инструмент является идеальной утилитой для тестирования операторов SQL при использовании источников данных ODBC. (Существует также версия для ADO<sup>2</sup>.)



### Получение Query Tool

Утилита Query Tool может быть загружена через Internet. Чтобы получить ее копию, перейдите на Web-страницу книги по ссылке: <http://www.forta.com/books/0672321289/>.

Чтобы воспользоваться утилитой Query Tool, выполните следующее:

1. Query Tool использует ODBC для взаимодействия с базами данных, поэтому источник данных ODBC должен уже иметься на компьютере, прежде чем вы начнете работу (см. инструкции, которые были даны выше).
2. Прежде чем вы начнете использовать Query Tool, эта утилита должна быть установлена на вашем компьютере. Просмотрите список установленных на нем программ и найдите ее.
3. В появившемся диалоговом окне вам предложат выбрать источник данных ODBC, который будет использоваться. Если нужный вам источник данных в спи-

<sup>2</sup> ADO (сокр. от *ActiveX Data Objects*) — технология доступа к данным, включающая набор высокоуровневых интерфейсов, которые позволяют разработчикам обращаться к данным на любом языке программирования. — *Прим. ред.*

ске не представлен, щелкните на кнопке New, чтобы создать его. После того как будет правильно выбран источник данных, щелкните на кнопке OK.

4. Введите ваш оператор SQL в правом верхнем окне.
5. Щелкните на кнопке Execute (на ней изображена голубая стрелка), чтобы выполнить оператор SQL и отобразить возвращаемые данные в нижней панели. (Вы можете также нажать клавишу <F5> или выбрать команду Execute в меню Query.)

## Использование Sybase

Sybase Adaptive Server поставляется с Java-утилитой SQL Advantage. Данная утилита очень похожа на Query Analyzer, поставляемую вместе с СУБД Microsoft SQL Server (эти продукты имеют общее прошлое). Для того чтобы воспользоваться SQL Advantage, выполните следующее:

1. Вызовите приложение SQL Advantage.
2. Получив предложение зарегистрироваться, введите свое регистрационное имя и пароль.
3. После того как появится окно для ввода запроса, выберите базу данных в раскрывающемся списке.
4. Введите в появившемся окне ваш SQL-код.
5. Чтобы выполнить запрос, щелкните на кнопке Execute, выберите команду Execute Query в меню Query или нажмите <Ctrl+E>.
6. Результаты (если они есть) будут отображены в новом окне.

## Конфигурирование источников данных ODBC

Несколько приложений из числа описанных выше используют для интеграции с базами данных протокол ODBC, поэтому мы начнем с краткого обзора ODBC и инструкций по конфигурированию источников данных ODBC.

ODBC — это стандарт, который используется для обеспечения того, чтобы клиентские приложения могли взаимодействовать с различными компьютерами, на которых размещены базы данных, или процессорами баз данных. При использовании ODBC можно написать код с помощью одного клиента, и он будет взаимодействовать почти с любой базой данных или СУБД.

ODBC сам по себе не является базой данных. Скорее ODBC представляет собой оболочку для баз данных, позволяющую всем базам данных вести себя непротиворечивым и четко выраженным образом. Он добивается этого за счет использования программных драйверов, выполняющих две основные функции. Во-первых, они инкапсулируют некоторые характерные для отдельных баз данных особенности и скрывают их от клиента. Во-вторых, они обеспечивают общий язык для взаимодействия с этими базами данных (при необходимости выполняя нужное преобразование). Язык, используемый ODBC, — это SQL.

Клиентские приложения ODBC не взаимодействуют с базами данных непосредственно. Вместо этого они взаимодействуют с источниками данных ODBC (ODBC Data Sources). Источник данных ODBC представляет собой логическую базу данных, которая включает в свой состав драйвер (база данных каждого типа имеет свой собственный драйвер) и информацию о том, как нужно подключаться к этой базе данных (пути к файлам, имена серверов и т.д.).

После того как источники данных ODBC определены, ими может пользоваться любое ODBC-совместимое приложение. Источники данных ODBC не специфичны для приложений, они специфичны для систем.



#### Различия в реализации

Существует много версий системных модулей ODBC, поэтому невозможно дать четкие инструкции, применимые ко всем версиям. Обратите внимание на приглашения, когда будете устанавливать свои источники данных.

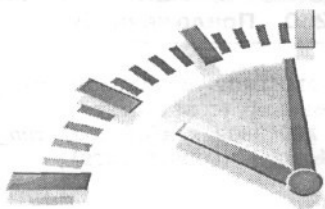
Источники данных ODBC определяются с помощью ODBC-модуля панели управления Windows. Чтобы установить какой-либо источник данных ODBC, выполните следующее.

1. Откройте ODBC-модуль панели управления Windows.
2. Большинство источников данных ODBC должны быть общесистемными (в противоположность источникам данных, специфичным для пользователя), поэтому выберите вкладку Системный DSN, если она доступна для вас.
3. Щелкните на кнопке Добавить, чтобы добавить новый источник данных.
4. Выберите драйвер, который будет использоваться. Обычно по умолчанию доступен набор драйверов, обеспечивающих поддержку большинства продуктов компании Microsoft. В вашей системе могут быть установлены другие драйверы. Вы можете выбрать драйвер, соответствующий типу базы данных, к которой вы собираетесь подключаться.
5. В зависимости от типа базы данных или СУБД вам предложат ввести имя сервера или путь к файлу и, возможно, регистрационную информацию. Введите запрашиваемую информацию и следуйте остальным инструкциям, чтобы создать источник данных.





## Приложение В



# Синтаксис операторов SQL

Для того чтобы помочь вам быстро найти образец нужного синтаксиса, в этом приложении приводятся образцы для наиболее часто выполняемых операций SQL. Каждый оператор начинается с краткого описания, затем приводится соответствующий синтаксис.

Для большего удобства даются также ссылки на уроки, в которых изучались соответствующие операторы.

При рассмотрении синтаксиса операторов помните следующее:

- Символ “|” служит для выбора одного из нескольких вариантов, поэтому NULL|NOT NULL означает указание NULL или NOT NULL.
- Ключевые слова или предложения, заключенные в квадратные скобки, [например, так], являются опциональными.
- Представленный ниже синтаксис подходит почти для любой СУБД. Рекомендуем обращаться к документации вашей СУБД за подробностями относительно возможных изменений в синтаксисе.

## ALTER TABLE

Оператор ALTER TABLE используется для обновления схемы существующей таблицы. Чтобы создать новую таблицу, используйте оператор CREATE TABLE. За более детальной информацией обратитесь к уроку 17, “Создание таблиц и работа с ними”.

### ВВОД

```
ALTER TABLE имя_таблицы
```

```
(
```

```

ADD|DROP имя_столбца тип_данных [NULL|NOT
☞NULL] [CONSTRAINTS],
ADD|DROP имя_столбца тип_данных [NULL|NOT
☞NULL] [CONSTRAINTS],
...
);

```

## COMMIT

Оператор COMMIT используется для создания запросов к базе данных в виде транзакций. За более детальной информацией обратитесь к уроку 20, “Обработка транзакций”.

### ВВОД

```
COMMIT [TRANSACTION];
```

## CREATE INDEX

Оператор CREATE INDEX используется для создания индекса одного или нескольких столбцов. За более детальной информацией обратитесь к уроку 22, “Расширенные возможности SQL”.

### ВВОД

```
CREATE INDEX название_индекса
ON имя_таблицы (имя_столбца, ...);
```

## CREATE PROCEDURE

Оператор CREATE PROCEDURE используется для создания хранимых процедур. За более детальной информацией обратитесь к уроку 19, “Работа с хранимыми процедурами”. В СУБД Oracle используется синтаксис, отличный от описанного в этом уроке.

### ВВОД

```
CREATE PROCEDURE имя_процедуры [параметры]
☞ [опции]
AS
SQL statement;
```

## CREATE TABLE

Оператор CREATE TABLE используется для создания новых таблиц базы данных. Чтобы обновить схему уже существующей таблицы, используйте оператор ALTER TABLE. За более детальной информацией обратитесь к уроку 17.

### ВВОД

```
CREATE TABLE имя_таблицы
(
  имя_столбца тип_данных [NULL|NOT NULL]
  ⚡ [CONSTRAINTS],
  имя_столбца тип_данных [NULL|NOT NULL]
  ⚡ [CONSTRAINTS],
  ...
);
```

## CREATE VIEW

Оператор CREATE VIEW используется для создания нового представления одной или нескольких таблиц. За более детальной информацией обратитесь к уроку 18, “Использование представлений”.

### ВВОД

```
CREATE VIEW имя_представления AS
SELECT имена_столбцов, ...
FROM tables, ...
[WHERE ...]
[GROUP BY ...]
[HAVING ...];
```

## DELETE

Оператор DELETE удаляет одну или несколько строк таблицы. За более детальной информацией обратитесь к уроку 16, “Обновление и удаление данных”.

### ВВОД

```
DELETE FROM имя_таблицы
[WHERE ...];
```

## DROP

Оператор DROP навсегда удаляет объекты базы данных (таблицы, представления, индексы и т.д.). За более детальной информацией обратитесь к урокам 17 и 18.

### ВВОД

```
DROP INDEX | PROCEDURE | TABLE | VIEW
```

```
☞ имя_индекса | имя_процедуры | имя_таблицы | имя_представления;
```

## INSERT

Оператор INSERT добавляет в таблицу одну строку. За более детальной информацией обратитесь к уроку 15, "Добавление данных".

### ВВОД

```
INSERT INTO имя_таблицы [(имена_столбцов, ...)]  
VALUES (значения, ...);
```

## INSERT SELECT

Оператор INSERT SELECT добавляет результаты выполнения оператора SELECT в таблицу. За более детальной информацией обратитесь к уроку 15.

### ВВОД

```
INSERT INTO имя_таблицы [(имена_столбцов, ...)]  
SELECT имена_столбцов, ... FROM имя_таблицы, ...  
[WHERE ...];
```

## ROLLBACK

Оператор ROLLBACK используется для аннулирования результатов работы блока транзакции. За более детальной информацией обратитесь к уроку 20.

### ВВОД

```
ROLLBACK [ TO точка_сохранения];
```

или

**ВВОД**

```
ROLLBACK TRANSACTION;
```

## SELECT

Оператор `SELECT` используется для выборки данных из одной или нескольких таблиц (или представлений). За более детальной информацией обратитесь к уроку 2, “Выборка данных”; уроку 3, “Сортировка выбранных данных”; и уроку 4, “Фильтрация данных”. (Во всех уроках со 2 по 14 рассматриваются аспекты применения оператора `SELECT`.)

**ВВОД**

```
SELECT имя_столбца, ...  
FROM имя_таблицы, ...  
[WHERE ...]  
[UNION ...]  
[GROUP BY ...]  
[HAVING ...]  
[ORDER BY ...];
```

## UPDATE

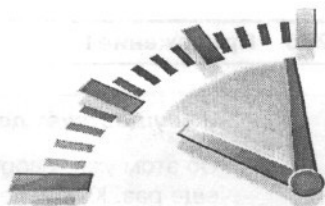
Оператор `UPDATE` обновляет одну или несколько строк таблицы. За более детальной информацией обратитесь к уроку 16.

**ВВОД**

```
UPDATE имя_таблицы  
SET имя_столбца = значение, ...  
[WHERE ...];
```



## Приложение Г



# Использование типов данных SQL

Как уже говорилось в уроке 1, “Что такое SQL”, типы данных представляют собой основные правила, определяющие, какие данные могут храниться в столбцах и в каком виде эти данные в действительности хранятся.

Типы данных используются по нескольким причинам.

- Типы данных позволяют ограничить разновидности данных, которые могут храниться в столбце. Например, столбцы с данными числового типа будут принимать только числовые данные.
- Типы данных позволяют более эффективно организовать хранение. Числовые значения и значения даты/времени могут храниться в более компактном формате, чем текстовые строки.
- Типы данных позволяют изменять порядок сортировки. Если все данные трактуются как строки, то 1 предшествует 10, а 10 предшествует 2.
- (Строки сортируются в лексикографической последовательности, по одному символу за один раз, начиная слева). Если выполняется числовая сортировка, то числа будут располагаться по возрастанию.

При разработке таблиц обращайтесь пристальное внимание на используемые в них типы данных. При использовании неправильных типов данных работа вашего приложения серьезно осложнится.

Изменение типов данных уже имеющихся и заполненных столбцов — задача нетривиальная. (Кроме того, при ее выполнении возможна потеря данных.)

В одном приложении невозможно дать исчерпывающую информацию по типам данных и методам их использования, однако здесь все же рассмотрены основные типы данных, рассказано, для чего они используются, а также раскрываются возможные проблемы совместимости.





### Не существует двух одинаковых СУБД

Об этом уже говорилось, но не лишним будет сказать еще раз. К сожалению, в разных СУБД используются существенно разные типы данных. Даже если названия типа данных звучат одинаково, пониматься под одним и тем же типом данных в разных СУБД может не одно и то же. Непременно обратитесь к документации своей СУБД и выясните, какие в точности типы данных она поддерживает и каким образом.

## Строковые данные

Чаще всего используются данные типа строки, или строковые данные. К ним относятся хранимые в СУБД строки, например, имена, адреса, номера телефонов и ZIP-коды.

Вам придется использовать строковые данные в основном двух типов — строки фиксированной длины и строки переменной длины (табл. Г.1).

Строки фиксированной длины относятся к типу данных, которые могут состоять из фиксированного числа символов, это число определяется при создании таблиц.

Например, вы можете разрешить ввод 30 символов в столбец, предназначенный для хранения имен, или 11 символов в столбец с номером карточки социального страхования. В столбцы для строк фиксированной длины нельзя вводить больше символов, чем разрешено. Столбцы также выделяют для хранения именно столько места, сколько разрешено. Так, если имя Бен сохраняется в поле столбца имен, рассчитанном на ввод 30 символов, будет сохранено ровно 30 символов (при необходимости текст дополняется пробелами или нулями).

В строках переменной длины можно хранить столько символов, сколько необходимо (максимальное значение ограничивается типом данных и СУБД). Некоторые типы данных переменной длины имеют ограничение снизу (фиксированное значение минимальной длины). Другие ограничений не имеют. В любом случае сохраняются только указанные данные (и никаких дополнительных).

Если тип данных переменной длины обладает такой гибкостью, зачем вам использовать типы данных фиксирован-

ной длины? Ответ прост: для повышения производительности. СУБД способна сортировать столбцы с данными фиксированной длины и манипулировать ими намного быстрее, чем столбцами с данными переменной длины. Кроме того, многие СУБД не способны индексировать столбцы с данными переменной длины (или переменную часть столбца). (За подробностями относительно индексов вам следует обратиться к уроку 22, “Расширенные возможности SQL”.)

**Таблица Г. 1. Строковые данные**

| <i>Тип данных</i>                                   | <i>Описание</i>  |
|---|--|
| CHAR  | Строка фиксированной длины, состоящая из 1–255 символов. Ее размер должен быть определен во время создания   |
| NCHAR   | Особая форма типа данных CHAR, разработанная с целью поддержки многобайтовых символов или символов Unicode. (Точная спецификация зависит от реализации)      |
| NVARCHAR  | Специальная форма типа данных TEXT, разработанная с целью поддержки многобайтовых символов или символов Unicode. (Точная спецификация зависит от реализации) |
| TEXT (также называется LONG, или MEMO, или VARCHAR) | Текст переменной длины   |



#### Использование кавычек

Независимо от формы используемых строковых данных значение строки должно быть всегда заключено в одинарные кавычки.



#### Когда числовые значения не являются таковыми

Вы можете подумать, что номера телефонов и ZIP-коды должны храниться в числовых полях (ведь они содержат только числовые данные), но поступать так нецелесообразно. Если вы сохраните ZIP-код 01234 в числовом поле, будет сохранено число 1234. Вы потеряете одну цифру.

Основное правило таково: если число предназначено для вычислений (сумм, средних значений и т.д.), его следует хранить в столбце, предназначенном для числовых данных. Если оно используется в качестве строкового литерала (пусть он и состоит только из цифр), его место — в столбце с данными строкового типа.

## Числовой тип данных

Числовые типы данных предназначены для хранения чисел. В большинстве СУБД поддерживаются многие числовые типы данных, каждый из которых предназначен для хранения чисел определенного диапазона.

Очевидно, чем больше поддерживаемый диапазон, тем больше нужно места для хранения. Кроме того, некоторые числовые типы данных поддерживают использование десятичных точек (и дробей), другие поддерживают только целые числа.

В табл. Г.2 представлены наиболее часто используемые различные типы данных. Не все СУБД следуют соглашениям о наименовании и описаниям, перечисленным в таблице.

Таблица Г.2. Числовые типы данных

| <i>Типы данных</i>                 | <i>Описание</i>   |
|------------------------------------|---|
| BIT                                | Одноразрядное значение, 0 или 1, используется в основном для битовых флагов   |
| DECIMAL (также называется NUMERIC) | Значения с фиксированной или плавающей запятой различной степени точности     |
| FLOAT (также называется NUMBER)    | Значения с плавающей запятой  |
| INT (также называется INTEGER)     | 4-разрядные целые значения, поддерживаются числа от -2147483648 до 2147483647 |
| REAL                               | 4-разрядные значения с плавающей запятой                                      |
| SMALLINT                           | 2-разрядные целые значения, поддерживаются числа от -32768 до 32767           |
| TINYINT                            | 1-байтовые целые значения, поддерживаются числа от 0 до 255                   |

**Кавычки не используются**

В отличие от строковых типов данных, числовые никогда не заключаются в кавычки.

**Денежные типы данных**

В большинстве СУБД поддерживается особый числовой тип данных для хранения денежных значений. Обычно он называется MONEY или CURRENCY. Эти типы данных обычно относятся к типу DECIMAL, но со специфическими диапазонами, делающими их удобными для хранения денежных значений.

## Типы данных даты и времени

Все СУБД поддерживают типы данных, разработанные для хранения значений даты и времени (табл. Г.3). Аналогично числовым значениям, большинство СУБД поддерживают многие типы данных даты и времени, каждый со своим диапазоном и степенью точности.

**Таблица Г.3. Типы данных даты и времени**

| <i>Тип данных</i>                     | <i>Описание</i>   |
|---------------------------------------|---|
| DATE                                  | Значения даты   |
| DATETIME (также называется TIMESTAMP) | Значения даты и времени   |
| SMALLDATETIME                         | Значения даты и времени с точностью до минуты (без значений секунд или миллисекунд) |
| TIME                                  | Значение времени  |



### Указание дат

Не существует стандартного способа указания даты, который подходил бы к любой СУБД. В большинстве реализаций приемлем формат типа 2004-12-30 или Dec 30th, 2004, но даже эти значения могут оказаться проблемой для некоторых СУБД. Обязательно обратитесь к документации своей СУБД и найдите список распознаваемых ею форматов.



### Даты в ODBC

Поскольку в каждой СУБД используется свой формат представления даты, ODBC создал свой собственный формат, который способен работать с любой СУБД при использовании ODBC. Формат ODBC выглядит так: {d '2004-12-30'} для значений дат, {t '21:46:29'} для значений времени и {ts '2004-12-30 21:46:29'} для значений даты и времени. Если вы выполняете SQL-код через ODBC, убедитесь в том, что значения даты и времени отформатированы подобным образом.

## Двоичные типы данных

Двоичные типы данных относятся к числу наименее совместимых (и реже всего используемых) типов данных. В отличие от всех других типов данных, рассмотренных нами до сих пор и предназначенных для весьма конкретного применения, двоичные типы данных могут содержать любые данные, даже информацию в двоичном виде, такую как графические изображения, мультимедиа и документы текстового процессора (табл. Г.4).

Таблица Г.4. Двоичные типы данных

| <i>Тип данных</i>                               | <i>Описание</i>  |
|---|--|
| BINARY  | Двоичные данные фиксированной длины (максимальная длина может быть от 255 байт до 8000 байт, в зависимости от реализации)      |
| LONG RAW  | Двоичные данные переменной длины объемом до 2 Гбайт  |
| RAW (в некоторых реализациях называются BINARY) | Двоичные данные фиксированной длины объемом до 255 байт  |
| VARBINARY                                       | Двоичные данные переменной длины (обычно максимальный объем варьируется от 255 байт до 8000 байт, в зависимости от реализации) |

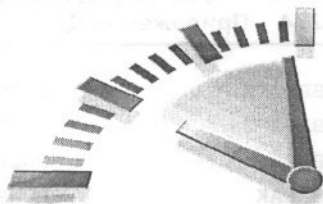


#### Сравнение типов данных

Чтобы выполнить реальный пример и провести сравнение типов данных различных СУБД, воспользуйтесь таблицей создания сценариев, используемых для построения экземпляров таблиц для этой книги (см. приложение А, "Сценарии демонстрационных таблиц"). Путем сравнения сценариев, используемых для различных СУБД, вы на личном опыте убедитесь в том, насколько сложна задача согласования типов данных.



## Приложение Д



# Зарезервированные слова SQL

В языке SQL широко используются ключевые слова — особые слова, применяемые для выполнения операций SQL. Нужно внимательно следить за тем, чтобы эти ключевые слова не были использованы в качестве имен баз данных, таблиц, столбцов и других объектов баз данных. Поэтому ключевые слова считаются зарезервированными.

В данном приложении содержится перечень зарезервированных слов, наиболее часто встречающихся в основных СУБД. Обратите внимание на следующие моменты.

- Ключевые слова — вещь весьма специфичная, поэтому не все приведенные ниже ключевые слова используются во всех СУБД.
- Во многих СУБД используется расширенный перечень зарезервированных слов SQL; в него включаются термины, специфичные для данной реализации. Многие из специфичных для отдельных СУБД ключевых слов не представлены ниже.
- Чтобы обеспечить совместимость и переносимость, следует избегать любого из зарезервированных слов, если даже некоторые из них не относятся к числу зарезервированных в вашей СУБД.

|               |           |                |
|---------------|-----------|----------------|
| ABORT         | ABSOLUTE  | ACTION         |
| ACTIVE        | ADD       | AFTER          |
| ALL           | ALLOCATE  | ALTER          |
| ANALYZE       | AND       | ANY            |
| ARE           | AS        | ASC            |
| ASCENDING     | ASSERTION | AT             |
| AUTHORIZATION | AUTO      | AUTO-INCREMENT |
| AUTOINC       | AVG       | BACKUP         |



|                   |              |               |
|-------------------|--------------|---------------|
| BEFORE            | BEGIN        | BETWEEN       |
| BIGINT            | BINARY       | BIT           |
| BLOB              | BOOLEAN      | BOTH          |
| BREAK             | BROWSE       | BULK          |
| BY                | BYTES        | CACHE         |
| CALL              | CASCADE      | CASCADED      |
| CASE              | CAST         | CATALOG       |
| CHANGE            | CHAR         | CHARACTER     |
| CHARACTER_LENGTH  | CHECK        | CHECKPOINT    |
| CLOSE             | CLUSTER      | CLUSTERED     |
| COALESCE          | COLLATE      | COLUMN        |
| COLUMNS           | COMMENT      | COMMIT        |
| COMMITTED         | COMPUTE      | COMPUTED      |
| CONDITIONAL       | CONFIRM      | CONNECT       |
| CONNECTION        | CONSTRAINT   | CONSTRAINTS   |
| CONTAINING        | CONTAINS     | CONTAINSTABLE |
| CONTINUE          | CONTROLROW   | CONVERT       |
| COPY              | COUNT        | CREATE        |
| CROSS             | CSTRING      | CUBE          |
| CURRENT           | CURRENT_DATE | CURRENT_TIME  |
| CURRENT_TIMESTAMP | CURRENT_USER | CURSOR        |
| DATABASE          | DATABASES    | DATE          |
| DATETIME          | DAY          | DBCC          |
| DEALLOCATE        | DEBUG        | DEC           |
| DECIMAL           | DECLARE      | DEFAULT       |
| DELETE            | DENY         | DESC          |
| DESCENDING        | DESCRIBE     | DISCONNECT    |
| DISK              | DISTINCT     | DISTRIBUTED   |
| DIV               | DO           | DOMAIN        |
| DOUBLE            | DROP         | DUMMY         |
| DUMP              | ELSE         | ELSEIF        |
| ENCLOSED          | END          | ERRLVL        |
| ERROREXIT         | ESCAPE       | ESCAPED       |
| EXCEPT            | EXCEPTION    | EXEC          |

|               |              |           |
|---------------|--------------|-----------|
| EXECUTE       | EXISTS       | EXIT      |
| EXPLAIN       | EXTEND       | EXTERNAL  |
| EXTRACT       | FALSE        | FETCH     |
| FIELD         | FIELDS       | FILE      |
| FILLFACTOR    | FILTER       | FLOAT     |
| FLOPPY        | FOR          | FORCE     |
| FOREIGN       | FOUND        | FREETEXT  |
| FREETEXTTABLE | FROM         | FULL      |
| FUNCTION      | GENERATOR    | GET       |
| GLOBAL        | GO           | GOTO      |
| GRANT         | GROUP        | HAVING    |
| HOLDLOCK      | HOURL        | IDENTITY  |
| IF            | IN           | INACTIVE  |
| INDEX         | INDICATOR    | INFILE    |
| INNER         | INOUT        | INPUT     |
| INSENSITIVE   | INSERT       | INT       |
| INTEGER       | INTERSECT    | INTERVAL  |
| INTO          | IS           | ISOLATION |
| JOIN          | KEY          | KILL      |
| LANGUAGE      | LAST         | LEADING   |
| LEFT          | LENGTH       | LEVEL     |
| LIKE          | LIMIT        | LINENO    |
| LINES         | LISTEN       | LOAD      |
| LOCAL         | LOCK         | LOGFILE   |
| LONG          | LOWER        | MANUAL    |
| MATCH         | MAX          | MERGE     |
| MESSAGE       | MIN          | MINUTE    |
| MIRROREXIT    | MODULE       | MONEY     |
| MONTH         | MOVE         | NAMES     |
| NATIONAL      | NATURAL      | NCHAR     |
| NEXT          | NEW          | NO        |
| NOCHECK       | NONCLUSTERED | NONE      |
| NOT           | NULL         | NULLIF    |
| NUMERIC       | OF           | OFF       |

|             |             |              |
|-------------|-------------|--------------|
| OFFSET      | OFFSETS     | ON           |
| ONCE        | ONLY        | OPEN         |
| OPTION      | OR          | ORDER        |
| OUTER       | OUTPUT      | OVER         |
| OVERFLOW    | OVERLAPS    | PAD          |
| PAGE        | PAGES       | PARAMETER    |
| PARTIAL     | PASSWORD    | PERCENT      |
| PERM        | PERMANENT   | PIPE         |
| PLAN        | POSITION    | PRECISION    |
| PREPARE     | PRIMARY     | PRINT        |
| PRIOR       | PRIVILEGES  | PROC         |
| PROCEDURE   | PROCESSEXIT | PROTECTED    |
| PUBLIC      | PURGE       | RAISERROR    |
| READ        | READTEXT    | REAL         |
| REFERENCES  | REGEXP      | RELATIVE     |
| RENAME      | REPEAT      | REPLACE      |
| REPLICATION | REQUIRE     | RESERV       |
| RESERVING   | RESET       | RESTORE      |
| RESTRICT    | RETAIN      | RETURN       |
| RETURNS     | REVOKE      | RIGHT        |
| ROLLBACK    | ROLLUP      | ROWCOUNT     |
| RULE        | SAVE        | SAVEPOINT    |
| SCHEMA      | SECOND      | SECTION      |
| SEGMENT     | SELECT      | SENSITIVE    |
| SEPARATOR   | SEQUENCE    | SESSION_USER |
| SET         | SETUSER     | SHADOW       |
| SHARED      | SHOW        | SHUTDOWN     |
| SINGULAR    | SIZE        | SMALLINT     |
| SNAPSHOT    | SOME        | SORT         |
| SPACE       | SQL         | SQLCODE      |
| SQLERROR    | STABILITY   | STARTING     |
| STARTS      | STATISTICS  | SUBSTRING    |
| SUM         | SUSPEND     | TABLE        |
| TABLES      | TAPE        | TEMP         |

|            |             |           |
|------------|-------------|-----------|
| TEMPORARY  | TEXT        | TEXTSIZE  |
| THEN       | TIME        | TIMESTAMP |
| TO         | TOP         | TRAILING  |
| TRAN       | TRANSACTION | TRANSLATE |
| TRIGGER    | TRIM        | TRUE      |
| TRUNCATE   | UNCOMMITTED | UNION     |
| UNIQUE     | UNTIL       | UPDATE    |
| UPDATETEXT | UPPER       | USAGE     |
| USE        | USER        | USING     |
| VALUE      | VALUES      | VARCHAR   |
| VARIABLE   | VARYING     | VERBOSE   |
| VIEW       | VOLUME      | WAIT      |
| WAITFOR    | WHEN        | WHERE     |
| WHILE      | WITH        | WORK      |
| WRITE      | WRITETEXT   | XOR       |
| YEAR       | ZONE        |           |

# Предметный указатель

## A

ANSI SQL, 26  
Aqua Data Studio, 243, 244

## C

ColdFusion, 245  
Command Center, 245

## D

Data Source ODBC, 255  
DB2, 245

## E

Enterprise Manager, 252

## M

Macromedia, 245  
Microsoft ASP, 248  
Microsoft ASP.NET, 249  
Microsoft Query, 243, 250  
Microsoft SQL Server, 251  
MySQL, 252

## O

ODBC, 256  
Oracle, 252

## P

PHP, 253  
PostgreSQL, 44, 253  
psql, 253

## Q

Query Designer, 246  
Query Tool, 243, 254

## S

SQL, 25  
ANSI, 26  
расширения, 26  
стандартный, 26  
SQL Advantage, 255  
Structured Query Language, 25  
Sybase Adaptive Server, 255

## A

Аргумент  
ALL, 96  
DISTINCT, 97

## Б

База данных  
безопасность, 231  
основные понятия, 20  
реляционная, 122  
типа ISAM, 212

## В

Внешнее объединение, 138  
Внешний ключ, 222  
Внутреннее объединение, 128  
Выборка  
всех столбцов, 33  
нескольких столбцов, 32  
отдельных столбцов, 30  
Вычисляемое поле, 69, 70

## Г

- Группа, 102
  - фильтрующая, 104
- Групповой символ, 34

## Д

- Данные
  - двоичные, 270
  - добавление, 153
  - итоговые, 101
  - неотсортированные, 31
  - распределение по столбцам, 22
  - сортировка, 35
  - строковые, 266
  - тип, 23
  - форма представления, 33
  - числовые, 268
- Декартово произведение, 126
- Добавление
  - выбранных данных, 158
  - данных, 153
  - нескольких строк, 160
  - полных строк, 153
  - части строки, 157

## Е

- Естественное объединение, 137

## З

- Запись, 24
- Запрос, 113
  - вложенный, 113
  - комбинированный, 145
  - подчиненный, 113
  - сложный, 145
- Зарезервированное слово, 273

## И

- Индекс, 227
  - создание, 229

- Индексно-последовательный метод доступа, 212
- Источник данных ODBC, 256
- Итоговые данные, 101

## К

- Кавычки, 47
- Каскадное удаление, 224
- Ключ
  - внешний, 222
  - первичный, 24, 122, 220
- Ключевое слово, 29, 273
  - AND, 52, 87
  - AS, 74, 134
  - ASC, 41
  - BETWEEN, 48
  - DEFAULT, 174
  - DESC, 41
  - FROM, 166
  - IN, 56, 57
  - INTO, 154
  - NOT, 57, 58
  - ON, 229
  - OR, 53
  - OUT, 199
  - PRIMARY KEY, 221
  - REFERENCES, 223
  - UNIQUE, 225
- Код переносимый, 80
- Комбинированный запрос, 145
- Копирование данных, 160
- Критерий поиска, 43
- Курсор, 211
  - закрытие, 217
  - открытие, 215
  - создание, 214

## Л

- Левое внешнее объединение, 140

## М

- Масштабирование, 123
- Метасимвол, 62
  - знак процента, 62
  - квадратные скобки, 66
  - символ подчеркивания, 65
- Модуль ODBC, 256

## Н

- Неявная фиксация, 207

## О

- Обновление данных, 163
- Объединение, 121
  - внешнее, 138
  - внешнее левое, 140
  - внешнее полное, 141
  - внешнее правое, 140
  - внутреннее, 128
  - естественное, 137
  - многих таблиц, 129
  - перекрестное, 128
  - по эквивалентности, 128
- Ограничение, 219
  - на значения столбца, 225
  - уникальности, 224
- Оператор
  - ALTER TABLE, 176, 259
  - CLOSE, 217
  - COMMIT, 207, 260
  - CREATE INDEX, 229, 260
  - CREATE PROCEDURE, 260
  - CREATE TABLE, 169, 259, 261
  - CREATE VIEW, 185, 261
  - DECLARE, 214
  - DELETE, 165, 261
  - DROP, 185, 262
  - DROP TABLE, 178
  - EXCEPT, 152
  - EXECUTE, 197
  - FETCH, 215

- GRANT, 232
- INSERT, 153, 262
- INSERT SELECT, 158, 262
- INTERSECT, 152
- LIKE, 61
- MINUS, 152
- OPEN CURSOR, 215
- RENAME, 179
- REVOKE, 232
- ROLLBACK, 207, 262
- SAVEPOINT, 209
- SELECT, 29, 263
- SELECT INTO, 160
- SQL, синтаксис, 259
- TRUNCATE TABLE, 167
- UNION, 146
- UNION ALL, 150
- UPDATE, 163, 263
  - в предложении WHERE, 51
  - завершение, 31
  - левого внешнего
    - объединения, 140
  - независимость от регистра, 32
  - правого внешнего
    - объединения, 140
  - условный, 45
- Операция
  - !=, 47
  - <>, 47
- Откат, 205
- Отмена, 205

## П

- Первичный ключ, 24, 122, 220
  - создание, 221
- Переименование таблиц, 179
- Перекрестное объединение, 128
- Подзапрос, 113
- Поле, 70
  - вычисляемое, 69, 70
  - таблицы, 22

- Полное внешнее объединение, 141
- Полностью определенное имя, 126
- Пользовательский тип данных, 226
- Порядок обработки операторов, 54
- Правое внешнее объединение, 140
- Префикс, 62, 98
- Предложение, 36  
 ALL, 103  
 FROM, 43  
 GROUP BY, 102  
 HAVING, 105  
 ORDER BY, 36  
 SET, 164  
 VALUES, 156  
 WHERE, 43  
 фильтрации, 43
- Представление, 181  
 методика создания, 185  
 удаление, 185
- Проверка  
 на диапазон значений, 48  
 на несовпадения, 46  
 на отсутствие значения, 48  
 на равенство, 44  
 одного значения, 46
- Псевдоним, 74, 133  
 столбца, 133  
 таблицы, 134

## Р

- Реляционная таблица, 121

## С

- Самообъединение, 135
- Символ  
 групповой, 34  
 звездочка, 34
- Скобки, 55

- Слово  
 зарезервированное, 273  
 ключевое, 273
- Сложный запрос, 145
- Создание  
 индекса, 229  
 курсора, 214  
 таблицы, 169  
 хранимой процедуры, 198
- Сортировка  
 в указанном направлении, 39  
 данных, 35  
 по невыбранным столбцам, 37  
 по нескольким столбцам, 37  
 по положению столбца, 38  
 результатов  
 комбинированных  
 запросов, 151
- Статистическая функция, 142
- Статистические вычисления, 96
- Столбец, 22, 70  
 полностью определенное  
 имя, 126  
 производный, 76
- Строка, 23  
 переменной длины, 266  
 фиксированной длины, 266
- СУБД, 20
- Схема, 22

## Т

- Таблица, 21  
 переименование, 179  
 реляционная, 121  
 создание, 169  
 создание копии, 162  
 схема, 21  
 удаление, 178  
 уникальное имя, 21
- Тип данных, 23, 265  
 даты и времени, 269



двоичный, 270  
денежный, 269  
переменной длины, 266  
пользовательский, 226  
совместимость, 23  
фиксированной длины, 267  
числовой, 268

**Точка**

отката, 205  
сохранения, 205, 209

**Транзакция, 203**

управляемая, 206

**Триггер, 229****У****Удаление**

данных, 165  
таблиц, 178

**Ф****Фиксация, 205**

неявная, 207  
частичная, 208

**Фильтрация**

в SQL, 44  
в приложении, 44  
посредством подзапросов,  
114

**Фильтрация данных**

расширенная, 51

**Функция, 79**

AVG(), 90

COUNT(), 92

DATAPART(), 87

LTRIM(), 74

MAX(), 93

MIN(), 94

RTRIM(), 73

SOUNDEX, 82

SUM(), 95

TRIM(), 74

UPPER(), 82

итоговая, 90

манипулирования датой и  
временем, 84

манипулирования текстом,  
82

манипулирования числами,  
87

статистическая, 90

**Х**

Хранимая процедура, 179,  
193

**Ш**

Шаблон поиска, 62

**Я**

Язык структурированных  
запросов, 25

*Научно-популярное издание*

**Бен Форта**  
**Освой самостоятельно SQL.**  
**10 минут на урок**

|                         |                          |
|-------------------------|--------------------------|
| Литературный редактор   | <i>П.Н. Мачуга</i>       |
| Верстка                 | <i>А.Н. Полинчик</i>     |
| Художественный редактор | <i>В.Г. Павлютин</i>     |
| Корректор               | <i>Л.В. Пустовойтова</i> |

Издательский дом "Вильямс".  
101509, Москва, ул. Лесная, д. 43, стр. 1.

Подписано в печать 29.04.2005. Формат 84×108/32.  
Гарнитура Times. Печать офсетная.  
Усл. печ. л. 15,1. Уч.-изд. л. 9,9.  
Тираж 3000 экз. Заказ № 1645.

Отпечатано с диапозитивов в ФГУП "Печатный двор"  
Министерства РФ по делам печати,  
телерадиовещания и средств массовых коммуникаций.  
197110, Санкт-Петербург, Чкаловский пр., 15.

# MySQL

## Учебное пособие

Люк Веллинг и Лора Томсон



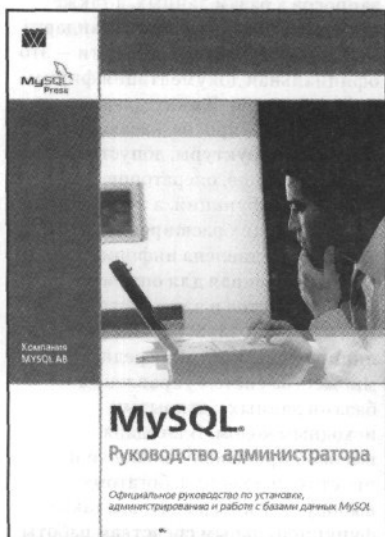
ISBN: 5-8459-0769-1  
В продаже

**К**нига представляет собой краткое, но ясное изложение как основных теоретических принципов, так и практических приемов работы с MySQL. Она научит начинающего пользователя MySQL создавать сложные базы данных, которые можно использовать дома, на работе или в Web. Независимо от того, кем вы являетесь — новичком в деле освоения баз данных или профессионалом, стремящимся понять особенности работы MySQL, — это учебное пособие предоставит вам всю необходимую информацию для начала работы с MySQL и быстрого освоения этой системы.

<http://www.williamspublishing.com>

# MYSQL. РУКОВОДСТВО АДМИНИСТРАТОРА

**Компания MySQL AB**



[www.williamspublishing.com](http://www.williamspublishing.com)

**ISBN 5-8459-0805-1**

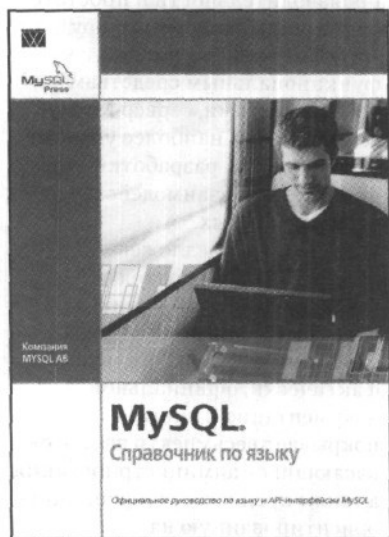
MySQL занимает лидирующие позиции среди множества систем управления базами данных с открытым исходным кодом. Благодаря высокой производительности и простоте настройки, богатому выбору API-интерфейсов, а также функциональным средствам работы с сетями, сервер MySQL стал одним из наиболее удачных вариантов для разработки Web-приложений, взаимодействующих с базами данных.

В этой книге предложен всеобъемлющий подход к установке, обслуживанию и администрированию сервера баз данных MySQL. Являясь, фактически, официальной документацией, книга покрывает весь спектр вопросов, касающихся администрирования, а также предлагает информацию, ориентированную на опытных пользователей и администраторов. Книга рассчитана на администраторов и разработчиков Web-приложений любой квалификации, а также на студентов и преподавателей соответствующих дисциплин.

**в продаже**

# MYSQL. СПРАВОЧНИК ПО ЯЗЫКУ

**Компания MySQL AB**



[www.williamspublishing.com](http://www.williamspublishing.com)

Эта книга, написанная специалистами компании MySQL AB, является всеобъемлющим справочником по языку SQL, который используется для организации запросов к базам данных, а также особенностях реализации стандарта SQL в сервере MySQL. По сути — это официальная документация фирмы-производителя. В книге рассмотрен весь спектр вопросов, касающихся языковой структуры, допустимых типов столбцов, операторов, операций и функций, а также существующих расширений MySQL; также представлена информация, предназначенная для опытных программистов и администраторов. Как известно, MySQL занимает лидирующие позиции среди множества систем управления базами данных с открытым исходным кодом. Благодаря высокой производительности и простоте настройки, богатому выбору API-интерфейсов, а также функциональным средствам работы с сетями, сервер MySQL стал одним из наиболее удачных вариантов для разработки Web-приложений, взаимодействующих с базами данных. Книга рассчитана на разработчиков Web-приложений и администраторов любой квалификации, а также на студентов и преподавателей соответствующих дисциплин.

**ISBN 5-8459-0804-3**

**в продаже**