Реляционные базы данных. Учебное пособие для бакалавров

Версия: 1.1

Фомин М.М.

Москва 2024

Оглавление

1. Базы данных 8

1.1. Появление БД 8

1.1.1. Исторические предпосылки 8
1.1.2. Понятие БД 9
1.1.3. Концепции БД 10
1.1.4. Первые базы данных 11
1.1.5. Реляционная модель БД 13

2. СУБД 16

2.1. Архитектура СУБД 16

2.1.1. Трехуровневая архитектура систем баз данных ANSI/SPARC 16
2.1.2. Файл-серверная архитектура 18
2.1.3. Встраиваемые СУБД 18
2.1.4. Клиент серверная архитектура 19
2.1.5. Трехзвенная архитектура 21

2.2. Функции СУБД 24

2.2.1. Управление данными во внешней памяти 24
2.2.2. Управление буферами оперативной памяти 26
2.2.3. Поддержка языков СУБД 28
2.2.4. Обеспечение согласованности БД 29
2.2.5. Управление транзакциями 30
2.2.6. Управление блокировками и клинчами 34
2.2.7. Последовательное выполнение транзакций 37
2.2.8. Уровни изоляции транзакций 38
2.2.9. Управление журналами изменений БД 39
2.2.10. Обеспечение безопасности БД 40
2.2.11. Копирование данных и резервные копии БД 43
2.2.12. Репликация данных 45
2.2.13. Ведение словаря БД 48

2.3. Развитие реляционных СУБД 49

3. Язык SQL 53

3.1. Становление языка 53

3.1.1. История возникновения 53
3.1.2. Стандарт языка SQL 54

3.2. Концепции языка SQL 55

3.2.1. Декларативность и простота 55
3.2.2. Непроцедурность и расширения языка 56
3.2.3. Замкнутость и стиль мышления 56
3.2.4. Зависимость от СУБД и независимость от платформы 56
3.2.5. Равенство формализмов и отход от реляционной теории 57
3.2.6. Срезы и консолидация 58
3.2.7. Защита данных и привилегии пользователей 58
3.2.8. Диалекты языка SQL 58

3.3. Типы данных SQL 59

3.4. Операторы SQL 60

3.4.1. Простые примеры 60
3.4.2. Некоторые общие замечания по синтаксису SQL 62
3.4.3. Оператор SELECT 62
3.4.4. Подзапросы 68

3.4.5. Выборка из нескольких таблиц ... 71
3.4.6. Конструкция JOIN. Внутренние соединения ... 74
3.4.7. Объединение результатов запросов. Внешние соединения ... 76
3.4.8. Конструкция JOIN. Внешние соединения ... 77
3.4.9. Понимание SQL. Как выполняется оператор SELECT ... 82
3.4.10. Все операторы языка SQL ... 87
3.4.11. Предложение WHERE ... 88
3.4.12. NULL и работа с ним ... 90
3.4.13. Создание объектов СУБД ... 91
3.4.14. Изменение и удаление объектов СУБД ... 91
3.4.15. Представления в языке SQL ... 92
3.4.16. Добавление данных. Оператор INSERT ... 94
3.4.17. Изменение данных. Оператор UPDATE ... 95
3.4.18. Удаление данных. Оператор DELETE ... 96
3.4.19. Защита данных в SQL ... 97
3.5. Функции в SQL ... 98
3.5.1. Классификация функций ... 98
3.5.2. Строковые функции ... 99
3.5.3. Математические функции ... 99
3.5.4. Агрегатные функции ... 100
3.5.5. Функции даты и времени ... 102
3.5.6. Системные функции. ... 103
3.5.7. Детерминированные и недетерминированные функции ... 103
4. ПРОЕКТИРОВАНИЕ БД ... 104
4.1. Исторический экскурс ... 104
4.2. Простой пример. Фразеологический словарь ... 105
4.3. Основные цели проектирования БД ... 108
4.4. Основные задачи и результат проектирования БД ... 109
4.4.1. Основные задачи ... 109
4.4.2. Результат проектирования БД ... 110
4.4.3. Выбор СУБД ... 110
4.5. Логическое проектирование БД ... 110
4.5.1. Анализ бизнес процессов ... 111
4.5.2. Определение сущностей ... 112
4.5.3. Верификация сущностей ... 115
4.5.4. От автоматизации процессов к автоматизации компании ... 116
4.5.5. Определение атрибутов сущностей ... 116
4.5.6. Первичный ключ – важный атрибут ... 116
4.5.7. Вычисляемые атрибуты (поля) ... 117
4.5.8. Верификация атрибутов ... 117
4.5.9. Виды связей между сущностями ... 118
4.5.10. Первичные и внешние ключи ... 119
4.5.11. «Один к одному» - редкий тип связи ... 120
4.5.12. «Много ко многим» - несуществующий тип связи ... 121
4.5.13. Экзотические связи ... 122
4.5.14. Верификация связей ... 122
4.5.15. Диаграмма «Сущность-связь» ... 123
4.6. Физическое проектирование базы данных ... 124
4.6.1. Формирование схемы таблиц ... 124
4.6.2. Создание представлений ... 125
4.6.3. Создание хранимых процедур ... 127
4.6.4. Триггеры ... 127

4.7. Разграничение доступа и создание ролей пользователей 131
4.7.1. Что такое разграничение доступа 131
4.7.2. Пользователи, их схемы и роли 131
4.7.3. Предопределённые роли пользователей СУБД 132
4.7.4. Создание ролей пользователей 133
4.7.5. Операторы SQL для определения доступа к данным 134
4.8. Обеспечение согласованности БД 135
4.8.1. Виды согласованности БД 135
4.8.2. Механизмы обеспечения согласованности БД 136
4.8.3. Удаление данных 137
4.9. Поддержка историчности 137
4.10. Обеспечение быстродействия БД 139
4.10.1. Создание индексов 139
4.10.2. Кластеризация таблиц 141
4.10.3. Секционирование таблиц 142
4.11. Нормализация базы данных 143
5. АДМИНИСТРИРОВАНИЕ БД 145
ПРИЛОЖЕНИЕ А. СОКРАЩЕНИЯ, ПРИНЯТЫЕ В ЭТОЙ КНИГЕ 152
ПРИЛОЖЕНИЕ Б. ОСНОВНЫЕ ТЕРМИНЫ И ПОНЯТИЯ 153
ПРИЛОЖЕНИЕ В. БАЗА ДАННЫХ ПРИМЕРА 158
ПРИЛОЖЕНИЕ Г. 12 ПРАВИЛ КОДДА 162
ПРИЛОЖЕНИЕ Д. ЗАДАЧИ И ДОЛЖНОСТНЫЕ ОБЯЗАННОСТИ АДМИНИСТРАТОРА БАЗЫ ДАННЫХ 164
ПРИЛОЖЕНИЕ Е. ПЕРЕЧЕНЬ ПРОФЕССИОНАЛЬНЫХ СТАНДАРТОВ 166
ПРИЛОЖЕНИЕ Ж. СТАНДАРТНЫЕ ФОРМЫ БЭКУСА — НАУРА (BNF) 167
ПРИЛОЖЕНИЕ З. СЕМЕЙСТВО СТАНДАРТОВ IDEF 168

Предисловие

Умение грамотно работать с современными базами данных является одним из ключевых требований к любому специалисту в области компьютерных технологий. Важную роль в освоении технологий баз данных играет понимание их основных сущностей и концепций. Предлагаемое учебное пособие нацелено именно на ознакомление студентов с фундаментальными понятиями баз данных. В нем дается общий обзор различных типов баз данных и соответствующих методов их использования. Особое внимание уделяется наиболее распространенным реляционным базам данных. Описываются основные конструкции языка SQL, являющегося фактическим стандартом работы с реляционными базами данных. Приведены методы проектирования баз данных. Отдельная глава посвящена описанию транзакций, являющихся основным средством обеспечения надежности и согласованности баз данных. Рассматриваются современные архитектуры СУБД.

Разумеется, в небольшом пособии невозможно охватить все грани работы с базами данных. Пособие отражает скорее опыт автора в работе с большими базами данных в крупных банках и корпорациях. Приведенный список литературы поможет значительно расширить кругозор знаний о базах данных.

Термины и сокращения

Термин	Определение
Программа	Последовательность инструкций, предназначенных для исполнения вычислительной машиной. Программа — один из компонентов программного обеспечения.
Программное обеспечение (ПО)	Компьютерные программы, процедуры и, возможно, соответствующая документация и данные, относящиеся к функционированию вычислительной системы
Программный продукт (ПП)	Программа как результат человеческой деятельности, выставленный на рынке в качестве товара и имеющий ненулевую потребительную стоимость.
Корпоративный стандарт	Внутренний стандарт компании, утвержденный и введенный в действие приказом ее руководителя, обязательный для соблюдения всеми ее сотрудниками.
Предметная область базы данных	Это та часть реального мира, о которой база данных хранит, собирает и анализирует информацию.
База данных (БД)	Организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.
Система управления базами данных (СУБД)	Совокупность программных, лингвистических и методических средств общего или специального назначения, обеспечивающих управление созданием и использованием БД.
SQL	Формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных.
Операционная система (ОС)	Операционная система — комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия прикладных программ с пользователем.
Информационная система (ИС)	Система, предназначенная для хранения, поиска и обработки информации вместе с соответствующими организационными ресурсами (человеческими, техническими, финансовыми и т. д.), которые обеспечивают сбор, обработку и распространение информации.
Сущность	Предмет, человек или событие, о котором собирается и хранится информация в БД.
Таблица	Объект базы данных, хранящий данные об одной сущности: поля таблицы (столбцы) — это атрибуты сущности, а записи (строки таблицы) — это данные о конкретном экземпляре сущности.
Запись (кортеж, строка таблицы)	Данные об одном конкретном экземпляре сущности - предмете, человеке или событии. Запись состоит из полей.
Поле записи (атрибут, столбец таблицы)	Отдельная характеристика (свойство) объекта.
Система хранения данных (СХД)	Комплексное программно-аппаратное решение по организации надёжного хранения информационных ресурсов и предоставления гарантированного доступа к ним.

Центр обработки данных (ЦОД)	Сложная система, которая включает в себя целый комплекс IT решений, высокотехнологичного оборудования и инженерных конструкций.

Для кого предназначена эта книга

Эта книга предназначена всем тем, кто хочет понять, что же такое база данных, а в первую очередь студентов высших учебных заведений, обучающихся по направлениям подготовки бакалавров "Информатика и вычислительная техника" и "Прикладная информатика". Может быть полезна для студентов, обучающихся по другим специальностям, при изучении дисциплин «Базы данных» и «Управление данными».

Это издание подразумевает возможность самостоятельного изучения предмета, но в то же время оно может сопровождаться (хотя и необязательно) дополнительным лекционным материалом. Я очень надеюсь, что представленная информация принесет студентам не просто набор фактов, но также и понимание того, как работает, проектируется и эксплуатируется современная БД.

Краткое содержание

Эта книга состоит из пяти глав и последовательно знакомит читателя с основными понятиями баз данных: сначала обсуждаются концепции БД, затем происходит знакомство с функциями СУБД, поддерживающими БД. Далее предлагается знакомство с языком SQL являющимся универсальным интерфейсом СУБД, и наконец, дается описание процессов проектирования и сопровождения БД.

Первая глава посвящена становлению концепций БД и в рамках исторического экскурса знакомит читателя с основными моделями данных.

Во второй главе даются основные понятия, описывается архитектура и главные функции СУБД. Приводится краткий экскурс в развитие и становление реляционных СУБД.

Третья глава содержит описание языка SQL, демонстрируется неразрывная связь языка SQL с СУБД и реляционной теорией. Обсуждаются основные принципы, стандарты, диалекты и операторы языка SQL.

Четвертая глава дает понятие о процессе проектирования БД. Предлагаются алгоритмы выделения и верификации сущностей и связей, обсуждается создание различных объектов БД.

Пятая глава описывает основные процессы администрирования и сопровождения БД.

Соглашения принятые в книге

Примеры (синтаксис и нотации)

В данном документе присутствуют примеры скриптов (программ) на языке SQL и различные схемы. Тексты скриптов приведены в соответствии со стандартом ANSI SQL2003 (SQL3) [1], схемы бизнес-процессов в нотации IDEF0 [19] (отступления оговариваются отдельно), а схемы сущностей и таблиц БД в виде, принятом в системе ORACLE SQL Developer Data Modeler.

Вопросы терминологии

В связи с развитием теории и практики проектирования БД, а также развитием стандартов СУБД и языка SQL родилось множество наслоений терминов, которые невольно порождают путаницу при изучении проектирования и работе с БД. Основные термины сведены в таблицу 1.

В настоящей книге используются термины, принятые в практической литературе и документации на системы управления базами данных (СУБД). К сожалению, в настоящее время все чаще применяется терминология ГОСТ, появившаяся в результате грубого перевода англоязычных стандартов.

Таблица1. Термины, применяемые в литературе по базам данных.

Реляционная алгебра (Теория)	**Базы данных (СУБД)** (Практика)	**ISO и ГОСТ** (Стандарты)

Отношение	Таблица	Таблица
Кортеж	Запись	Строка
Атрибут	Поле	Столбец

Часть определений терминов и сокращений приведены в разделе «Термины и сокращения» и в приложении Б.

1. Базы данных

1.1. Появление БД

История возникновения и развития подхода к организации хранения и обработки баз данных (БД) насчитывает более 50 лет и неразрывно связана с историей использования компьютеров и информационных технологий.

1.1.1. Исторические предпосылки

На первом этапе (50-60е годы) на компьютерах, их тогда называли электронными вычислительными машинами (ЭВМ), выполнялись лишь вычислительные задачи. Типичными примерами решаемых в то время задач является решение систем обыкновенных и дифференциальных уравнений различными численными методами. Еще один пример - управление системами ведения боя, сопровождение целей и наведение на них управляемых или неуправляемых огневых средств. Входные и выходные данные для таких задач представляют собой наборы чисел небольшого объема (коэффициенты, координаты, курсы, скорости).

По мере увеличения быстродействия, а главное памяти ЭВМ, появилась возможность решать не только вычислительные, но и информационные задачи. Это задачи управления предприятием, ведение материального и бухгалтерского учета, делопроизводство. Для решения таких задач необходимо большое количество входных данных, при этом значительная часть информации совместно используется несколькими программами. Так данные о сотрудниках предприятия используются и в кадровом, и в бухгалтерском учете, и в делопроизводстве. На этом этапе появилась необходимость в надежном хранении данных независимых от программ с одновременным доступом к ним из нескольких приложений. Эта необходимость и послужила толчком к разработке концепций баз данных, которым и будет посвящен раздел 1.1.3. С историей развития БД тесно связана история развития СУБД, которая обсуждается в разделе 2.3. В настоящее время подавляющее большинство БД используются в автоматизированных системах управления и позволяют человеку осуществлять свои рабочие функции. Рассмотрим классический цикл обработки данных в АСУ, представленный на рис. 1.0.:

- Поступающие данные из разных источников проверяются, дополняются, приводятся к структуре модели предметной области и сохраняются в БД;
- Данные агрегируются т.е. объединяются в новые сущности характеризующие процессы, происходящие в системе и в предметной области. После этих преобразований данные выгружаются в аналитическую систему (специальным образом организованную БД);
- Аналитическая обработка данных – поиск сведений о закономерностях, тенденциях или взаимозависимостях между какими-либо данными, позволяющие принимать определенные решения.
- После рассмотрения аналитических данных, лица принимающие решения изменяют бизнес-процессы и/или параметры предметной области для лучшего выполнения системой своих задач.
- В результате изменения процессов в системе, собираются новые данные, с накоплением которых начинается новый цикл обработки данных.

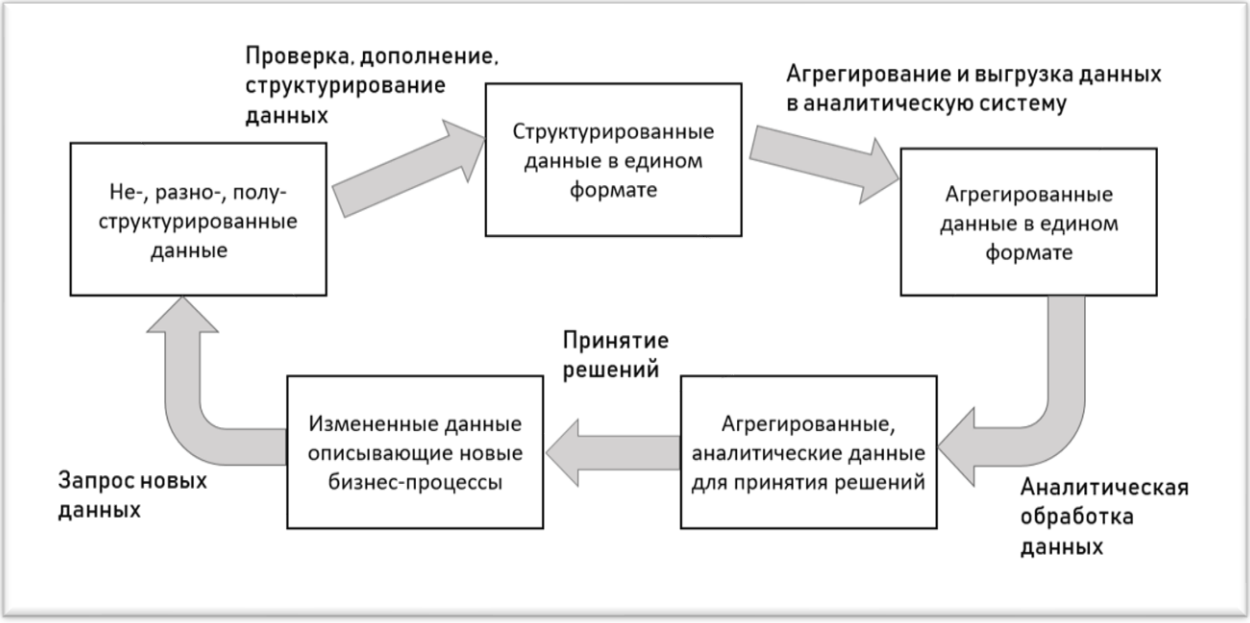

Рис. 1.0. Цикл обработки данных в АСУ

Цикл преобразования информации действует непрерывно, обеспечивая функционирование системы, причем под системой понимается любая АСУ: от банковской системы до системы управления ракетами. Надо заметить, что цикл преобразования информации имеет разные длительности, так в банковской системе различная информация циркулирует с темпом от одного до нескольких месяцев, тогда как в системе управления противовоздушной обороной цикличность составляет от 10 секунд до нескольких минут. Важно, что на всех этапах необходимо присутствие базы данных, как хранилища и инструмента преобразования данных.

1.1.2. Понятие БД

База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей [13].

Обсудим это определение.

Предметная область — это та часть реального мира, о которой база данных хранит, собирает и обрабатывает информацию. Конечно, в реальной БД хранятся не все данные о предметной области, а только те, которые позволяют нам решать определенные задачи, связанные с управлением системами предметной области. Например, для функционирования банка не особенно важен цвет стен в переговорных или тип прически главного бухгалтера, на управление балансом и прибылью эти данные не влияют. Значит в БД хранятся данные о некоторой абстрактной модели предметной области, достаточной для «удовлетворения информационных потребностей пользователей», а сама БД, в сущности, и является информационной моделью предметной области

Из определения видно, что данные имеют жесткую структуру и удовлетворяют некоторым правилам и ограничениям. Эта структура, правила и ограничения диктуются опять-таки предметной областью. Например, БД библиотеки должна по структуре напоминать (моделировать) предметный указатель, а данные о коде ISBN должны удовлетворять определенным правилам, БД ВУЗа должна отражать иерархию структурных подразделений, а номер учебной группы должен формироваться в соответствии с правилами, принятыми в ВУЗе.

Необходимо несколько слов сказать о пользователях: под пользователями мы понимаем любых потребителей информации – это могут быть люди или программы, а обращение к БД должно быть стандартизировано использованием единого языка запросов. Пользователи периодически выполняют определенные действия, позволяющие им выполнять свои функции

(для сотрудников предприятия это должностные обязанности). Набор и порядок этих действий называются «бизнес-процессами»

По своей сути любая БД представляет собой скрупулезно построенное хранилище структурированных данных. Само по себе хранилище ни для кого особого интереса не представляет до тех пор, пока оно не интегрируется в состав более сложной надсистемы, называемой информационной, и именно она создается для «удовлетворения информационных потребностей пользователей».

Все элементы информационной системы объединены общей целью – обеспечить поддержку эффективного выполнения бизнес-процессов в соответствующей предметной области.

1.1.3. Концепции БД

В середине 60-х годов прошлого столетия начали вырабатываться основные принципы, по которым должны быть построены базы данных. Правильней назвать этот процесс выработкой основных концепций баз данных. Под термином ***концепция*** в этом разделе понимается определённый способ понимания, трактовки каких-либо явлений; основная точка зрения, руководящая идея. Итак, основные концепции БД:

- Отчуждение данных от программ.

Суть этой концепции в том, что данные должны быть отделены от программ, структура данных не должна зависеть от программы и наоборот - структура данных не должна влиять на организацию программ. Т.е. БД с данными о студентах ВУЗа используется в различных программах – начисление стипендии, учет успеваемости, учет посещаемости, учет мест в общежитии и т.д. Структура, заполнение и корректировка информации в этой базе не зависит ни от одной из этих программ, более того, дополнение номенклатуры информации не сказывается на работоспособности этих программ.

- Хранение описания данных вместе с самими данными.

Необходимо, чтобы любой пользователь (человек или программа) мог узнать какая информация храниться в БД, какова структура этих данных, какие типы имеют конкретные данные. Такая служебная информация (метаданные) должны храниться в самой БД — это сильно упрощает реализацию концепции отчуждения данных от программ.

Описание данных имеет еще и глубокий смысл, связанный с интерпретацией данных, связанный с «преобразованием» данных в информацию. Современные стандарты дают такое определение информации:

Информация — это наборы данных относительно фактов, событий, вещей, идей и понятий, которые в определённом контексте имеют конкретный смысл и приносят пользу.

Без описания, без контекста данные превращаются просто в цифровой шум.

- Отчуждение данных от носителей.

Пользователь не должен знать где хранится информация БД: на дисках, на лентах, на серверах, находящихся на другом континенте. Пользователь должен получать информацию, не зная где она находится.

- Поддержание БД в согласованном (целостностном) состоянии.

Согласованность БД (database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным ограничениям предметной области. Одна из важнейших задач БД - поддержание в любой момент времени взаимной непротиворечивости, правильности и точности хранящихся данных (подробнее см. раздел 4.8).

- Защита информации.

Защита информации — это очень широкое понятие, охватывающее такие процессы, как защита от сбоев аппаратной или программной природы, защита от злонамеренного или незлонамеренного искажения и порчи информации, защита от ошибок программ или пользователей БД.

- Поддержка многозадачной (многопользовательской) работы.

Как следует из первой концепции (отделения данных от программ) многочисленные пользователи (люди или программы) будут работать с данными, находящимися в БД. Если пользователи будут постоянно ждать доступа, вступая в конфликты друг с другом, то все преимущества БД сойдут на нет. Значит БД должна поддерживать многозадачный (многопользовательский) режим работы, обеспечивая постоянный эффективный доступ к содержащейся в ней информации.

1.1.4. Первые базы данных и первые модели представления данных

Иерархическая модель

Концепция иерархических баз данных преследовала нас с 1960-х годов и остается популярной и по сегодняшний день. Иерархическую модель можно назвать самой интуитивно понятной. Как следует из ее названия, информация в такой базе данных хранится в структуре, напоминающей родословное дерево или организационную диаграмму предприятия. В принципе, файловая система компьютера, представленная в графическом интерфейсе, выглядит как иерархическая структура.

Наиболее популярной иерархической базой данных была IMS (Information Management System) от IBM, предназначенная для промышленных компьютеров. Впервые представленная в 1968 году, она (после нескольких реинкарнаций) применяется до сих пор в основном благодаря впечатляющей производительности некоторых типов запросов.

Иерархическая база данных основывается на парадигме “родительский/дочерний объект”: каждый “родитель” может иметь множество “детей”, но каждый “ребенок” может иметь одного и только одного “родителя”. Показать такую структуру в графическом интерфейсе можно, последовательно раскрывая ее узлы на разных уровнях (рис. 1.1).

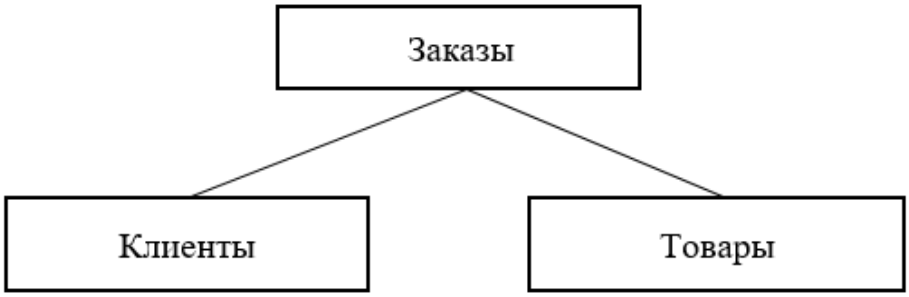

Рис. 1.1. Пример иерархической базы данных

Каждый квадратик на рис. 1.1. представляет из себя таблицу с записями о некоторой сущности предметной области «Клиентах», «Заказах», «Товарах» и т.д.

Так как записи дочерней таблицы доступны путем раскрытия иерархии уровней, все ее записи должны иметь непосредственные связи с соответствующими элементами родительской таблицы, например, «Клиенты некоторого Заказа» или «Товары, поставляемые в соответствии с Заказом». Такую организацию связей можно сравнить с системой управления файлами (например, с деревом папок, которое можно увидеть в Проводнике Windows), чтобы получить доступ к файлу в каталоге, нужно открыть родительскую папку.

Данная конструкция будет работать великолепно, до тех пор, пока не потребуется информация, извлекаемая не интуитивным путем. (Независимо от того, какие данные нужны, поиск приходится начинать с корневой таблицы.) Если требуется извлечь только имена заказчиков, иерархическая база отработает исключительно быстро, напрямую переходя от родительской таблицы к дочерней. В то же время для получения любой информации из такой базы требуется четкое знание ее структуры, которую вряд ли можно назвать гибкой. К примеру, если вы решите оформлять заказы через посредников, придется полностью перестраивать все

связи (см. рис. 1.2.), так как теперь таблица «Заказы» более не будет связана напрямую с таблицей «Клиенты».

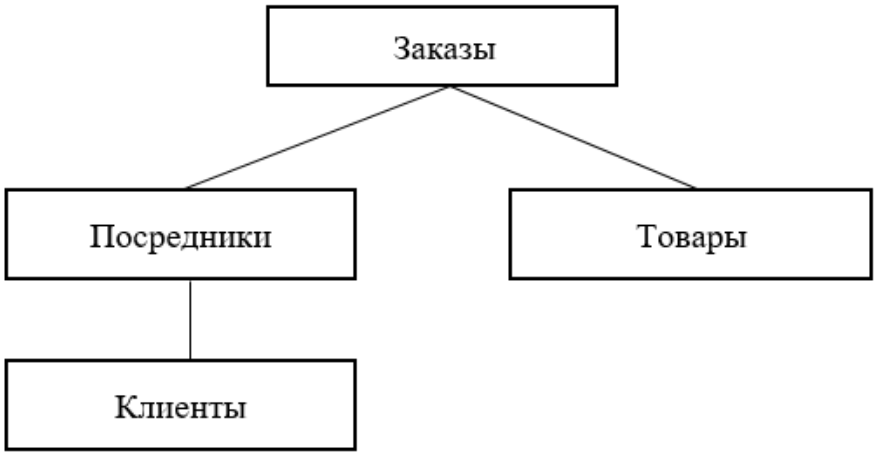

Рис. 1.2. Модифицированная иерархическая база данных

Во все запросы теперь придется внедрить дополнительный этап: поиск посредника, оформившего заказ, и определение конечного потребителя. Также совершенно очевидно, что проблемы избыточности избежать не удастся: если найдется такой конечный потребитель, который оформляет заказы более чем через одного агента, информацию о посредниках придется хранить в нескольких таблицах. Спрашивается, как поступать, если некоторые клиенты не оформляли заказы, или существуют товары, которые еще никто не покупал? С такой ситуацией иерархическая база данных попросту не справится, поскольку добраться до информации в дочерней таблице при отсутствии указателя на нее в родительской невозможно. Из самого определения иерархии следует, что не может существовать как товаров, так и клиентов без заказов, хотя на практике такие ситуации вполне естественны.

Иерархические базы данных неплохо поддерживают отношения “один ко многим” (об этом речь пойдет в разделе 1.15.). Однако во многих ситуациях требуется, чтобы один дочерний объект был связан со множеством родительских. К примеру, один товар может присутствовать во множестве заказов, а каждый заказ содержать несколько товаров. Для подобных вопросов иерархические базы данных не позволяют получить решение (по крайней мере простое).

Сетевая модель

Попытка решить проблемы, связанные с иерархическими базами данных, породила модель сетевых баз данных. Своим происхождением она обязана Комитету по языкам систем данных (CODASYL) — организации, созданной в 1957 году Министерством обороны США. Благодаря CODASYL появился на свет один из первых популярных языков программирования — COBOL, а в 1971 году был опубликован стандарт сетевых баз данных. Наиболее популярной коммерческой реализацией сетевой модели была СУБД Adabas (впоследствии преобразованная на поддержку реляционной модели).

Сетевая модель во многом сходна с иерархической. Она также основана на концепции отношений родительского и дочернего объекта, однако не ограничивает пользователя наличием всего одного родителя. В сетевой модели базы данных один родительский элемент может иметь множество дочерних, а дочерний — несколько родительских (рис. 1.3).

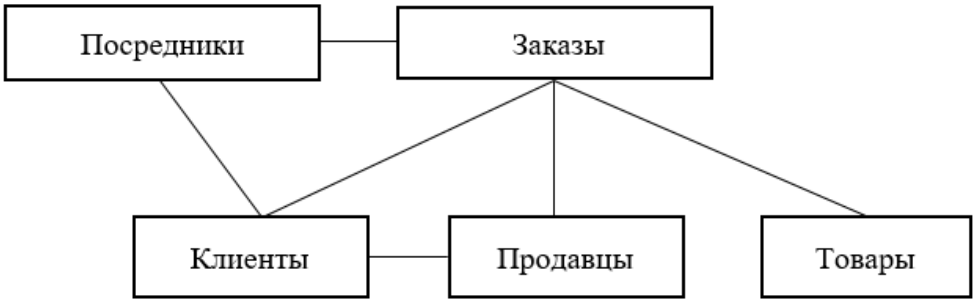

Рис. 1.3. Пример сетевой базы данных

Саму структуру в данном случае можно представить, как несколько деревьев с переплетенными ветвями. В терминологии сетевых баз данных такие переплетения называют множествами. Процесс доступа к данным в сетевых базах данных не обязательно должен начинаться с корневого элемента. Можно обратиться к любому элементу базы и начать свой путь в любом направлении, опираясь на связи между таблицами (предполагается, что таковые существуют).

В примере, показанном на рис. 1.3., для поиска товаров, проданных определенному заказчику, можно начать путь с таблицы «Заказ» и затем продолжить его к таблицам «Клиент» и «Товары» — в этом нет ничего нового. Однако в случае, когда один клиент оформляет заказы у разных посредников, больше не придется для получения списка клиентов проходить лишний этап составления списка агентов.

Несмотря на очевидные преимущества, в сетевых базах данных были устранены не все проблемы, связанные с иерархическими базами. Обе эти модели недостаточно гибкие, а изменения в структуре (например, добавление новой сущности) связаны с перестройкой всей базы. Также, все множество связей, структура записей и таблиц должны быть предварительно определены.

Главным недостатком иерархической и сетевой модели является необходимость в жесткой программной поддержке. Чтобы получить ответ на простейший запрос, приходилось создавать программу, которая проходит по структуре и формирует вывод. Такая программа должна быть написана на процедурном языке, часто привязанном к конкретной реализации базы, что требует основательных знаний как программирования, так и операционной системы. В результате такие программы были непереносимыми и требовали больших временных затрат на написание и поддержку.

1.1.5. Реляционная модель БД

Впервые термин "реляционная модель данных" появился в статье сотрудника фирмы IBM д-ра Эдгара Кодда опубликованной 6 июня 1970г. [16]. Будучи математиком по образованию, Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных может сводится к совокупности двумерных таблиц, которые он назвал отношениями - relation (отсюда и название модели – ***реляционная***). Реляционной является БД, в которой все данные доступные пользователю, организованы в виде набора связанных двумерных таблиц, а все операции над данными сводятся к операциям реляционной алгебры.

В сравнении с иерархической и сетевой моделью данных, реляционная модель отличается более высоким уровнем абстракции данных, и в настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные коммерческие СУБД. Как следует из названия основным понятием в реляционной СУБД является «таблица»

Таблица в реляционной модели данных соответствует (содержит данные) одной сущности предметной области и состоит из фиксированного числа полей, собранных в записи, каждая ***запись*** соответствует экземпляру сущности. Например, таблица «Студенты» содержит данные о

физических лицах, обучающихся в высшем учебном заведении. Одна запись в таблице «Студенты» содержит информацию о конкретном физ. лице - конкретном экземпляре сущности, например, Иванове И.И. Запись состоит из полей и каждое ***поле*** в этой записи содержит данные об Иванове И.И. - дата рождения, номер студенческого билета и т.д. Поля идентифицируются своими именами («ФИО», «Дата рождения», «Номер студенческого билета»), так же и таблицы («Студенты», «Преподаватели»), а вот записи идентифицируются содержанием некоторого поля, называемого ***первичным ключом***, в данном случае это может быть поле «Номер студенческого билета». Первичный ключ обладает свойством уникальности, ну, действительно, номера студенческих билетов не повторяются, в отличии от значения поля «ФИО» или поля «Дата рождения». Т.е. нужного студента можно найти в таблице «Студенты» только по номеру его студенческого билета. Все остальные поля неуникальны и выбрать по ним одну нужную запись невозможно. Таблица «Студенты» представлена на рис. 1.4.

Студенты

Номер студенческого билета	*ФИО*	*Дата рождения*	*Пол*	*Номер группы*
135236	Иванов И.И.	31.12.2000	М	ИУ6-21
772899	Петрова П.П.	12.12.2000	Ж	ИУ6-21
566334	Кузнецов К.К.	11.11.2001	М	ИУ6-22
122344	Иванов И.И.	12.12.2000	М	ИУ6-23

Рис. 1.4. Пример таблицы «Студенты» в реляционной модели данных

Еще одна таблица в представлении предметной области «ВУЗ» - таблица «Преподаватели» представлена на рис. 1.5. У преподавателей нет студенческого билета и роль первичного ключа выполняет поле «Номер паспорта».

Преподаватели

Номер паспорта	*ФИО*	*Дата рождения*	*Пол*
4004 677210	Иванова Г.С.	31.12.1955	Ж
4404 614255	Смирнова Е.В.	12.12.1954	Ж
4404 554399	Фомин М.М.	23.11.1954	М
0444 834567	Иванов И.И.	12.12.1950	М

Рис. 1.5. Пример таблицы «Преподаватели» в реляционной модели данных

Предыдущие две таблицы никак не связаны между собой, а для иллюстрации связей в реляционной модели приведем еще одну таблицу - «Группы» (см. рис. 1.6.) представляющие сущность «Учебные группы».

Группы

Номер группы	*Номер приказа о создании*	*Староста*	*Куратор*
ИУ6-21	143927	772899	4404 614255
ИУ6-22	112095	566334	4404 554399

Рис. 1.6. Пример таблицы «Группы» в реляционной модели данных

В таблице «Группы» первичный ключ – поле «Номер группы». А теперь вспомним про связи между сущностями в предметной области «ВУЗ»:

- каждый студент состоит в группе;
- в каждой группе есть староста из числа студентов;
- у каждой группы есть куратор из числа преподавателей.

Если внимательно посмотреть на рисунки, то видно, что поля «Куратор» и «Староста» в таблице «Группы» заполнены какими-то цифрами – это не что иное, как значения первичных ключей других таблиц – «Номер студенческого билета» из таблицы «Студенты» и «Номер паспорта» из таблицы «Преподаватели», таким образом осуществляются связи между таблицами (сущностями). Если необходимо найти имя старосты определенной группы, надо просто найти нужную запись в таблице «Студенты», взяв из записи таблицы «Группы» номер студенческого билета старосты. Куратора группы можно найти по номеру паспорта, взятого из записи нужной нам группы. В каждой записи студента есть номер группы, в котором содержится значение первичного ключа таблицы «Группы». По этому ключу можно легко найти информацию по группе, в которой состоит студент. А если мы знаем номер группы, то можно легко найти всех студентов этой группы, по номеру группы в записях таблицы «Студенты».

Поля, содержащие значения первичных ключей других (внешних) таблиц называются ***внешними ключами***. При помощи этих полей и осуществляются связи между таблицами. Более подробное рассмотрение связей между сущностями и таблицами приведено в разделах 4.5.10. – 4.5.15.

Простота структурных единиц реляционной модели позволила привлечь формальные математические методы для описания обработки данных. С этой целью Э. Коддом были разработаны языки *реляционной алгебры* и *реляционного исчисления*, обладающие необходимой полнотой, требуемой для такой обработки.

Понятие реляционной модели является центральным в реляционной теории, благодаря которой разработаны принципы создания реляционных СУБД и языка SQL. Основным достижением реляционной теории является доказательство того, что любая структура данных может быть представлена в виде реляционной базы данных.

Можно сравнить переход к реляционным базам данных с переходом от ассемблера к высокоуровневым языкам программирования. И кстати, одной из целей Э. Кодда было предоставить конечным пользователям средства для работы с базами данных без обращения к услугам профессиональных программистов.

Наиболее полно реляционная модель рассмотрена в книге соратника Э. Кодда - К. Дж. Дейта [2].

2. СУБД

Вспомним определение БД – «организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей». Исходя из этого определения можно понять, что для поддержки функционирования БД необходим компьютер с некоторым программным обеспечением, позволяющим обеспечить хранение, изменение и выдачу данных по запросу пользователей. Отсюда можно дать определение ***системы управления базами данных (СУБД)*** – «совокупность программных, лингвистических и методических средств общего или специального назначения, обеспечивающих управление созданием и использованием БД» [12]. Во многих источниках возникает путаница между терминами БД и СУБД – мы будем строго разделять эти понятия.

Прежде чем читать эту книгу дальше, ответьте для себя на один вопрос: «Нужно ли программисту прикладных приложений понимать, как работает СУБД?». Том Кайт, признанный специалист ORACLE, в своей книге «ORACLE для профессионалов» [4] настаивает, что это просто необходимо.

2.1. Архитектура СУБД

Под ***архитектурой СУБД*** понимается совокупность ее функциональных компонентов, а также средств обеспечения их взаимодействия друг с другом, с пользователями и с системным персоналом.

Ниже в упрощенном виде рассматривается обобщенная архитектура СУБД, предложенная подкомитетом SPARC (*англ.* Standards Planning and Requirements Committee, комитет по планированию стандартов) американского национального института стандартов ANSI, так называемая логическая архитектура ANSI/SPARC, впервые представленная в 1975 г. (см. раздел 2.1.1).

Далее рассматривается эволюция физической организации СУБД и ИС, построенных на основе этих СУБД от файл-серверной до трехзвенной архитектуры (см. разделы 2.1.2 – 2.1.5).

2.1.1. Трехуровневая логическая архитектура систем баз данных ANSI/SPARC

Архитектура СУБД ANSI/SPARC описывает **логическую** организацию системы с точки зрения представления данных пользователям и включает три уровня:

- внутренний (физический);
- промежуточный (концептуальный);
- внешний (пользовательский).

Каждый из этих уровней состоит из одного или нескольких представлений (рис. 2.1).

В основе архитектуры ANSI-SPARC лежит концептуальный уровень. В современных СУБД он может быть реализован при помощи представлений (см. раздел 3.4.15). Концептуальный уровень описывает данные и их взаимосвязи с наиболее общей точки зрения, — концепции архитекторов базы, использующих реляционную или другую модель.

Внутренний уровень позволяет скрыть подробности физического хранения данных (носители, файлы, таблицы, триггеры и т.д.) от концептуального уровня. Отделение внутреннего уровня от концептуального обеспечивает так называемую ***физическую независимость*** данных.

На внешнем уровне описываются различные подмножества элементов концептуального уровня для представлений данных различным пользовательским программам. Каждый пользователь получает в своё распоряжение часть представлений о данных, но полная концепция

скрыта. Отделение внешнего уровня от концептуального обеспечивает ***логическую независимость*** данных.

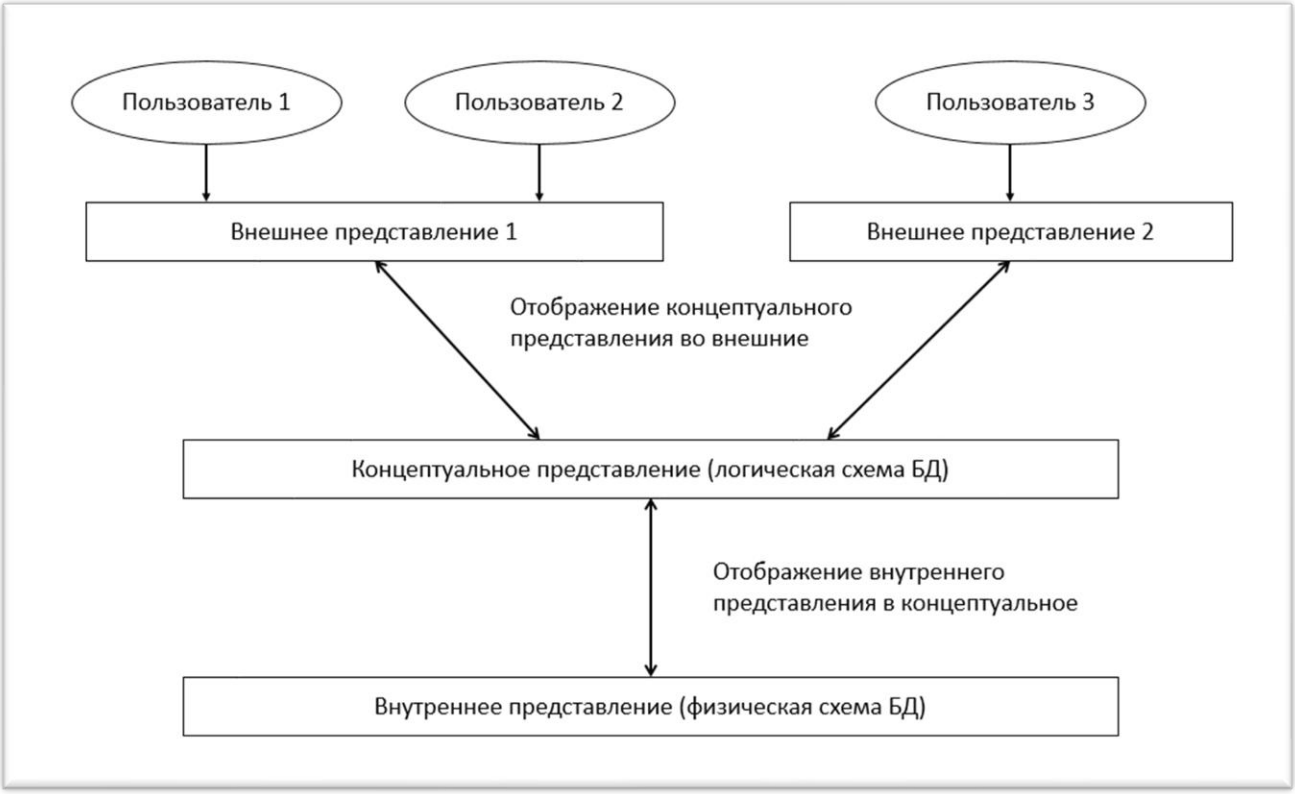

Рис. 2.1. Схематичное представление архитектуры ANSI/SPARC

Внешний уровень.

Внешний уровень – представление данных с точки зрения пользователей. Обычно он состоит из нескольких различных внешних представлений БД.

Каждый пользователь имеет дело с представлением предметной области в наиболее удобной для него форме. Внешнее представление пользователя содержит только те сущности, атрибуты и связи, которые ему необходимы, другие, содержащиеся в базе сущности, ему недоступны. Кроме того, различные представления могут по-разному отображать одни и те же данные (например, форматы дат, размерность физ. величин и т.д.). Например, бухгалтеру нужны объемы продаж в стоимостном (денежном) выражении, начальнику склада интересны объемы продаж в штуках, а начальнику перевозок объемы продаж нужны в тоннах. Некоторые представления могут содержать производные или вычисляемые данные, которые не хранятся в БД, а создаются по мере надобности. В современных СУБД внешний уровень формируется посредством механизма представлений языка SQL (см. разделы 3.4.15. и 4.6.2.).

Концептуальный уровень

Концептуальный уровень – обобщающее представление БД. Описывает то, какие сущности содержатся в БД, а также связи между ними. Этот уровень содержит логическую структуру БД. На концептуальном уровне описываются следующие компоненты:

- все сущности, их атрибуты и связи;
- ограничения, накладываемые на данные;
- семантическая информация о данных;
- информация о мерах обеспечения безопасности и поддержки согласованности данных.

Концептуальный уровень поддерживает все внешние представления. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных, их длине или максимальном количестве символов, но не должно содержать сведений об организации хранения, например, об конкретных устройствах хранения или объеме отведенного пространства в байтах.

Внутренний уровень.

Внутренний уровень – это физическое представление БД в памяти компьютера. Этот уровень описывает, как информация хранится на конкретных носителях. Внутренний уровень содержит описание структур данных и организации отдельных файлов и устройств, используемых для хранения данных. На этом уровне осуществляется взаимодействие СУБД с методами доступа ОС. Вот некоторая информация, которая хранится на внутреннем (физическом) уровне:

- распределение дискового пространства для хранения данных и индексов;
- сведения о размещении записей;
- сведения о сжатии данных и выбранных методах их шифрования.

Архитектура СУБД ANSI/SPARC позволяет обеспечить логическую и физическую независимость данных от программ и пользователей, т.е. поддерживает концепции БД (см. раздел 1.1.2). Логическая независимость предполагает возможность изменения одного приложения без корректировки других приложений, работающих с этой же БД. Физическая независимость предполагает возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений, работающих с этими данными.

2.1.2. Файл-серверная архитектура

Файл-серверная архитектура была самой первой и самой несовершенной физической архитектурой построения ИС на основе СУБД. В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть путем пересылки файлов, содержащих запрашиваемую информацию. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок, т.е. при необходимости изменения блокируются целиком файлы БД, которые могут содержать даже несколько таблиц.

Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера.

Недостатки: потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность. Применялась чаще всего в локальных приложениях, использующих функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

На данный момент файл-серверная технология считается устаревшей и практически не используется.

Примеры: Paradox, dBase, FoxPro, Visual FoxPro. Все эти системы не являются полноценными СУБД, а представляют собой хранилища записей.

2.1.3. Встраиваемые СУБД

Встраиваемая СУБД — та, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети.

Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить посредством языка SQL либо через специальные программные интерфейсы.

Примеры: OpenEdge, SQLite, BerkeleyDB, Firebird Embedded, Microsoft SQL Server Compact.

2.1.4. Клиент серверная архитектура

Суть этой архитектуры в том, что система разбивается на две физические части, которые выполняются на разных узлах сети, - клиентскую и серверную части. Пользователь взаимодействуют с клиентской частью системы, которая преобразует запросы пользователя в форму языка SQL. Серверная часть в свою очередь выполняет запрос клиента и возвращает результаты выборки данных. Клиентское приложение отображает полученные данные в удобной для пользователя форме.

Клиент-серверная архитектура позволяет обмениваться клиенту и серверу минимально необходимыми объёмами информации. При этом основная вычислительная нагрузка ложится на сервер. Клиент может выполнять функции предварительной обработки перед передачей информации серверу, но в основном его функции заключаются в организации доступа пользователя к серверу.

В большинстве случаев клиент-серверная архитектура гораздо менее требовательна к пропускной способности компьютерной сети, чем файл-серверная, особенно при выполнении операции поиска в базе данных по заданным пользователем параметрам, т.к. для поиска нет необходимости передавать на клиент весь массив данных: клиент передаёт параметры запроса серверу, а сервер производит поиск по полученному запросу в базе данных. Результат выполнения запроса, который обычно на несколько порядков меньше по объёму, чем весь массив данных, возвращается клиенту, который обеспечивает отображение результата пользователю. В простейшем случае клиент-серверная архитектура представлена на рис. 2.2.

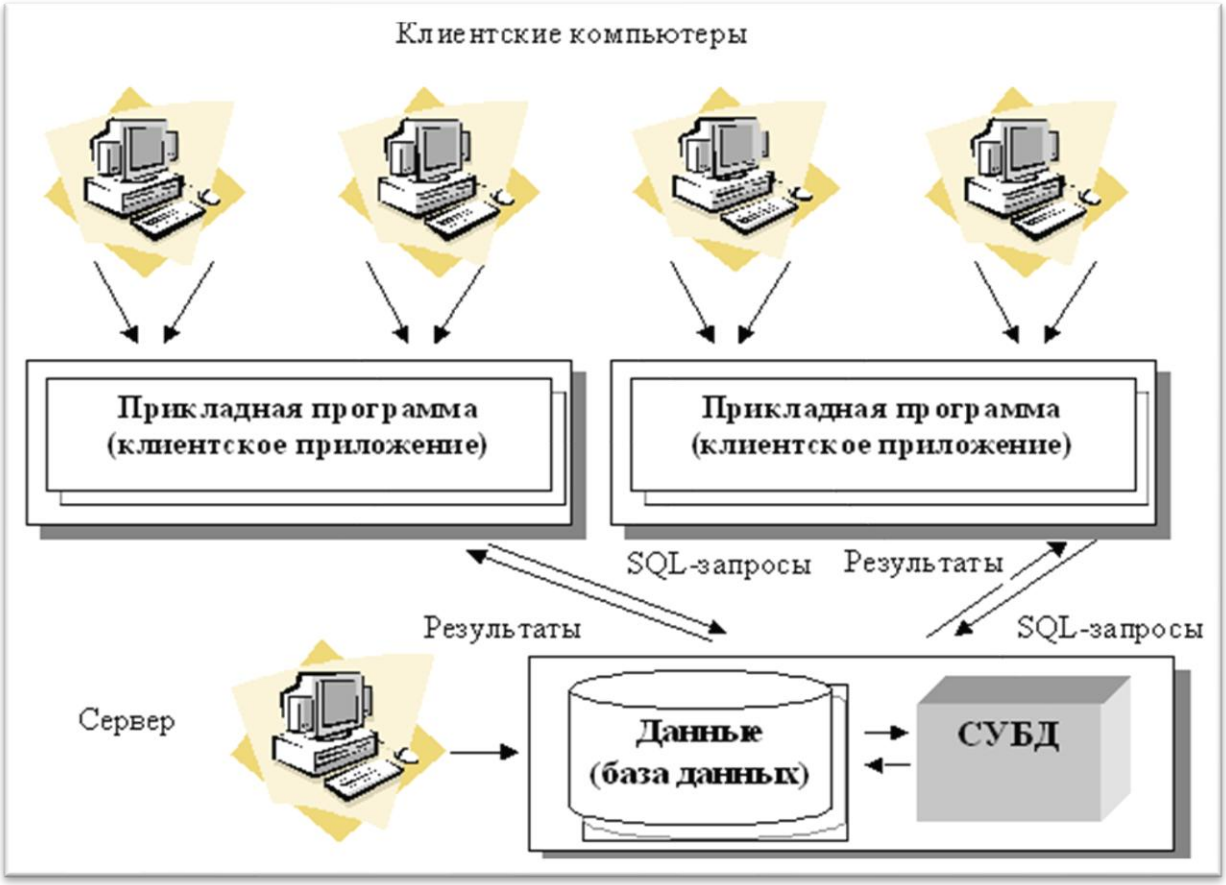

Рис. 2.2. Клиент серверная архитектура

Архитектура клиент-сервер не делит машины на только клиент или только сервер, а скорее позволяет распределить нагрузку и разделить функционал между клиентской частью и серверной.

В общем случае можно выделить три логических слоя и три задачи, выполняемые системой клиент-сервер (не путайте с уровнями архитектуры СУБД ANSI/SPARC, раздел 2.1.1):

- слой отображения данных;
- слой обработки бизнес-логики;
- слой доступа к данным.

Ясно, что слой доступа к данным должна располагаться на сервере, а вот слои отображения данных и обработки бизнес-логики могут располагаться как на клиенте, так и на сервере. Таким образом мы получаем четыре возможных варианта размещения слоев обработки данных на клиенте и сервере (см. рис. 2.3). Первые два варианта («а» и «б») — это варианты с использованием «тонкого» клиента, в то время как варианты («в» и «г») используют «толстый» клиент.

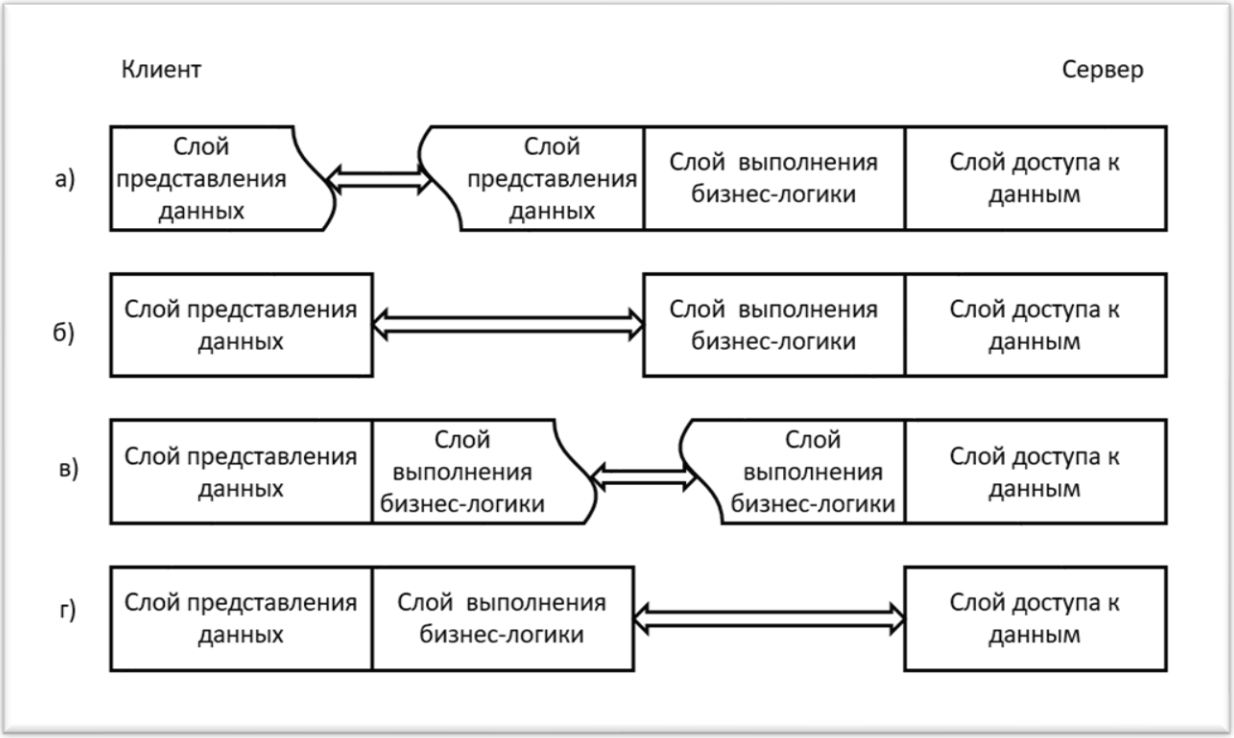

Рис. 2.3. Варианты клиент серверной архитектуры

Тонкий клиент (англ. *thin client*) в компьютерных технологиях — компьютер и программа-клиент, которая переносит все или большую часть задач по обработке информации на сервер. Примером тонкого клиента может служить компьютер с браузером. Все современные СУБД работают как веб-приложения т.е. функционируют как сервер, работающий с тонким клиентом. Тонкий клиент имеет ряд преимуществ перед толстым:

- низкая стоимость внедрения;
- очень простая поддержка;
- независимость от ОС;
- доступность из любой точки мира.

Однако, не следует забывать, что тонкий клиент обеспечивает ограниченную функциональность и предполагает большую нагрузку на сервер.

Толстый клиент (англ. *rich client; heavy client*) — в архитектуре клиент - сервер — это приложение, обеспечивающее (в противовес тонкому клиенту) расширенную функциональность. Сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на компьютер клиента.

Использование толстых клиентов имеет ряд преимуществ в определенных сценариях использования:

- высокая производительность и отзывчивость интерфейса. Поскольку большинство операций выполняется локально, нет задержек при низкой скорости подключения к серверу;
- возможность работы офлайн с последующей синхронизацией данных при появлении подключения к сети;
- более эффективное использование ресурсов пользовательского устройства, что важно при использовании ресурсоемких приложений;
- меньшая нагрузка на сетевую инфраструктуру, так как основные вычисления происходят на стороне клиента.

Выбор между толстым и тонким клиентом должен основываться на комплексном анализе бизнес-задач, технических требований и финансовых возможностей компании. Каждый подход имеет свои сильные и слабые стороны, которые необходимо учитывать при проектировании информационной системы.

Очевидно, что клиент-серверная архитектура обладает рядом достоинств:

- снижаются требования к пропускной способности сети;
- снижаются требования к вычислительной мощности клиента;
- повышается защищенность данных из-за централизованного хранения данных «под присмотром» квалифицированных администраторов.

Однако, у этой архитектуры есть и недостатки – недостаточно высокая надежность (зависимость от работоспособности сервера) и сложность масштабирования (решается только увеличением мощности сервера).

Примеры: ORACLE Database, IBM DB2, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL.

2.1.5. Трехзвенная архитектура

Постепенно в среде разработчиков IT-решений родилось понимание, что три слоя обработки данных можно разнести на три физических уровня – так родилась трехзвенная архитектура.

Базы данных редко используются сами по себе — чаще всего они служат ядром ИС (транзакционных или аналитических) и встроены в сложные системные архитектуры. Классическая архитектура, которая сейчас применяется в подавляющем количестве ИС, состоит из трех звеньев, или слоев: представления данных, бизнес-логики и доступа к данным, как и в двухзвенной архитектуре «клиент-сервер» (см раздел 2.1.4.). Каждый из слоев предоставляется в виде сервиса и обычно размещен на одном или более выделенных серверах. На рисунке 2.4 слой доступа к данным представлен сервером БД, слой бизнес-логики – серверами приложений, а слой представления данных расположен на клиентских компьютерах.

Такая многозвенная система имеет немало преимуществ: на каждом из слоев проще выполнять масштабирование, тиражирование, секционирование и кэширование. Более того, путем переноса вычислений со слоя на слой можно эффективно регулировать нагрузку на сервера. Масштабируемость и отказоустойчивость ИС может быть повышена путем горизонтального масштабирования слоя бизнес-процессов (добавления новых серверов), не заботясь о сложных протоколах синхронизации. Горизонтальное масштабирование слоя доступа к данным требует более сложных управляющих протоколов обеспечения согласованности данных (например, протокола двухфазной фиксации), поэтому на этом уровне достичь высокой масштабируемости и отказоустойчивости сложнее.

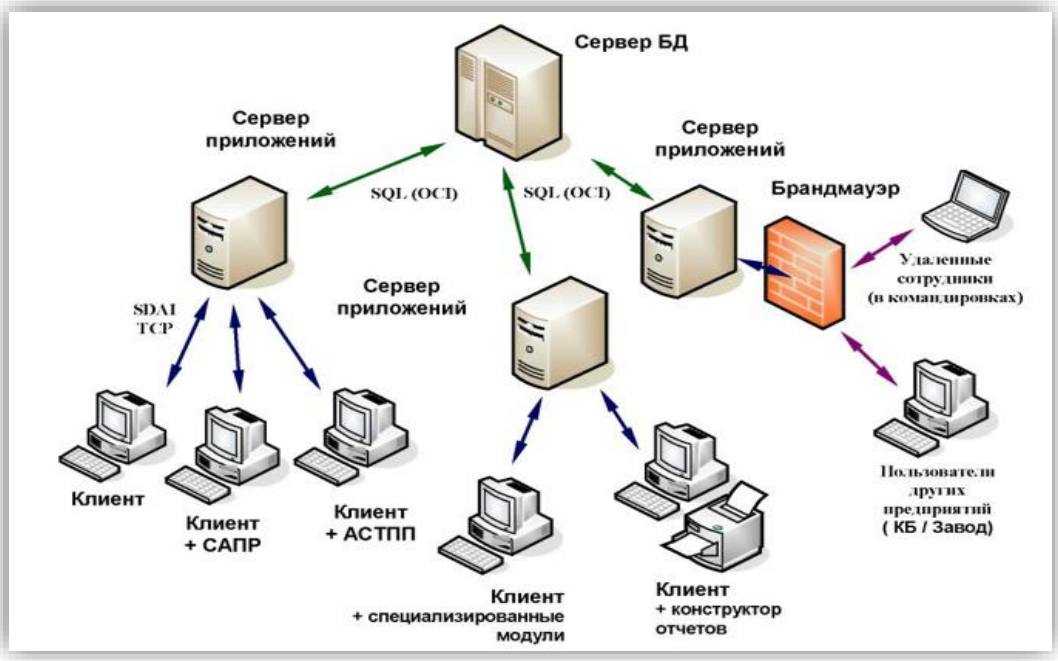

Рис. 2.4. Трехзвенная архитектура

На самом деле полную мощь трехзвенной архитектуры иллюстрирует рисунок 2.5, на котором явно определены слои обработки данных. Видно, что при необходимости масштабирования слоя бизнес-логики необходимо просто добавить сервера, подключив их к балансировщику.

Балансировщик нагрузки серверов — аппаратная схема или один сервер, распределяющий запросы к серверам приложений. Использование балансировщика повышает отказоустойчивость и позволяет равномерно распределять нагрузку на сервера.

Очевидно, что слой БД может быть разнороден, т.е. включать в себя разные СУБД (кстати, не только реляционные), что обеспечивает масштабируемость и гибкость системы. Введение слоя обработки бизнес-логики обеспечивает дополнительную защищенность данных. Каждый из слоев обеспечивает дополнительный уровень отказоустойчивости системы путём включения резервных серверов, которые принимают на себя нагрузку при отказах отдельных звеньев системы. Такое резервирование также повышает возможности модернизации системы: появляется возможность вывода отдельных звеньев из эксплуатации для обновления и отладки, не ухудшая общей производительности системы за счет резервных серверов.

Необходимо отметить, что в подавляющем числе внедрений в трехзвенной архитектуре применяется тонкий клиент.

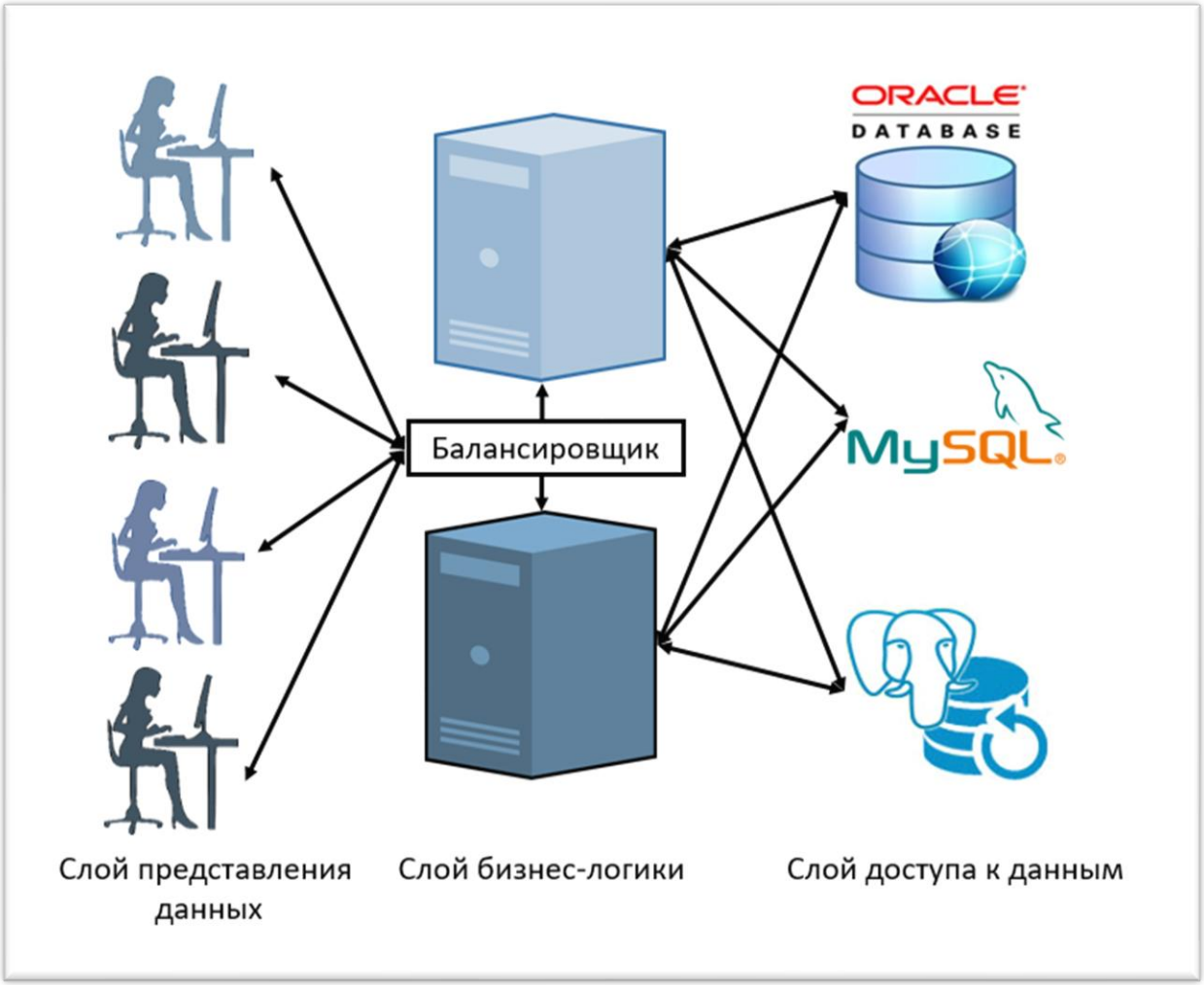

Рис. 2.5 Трехзвенная архитектура

Итак, подытожим преимущества и недостатки трехзвенной архитектуры.

Преимущества трехзвенной архитектуры:

- простота модификации;
- простота расширения;
- простота интеграции;
- повышение безопасности;
- возможность работы тонкого клиента позволяет:
    - получить низкую стоимость внедрения клиента;
    - снизить требования к поддержке клиента;
    - получить независимость клиента от ОС;
    - обеспечить доступность системы из любой точки мира.

Недостатки трехзвенной архитектуры скорее являются платой за преимущества:

- трудность проектирования;
- трудность отладки;
- эксплуатация требует высококвалифицированного персонала.

В настоящее время в связи с развитием технологий разработки ИС трудности проектирования и внедрения трехзвенной архитектуры быстро уменьшаются, и эта архитектура все больше и больше завоевывает пространство IT решений.

2.2. Функции СУБД

Основные функции СУБД тесно связаны с концепциями БД (см. раздел 1.1.2), которые СУБД должны поддерживать (см. Таблицу 2.1).

Таблица 2.1 Соответствие концепций БД и функций СУБД

Концепции БД	Функции СУБД
Отчуждение данных от носителей.	Управление данными во внешней памяти
Отчуждение данных от программ.	Поддержка языков БД Ведение словаря БД
Хранение описания данных вместе с самими данными	Ведение словаря БД
Поддержание БД в согласованном (целостном) состоянии.	Обеспечение согласованности БД Управление транзакциями Управление блокировками и клинчами
Защита информации	Управление журналами изменений в БД Управление транзакциями Обеспечение безопасности БД
Поддержка многопользовательской (многозадачной) работы	Управление транзакциями Управление блокировками и клинчами

Дале подробно рассматриваются функции СУБД.

2.2.1. Управление данными во внешней памяти

Управление СУБД данными во внешней памяти обеспечивает пользователя возможностью оперирования абстрактными данными, позволяет пользователю не задумываться о том, где и в каких физических структурах содержатся данные составляющие БД. Одним из таких абстрактных понятий хранения данных являются **табличные пространства** (tablespace). **Табличное пространство** — это логический контейнер хранения информации в СУБД, который находится на вершине иерархии хранения и образован из одного или нескольких файлов данных.

Основные табличные пространства, используемые в СУБД ORACLE:

- System содержит важнейшую информацию словаря СУБД;
- Sysaux дополняет табличное пространство System, храня метаданные для различных приложений ORACLE;
- пользовательские табличные пространства содержат объекты БД – таблицы и индексы;
- временные табличные пространства СУБД использует в качестве рабочих областей для выполнения таких задач, как операции сортировки при выполнении запросов пользователей или при создании индексов;
- UNDO - табличное пространство отмены содержит образы данных на момент начала транзакции и служат для отката транзакций, восстановление прерванных транзакций и обеспечения согласованности чтения.

При инициализации кластера PostgreSQL создаются два табличных пространства:

- pg_default — располагается в каталоге PGDATA/base и используется как табличное пространство по умолчанию;
- pg_global — располагается в каталоге PGDATA/global и хранит общие для всего кластера объекты.

Важное место в логической структуре БД занимает понятие «схемы». Схема — это именованный набор объектов и данных, принадлежащих конкретному пользователю. По-другому говорят, что пользователь «обладает» схемой данных, так главный конструктор разработки ИС обладает всей схемой (всей БД) этой ИС, и только он может раздавать права на изменения

объектов и данных в этой схеме. Понятие схемы примерно совпадает с понятием БД конкретной разработки.

Табличные пространства не имеют отношения ни к логической структуре базы данных, ни к схеме, а предназначены для указания места хранения данных на физических носителях. Различные объекты одной базы данных, например, индекс и таблица, могут физически храниться в разных пространствах. С помощью табличных пространств администратор имеет возможность контролировать использование базой данных доступного объема памяти и оптимизировать быстродействие. Например, пространство, используемое для индексов, можно разместить на быстрых накопителях, а пространство, используемое для архивных данных, можно разместить на менее быстрых, но значительно более ёмких.

Минимальной логической единицей хранения данных является **блок** - базовая единица операций чтения и записи для базы данных (размер блока обычно от 4 до 16 КБ). Управление дисковым пространством по одному блоку было бы очень трудоёмкой задачей, поэтому блоки группируются в экстенты (extent). **Экстентом** называется набор последовательных блоков внутри одного файла данных. Экстенты могут находиться в любом или во всех из доступных для табличного пространства файлов данных. Следующий уровень хранения – **сегмент**, который состоит из одного или более экстентов, последовательно пронумерованных. Экстент можно идентифицировать как внутри сегмента (экстенты последовательно пронумерованы в пределах сегмента), так и внутри файла данных: каждый экстент находится только в одном файле данных. Табличное пространство — это набор сегментов (см. рис. 2.6.), т.е. табличные пространства, также, как и сегменты могут размещаться в нескольких физических файлах, а объекты БД (таблицы, индексы) размещаются в отдельном сегменте.

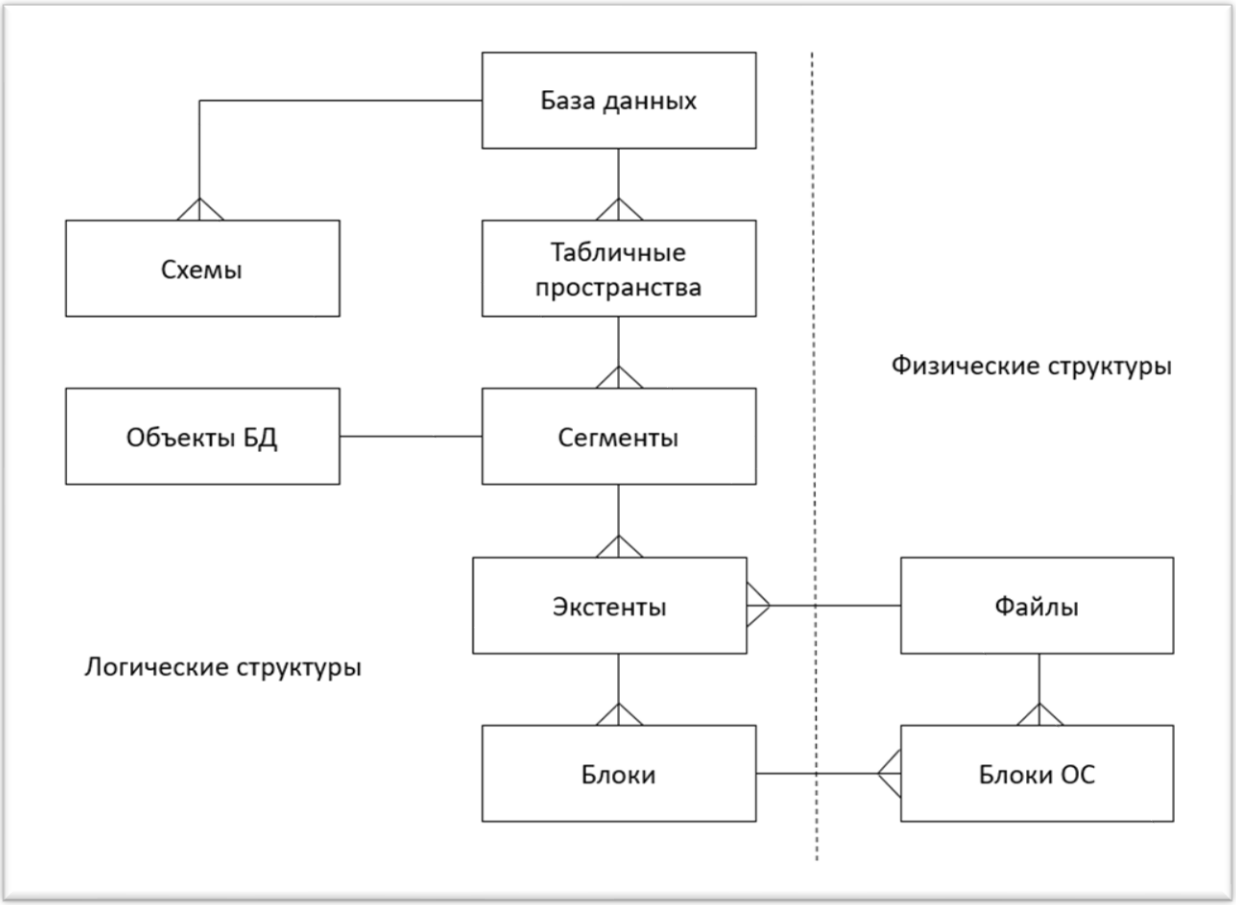

Рис. 2.6. Иерархия логических единиц хранения БД

Контейнеры, в которых размещаются экстенты могут быть файлами ОС, чистыми (RAW) разделами, или файлами в кластеризованной файловой системе. Большинство СУБД позволяют администратору системы выбрать один из двух способов размещения файлов БД на дисках: на

«*чистых*» дисках или в *файловой системе* ОС. В первом случае управление данными, хранящимися на отдельных носителях, производится низкоуровневыми средствами самих СУБД. Рассмотрим достоинства и недостатки обоих вариантов размещения файлов на диске.

Чистые (RAW) разделы — это не файлы, это диски или их части без файловой системы. Их нельзя просматривать с помощью команды ls или проводника Windows. Эти разделы представляют собой всего лишь большие области диска без какой-либо файловой системы внутри.

Достоинством хранения информации на «чистых» дисках является то, что внешняя память используется более эффективно и, как правило, увеличивается производительность обмена с дисками. Экономия внешней памяти от 10 до 20% достигается на основе устранения необходимости организации самой файловой системы. Так, в ОС UNIX служебная информация файловой системы составляет примерно 10% от емкости дисков. Примерно столько же памяти резервируется для последующего возможного расширения файлов. Производительность (обычно порядка 10%) повышается на основе удаления дополнительного слоя программного обеспечения при обращении к дискам.

Некоторые СУБД, например, Ingres и Interbase требуют обязательного использования файловой системы**,** которая обеспечивает следующие *достоинства.* Во-первых, использование файловой системы обладает большей *гибкостью,* так как дает администратору стандартные средства обслуживания файлов: утилиты резервного копирования, архивации, восстановления, а также возможность пользования другими программами работы с файлами (редакторами, антивирусными программами, утилитами контроля качества носителя и т. д.). Во-вторых, в некоторых случаях выполнение операций ввода/вывода через файловую систему обеспечивает оптимизацию, которую СУБД не может реализовать.

В настоящее время часто применяются системы хранения данных (СХД) - комплексное программно-аппаратное решение по организации надёжного хранения данных и предоставления гарантированного доступа к ним. Такие системы состоят из управляющего сервера и массивов дисков, причем диски объединены в сложные структуры (RAID) позволяющие повысить отказоустойчивость и быстродействие. Снаружи СХД выглядит как сплошной массив памяти, возможно разбитый на LUN (виртуальные диски), которые монтируются на файловую систему ОС или предоставляются СУБД как чистые (RAW) разделы.

Пользователи СУБД не знают на каких носителях находятся логические единицы хранения – этот механизм обеспечивает отчуждение данных от носителей.

2.2.2. Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной обслуживания буферов [12].

Эта обширная цитата показывает общую необходимость в буферизации данных, каждая конкретная СУБД выполняет эту функцию по-своему в зависимости от сложности внутренней логики и реализуемых механизмов. Например, общая схема разделов оперативной памяти СУБД ORACLE показана на рис. 2.7.

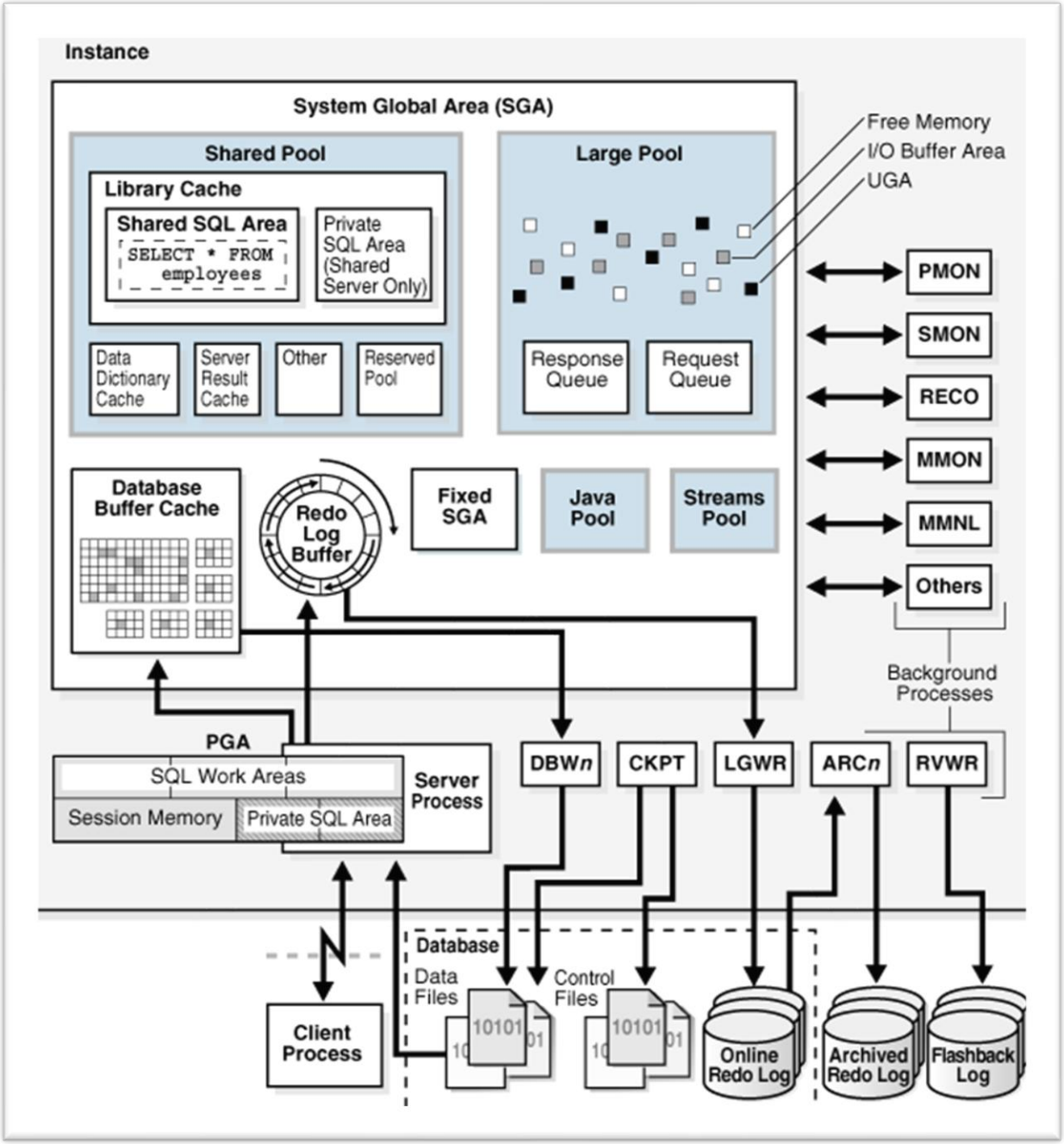

Рисунок 2.6. Общая схема буферов СУБД ORACLE.

- **Пул Java** (Java pool). Это фиксированный объем памяти, выделенной для виртуальной машины Java (Virtua\Java Machine - JVM), которая действует в базе данных.
- **Большой пул** (large pool). Этот пул используется подключениями пользователей для памяти сеанса, средствами параллельного выполнения для буферов сообщений и резервным копированием RMAN для буферов дискового ввода-вывода.
- **Разделяемый пул** (shared pool). Этот пул содержит разделяемые курсоры, хранимые процедуры, объекты состояния, кэши словаря СУБД и многие десятки других элементов данных.
- **Буфера кэшированных блоков базы данных** (database buffer cache) содержат образы блоков БД (см. раздел 2.2.1.), хранящихся в памяти внешних устройств или созданные динамически, чтобы реализовать транзакционную модель (см. раздел 2.2.5.). Все транзакции, которые одновременно выполняются в СУБД, совместно используют буферный кэш базы данных. Измененные блоки сбрасываются обратно на внешние

носители. Поскольку изменение информации и сброс измененных данных на внешние носители выполняется асинхронно, данные в оперативной памяти и на дисковых носителях могут отличаться. Для устранения этой рассогласованности специальный процесс обеспечивает создание «точек согласованности» - моментов, когда содержимое операционной памяти точно соответствует состоянию данных на внешних носителях. Данные об этих «точках согласованности» записываются в журналах СУБД.

- **Буфера журнала повторов** (redo log buffer)– это циклический буфер в SGA, который содержит информацию об изменениях, произведенных в базе данных. Эта информация хранится в записях повторов, которые должны быть записаны в оперативные журналы повторения (online redo log). Записи повторов содержат информацию, необходимую, чтобы реконструировать (или повторить) изменения, которые производятся в базе. Записи повторов используются для восстановления базы данных (см. раздел 2.2.3.) и при репликации (см. раздел 2.2.5.).
- **Фиксированная область SGA**. (fixed SGA)–область памяти, которая содержит набор переменных, которые указывают на другие компоненты SGA и переменных, содержащих значения различных параметров СУБД.

Управление таким количеством структур памяти требует значительных расходов процессорных мощностей, однако, чем больше работы выполняет СУБД в памяти по сравнению с обращениями к дискам, тем быстрее она реагирует на запросы. Конечно, по мере сокращения операций ввода-вывода загрузка процессора также сокращается, что повышает общую эффективность системы.

Настройка размера буферов оперативной памяти — один из ключевых навыков DBA любой СУБД. Стандартные значения, выставленное по умолчанию, подходят разве что для тестовых стендов. Грамотная настройка буферов — обязательное условие стабильной и быстрой работы баз данных.

Опыт показывает, что оптимальный баланс производительности и размера кэша всегда достигается путём тестирования и анализа рабочих нагрузок конкретной базы данных. Поэтому регулярный мониторинг, а также использование технологических средств конкретной СУБД помогут понять какие размеры буферов наиболее эффективны для ваших задач.

2.2.3. Поддержка языков СУБД

Еще Эдгар Кодд в своих «12 правилах определения реляционной СУБД» (см. Приложение Г) писал:

«Система управления реляционными базами данных должна поддерживать хотя бы один реляционный язык, который

(а) имеет линейный синтаксис,

(б) может использоваться как интерактивно, так и в прикладных программах,

(в) поддерживает операции определения данных, определения представлений, манипулирования данными (интерактивные и программные), ограничители согласованности, управления доступом и операции управления транзакциями (begin, commit и rollback).»

В современных реляционных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком реляционных СУБД является язык SQL (см. главу 3).

Основным инструментом поддержки языков в СУБД является «процессор языка базы данных» (чаще по традиции называемый «оптимизатором запросов»), обеспечивающий оптимизацию и исполнение запросов и создающий, как правило, машинно-независимый исполняемый внутренний код. Подобно программе на языке Java оптимизатор СУБД

компилирует программу на языке SQL во внутренний код, а выполняет этот код ядро СУБД, подобно Java-машине.

Поскольку SQL является непроцедурным языком, каждая СУБД поддерживает процедурное расширение SQL, содержащее привычные программные конструкции ветвлений, циклов и т.д. (подробнее см. разделы 3.2.2 и 3.2.4). Некоторые СУБД поддерживают другие широко распространенные процедурные языки. Ниже представлена таблица из документации PostgreSQL, показывающая широкий спектр поддерживаемых языков программирования.

Таблица 2.2. Поддерживаемые в СУБД PostgreSQL процедурные языки

Имя	Язык	Сайт с документацией
PL/Java	Java	http://pljava.projects.postgresql.org/
PL/PHP	PHP	http://www.commandprompt.com/community/plphp/
PL/Py	Python	http://python.projects.postgresql.org/backend/
PL/R	R	http://www.joeconway.com/plr/
PL/Ruby	Ruby	http://raa.ruby-lang.org/project/pl-ruby/
PL/Scheme	Scheme	http://plscheme.projects.postgresql.org/
PL/sh	Оболочка Unix	http://plsh.projects.postgresql.org/

2.2.4. Обеспечение согласованности БД

Согласованность БД по другому «целостность» (database integrity), — соответствие имеющейся в БД информации её внутренней логике, структуре и всем явно заданным правилам предметной области. Каждое правило, налагающее некоторое ограничение на возможное состояние БД, называется ограничением согласованности (integrity constraint). В большинстве случаев эти ограничения определяются особенностями предметной области.

Примеры правил: в неделе семь дней; вес детали должен быть положительным; количество знаков в телефонном номере не должно превышать 15; возраст родителей не может быть меньше возраста их биологического ребёнка и т.д.

Понятие согласованности, или целостности данных является ключевым понятием БД. Одной из важнейших задач, решаемой СУБД, является поддержание в любой момент времени взаимной непротиворечивости, правильности и точности данных, хранящихся в БД. Этот процесс называется обеспечением согласованности базы данных.

Очевидно, что ограничения должны быть формально объявлены для СУБД, после чего СУБД должна отслеживать их выполнение. Объявление ограничений сводится просто к использованию соответствующих средств языка базы данных, а соблюдение ограничений осуществляется с помощью контроля со стороны СУБД операций с данными, которые могут нарушить эти ограничения, и запрещения тех операций, которые их действительно нарушают. Более подробно о методах обеспечения согласованности см. в разделе 4.8.

На самом деле в литературе нет устоявшегося общепринятого термина: где-то употребляется термин «согласованность», а где-то «целостность», а в некоторых изданиях одновременно оба термина. Являясь ярым последователем Оккама, мы будем использовать один термин, а именно «согласованность» как более созвучный идеологии ACID, обсуждаемой в следующей главе.

2.2.5. Управление транзакциями

Если внимательно посмотреть на таблицу 2.1, станет ясно, что управление транзакциями - одна из основных функций СУБД. Поддержка транзакций является основой обеспечения согласованности БД, позволяют достичь изолированности пользователя во многопользовательских системах, обеспечивает надежную защиту данных от сбоев. Многие механизмы СУБД, такие как копирование и репликация невозможны без полноценной поддержки транзакций.

Термин **«транзакция»** (англ. transaction, от лат. transactio) имеет латинское происхождение и, в прямом переводе, означает «обмен» или «соглашение». В широком, общеупотребимом смысле термин используется для обозначения любой операции по обмену данными, в следствие которой в систему вносятся изменения. Сегодня это слово часто употребляется в финансовой сфере и обозначает операцию, по итогу которой изменились банковские счета участников.

Разберем как происходит банковская транзакция, основные действия которой представлены на рис. 2.10. Главная цель банковской транзакции состоит в том, чтобы гарантированно одновременно провести изменения состояния счетов и документов. Если транзакция будет выполнена частично, деньги с одного счета уйдут, а на другой не придут, что нарушит баланс банка, а это недопустимо. Значит, главное свойство транзакции – неделимость (выполнение её единым блоком), что в свою очередь гарантирует перевод состояния банковских счетов из одного согласованного состояние в другое согласованное состояние с обеспечением постоянного сохранения баланса банка. Исходя из этого описания транзакции, можно сформулировать определение транзакции в терминах СУБД.

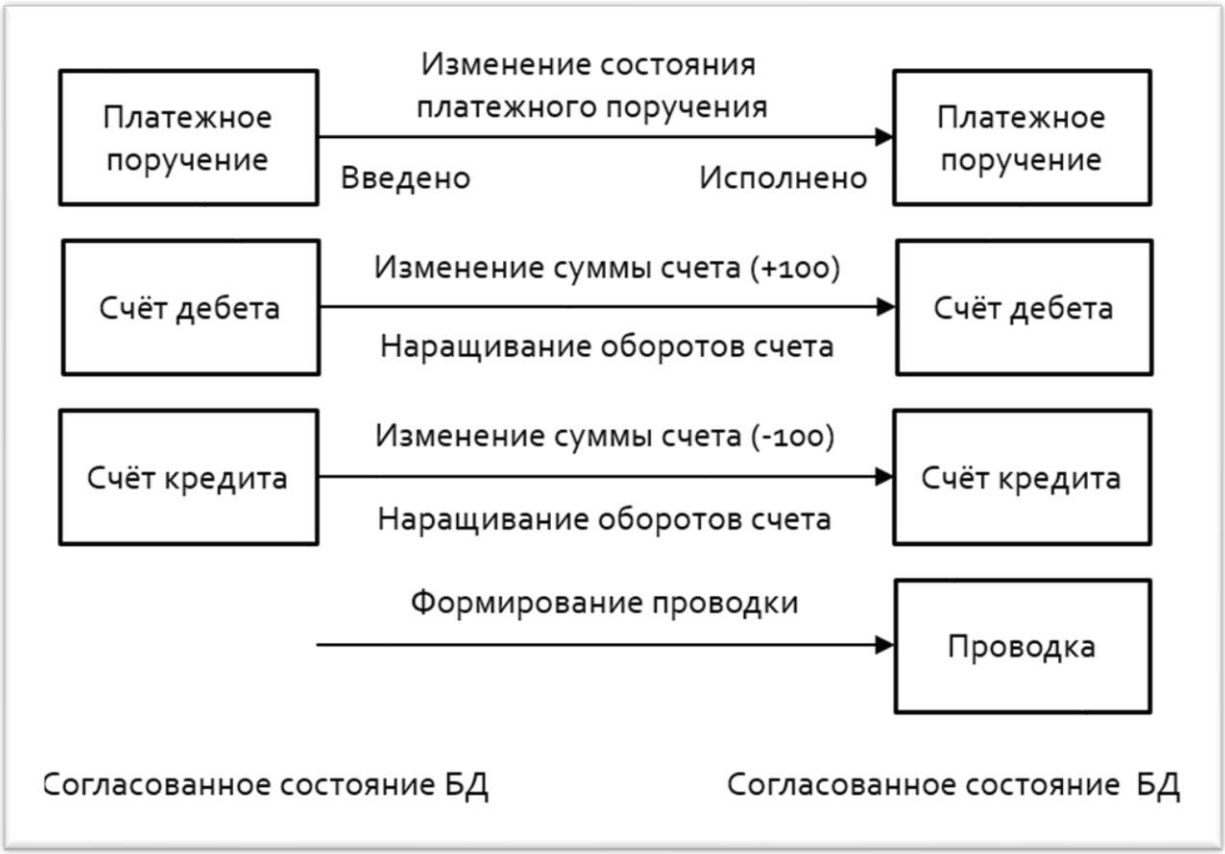

Рис. 2.10. Банковская транзакция

Далее мы будем употреблять термин «транзакция» только в смысле операций в среде СУБД.

Транзакция — логическая единица работы, состоящая из одного или нескольких операторов SQL, которую СУБД рассматривает и обрабатывает как неделимое действие, переводящее БД из одного согласованного состояния в другое согласованное состояние.

Допускается, что в процессе транзакции согласованность БД может нарушаться, но извне транзакции этого не видно.

Одним из наиболее распространённых наборов требований к транзакциям и транзакционным системам является набор ACID. Требования ACID были в основном сформулированы в конце 1970-х годов Джимом Греем.

Свойства транзакции (ACID):

- **Atomic** - атомарность. Транзакция — это неделимая единица, которая должна быть либо выполнена, либо отменена. То есть не может быть такого, что база данных выполнит только часть транзакции. Например, если был выдан оператор SQL, который должен удалить 1000 записей, то вся транзакция должна быть отменена (проведен ее откат), если база даст сбой после удаления 999 записей.
- **Coordination** - согласованность. Смысл транзакции состоит в том, чтобы база данных переходила из одного согласованного состояния в другое. База данных всегда должна пребывать в согласованном состоянии. Например, в банковской транзакции, включающей снятие денег с одного счета и перенос на другой счет, программа не может просто снять деньги со счета и остановиться. Это привело бы к несогласованным данным – нарушению баланса банка, а свойство согласованности транзакций гарантирует, что база никогда не перейдет в такое несогласованное состояние. Все транзакции должны обеспечивать согласованность базы данных. Например, если требуется удалить запись отдела из таблицы «Отделы», то транзакция не должна допустить такого действия, если в таблице «Сотрудники» имеются записи, относящиеся к удаляемому отделу. Понятие согласованности БД обсуждается в разделе 2.2.4.
- **Insulativity** - изолированность. Каждая транзакция, которая выполняется, не зависит от остальных. Изоляция означает, что несмотря на параллельный доступ к базе данных множества транзакций, каждая из них должна проходить изолированно от остальных. Свойство изоляции транзакций гарантирует, что транзакция не увидит изменений, внесенных другими транзакциями в БД, пока они выполняются. Т.е. промежуточные результаты работы транзакции не видны никому. Например, для любых двух транзакций T1 и T2 справедливо следующее утверждение: T1 сможет увидеть обновление T2 только после выполнения T2, а T2 сможет увидеть обновление T1 только после выполнения T1. Это свойство обеспечивается механизмом параллелизма базы данных. Хотя параллельный доступ — неотъемлемый свойство реляционной базы данных, техника изоляции создает впечатление, что пользователи выполняют транзакции последовательно, одну за другой (см. раздел 2.2.7).
- **Duration** - надежность. Все результаты, которые были достигнуты в ходе успешной транзакции, наверняка сохраняются в базе данных. Как только транзакция завершена, база данных должна гарантировать, что ее результаты не будут потеряны. Это свойство обеспечивается механизмами восстановления базы данных, гарантируя сохранность всех зафиксированных транзакций.

Рассмотрим один из вариантов механизма реализации транзакций, который показывает за счет чего достигаются все свойства транзакции.

Описанный ниже механизм реализации транзакций является строго логическим, то есть он лишь объясняет результат, который должен получиться при выполнении транзакции, но не предписывает, чтобы конкретная СУБД обеспечивает именно такой механизм реализации. Этот механизм транзакции связан с созданием мгновенных снимков – снапшотов (snapshot) и основан на многоверсионной модели (Multiversion Concurrency Control, MVCC). Предполагается, что каждая транзакция видит так называемый снимок данных, т. е. то согласованное состояние (версию) базы данных, которое она имела на определенный момент времени.

Снапшот (от англ. *snapshot* — мгновенный снимок), — фиксированное состояние данных на определённый момент времени. Создание снапшота иллюстрирует рис. 2.11. Представим себе буферный кэш базы данных (см. раздел 2.2.2), состоящий из блоков памяти, помеченных от **A** до **Q**; обращение к этим блокам идет через «таблицу блоков», в которой сосредоточена информация о содержимом блоков и их физических адресах в оперативной памяти. В момент начала работы транзакции создается снапшот памяти т.е. «основная таблица блоков» копируется и обращение к данным из новой транзакции (будем называть ее транзакция T1) идет через новую «таблицу блоков снапшота».

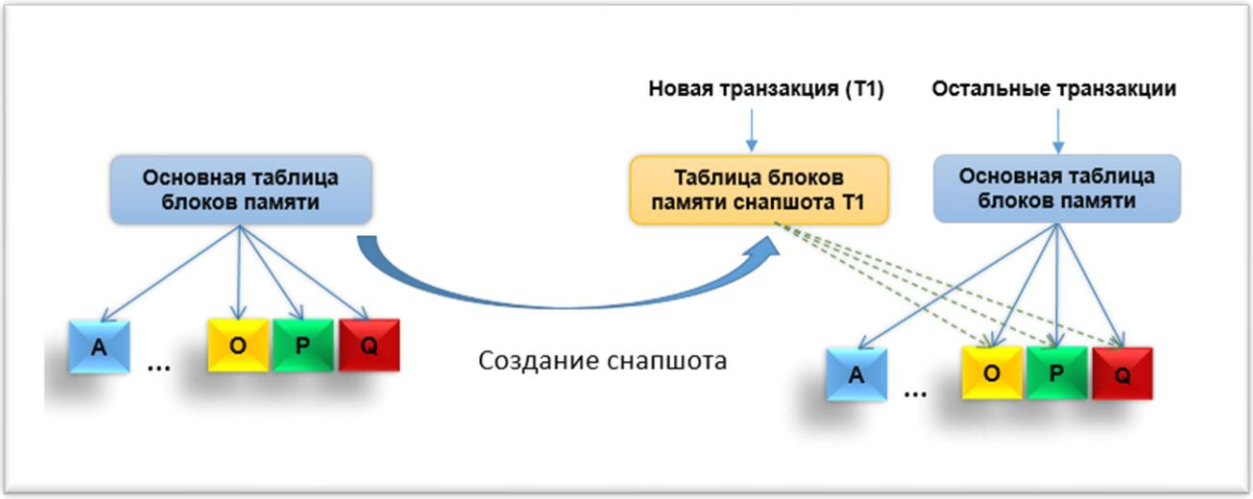

Рис. 2.11. Создание снапшота транзакции

При необходимости добавления данных транзакцией СУБД выделяет транзакции T1 новый блок **R** куда и записывается новая информация. Обратите внимание, что информация об этом блоке содержится только в «таблице блоков памяти снапшота T1», связанной с транзакцией T1, а остальные транзакции этот блок просто не видят, т.к. его нет в «основной таблице блоков» памяти (см. рис. 2.12.).

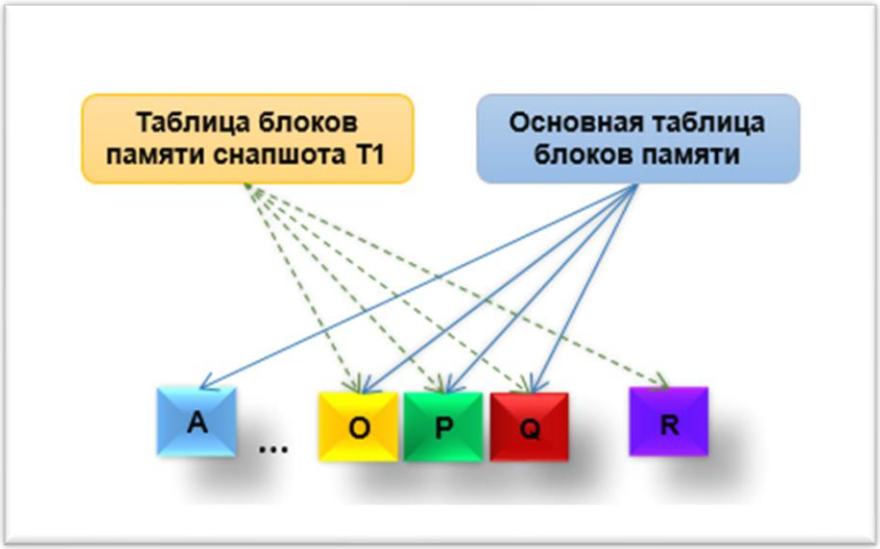

Рис. 2.12 Добавление данных

При необходимости изменения данных транзакцией СУБД опять выделяет транзакции T1 новый блок **Q1** куда и записывается измененная информация, а блок **Q** со «старой» информацией продолжает хранится в «буферном кэше базы данных» и именно на него ссылается «основная таблица блоков памяти». Теперь «таблица блоков памяти снапшота T1» ссылается на блок **Q1**, а «основная таблица блоков» ссылается на блок **Q.**

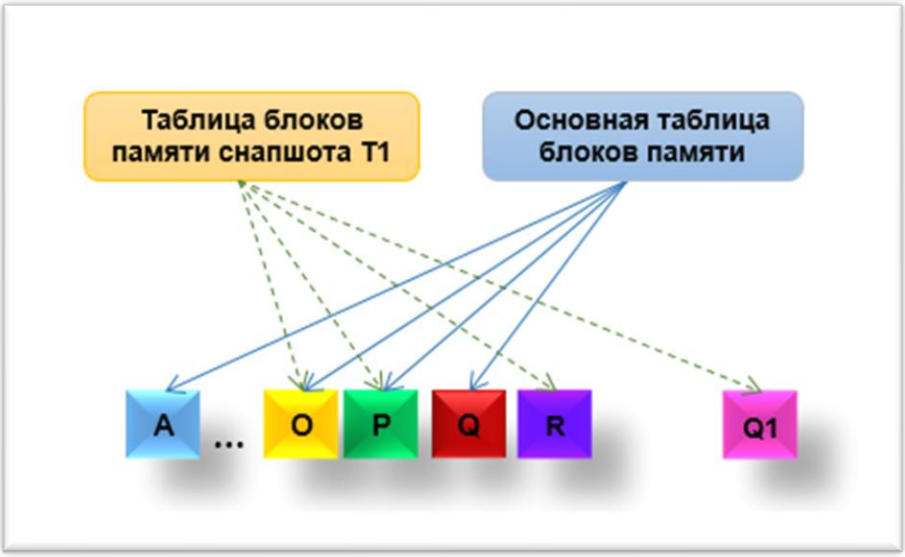

Рис. 2.13. Изменение данных

Таким образом достигается изолированность транзакций, ведь все транзакции полностью независимы от изменений, проходящих внутри транзакции T1. Так, например, транзакция чтения, создав свой снапшот просто читает данные из блоков памяти своего снапшота, не замечая, что происходит с блоками изменяемыми другими транзакциями.

Рассмотрим теперь завершение транзакции: если завершение транзакции успешно, то «таблица блоков памяти снапшота T1» копируется в «основную таблицу блоков памяти», а все ненужные блоки памяти, в данном случае это блок **Q**, освобождаются (рис. 2.14). Таким образом все результаты работы транзакции вступают в силу одномоментно, в момент переписи «основной таблицы блоков». Заметьте, что перепись «основной таблицы блоков памяти», в свою очередь тоже является транзакцией, т.е. может быть выполнена только целиком и переводит «основную таблицу блоков памяти», а вместе с ней и всю БД, из одного согласованного состояние в другое согласованное состояние.

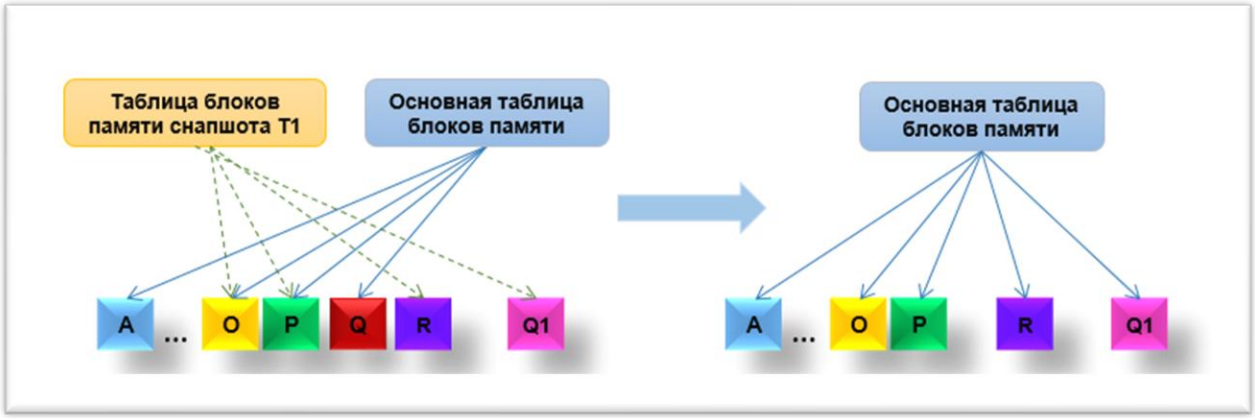

Рис. 2.14. Успешное завершение транзакции

Если во время обработки транзакции происходит сбой, транзакция откатывается - восстанавливается состояние БД на момент начала транзакции. В этом случае нужно просто стереть «таблицу блоков памяти снапшота T1» и освободить блоки, связанные только с транзакцией T1, при этом состояние БД, определяемое «основной таблицей блоков», не меняется (рис. 2.15).

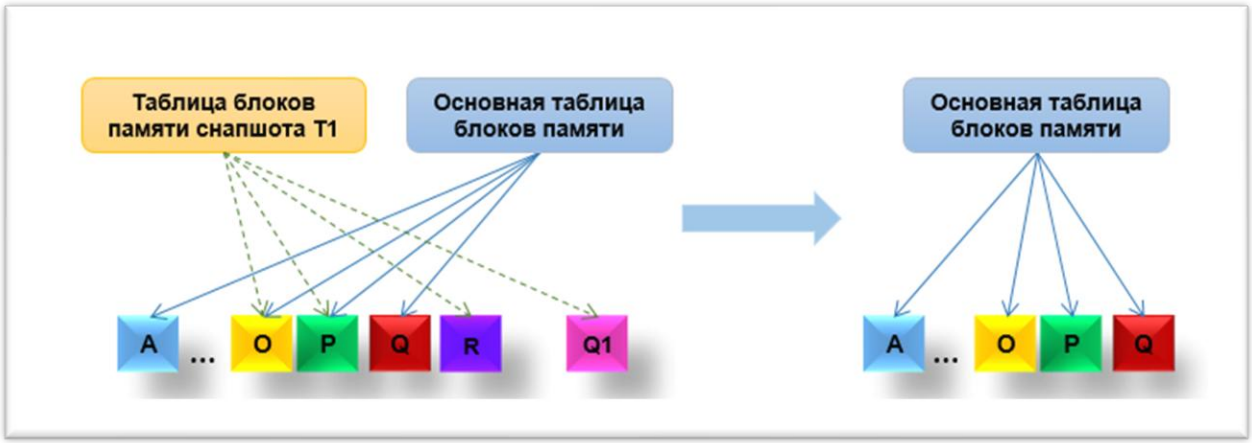

Рис. 2.15. Ошибочное завершение транзакции

Транзакция — это последовательность операторов на языке SQL выделенная специальным образом: операторами START TRANSACTION, COMMIT и ROLLBACK (более подробно см. в главе 4). Любой оператор языка SQL находящийся вне явных рамок транзакции рассматривается СУБД как отдельная транзакция, таким образом все действия с данными БД проходят в контексте транзакционной концепции.

Действий с данными в реляционной СУБД вне транзакций не бывает.

В некоторых СУБД возможно использование вложенных транзакций и при неверном завершении вложенной транзакции (ошибка или достижение команды ROLLBACK) откатываются ВСЕ внешние транзакции.

2.2.6. Управление блокировками и клинчами

Применение механизма снапшотов не может гарантировать полной изоляции транзакций, ведь возможны ситуации, когда две или более транзакций пытаются редактировать один и тот же блок данных. В этом случае должны быть порождены два блока с изменёнными данными – на рис. 2.16 это блоки **Q1** и **Q2.** Какой из них попадет в итоговую таблицу блоков? Ясно одно: какой бы блок из этих двух не попал, результаты транзакции, записанные в другом блоке, пропадут. Чтобы такого не случилось СУБД использует механизм блокировки – как только появляется копия блока памяти, предназначенная для внесения изменений (**Q1**), доступ к основному блоку (**Q**) ограничивается, т.е. этот блок памяти становиться доступен только для чтения, а все операция изменения блокируются. Блокирование операций записи выражается в том, что транзакции, пытающиеся изменить заблокированный блок, переводятся в режим ожидания, а как только блокировка будет снята (т.е. закончится блокирующая транзакция) приостановленная транзакция будет продолжена. Таким образом ситуация, показанная на рис. 2.16. становится невозможной и блок **Q2** не будет создан.

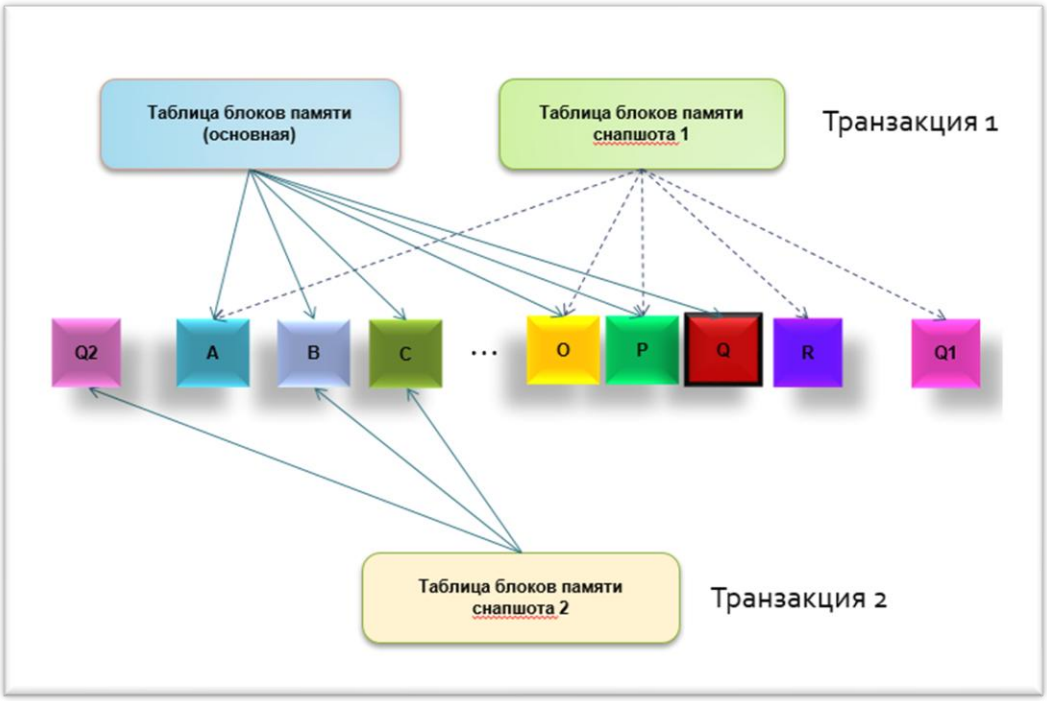

Рис. 2.16. Изменение одного блока двумя транзакциями

В реальной работе СУБД, когда одновременно обрабатываются тысячи транзакций, блокируются сотни блоков и несколько десятков таких блокировок приводят к приостановке выполнения транзакций. Одна из таких блокировок и представлена на рис. 2.17. Обычно эти блокировки живут недолго и быстро разрешаются.

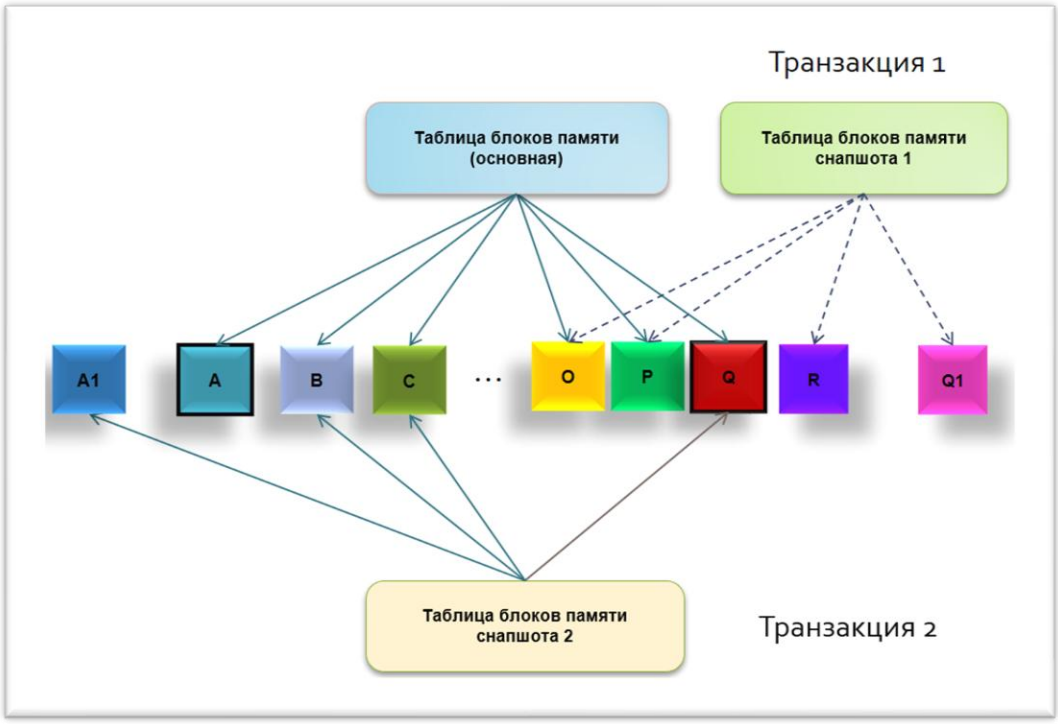

Рис. 2.17. Блокировка транзакции

Однако, бывают ситуации, когда разрешение блокировок не такое уж и простое дело. Одна из таких ситуаций представлена на рис. 2.18.: транзакция 2 ожидает освобождения блока **Q**, а транзакция 1, в свою очередь запросила для изменений блок **C** и была приостановлена до

завершения транзакции 2. Таким образом получилось, что транзакции ждут друг друга и эта ситуация кажется неразрешимой. На самом деле решение простое – СУБД принудительно завершает транзакцию 2, все заблокированные ей блоки освобождаются и, после окончания работы транзакции 1, СУБД снова запускает транзакцию 2.

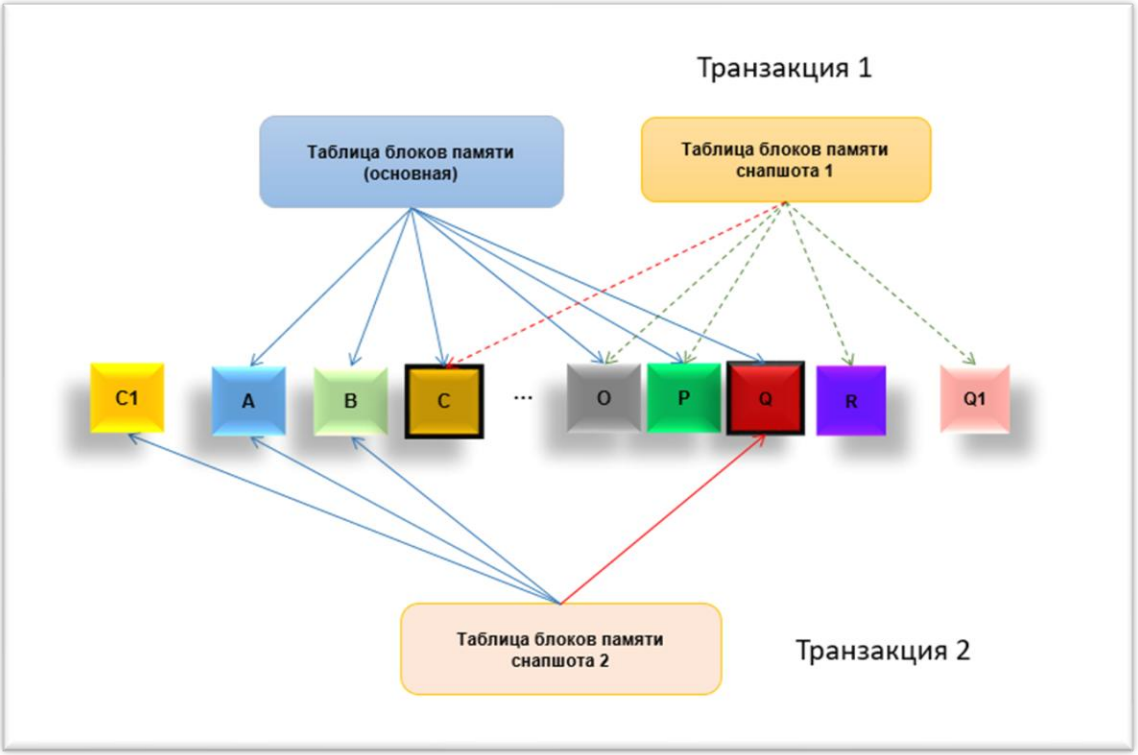

Рис. 2.18. Клинч транзакций

Гораздо сложнее случай «кольцевых» блокировок, показанный на рис. 2.19. Здесь не просто взаимная блокировка двух транзакций, здесь множественные блокировки образуют цепочку транзакций, которая в конце концов замыкается сама на себя. В замкнутом кольце все транзакции переведены в режим ожидания и разорвать этот «порочный» круг можно только принудительно завершив одну из транзакций. Если этого не сделать, то у каждой из ожидающих транзакций возникают свои дополнительные цепочки, замыкающиеся друг на друга, что в конечном счете приведет к коллапсу системы. СУБД достаточно просто находит подобные кольца и завершает одну из транзакций, кольцо распадается и блокировки разрешаются. Остается один вопрос - какую транзакцию завершать: самую «старую», работающую дольше всего или самую «молодую», только что начавшую работу?

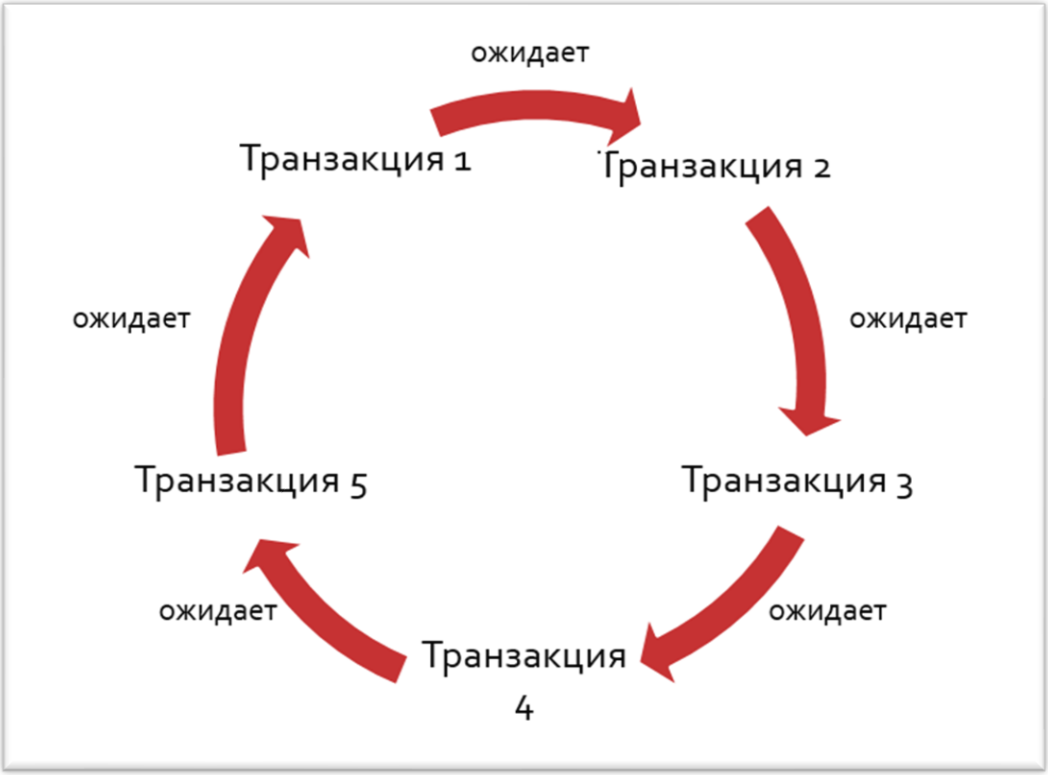

Рис. 2.19. Множественные клинчи

И та и другая политика имеет право на жизнь: завершив самую «молодую» транзакцию, мы дадим доработать транзакциям, уже затратившим большое количество процессорного времени и их не придется запускать заново, с другой стороны может быть стоит завершить самую «старую» транзакцию, ведь не известно сколько она будет всех держать, может быть, она и вовсе зациклилась. СУБД может выполнять любую политику разрешения клинчей, а вот выбирает эту политику администратор СУБД, на остове анализа статистики выполнения транзакций и типов выполняемых в СУБД запросов.

2.2.7. Последовательное выполнение транзакций

Исходя из свойства атомарности, можно сделать вывод, что все транзакции в СУБД выполняются последовательно и в простейших системах типа MS ACCESS все происходит именно так, но в больших СУБД управляющих огромными многопроцессорными комплексами подобная организация работы приведет к чудовищным простоям оборудования. В таких системах, конечно, транзакции выполняются параллельно, но в силу свойства изолированности, каждая транзакция не видит изменений данных, происходящих внутри других транзакций, а все результаты работы транзакции вступают в силу одномоментно, когда транзакция завершается, таким образом пользователь СУБД видит только последовательное выполнение транзакций как показано на рис. 2.20.

На нижней оси показана последовательность выполнения транзакций, обозначенных на осях выше. И хотя многие транзакции выполняются параллельно, общая картина соответствует последовательному выполнению транзакций. Первой заканчивается транзакция 2 и ее изменения вступают в силу. Транзакция 4 использует те же блоки, что и транзакция 1, поэтому в момент конкуренции за блоки памяти транзакция 4 приостанавливается и ждет завершения транзакции 1. После завершения первой транзакции (см. нижнюю ось) работа транзакции 4 возобновляется и успешно завершается через некоторое время. Самая нижняя транзакция – транзакция чтения не может ни с кем конфликтовать, поскольку не изменяет блоки памяти БД, поэтому она работает независимо от остальных транзакций. Обратите внимания, что снапшот транзакции чтения создается раньше начала выполнения всех других транзакций, а значит ее результаты не будут

учитывать изменения, внесенные транзакциями работающими и заканчивающимися во время её выполнения.

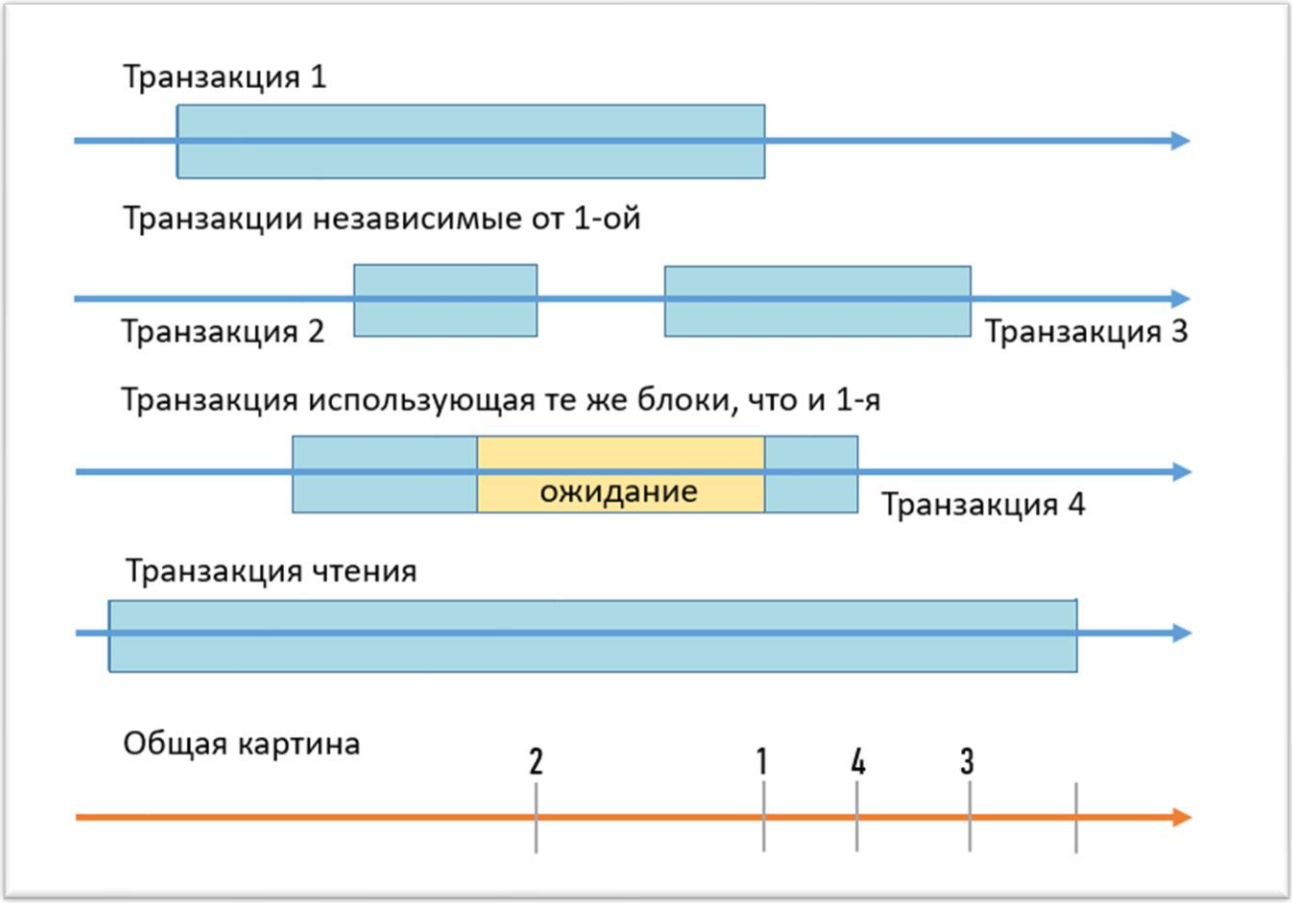

Рис. 2.20. Последовательное выполнение транзакций

2.2.8. Уровни изоляции транзакций

Реализация полной изоляции, описанной в предыдущих разделах — технически сложная задача, сопряженная с уменьшением производительности системы. Поэтому на практике почти всегда применяется ослабленная изоляция, которая предотвращает некоторые, но не все аномалии одновременного выполнения. Поэтому очень важно понимать, какой уровень изоляции используется в системе, какие гарантии он дает, а какие — нет, и как в таких условиях писать корректный код [30].

Уровни изоляции транзакций тесно связаны с аномалиями совместного исполнения, которые возможны на том или другом уровне изоляции. Существуют следующие аномалии, определенные в стандарте SQL:

Грязное чтение

Возникает, когда в транзакции читаются данные, измененные, но не зафиксированные другой транзакцией. В этом случае могут быть внесены изменения в записи, которые не были (а возможно и не будут) зафиксированы.

Неповторяющееся чтение

Возникает, когда в транзакции читается запись, модифицируемая из другой транзакции. Если первая транзакция попытается вновь прочитать запись, то запись в первоначальном виде не будет найдена.

Фантомные записи

Возникают в том случае, когда одна транзакция пытается прочитать запись, которая еще не существует в начале данной транзакции, но вставляется второй транзакцией,

прежде чем закончится первая транзакция. Если первая транзакция снова выполнит поиск этой строки, то обнаружит ее неожиданное появление. Эта новая строка называется фантомной записью.

Таблица 2.3 показывает влияние уровня изоляции транзакции на возможность возникновения аномалий.

Таблица 2.3. Уровни изоляции и аномалии

Уровень изоляции	Грязное чтение	Неповторяющееся чтение	Фантомные записи
READ UNCOMMITED	Возможно	Возможно	Возможно
READ COMMITED	Невозможно	Возможно	Возможно
REPEATABLE READ	Невозможно	Невозможно	Возможно
SERIALIZABLE	Невозможно	Невозможно	Невозможно

Надо иметь в виду, что в конкретных реализациях СУБД, часто реализуются не все уровни изоляции или реализуются с отклонениями и особенностями.

Внимательно ознакомьтесь с документацией на СУБД прежде, чем писать сложные транзакции или менять уровень изолированности транзакций.

2.2.9. Управление журналами изменений БД

Обеспечение надежности и безопасности хранения данных всегда требует дополнительных затрат как памяти, так и быстродействия СУБД, а любое восстановление после сбоя требует дополнительной информации. Такая избыточная информация хранится в журналах изменений, т.е. при любых операциях, совершаемых СУБД, сведения об изменениях данных заносятся в специальные журналы изменений. Все изменения в реляционной БД происходят в рамках транзакций, поэтому **журнал изменений** часто называют **журналом транзакций**.

Журнализация транзакций - функция СУБД, суть которой в сохранении информации, необходимой для восстановления базы данных в согласованное состояние в случае логических или физических сбоев.

В простейшем случае журнализация транзакций заключается в последовательной записи во внешнюю память всех изменений, выполняемых в базе данных. Записывается следующая информация:

- индикаторы начала транзакции (порядковый номер, тип и время изменения);
- пользователь, от имени которого проводились изменения;
- код транзакции на языке SQL;
- имя объекта, подвергшегося изменению (например, прикладной таблицы);
- предыдущее состояние объекта и новое состояние объекта;
- индикаторы фиксации, показывающие, была ли завершена транзакция, и, если да, то когда.

На самом деле СУБД поддерживает несколько журналов, так СУБД ORACLE использует оперативный журнал повтора (*redo log*), архивный журнал повтора (*archive log*), и оперативный журнал отката (*undo log*).

Изначально содержимое журнала повтора хранится в области памяти, называемой буфером журнала повторного выполнения (*redo log buffer*), но оттуда оно очень быстро сбрасывается на диск и такие сохраненные журналы называются архивными журналами повторного выполнения (*archived logs*). Онлайновый журнал повторного выполнения (redo logs) — это средство ORACLE, благодаря которому гарантируется, что все изменения, проведенные пользователями, не будут потеряны в случае, если произойдет сбой между моментом проведения изменений и моментом записи их в постоянное хранилище. Итак, фактически, система ORACLE ведет два журнала – один для отката транзакции (*undo log*) в случае, например, выполнения

команды ROLLBACK, а другой для повторного выполнения транзакции (*redo log*), например, в случае восстановления БД после сбоя (см. раздел 2.2.9).

Когда сервер БД терпит крах, все журналы изменений активно используются для восстановления состояния сервера СУБД и самой БД.

Функция журнализации транзакций позволяют решать следующие задачи:

- восстановление данных после сбоев (см. раздел 2.2.3);
- репликация данных (см. раздел 2.2.4);
- анализ работы СУБД и действий пользователей;

На основании информации, хранящийся в журналах изменений СУБД, можно проанализировать загрузку СУБД и работу пользователей по времени и по видам операций, востребованность различных данных, циклы наибольшей загрузки СУБД и т.д. Полученная таким образом информация позволяет оптимизировать работу СУБД, выработать методические указания для пользователей, ускорить работу отдельных операторов SQL, представлений и встроенных процедур.

2.2.10. Обеспечение безопасности БД

Безопасность БД (англ. *Database security*) - широкий спектр средств защиты информации включая данные, приложения баз данных и хранимые функции, системы управления базами данных, серверы баз данных. Это понятие включает в себя различные типы и категории контроля - технические, процедурные, административные и физические, которые противостоят компрометации и поддерживают конфиденциальность, согласованность и доступность информации, содержащейся в БД.

Здесь будут обсуждаться только некоторые аспекты защиты безопасности, относящиеся к функциям и задачам СУБД:

Защита от сбоев

Защита от злонамеренных действий

Защита от сбоев

Сбой — утрата системой работоспособности.

Сбои в работе СУБД, вызывает неправильно написанное программное обеспечение (программные ошибки), неверные инструкции, переданные оператором (ошибки пользователя), отказ аппаратуры компьютера (аппаратные ошибки), внешние воздействия на аппаратный комплекс (катастрофы).

Мягкий сбой

Мягкий сбой системы обычно возникает при аварийном выключении электрического питания, при возникновении сбоя аппаратной части сервера, не затрагивающего устройства долговременного хранения. При таком сбое данные, хранящиеся на дисках, остаются неповрежденными, но утрачивается содержимое буферов базы данных в оперативной памяти.

Для восстановления состояния оперативной памяти используются данные «точки согласованности». В журнале транзакций отмечаются точки физической согласованности базы данных, так называемые *точки согласованности* – моменты времени, когда в долговременной памяти содержатся данные, точно соответствующие состоянию оперативной памяти, и при этом в долговременную память полностью вытолкнут буфер журнала транзакций. По сути «точка согласованности» — это дамп памяти сервера вместе с некоторой служебной информацией. В момент точки согласованности содержимое ОЗУ согласованно с содержимым долговременной памяти (не путайте с согласованностью БД). Сервер ORACLE обычно делает точки согласованности не реже, чем раз в две секунды.

На рис. 2.22 представлена ситуация мягкого сбоя: на оси времени отмечены собой, в результате которого утеряно содержимое оперативной памяти и момент создания ближайшей к

сбою точки согласованности. Возврат состояния ОЗУ к точке согласованности – первое действие, которое осуществляет СУБД после мягкого сбоя, т.е. в очищенную память «заливается» содержимое точки согласованности, сохраненное в журнале изменений СУБД.

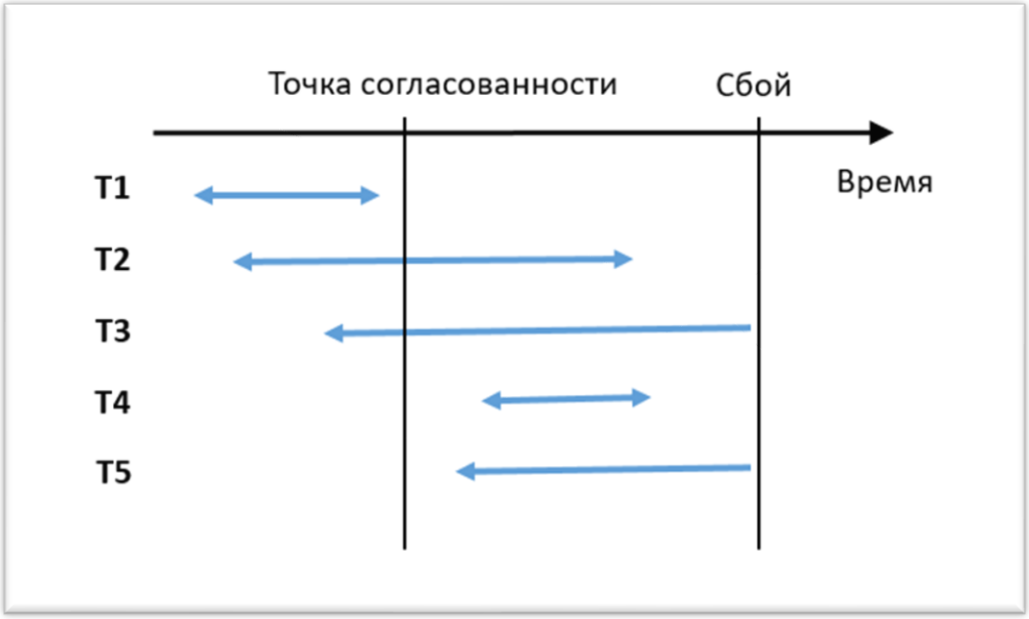

Рис. 2.22. Восстановление СУБД после мягкого сбоя

Отметим, что после восстановления ОЗУ в памяти находится информация о промежуточных результатах незавершенных транзакций. После восстановления ОЗУ СУБД анализирует журнал транзакций и проделывает следующие операции:

1. Откатываются все транзакции, которые начались до точки согласованности и окончились после нее – как **T2** или вовсе не закончились, как **T3**. Это действие выполняется по журналу *undo log* и очищает ОЗУ от промежуточных результатов незавершенных транзакций.
2. Накатываются все транзакции, которые завершились после точки согласованности – такие как **T2** и **T4**. Это действие выполняется по журналу *redo log*.
3. Запускаются на выполнение транзакции, которые выполнялись в момент сбоя, такие как **T3** и **T5**, т.е. не имеющих отметок о завершении.
4. Для транзакции типа **T1**, которые закончилась до момента «точки согласованности» никаких действий производить не требуется, их результаты уже надежно сохранены в базе данных.

После выполнения этих действий СУБД полностью восстанавливается и может продолжать работу.

Жесткий сбой

Жесткий сбой системы возникает из-за разрушения данных, записанных на устройства долговременного хранения. Отказ систем хранения данных в настоящее время это очень редкое событие, гораздо чаще возникает ситуация, когда некомпетентные или неосторожные действия пользователя приводят к утрате данных. Процесс восстановления БД после жесткого сбоя показан на рис. 2.23.

Предположим, что жесткий сбой СУБД произошёл в момент времени **t1** и обнаружен был не сразу, а в момент **t2,** когда пользователи сообщили администратору СУБД о невозможности работы. После анализа ситуации администратор останавливает СУБД в момент **t3** и приступает к устранению результатов сбоя.

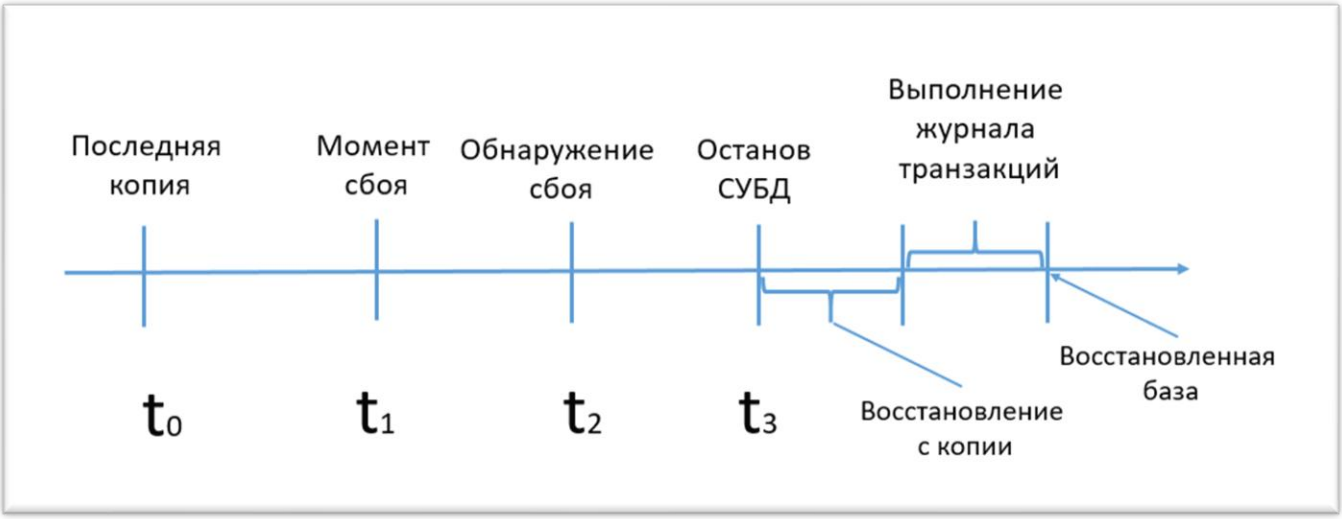

Рис. 2.23. Процесс восстановления БД после жесткого сбоя

Теперь надо восстановить состоянии БД с копии, сделанной в момент **t0** (о копировании БД см. раздел 2.2.4). Восстановление с резервной копии может быть проведено средствами операционной системы или средствами СУБД, запущенной уже в однопользовательском режиме. Остается выполнить все транзакции, записанные в журнал *archived log* с момента **t0** до момента **t3**. Если к сбою привела ошибка пользователя, то перед выполнением журнала необходимо убрать оттуда транзакцию, которая привела к уничтожению или искажению данных. Выполнение (или как говорят «накат») журнала может занять очень большое время, если последняя резервная копия создана давно, поэтому политика резервного копирования должна учитывать сроки возможного восстановления БД после жесткого сбоя.

Катастрофа

Под катастрофой мы будем здесь понимать крупное неблагоприятное событие (авария, стихийное бедствие, пожар и др.), влекущее за собой полный отказ узла сети, без возможности его быстрого восстановления. Надо понимать, что обычно крупные узлы больших систем размещаются на специальных объектах, называемых ЦОД (Центр Обработки Данных). ЦОДы (как и сети ЦОДов) создаются таким образом, чтобы минимизировать риски от катастроф. Единственным выходом при потере ЦОДа в результате катастрофы, являет переход на резервный ЦОД, который должен содержать реплики серверов вышедшего из строя ЦОДа (про репликацию см. раздел 2.2.11). Создание сетей ЦОДов — это прежде всего инженерно-строительная задача и ее рассмотрение выходит за границы данной книги. Более подробно см. [23].

Злонамеренные угрозы и борьба с ними

Все вышеперечисленные угрозы работы БД являются незлонамеренными, т.е. не предполагающие наличия злоумышленника. Однако основные угрозы безопасности исходят от людей.

Запомните: взламывают не системы, а людей! Практика показывает, что 90 процентов случаев, когда возникают уязвимости и куда-то утекает информация, связаны с ошибками разработчиков и сопровожденцев или наличием сообщника злоумышленников в обслуживающем персонале системы. Борьба с подобными угрозами лежит в плоскости неформальных средств защиты информации – это нормативные (законодательные), административные (организационные) и морально-этические средства рассмотрение которых выходит за границы данной книги. Более подробно см. [23].

Простые правила обеспечения безопасности БД

Про обеспечение безопасности написано немало статей, книг, методичек и стандартов, в рамках же этой книги можно обойтись несколькими простыми, базовыми правилами, обязательное следование которым позволит устранить большую часть угроз безопасности ИС.

- Настройка поддерживаемых систем

  Тщательная настройка всех модулей системы позволяет убрать все ненужные функции ПО, что сужает количество уязвимостей. Злоумышленники чаще всего предполагают, что настройки системы – стандартные, поэтому тщательная настройка системы устраняет многие угрозы.
- Обновление поддерживаемых систем

  Постоянное обновление систем, поддержка их в актуальном состоянии позволяет устранить ряд угроз, связанных с известными вирусными атаками и избежать проявления ошибок, выявленных и исправленных в последних версиях ПО.
- Обучение персонала (пользователей)

  Персонал, работающий в системе, должен быть обучен быстро и грамотно реагировать на нештатные ситуации, сбои и попытки проникновения в систему. Это позволит сократить время реакции администраторов на сбои и не даст многим мелким инцидентам развиться в угрожающие ситуации.
- Внедрение морально-этических средств

  К морально-этическим средствам можно отнести сложившиеся в обществе или данном коллективе моральные нормы или этические правила, соблюдение которых способствует защите информации, а нарушение их приравнивается к несоблюдению правил поведения. Эти нормы не являются обязательными, однако, их несоблюдение должно вести к падению авторитета, престижа человека в коллективе.
- Распределение ролей и ограничение доступа к данным

  Разграничение доступа персонала к данным позволяет свести к минимуму неумышленную порчу данных, а также злоумышленные попытки доступа к закрытой информации.
- Создание программ проверки целостности

  Постоянные проверки целостности, непротиворечивости и достоверности информации обеспечивают своевременное выявление нештатных ситуаций и не дают разиться им в искажение или потерю данных. Для подобных проверок необходимо разрабатывать специальные программные средства.
- Создание всеохватывающей системы аудита

  Сохранение и анализ всех действий персонала дает возможность понять причины сбоев и искажения данных.

2.2.11. Копирование данных и резервные копии БД

Современные СУБД – весьма надёжные программные комплексы, однако изредка всё же происходит повреждение внутренних структур базы данных, после которого доступ к данным пропадает. Что особенно обидно, такое нарушение происходит обычно при высокой нагрузке или при установке какого-нибудь обновления. Но как высокая нагрузка, так и регулярные обновления говорят о том, что база данных – отнюдь не тестовая, и данные, хранящиеся в ней, ценны. В этих редких, но очень болезненных случаях на помощь приходит резервная копия, с помощью которой можно восстановить «испорченные» данные. Кроме аппаратных сбоев, резервная копия позволяет восстановить данные после логической ошибки. Например, бухгалтер удалил группу проводок или администратор БД уничтожил табличное пространство: обе операции абсолютно легитимны с точки зрения базы данных, и будут выполнены СУБД. Наконец, еще одна задача, решение которой требует наличия резервной копии, – это клонирование базы, например, для целей тестирования.

Создание копии БД может происходить различными способами:

- Средствами операционной системы делается копия файлов СУБД, при этом на время копирования СУБД надо остановить, иначе можно получить несогласованную копию.
- Копирование устройств. Создается копия устройства, на котором хранится БД, аппаратными средствами системы хранения данных (СХД).
- Копирование специальными программами. Создается копия базы данных средствами СУБД, например, создается снапшот всей БД и с него производится копирование, таким образом мы получаем согласованную копию БД.

Первые два способа используют средства внешние по отношению к СУБД и не описываются в данной книге, то третий способ относится как раз к функциям СУБД, рассматриваемым в этой главе.

Резервное копирование баз данных так или иначе базируется на одном из двух принципов:

- Выборка данных с последующим сохранением в произвольном формате;
- Снимок состояния БД и сохранение журналов.

Первый принцип базируется на утилитах выгрузки и загрузки данных БД. Данные сохраняются этими утилитами в файлах либо в текстовом, либо в двоичном формате, специфичном для конкретной СУБД, затем утилиты загрузки достают сохраненные данные и загружают их в пустую БД, создавая, таким образом, точную копию основной БД.

Текстовый формат хорош тем, что его можно редактировать или даже создавать внешними программами, а двоичный в свою очередь хорош тем, что позволяет быстрее выгружать и загружать данные за счёт распараллеливания загрузки и экономии ресурсов на преобразовании форматов.

Несмотря на простоту и очевидность идеи выгрузки данных, для резервирования нагруженных промышленных баз такой метод применяют редко. Ниже приведены некоторые причины, по которым выгрузка не подходит для полноценного резервного копирования:

- процесс выгрузки создаёт значительную нагрузку на систему-источник;
- выгрузка занимает много времени – к моменту окончания выгрузки она станет уже неактуальной;
- сделать согласованную выгрузку всей базы данных при высокой нагрузке практически невозможно, если только не остановить работу БД;
- выгрузка сохраняет логическую структуру данных, но не сохраняет их физическую структуру – параметры физического хранения таблиц, индексы и другие объекты БД; так восстановление индексов при загрузке может занять значительное время.

Есть, правда, у этого способа копирования неоспоримое преимущество - выгруженные данные можно загрузить в базу данных другой версии, а если выгрузка сделана в текстовом формате, то и в другую СУБД.

Для того, чтобы не останавливать работу БД на время копирования и получить согласованную копию используется механизм «снимков состояния БД и сохранение журналов».

Работа по сохранению копии начинается с запоминания момента начала резервного копирования (резервная копия должна будет содержать журналы базы данных начиная с этого момента). Далее выполняется точка согласованности, то есть все изменения, которые произошли в блоках данных на оперативном запоминающем устройстве сбрасываются на диск (это гарантирует, что журналы до момента начала резервного копирования при восстановлении не потребуются). Затем происходит получение быстрой согласованной копии, которая и будет сохраняться… здесь уместно применение таких средств как моментальный снимок (snapshot) всей БД или разрыв зеркала (BCV) в дисковом массиве СХД. Одни СУБД (ORACLE, PostgreSQL) оставляют администратору возможность самостоятельно выбрать способ копирования, другие (Microsoft SQL Server) предоставляют интерфейс для интеграции собственных утилит резервного копирования с механизмами файловых систем или СХД.

После создания моментального снимка БД (программным или аппаратным способом) можно копировать файлы БД средствами операционной системы – cp, rsync и другими. Во время этого копирования, которое может занять достаточно продолжительное время, БД продолжает работу, а все транзакции с момента начала копирования сохраняются в специальных журналах изменений. Процесс копирования представлен на рис. 2.24.

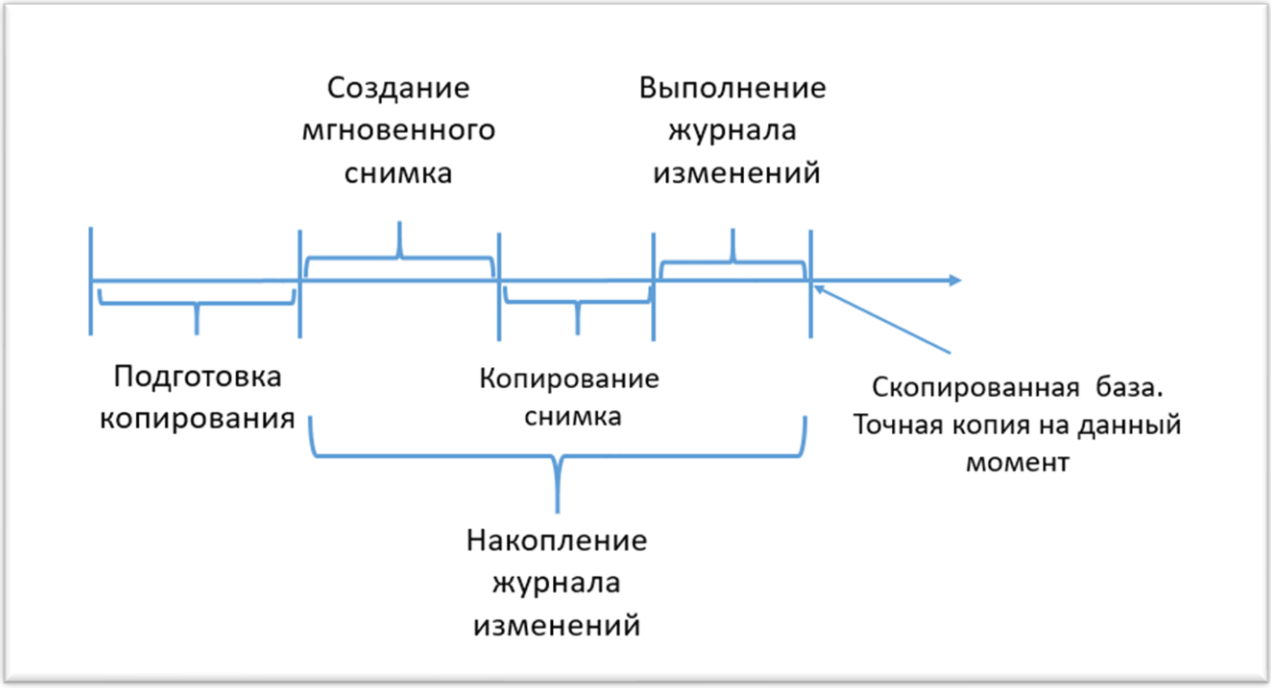

Рис. 2.24. Процесс копирования БД

Завершается копирование накатом на копии журнала изменений, накопленных с момента начала копирования до момента его окончания. Политика копирования БД обычно составляется администратором СУБД и проводится в рамках сопровождения (см. главу 5)

2.2.12. Репликация данных

Репликация (replication) — это процесс поддержание в синхронном состоянии копий одних и тех же данных на нескольких машинах (узлах, нодах), соединенных с помощью сети. Существует несколько причин репликации данных:

- для горизонтального масштабирования количества серверов, обслуживающих запросы на чтение (и повышения пропускной способности системы по чтению);
- часто репликация необходима для того, чтобы система могла продолжать работать при отказе некоторых ее частей (и повышения, таким образом, надежности и доступности) т.е. при отказе основного узла одна из реплик берет на себя его функции;
- ради хранения данных географически близко к пользователям (и сокращения, таким образом, трафика сети и задержек).

Механизмы репликации строятся на использовании журналов изменения БД. Поначалу журналы изменений служили обеспечению устойчивости, защиты от сбоев аппаратуры, но постепенно журналы изменений, а потом и журналы транзакций эволюционировали до механизма, используемого при репликации данных с основного (мастер) сервера на серверы реплики, в целях обеспечения доступности и масштабируемости БД. В какой-то момент были созданы полномасштабные сервисы для использования журналов с целью переноса данных между различными БД при наличии возможности преобразования данных между ними. Затем журналы превратились в полноценную систему обмена сообщениями, где журнальные записи играют роль событий, которые сервис издатель может использовать для выполнения различных

работ на серверах, подписанных на получение всей или части подписки издателя. Обратите внимание на терминологию в предыдущем предложении, она пришла в IT из издательского дела и успешно применяется для описания механизмов репликации и распространения изменений.

Суть репликации в том, что изменения данных, происходящие на мастере, повторяются на репликах (но не наоборот). Поэтому запросы на изменение данных (INSERT, UPDATE, DELETE и т. д.) выполняются только на мастере, а запросы на чтение данных (проще говоря, SELECT) могут выполняться как на репликах, так и на мастере. Процесс репликации на одной из реплик не влияет на работу других реплик, и практически не влияет на работу мастера.

Существует несколько подходов к реализации репликации. У каждого из них есть свои плюсы и минусы, и во многих СУБД можно из нескольких типов репликации выбрать тот, какой подходит вам лучше всего. Давайте рассмотрим эти типы репликации.

Поблочная репликация. Журнал упреждающей записи (Write-Ahead Log), также известный как журнал повтора (Redo Log), содержит сведения о серии событий, каждое из которых сопоставляется с транзакцией или записью. В журнале перечислены все блоки, измененные на дисках в результате такого события. В таких системах, как PostgreSQL, где используется этот метод, журнал изменений отправляется непосредственно всем приложениям-получателям, которые и выполняют необходимые изменения на дисках реплик.

Построчная репликация. При построчной репликации операции записи заносятся в журналы репликации ведущего узла как события, указывающие на то, как изменяются отдельные записи таблицы. Указываются поля с новыми данными; данные приводятся как до, так и после изменений, а также отмечаются удаленные записи. В репликах эти данные применяются не для выполнения исходного оператора, а для непосредственного изменения записей.

Логическая репликация (репликация транзакций). При логической репликации каждая транзакция (атомарный блок SQL-операторов), выполняющая изменения в БД, записывается и отправляется от ведущего узла к подписчикам. Это означает, что на каждом из подписчиков будет выполняться весь набор операторов транзакции. При таком способе передачи изменений значительно уменьшается нагрузка на сеть, ведь не надо передавать все измененные блоки данных – один запрос (несколько килобайт) может вызвать изменения гигабайтов данных. Кроме того, есть некоторые другие интересные возможности. Поскольку на реплики передаются не сами данные, а запросы, вызывающие их изменения, мы можем использовать различную структуру таблиц на мастере и репликах. В частности, может отличаться тип таблицы, сегментирование, кластеризация или набор индексов.

Репликация моментальных снимков. При использовании репликации снимков периодически создается снимок (снапшот) базы данных, который распространяется подписчикам. Хотя этот вид репликации все чаще упоминается в литературе, по сути, он репликацией не является: на самом деле это просто периодическое автоматическое копирование БД (см. 2.2.10).

Независимо от выбора вышеперечисленных способов синхронизации мастера и реплик, собственно «репликация» состоит из двух этапов:

1. Создание первоначальной копии;
2. Распространение и применение изменений.

Создание первичной копии может происходить различными методами, которые описаны в разделе 2.2.10. Работа СУБД на этапе распространения и применения изменений показана на рис. 2.26.

Рис. 2.26. Процесс репликации БД

На основном сервере формируются журналы изменения данных, которые передаются на репликационный сервер и распространяются на сервера реплики. На серверах репликах производятся действия по применению журналов для изменения информации точно так же, как и на основном сервере. Обратите внимание этот механизм работает независимо от типа репликации. Все чаще в литературе используется терминология, принятая в СУБД «MS SQL SERVER» – издатель, дистрибьютор, подписчик (см. рис. 2.26).

Информация об изменениях передается в одну сторону поэтому на серверах репликах возможны только запросы на чтение. Если позволить пользователям изменять данные на репликах, и эти изменения попытаться передать на остального сервера, в системе возникнут многочисленные конфликты изменения данных.

Учитывая, что можно реплицировать не только БД целиком, но и отдельные таблицы (а иногда и части таблиц), возможно создание разнообразных схем репликации, позволяющих проводить как консолидацию, так и разделение информации по узлам сети. За передачу на реплики информации, предназначенной только для них, отвечает репликационный сервер, настроенный на разделение или слияние потоков журналов изменений.

Один из таких сложных вариантов репликации — это «консолидация», показанная на рис 2.27. Все изменения, происходящие в БД структурных подразделение корпорации, в виде журналов поступают на репликационный сервер, где происходит слияние потоков. Далее все журналы попадают на «Сервер хранилища», где выполнятся (здесь применяется «логическая репликация»). Структуры всех БД структурных подразделений, как и схема «хранилища» одинаковы, но данные в них разные, при этом первичные ключи всех таблиц на различных серверах подразделений не пересекаются – каждая БД имеет свой разрешенный диапазон. После выполнения всех транзакций на «Сервере хранилища» мы получаем единую БД с информацией по всем структурным подразделениям.

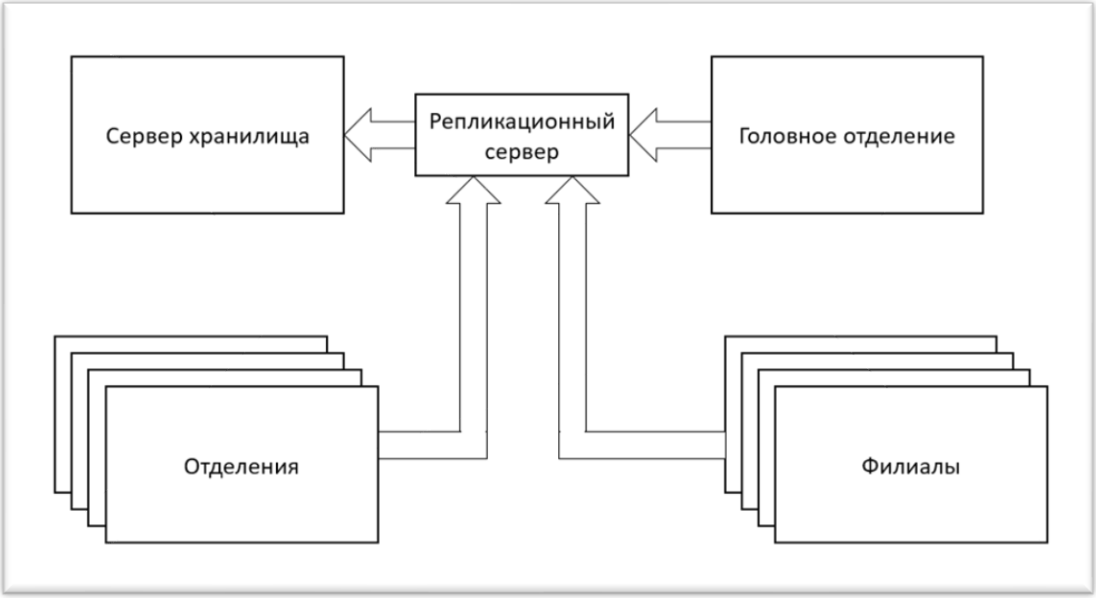

Рис. 2.27. Консолидация БД

Из такого хранилища удобно получать всю отчетность по корпорации в целом и аналитические отчеты для высшего руководства.

2.2.13. Ведение словаря БД

Каждая СУБД содержит набор таблиц, доступных обычному пользователю только для чтения, и известных как **словарь данных** (data dictionary), который содержит **метаданные** (информацию о различных компонентах базы данных). Словарь данных — сердце системы управления базой данных, и знание его необходимо администраторам, ведь здесь содержатся все настройки СУБД. Словарь данных – это первое, что создается при инсталляции СУБД, и в этот момент он заполняется в соответствии с «заводскими настройками» - усредненными данными позволяющими работать СУБД с некоей «средней» БД.

По сути, «Словарь данных» — это БД описывающая предметную область «служебная информация СУБД». Эта БД - реляционная, т.е. словарь данных состоит из таблиц и связей между ними, и СУБД работает со словарем данных, как с обычной БД.

Таблицы словаря данных описывают все БД, которыми управляет СУБД: их логическую и физическую структуру, объекты с их ограничениями, а также информацию о пользователях, размеры и расположение табличных пространств, настройки СУБД и другую служебную информацию. Словарь данных реализует концепцию БД «хранение описания данных вместе с самими данными».

Структура таблиц словаря данных оптимизирована для ускорения работы ядра СУБД и для человека трудно читаема, поэтому пользователь работает с представлениями, показывающими информацию словаря в удобной для него форме, кроме того, использование представлений создает еще один защитный барьер от ошибочных изменений данных в словаре (подробнее о представлениях см. разделы 3.4.15 и 4.6.2).

На рисунке 2.28 представлен небольшой фрагмент схемы словаря ORACLE - это основная его часть, в которой содержится информациях об основных объектах БД: таблицах, индексах, ограничениях.

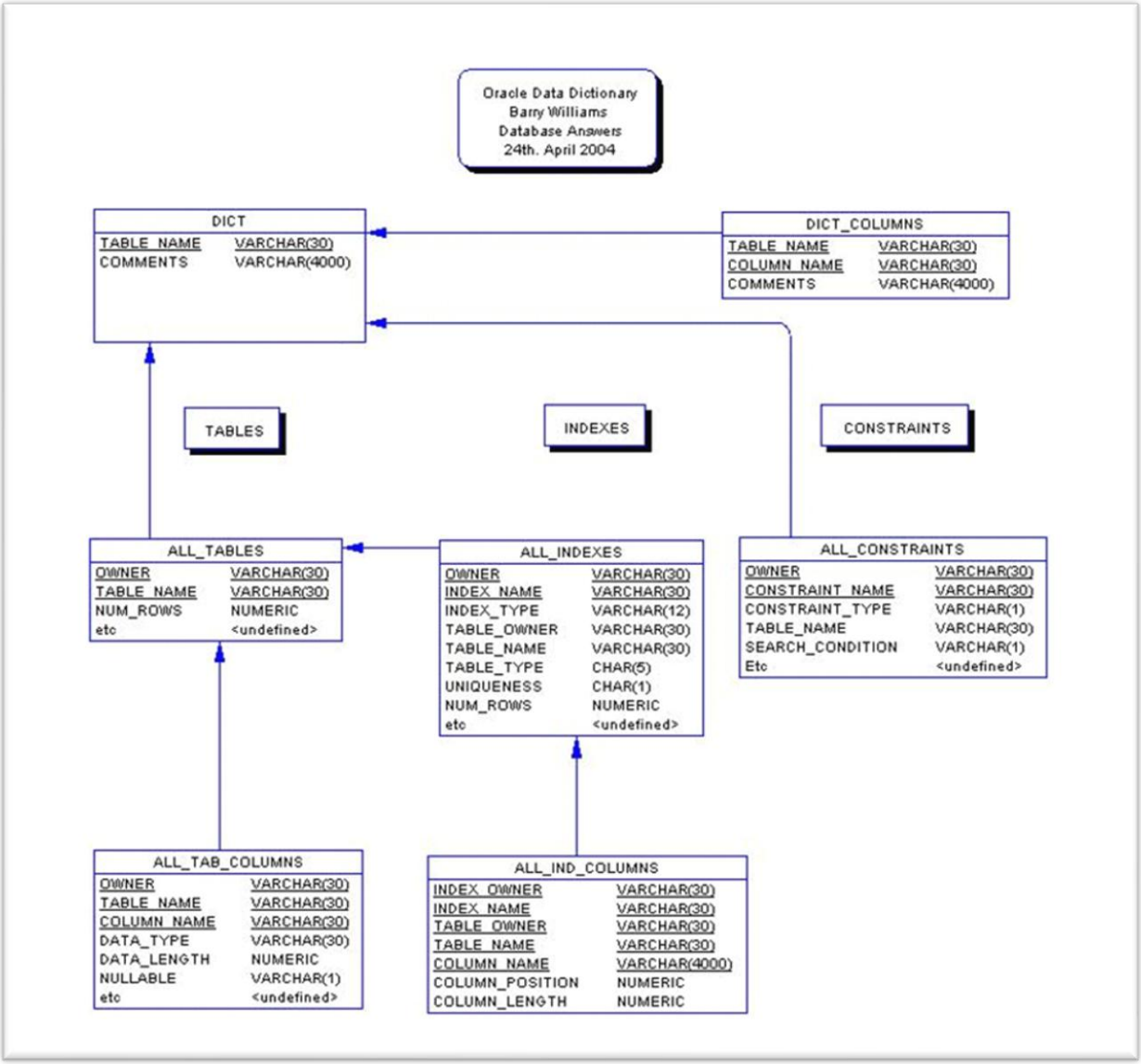

Рис. 2.28. Фрагмент схемы словаря данных ORACLE

«Словарь данных ORACLE — настоящие джунгли!

Он изобилует полезной информацией, но найти путь к ней порой бывает очень непросто. В нем сотни представлений, основанных на сотнях таблиц, множество сложных взаимосвязей и специальных кодов» [17].

2.3. Развитие реляционных СУБД

В 1974 году компания IBM начала исследовательский проект по разработке РСУБД, получивший название System R. Первым коммерческий продуктом IBM была реляционная СУБД (РСУБД) SQL/DS, выпущенная в 1982 году. Еже один научно-исследовательский проект начался Калифорнийском университете в Беркли в 70-х годах. Закончился этот проект в начале 1980-х созданием РСУБД Ingres работающей под управлением ОС UNIX. Исходный код, как и для других проектов того же происхождения, был доступен за минимальную плату под лицензией BSD.

Первой коммерчески успешной РСУБД стала ORACLE, выпущенная в 1979 году компанией Relational Software, которая впоследствии была переименована в ORACLE Corporation, чуть позже была выпущена первая коммерческая СУБД от IBM – DB2.

В 1970-е годы, когда уже были получены почти все основные теоретические результаты и даже существовали первые прототипы реляционных СУБД, многие авторитетные специалисты отрицали возможность добиться эффективной реализации таких систем. Однако преимущества реляционного подхода и развитие методов и алгоритмов организации и управления реляционными базами данных привели к тому, что к концу 1980-х годов реляционные системы заняли на мировом рынке СУБД доминирующее положение.

В связи с резким ростом популярности РСУБД в 1980-х годах многие компании стали позиционировать свои СУБД как «реляционные» в рекламных целях, иногда не имея для этого достаточных оснований, вследствие чего автор реляционной модели данных Эдгар Кодд в 1985 году опубликовал свои знаменитые «12 правил Кодда» (см. Приложение Б), которым должна удовлетворять каждая РСУБД. В настоящее время можно определить СУБД как реляционную, если она поддерживает язык SQL.

Одним из ключевых этапов в развитии СУБД стал 1986 год. К этому времени появилось ещё несколько компаний-разработчиков СУБД, одной из самых заметных из них стала компания Sybase, основанная двумя годами ранее. К 1986 году Sybase начала интегрировать интеллектуальные рабочие станции (как правило, разработки Sun Microsystems или Apollo Computer) с серверами базы данных (разработанных, например, ORACLE). При этом сама клиент-серверная технология сделала возможным отделение слоя обработки информации (т. н. back end) от интерфейсного слоя (т. н. front end). Учтя проникновение компьютерных сетей во все сферы человеческой деятельности, поставщики решений перешли к распределению и других задач (например, форматирование отчётов, проверка данных и т. д.) среди рабочих станций сети, оставив серверу выполнять лишь задачи, требующие централизованного решения - хранение и защита данных, оптимизация потока выполнения запросов и т. д.

К концу 1986 года использование языка SQL в качестве основного для работы с данными в СУБД стало практически повсеместным. IBM, ORACLE, Sybase и Gupta использовали схожий синтаксис языка SQL для отправки сообщений от клиентской части СУБД (front end) к серверной (back end), что позволяло сочетать клиентские и серверные части разных производителей. В том же году Американский национальный институт стандартов утвердил версию языка SQL в качестве международного стандарта обработки данных, что поставило под угрозу благополучие СУБД, не обладавших поддержкой языка SQL. Так, например, компания Cullinet хотя и анонсировала поддержку языка SQL в своих СУБД для мини-компьютеров, но из-за задержки в её реализации потеряла свою долю рынка СУБД, уступив IBM и её продукту DB2. В 1988 году в гонку лидеров вступила фирма Microsoft со своим детищем MS SQL Server. Сначала это была совместная разработка Microsoft и Sybase, но в апреле 1994 года пути компаний разошлись, а о совместном прошлом напоминает нам Transact-SQL – диалект языка SQL поддерживаемый в обеих СУБД.

Еще один некоммерческий проект Калифорнийского университета в Беркли - PostgreSQL. К разработке Postgres, начавшейся в 1986 году, имел непосредственное отношение Майкл Стоунбрейкер, руководитель более раннего проекта Ingres, на тот момент уже приобретённого компанией Computer Associates. Название расшифровывалось как «**Post** In**gres**», и при создании Postgres были применены многие уже ранее сделанные наработки как Ingres, так и ORACLE.

Последней из крупных РСУБД была MY SQL, созданная в конце прошлого века. Эта СУБД разрабатывалась в недрах корпорации SUN вместе с легендарным языком JAWA, а в 2010 СУБД MY SQL перешла к ORACLE вместе с поглощенной SUN.

Дерево развития основных РСУБД приведено на рис. 2.29.

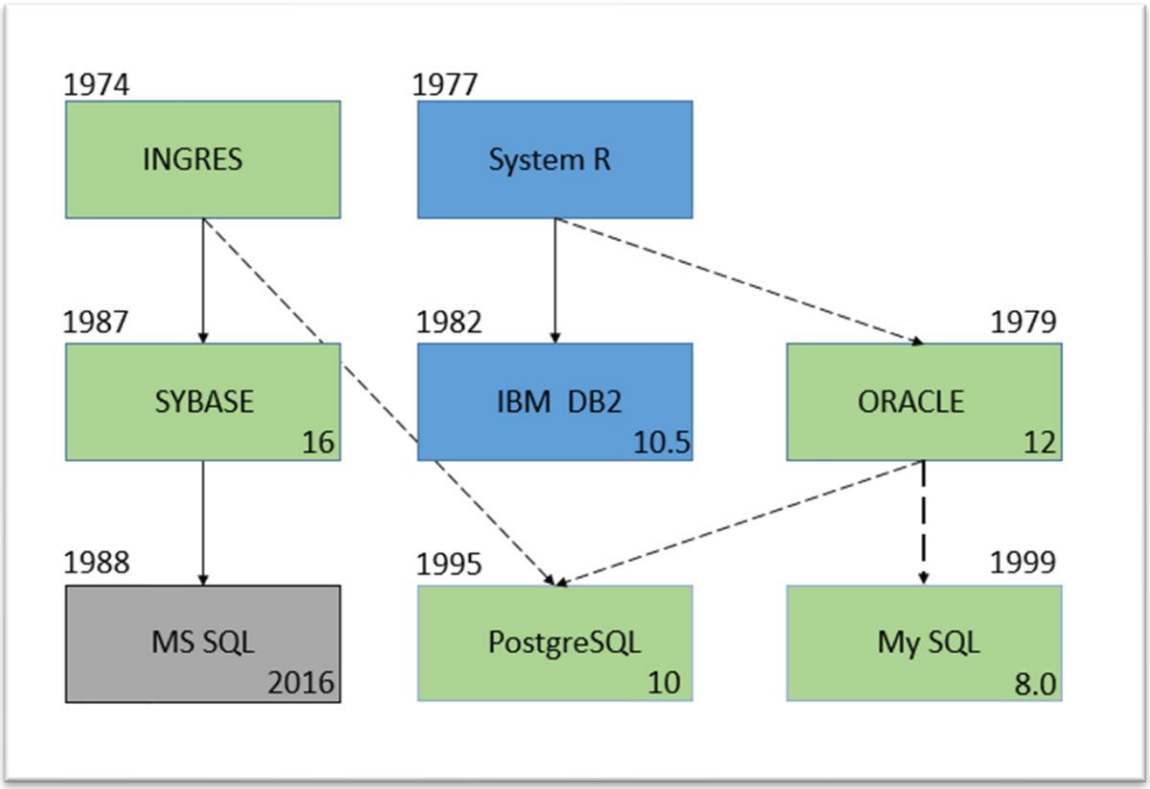

Рис. 2.29 Развитие реляционных баз данных

Реляционная модель баз данных и РСУБД гарантируют, что все операции манипулирования данными обладают ACID свойствами (см. раздел 2.2.4), с одной стороны это обеспечивает постоянную согласованность БД, с другой стороны мешает обеспечить высокую доступность и быстродействие особенно в случае, когда данные распределены по нескольким серверам. Для решения проблемы обеспечения высокого быстродействия при распределенном способе хранения информации была разработана специальная разновидность баз данных NoSQL (Not only SQL).

Нереляционная база данных — термин, обозначающий ряд подходов, направленных на реализацию хранилищ данных, в которых делается попытка решить проблемы быстродействия, масштабируемости и доступности данных за счет их атомарности и согласованности. Крупные корпорации, оперирующие гигантскими объемами данных, разработали несколько нереляционных СУБД и это мода была активно поддержана IT-сообществом. Вскоре, однако, выяснилось, что нереляционные СУБД поддерживают весьма узкий круг задач, связанных с накоплением, очисткой и выборкой малосвязанных данных. В настоящее время большинство существующих СУБД являются реляционными. Это ясно видно из диаграммы на рис. 2.30.

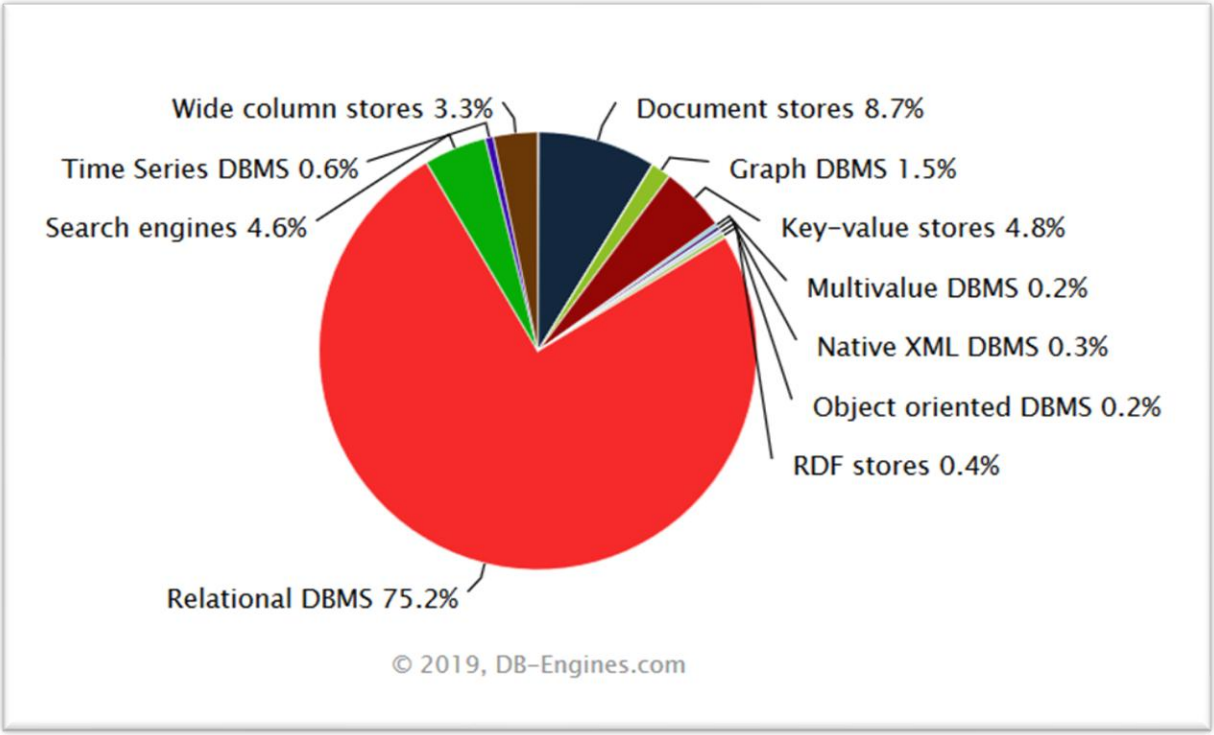

Рис. 2.30. Распределение БД по типам используемой модели данных

Реляционные СУБД, появившиеся в конце 70-х годов прошлого века, до сих пор удерживают лидирующие позиции при построении промышленных ИС.

3. Язык SQL

У языка SQL имеются два неоспоримых преимущества. Во-первых, почти за 50 лет существования языка к нему привыкли (и даже сроднились с ним) тысячи профессионалов в области баз данных. Во-вторых, (и это проверено многолетней практикой) язык SQL допускает эффективную и масштабируемую реализацию, и даже объектные расширения языка не вызывают какой-либо деградации производительности систем. Одним словом, нам предстоит еще долгая совместная жизнь с SQL, и, чтобы она была удачной, нужно хорошо знать и достоинства, и недостатки этого языка [4].

3.1. Становление языка

3.1.1. История возникновения

Язык SQL уже совсем не молод. В 2024 г. сообщество баз данных будет отмечать его 50-летний юбилей. Поэтому, чтобы правильно понимать и трактовать современные варианты SQL, нужно знать историю языка (хотя бы в общих чертах).

Язык SQL, предназначенный для взаимодействия с базами данных, появился в середине 70-х (первые публикации датируются 1974 г.) и был разработан в компании IBM в рамках проекта экспериментальной реляционной СУБД System R. Разработка языка началась в начале семидесятых, когда Рэймонд Бойс (англ. *Raymond Boyce*, 1947–1974) и Дональд Чемберлин (англ. *Donald D. Chamberlin*) (род. 1944), работая в IBM, создали язык запросов SEQUEL (тот самые “сиквел”), позднее переименованный в SQL и впервые стандартизированный в 1986 как “SQL-86”. Исходное название языка SEQUEL (Structured English Query Language) только частично отражало суть этого языка. Название SQL появилось несколько позже, когда выяснилось, что аббревиатура SEQUEL уже зарегистрирована как товарный знак сторонней компании. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционным БД. SQL задумывался как язык общения (взаимодействия) пользователя (непрограммиста) с базами данных. Идея такого языка сводилась к набору из нескольких фраз-примитивов английского языка («выбрать», «обновить», «вставить», «удалить»), через которые пользователь-непрограммист ставил бы «вопросы» к СУБД, реализуя свои информационные потребности. В этом случае дополнительной функцией СУБД должна быть интерпретация этих «вопросов» на низкоуровневый язык машинных кодов для непосредственной обработки данных и предоставление результатов пользователю. Так родилась идея-понятие «машины данных», то есть такой части СУБД, которая разделяет собственно данные и задачи по их обработке.

Несмотря на трудности реализации и на очень слабую, на первых порах, эффективность СУБД, ориентация компаний на поддержку разных аппаратных платформ и заинтересованность пользователей в переходе к реляционным системам позволили компаниям добиться коммерческого успеха и приступить к совершенствованию своих реализаций. В текущих версиях ORACLE, Sybase, PostgreSQL и Microsoft SQL Server поддерживаются очень мощные диалекты SQL.

Удачная идея универсального языка для всех СУБД дала дополнительный импульс широкому внедрению СУБД и особенно реляционных СУБД, на долгие годы закрепив их лидирующие позиции по сравнению с другими СУБД.

В настоящее время язык SQL реализован во всех коммерческих и некоммерческих реляционных СУБД и почти во всех СУБД, которые исходно основывались не на реляционном подходе. Все компании-производители провозглашают соответствие своей реализации стандарту SQL, и на самом деле реализованные диалекты SQL очень близки. Это произошло не сразу и не просто.

3.1.2. Стандарт языка SQL

С тех пор как был принят первый стандарт языка SQL-89, — SQL не изменил своему исходному предназначению: быть стандартизированным, обобщенным, непроцедурным и независимым от производителей языком работы с реляционными базами данных. Он всегда следовал новым тенденциям, и последняя версия стандарта — SQL:2003 — в очередной раз это доказала.

Язык SQL уже пережил множество ревизий (в среднем они происходили каждые 2–3 года), которые либо становились частью стандарта, либо не приживались к нему. Последний стандарт языка SQL был представлен в 2003 году, и очередная его редакция имела место в 2008-м. Стандарт развивается до сих пор, и последняя редакция датирована 2003 годом, однако ядро языка SQL уже давно остается неизменным. Основные вехи стандартизации языка SQL можно видеть на рис. 3.1

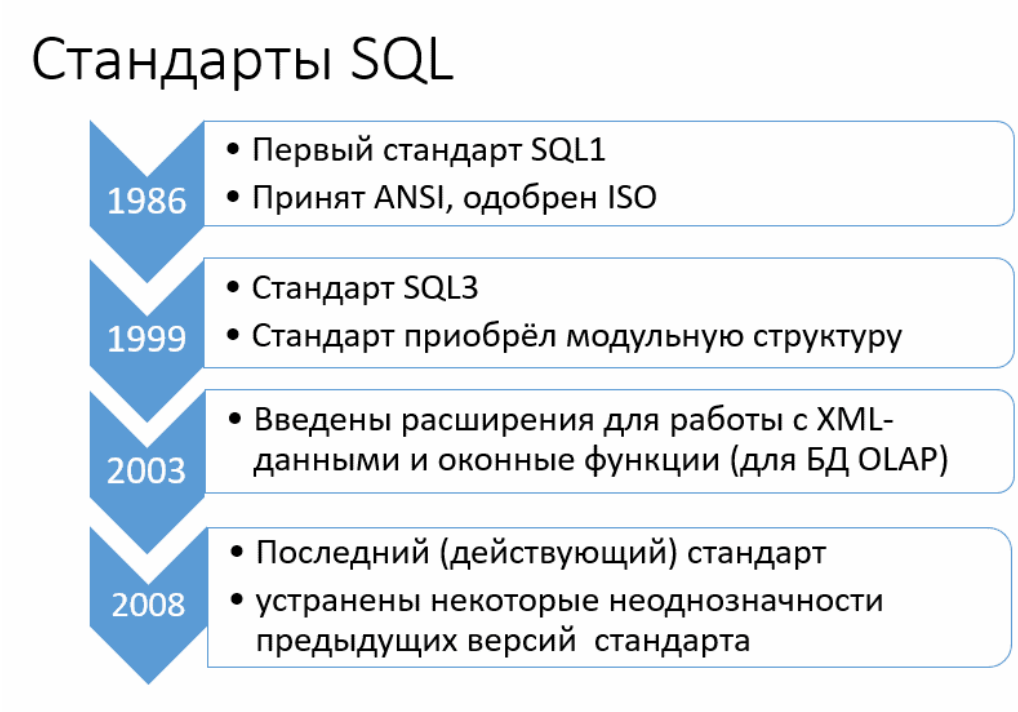

Рис. 3.1. Основные вехи стандартизации языка SQL

Стандарт SQL3 получился весьма объемным, он содержал около 1500 страниц. В связи с этим разработчики SQL3 пришли к выводу, что при таких объемах стандарта вероятность его принятия и последующей успешной поддержки заметно уменьшается. Поэтому они решили разбить стандарт на относительно независимые части, которые можно было бы разрабатывать и поддерживать по отдельности. В современном виде стандарт SQL содержит следующие части (документы):

- **SQL/Framework — описывает логические основы стандарта.**

  Первая часть (SQL/Framework) посвящена описанию концептуальной структуры стандарта. В этой части приводится развернутая аннотация следующих частей и формулируются требования к реализациям, претендующим на соответствие стандарту.
- **SQL/Foundation** — определяет содержание каждого раздела стандарта и описывает функциональное ядро стандарта (Core SQL99).

  Вторая часть SQL:1999 (SQL/Foundation) образует базис стандарта. Вводится система типов языка, формулируются правила определения функциональных зависимостей и возможных ключей, определяются синтаксис и семантика основных операторов SQL.

- **SQL/CLI —** описывает, как именно программа должна отправлять SQL-запросы к СУБД и как именно возвращённый набор записей должен быть обработан приложением. На этом стандарте основан протокол ODBC.
- **SQL/PSM —** определяет процедурные расширения языка SQL.
- **SQL/Bindings —** определяет механизм взаимодействия языка SQL с другими языками программирования.
- **SQL/MM —** описываются средства языка SQL, предназначенные для работы с мультимедийными данными.
- **SQL/OLB —** определяет синтаксис и семантику SQLJ, являющегося подмножеством SQL, встроенным в язык Java.
- **SQL/Schemata -** определяет Информационную Схему и Схему Определений, обеспечивая общий набор инструментов достаточных, чтобы сделать SQL-базы данных и их объекты самоописываемыми.
- **SQL/JRT –** описывается использование подпрограмм и типов SQL в языке Java.
- **SQL/XML -** описываются спецификации SQL позволяющие работать с XML-документами.

В настоящее время считается, что полностью закончены только первые три части, остальные продолжают развиваться и дополняться.

SQL — живой язык: он продолжает развиваться и приспосабливаться к требованиям современной жизни. Несмотря на тенденции рынка в направлении стандартизации функций и методов обмена данными, производители по понятным причинам стремятся удержать пользователей в среде конкретных СУБД, завлекая их нестандартными функциями. Эти расширения порой существенно повышают эффективность выполнения процедур, но одновременно затрудняют, а иногда даже делают невозможным перенос программ SQL на другую платформу СУБД. Тем не менее, можно сказать, что базовый набор операторов SQL, включающий операторы определения схемы БД, выборки и манипулирования данными, авторизации доступа к данным, поддержки встраивания SQL в языки программирования и операторы динамического SQL, в коммерческих реализациях устоялся.

3.2. Концепции языка SQL

SQL (structured query language — «язык структурированных запросов») — формальный непроцедурный декларативный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей СУБД.

3.2.1. Декларативность и простота

SQL не является императивным языком, и сейчас его принято относить к декларативным. Другие декларативные языки – это, например, различные языки разметки, скажем, HTML или XML.

Декларативность SQL заключается в том, что, вместо того, чтобы указывать, *как* достичь результата (как в императивных языках – C, Java), вы указываете, *какого* результата надо достичь и на основе каких *входных данных*. Всю остальную работу выполняет компонент СУБД, по традиции называемый *оптимизатором*.

Изначально SQL был основным способом работы пользователя с базой данных, причем предполагалось, что этот пользователь ничего не понимает в программировании. Однако со временем SQL превратился в мощный инструмент разработчиков приложений, работающих с БД. Язык обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и т.д.) — и стал приобретать черты, свойственные языкам программирования общего назначения. В результате SQL стал одним из самых сложных языков программирования.

3.2.2. Непроцедурность и расширения языка

Непроцедурность – это отсутствие в языке конструкции передачи управления, а это значит, что в SQL отсутствуют все привычные конструкции: циклы, условный оператор, аппарат процедур, передача параметров, помните – SQL разрабатывался для обычных людей – непрограммистов. Следовательно, работать с этим языком программистам очень сложно, для этого надо выработать особый стиль мышления. Многим разработчикам СУБД казалось, что гораздо проще расширить язык процедурными операторами для сложных процессов обработки данных. Так появились процедурные расширения языка SQL, которые представляют собой полноценные языки программирования, поддерживающий возможность использования в нем операторов SQL.

К сожалению, каждая СУБД поддерживает свой собственный язык процедурного расширения SQL, что усложняет задачи переносимости программного обеспечения.

Вот несколько распространенных расширений языка SQL:

PL/SQL – Используется в ORACLE. Сокращение обозначает Procedural Language/SQL. Диалект во многом похож на язык программирования Ada;

Transact-SQL - Используется в Microsoft SQL Server и Sybase Adaptive Server. С тех пор как в начале 1990_х Microsoft и Sybase начали развивать разные платформы, их реализации Transact-SQL также начали отличаться хотя и незначительно;

PL/pgSQL - Расширение SQL, реализованное в PostgreSQL. Сокращение расшифровывается как Procedural Language/PostgreSQL.

Основное предназначение этих языков написание процедур и функций, постоянно хранящихся в базе данных и исполняемых в среде СУБД.

Следует отметить, что хранимые процедуры и функции не следует рассматривать как альтернативу сложным SQL-запросам. По сравнению с «чистым» SQL любая процедурная альтернатива является худшим вариантом с точки зрения производительности, поскольку SQL-запросы любая СУБД обрабатывает с наивысшей эффективностью. Поэтому не нужно поддаваться соблазну и переходить на уровень процедурной логики там, где можно, пусть ценой больших мыслительных усилий, написать сложный, но эффективный запрос с использованием только средств языка SQL.

3.2.3. Замкнутость и стиль мышления

Реляционная алгебра, положенная в основу языка SQL, представляет собой набор таких операций над отношениями, что результат каждой из операций также является отношением. Это свойство реляционной алгебры называется **замкнутостью**, и оно полностью распространяется на язык SQL, только вместо термина «отношение» пользуются термином «таблицы». Входные данные для всех операторов манипулирования данными языка SQL – таблицы, выходные данные – таблица, таким образом можно использовать выход одного оператора как входные данные для другого. Только научившись мыслить таблицами, а не переменными, освоившись с вложенностью операторов, а не процедур, вы научитесь программировать на языке SQL.

3.2.4. Зависимость от СУБД и независимость от платформы

SQL отличается от стандартных языков программирования, поскольку он не может быть использован для создания обособленных приложений. Он не может существовать вне ядра СУБД, которое необходимо для преобразования инструкций в выполняемые машинные команды.

В некотором смысле СУБД работает подобно виртуальной машине Java. Даже если операторы SQL внедряются в язык высокого уровня, такой как C, они остаются не более чем текстом. Так же как веб-страницы на языке HTML, представляющие собой набор символов ASCII, интерпретируются для отображения веб-браузером, программы SQL для выполнения должны быть отправлены в некоторую СУБД, преобразующую текстовые инструкции в машинный код. Всем этим программам нужны для выполнения свои специфичные интерпретаторы.

Одной из главных платформенно-независимых особенностей языка SQL является набор базовых типов данных: в какой бы операционной системе ни выполнялась программа — UNIX или Windows, 32- или 64-разрядной, — размер и структура резервируемых блоков хранения информации остаются неизменными. Тип данных INTEGER всегда будет занимать 4 байта, а DOUBLE — 8 байт (правда, существуют и дополнительные типы данных, свойственные конкретным реализациям СУБД (О типах данных см. раздел 3.3.)

Принцип независимости от платформы SQL сходен с используемым языке Java: у первого в роли интерпретатора текстовых команд выступает СУБД, а у второго — виртуальная машина JVM. Так, программа, набранная в системе Windows с помощью программы Блокнот, может быть выполнена в СУБД ORACLE, установленной на следующих платформах:

● IBM AIX 5L версий 5.2 и 5.3 (64-разрядные);

● HP OpenVMS Alpha 8.2 и Itanium 8.2-1;

● HP Tru64 UNIX V5.1B;

● HP-UX Itanium 11i v1 (11.11), 11i V2 (11.23) и PA RISC (64-bit);

● IBM z/OS (OS/390) V1.4 и z/OSe V1.4 или выше;

● IBM zSeries Linux: Red Hat Enterprise Linux (RHEL) 4.0 (Update 2 или новее) и SUSE Linux Enterprise Server (SLES) 9 (Service Pack 2 или выше);

● Linux (Itanium, POWER, x86 и x86-64): Red Hat Enterprise Linux AS/ES 3.0 (Update 4 или новее); Red Hat Linux 4.0 (Update 1 или новее); SUSE Linux Enterprise Server 8.0 с установленным пакетом SP4 или выше; SUSE Linux Enterprise Server 9.0 с установленным пакетом SP2 или выше и Asianux 1.0/2.0;

● Microsoft Windows 2000 с установленным пакетом SP1 или выше, все редакции, включая Terminal Services и Microsoft Windows 2000 MultiLanguage Edition(MLE); все редакции Windows Server 2003; Windows XP Professional; редакции Business, Enterprise и Ultimate системы Windows Vista;

● Sun Solaris Operating System: 64-разрядная SPARC (Solaris 8 Update 7 или новее; Solaris 9 Update 6 или новее; Solaris 10), а также x86 и x86-64 (Solaris 10).

3.2.5. Равенство формализмов и отход от реляционной теории

Размышляя над языком SQL как средством общения с реляционными БД Эдгар Кодд провозгласил так называемую «теорему Кодда». Говоря простым языком, она звучит так: все три формализма - SQL, «реляционная алгебра» и «реляционное исчисление» - равны. (13).

Это означает, что любой запрос, выраженный на одном языке может быть переформулирован на другом. Данный результат прежде всего удобен тем, что позволяет использовать наиболее удобный формализм для анализа запроса, а во-вторых он связывает декларативные языки SQL и реляционное исчисление с императивной реляционной алгеброй. То есть, транслируя запрос из SQL в реляционную алгебру, мы уже получаем способ выполнения запроса (и его оптимизации).

Однако в мире IT постоянно ведутся дискуссии на тему соответствия SQL и реляционной теории. Вот, например, цитата - «Как это ни печально, SQL слишком часто отходит от принципов реляционной теории» (3).

А вот другая цитата, по сути ставящая точку во всех подобных дискуссиях:

«В области информационной технологии любой практически используемый инструмент не может быть полностью свободен от компромиссов. Идеологически чистые решения возможны только в научно-экспериментальной работе. «Великий и ужасный» язык SQL – это порождение ряда компромиссов между теорией, практикой и маркетинговой деятельностью. Этот язык является настолько реляционным, насколько это понадобилось потребителям коммерческих СУБД, прямо или косвенно финансировавшим разработку языка» (11).

3.2.6. Срезы и консолидация

В реальной предметной области можно по-разному представить и даже интерпретировать одни и те же данные. Например, бухгалтеру нужны объемы продаж в стоимостном (денежном) выражении, начальнику склада интересны объемы продаж в штуках, а начальнику перевозок объемы продаж нужны в тоннах. При помощи языка SQL легко предоставить пользователям данные в нужных форматах и разрезах, более того можно еще и выполнить сложные преобразования, связанные с группированием, суммированием, усреднением данных. Например, руководителю сети магазинов нужны не просто объемы продаж, а объемы по различным группам товаров, с разбивкой по регионам, и итоговыми цифрами за год в тысячах штук, рублей и т.д. Не всегда можно при помощи SQL построить подобные отчеты и представить их в красивом виде, но вот выбрать все необходимые данные и провести различные группировки и консолидации – это дело SQL.

3.2.7. Защита данных и привилегии пользователей

В отличии од других языков в SQL сразу был встроены механизмы защиты данных. Механизм защиты, предлагаемый SQL, многоуровневый, при этом степень детализации большей частью зависит от конкретной реализации СУБД. В сущности, все сводится к управлению правами доступа к различным объектам базы данных и к выполнению конкретных операций. С защитой данных в SQL связаны три основные концепции:

- Действующими лицами в базе данных являются пользователи. Каждый раз, когда СУБД извлекает, вставляет, удаляет или обновляет данные, она делает это от имени какого-то пользователя. СУБД выполнит требуемое действие или откажется его выполнять в зависимости от того, какой пользователь запрашивает это действие.
- Объекты базы данных являются теми элементами, чья защита может осуществляться посредством SQL. Обычно обеспечивается защита таблиц и представлений. Большинству пользователей разрешается использовать одни объекты базы данных и запрещается использовать другие.
- Привилегии – это права пользователя на проведение тех или иных действий над определенным объектом базы данных. Например, пользователю может быть разрешено, извлекать записи из некоторой таблицы и добавлять их в неё, но запрещено удалять или обновлять записи этой таблицы. У другого пользователя может быть другой набор привилегий.

3.2.8. Диалекты языка SQL

Общая концепция языка SQL для всех реляционных СУБД остается одинаковой и соответствует стандарту, однако есть и небольшие различия. Отличие операторов SQL в разных СУБД в основном заключается в типах данных (здесь могут отличаться не только их наименования, но и детали их реализации), так же может немного отличаться и сама специфика реализации языка SQL (т.е. суть операторов одна и та же, но в синтаксисе могут быть небольшие различия, обычно расширяющие стандарт языка). Владея основами SQL, вы легко сможете перейти с одной СУБД на другую, т.к. вам в данном случае нужно будет только разобраться в деталях реализации команд в новой СУБД, т.е. в большинстве случаев достаточно будет просто провести аналогию. Необходимо отметить, что различия в синтаксисе реализаций SQL затрудняют перенос приложений из одной системы в другую.

Если вы планируете плотно работать с одной из СУБД, необходимо изучить все особенности диалекта, используемого этой платформой.

3.3. Типы данных SQL

Изначально применение СУБД ограничивалось преимущественно решением финансово-экономических задач. Независимо от модели данных, базы данных обрабатывали такие основные типы данных:

- числовые – наиболее используемыми являются целочисленные, вещественные и денежные (финансовые);
- символьные, например, значения данных «четверг», «столбец», «менеджер»;
- логические, которые принимают значения «истина» или «ложь»;
- даты, которые задаются с помощью типа «Дата» или в виде обычных символьных данных.

В разных СУБД эти типы несущественно отличались названиями, диапазоном значений и видом представления. С развитием информационных технологий разрабатывались специализированные системы обработки данных (системы обработки видеоизображений, геоинформационные системы и т.д.), что привело к введению в СУБД их разработчиками поддержки новых типов данных:

- время и дата/время, которые были предназначены для хранения информации о времени и/или дате (диапазоне дат);
- символьные переменной длины, которые хранили текстовую информацию большой длины (например, документ);
- двоичные для хранения графических, аудио- и видео- объектов, хронологической, пространственной и другой специальной информации. Такие данные часто называются мультимедиа-данными. Реляционные системы могут только хранить мультимедиа-данные, а создавать и редактировать такие данные способны только специализированные программы;
- данные в формате XML.

Далее в таблице 3.1. приведены все типы данных определенные в стандарте SQL2003.

Таблица 3.1. Типы данных SQL2003

Категория	**Примеры типов данных (и сокращенные названия)**	**Описание**
BINARY	BINARY LARGE OBJECT (BLOB)	Этот тип хранит массив двоичных данных. Значение хранится без указания кодовой страницы и ограничения на длину.
BOOLEAN	BOOLEAN	Этот тип данных используется для хранения булевых значений (TRUE или FALSE)
CHARACTER	CHAR CHARACTER VARYING (VARCHAR, VARCHAR2)	боры символов конкретной кодовой страницы. Тип VARCHAR допускает хранение значений переменной длины, а тип CHAR – только фиксированной. Также в значениях типа VARCHAR автоматически удаляются пробелы в конце строки, а в CHAR, напротив, все оставшееся место заполняется пробелами.
	NATIONAL CHARACTER (NCHAR), NATIONAL VARYING CHARACTER (NCHAR VARYING)	Национальные символьные типы данных используются для поддержки специальной кодовой страницы, определяемой реализацией.
	CHARACTER LARGE OBJECT (CLOB)	Типы данных CHARACTER LARGE OBJECT (CLOB) и BINARY LARGE OBJECT (BLOB) относятся к типу LARGE OBJECT
	NATIONAL CHARACTER LARGE OBJECT (NCLOB)	То же, что и CHARACTER LARGE OBJECT, но с поддержкой кодовой страницы, определяемой реализацией.
DATALINK	DATALINK	Описывает указатель на файл или другой внешний источник данных, не являющийся частью базы данных.

NTERVAL	NTERVAL	Описывает набор значений времени или временной интервал.
COLLECTION	ARRAY MULTISET	ARRAY был введен в SQL99, а MULTISET был добавлен в SQL2003. ARRAY – это упорядоченная коллекция элементов, имеющая ограниченную длину. MULTISET – неупорядоченная коллекция, не имеющая ограничений на размер. Элементы MULTISET и ARRAY должны быть стандартного типа.
NUMERIC	INTEGER(INT) SMALLINT BIGINT NUMERIC(p,s) DEC[IMAL](p,s) FLOAT(p,s) REAL DOUBLE PRECISION	Эти типы данных хранят точные (целые и десятичные) или приблизительные (с плавающей точкой) числовые значения. INT, BIGINT и SMALLINT хранят точные числовые значения с фиксированным масштабом и точностью. NUMERIC и DEC хранят точные числовые значения с настраиваемыми масштабом и точностью. FLOAT хранит приблизительное числовое значение с настраиваемой точностью, а точность REAL и DOUBLE PRECISION фиксирована. Вы можете указать точность (precision) и масштаб (scale) типов DECIMAL, FLOAT и NUMERIC для указания общего числа хранимых цифр и десятичных знаков. INT, SMALLINT, DEC иногда называют точными типами данных, а FLOAT, REAL и DOUBLE PRECISION – приблизительными.
TEMPORAL	DATE TIME TIME WITH TIMEZONE TIMESTAMP TIMESTAMP WITH TIMEZONE	Эти типы данных используются для хранения значений времени. TIME и DATE говорят сами за себя. Типы данных с суффиксом WITH TIMEZONE содержат указание на часовой пояс. Тип TIMESTAMP используется для хранения более точного момента времени. Эти типы данных часто называют временными.
XML	XML	Хранит XML данные и может использоваться везде, где может использоваться любой другой стандартный тип данных (например, для столбца таблицы, поля в записи и т. д.). Операции со значениями типа XML предполагают наличие древовидной внутренней структуры данных. Внутренняя структура основана на XML Information Set Recommendation (Infoset), и использует новый внутренний информационный элемент с названием корневой информационный элемент XML

Не каждая платформа поддерживает все типы данных ANSI SQL, кроме того некоторые платформы поддерживают дополнительные типы данных, например, ROWID (тип присутствует в СУБД ORACLE, в SQL2003 отсутствует) - уникальный идентификатор любой записи в базе данных, тип этого псевдостолбца называется также - ROWID. А вот другие псевдостолбцы - ROWNUM и LEVEL имеют тип NUMERIC.

В некоторых реализациях СУБД преобразование типов данных выполняется автоматически, в других же для этого нужно явно его определять. На практике не рекомендуется полагаться на неявное преобразование данных, а определять его явно.

Синтаксис функции CAST, преобразующую данные из одного типа в другой, следующий:

CAST (<выражение> AS <тип_данных_назначения>)

В разных реализациях функции CAST существуют некоторые различия, но в общем случае она преобразовывает любой встроенный тип данных в другой.

Если какое-либо из преобразований не может быть выполнено, генерируется соответствующая ошибка. К примеру, если некоторое выражение не может быть преобразовано в число, ORACLE возвратит ошибку "ORA-01722: invalid number".

3.4. Операторы SQL

3.4.1. Простые примеры

Первая фраза

Изучение любого языка начинается с вывода строки «Hello, Word!», ну что ж, мы начнем так же.

Язык SQL не может существовать вне ядра СУБД (см. раздел 3.2.4) и работать может только с данными помещенными в БД, поэтому сначала надо создать некоторую структуру и заполнить ее данными.

Создаем таблицу из двух полей: первое "НОМЕР" – первичный ключ таблицы, второе поле "ТЕКСТ" – здесь будет храниться текст фразы; затем заполним одну строку таблицы «Фразы» и осуществим выдачу строки «Hello, Word!».

Листинг 3.4.1. Hello, Word!

```
CREATE TABLE "FMM"."ФРАЗЫ" -- создание таблицы «Фразы» в схеме «FMM»
     ("НОМЕР" NUMBER NOT NULL ENABLE, -- первичный ключ, цифровой,
                                      -- значение обязательно
     "ТЕКСТ" VARCHAR2(100 BYTE) NOT NULL ENABLE); -- символьное поле,
                                      -- значение обязательно
INSERT INTO "ФРАЗЫ" (НОМЕР, ТЕКСТ) VALUES ('1', 'Hello, Word!');
                                -- вставка записи в таблицу «Фразы»
SELECT ТЕКСТ FROM ФРАЗЫ; -- выборка поля ТЕКСТ из таблицы «Фразы»
```

На рис. 3.2. вы видите выполнение приведенного выше скрипта. Правда есть еще и другой, более простой способ вывести строки «Hello, Word!»:

```
SELECT 'Hello, Word!' FROM DUAL
```

Однако, этот оператор сработает только в СУБД ORACLE и не может быть перенесен в другую СУБД.

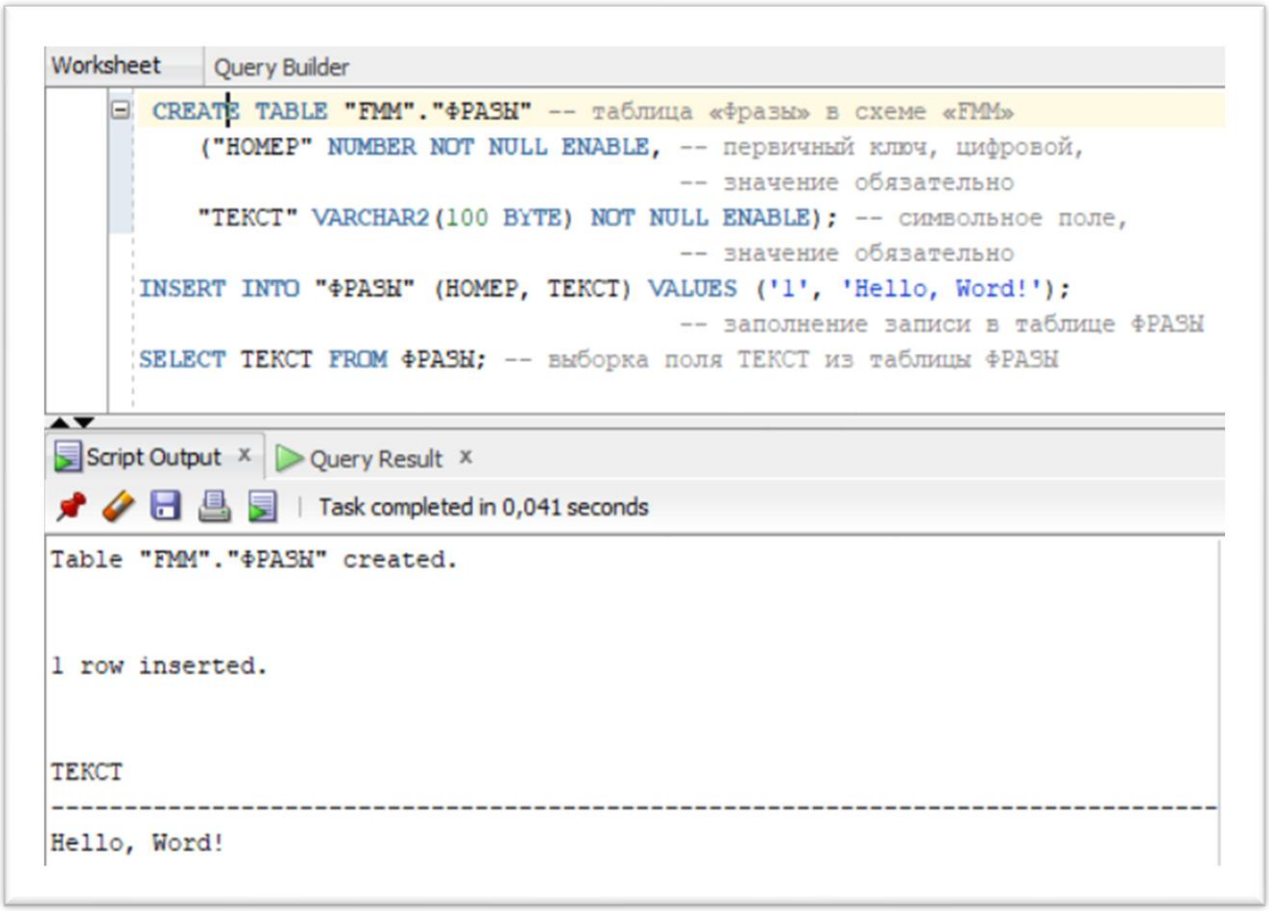

Рис. 3.2 Выполнение скрипта «Hello, Word!» в ORACLE SQL Developer

3.4.2. Некоторые общие замечания по синтаксису SQL

SQL использует типичные для многих языков программирования правила – имена должны записываться буквами, цифрами и символом подчеркивания, причем на первой позиции не допускается цифра. Большинство СУБД разрешают использование русских букв в именах полей.

Стандартный SQL не различает заглавные и строчные буквы. В этой книге заглавными буквами используются для лучшего понимания SQL.

Стандартный SQL игнорирует переходы на новую строку и избыточные пробелы. Т.е. команду SQL можно записывать в одну строку или в несколько строк – результат будет один и тот же. Поэтому разбиение на строки операторов SQL следует выполнять из соображений обеспечения лучшей читаемости программы.

Заметим также, что многие СУБД (либо инструментальные программы для работы с СУБД) в командах SQL при работе с текстовыми данными допускают использование как одинарных апострофов « ' » (по стандарту SQL), так и двойных кавычек «"».

В некоторых руководствах указано, что операторы SQL должны заканчиваться символом «;» <точка с запятой>. Однако на самом деле этот символ необходим только, если вы работаете с СУБД при помощи программы, позволяющей выполнить несколько последовательных операторов SQL (см. рис. 3.2). Тексты, объединяющие несколько операторов SQL, традиционно называются «скриптами», а инструментальные программы, позволяющие их записывать и выполнять в одной из СУБД – «редакторами скриптов». При выполнении одиночного оператора символ <точка с запятой> в конце команды необязателен.

В данной книге в качестве «редактора скриптов» используется ORACLE SQL Developer, который позволяет выполнить сразу несколько операторов SQL (см. рис. 3.2).

Все примеры операторов языка SQL в главе 3 построены на схеме данных учебного примера, описанного в Приложении В.

Синтаксис операторов языка SQL описывается в формах Бэкуса-Наура, которые описаны в Приложении Ж.

3.4.3. Оператор SELECT

Изучение SQL мы начнём с простейших примеров – запросов к базе данных. Оператор SELECT, синтаксис которого приведен ниже на рис. 3.3 извлекает данные упомянутые в предложении SELECT из таблиц, указанных в предложении FROM в соответствии с правилами, устанавливаемыми в остальных предложениях оператора SELECT.

SELECT [{ALL | DISTINCT}] *извлекаемый_ элемент*
    [AS *псевдоним*][, ...]
FROM {*таблица* [[AS] *псевдоним*] | *представление*
    [[AS] *псевдоним*]}[, ...]
    [[*тип_ соединения*] JOIN *условие_ соединения*]
[**WHERE** *условие_ отбора*] [{AND | OR | NOT} *условие_ отбора* [...]]
[**GROUP BY** *выражение_ группировки*]
[**HAVING** *условие_ отбора*]]
[**ORDER BY** {*выражение_сортировки* [ASC | DESC]}[, ...]]

Рис. 3.3 Синтаксис оператора SELECT

После ключевого слова SELECT в операторе размещается несколько «извлекаемых элементов», описывающих, какие поля должны составлять результирующую таблицу.

Каждый «извлекаемый элемент» может быть литералом, агрегатной или скалярной функцией, математическим выражением, параметром или переменной, подзапросом, но чаще всего просто полем таблицы или представления.

Предложение FROM указывает список источников данных, из которых выбираются данные для результата. Источниками данных могут быть таблицы базы данных, представления или любые конструкции SQL, возвращающие таблицы (такие конструкции называются табличными выражениями).

Предложение WHERE задает условия, которым должны удовлетворять записи входных таблиц, для того чтобы эти строки использовались при выполнении оператора. *Условие_отбора* это по сути выражение, результатом которого является булево значение (TRUE или FOLSE), если значение этого выражения - TRUE, то запись отбирается для дальнейшей обработки или выдаче пользователю.

Предложение GROUP BY группирует записи в итоговой таблице таким образом, что в группу попадают записи с одинаковым значением *выражения_группировки.* Каждая группа заменяется одной итоговой записью с агрегатными значениями в полях.

Группы могут быть отфильтрованы так же, как и отдельные записи входящих таблиц. Предложение HAVING работает аналогично предложению WHERE, отбирая только те группы, для которых *условие_ отбора* принимает значение TRUE.

Порядок выдачи записей в итоговой таблице определяется в предложении ORDER BY и производится в соответствии со значением *выражения_сортировки*, которое вычисляется для каждой записи – от меньших значений к большим, т.е. сначала выводится запись с минимальным значением *выражения_сортировки*, потом с большим и т.д.

Более подробно процесс выполнения оператора SELECT описан в разделе 3.4.9.

Простейший вариант запроса на языке SQL выглядит так:

SELECT * FROM <таблица>

Символ "*" (звездочка) здесь обозначает, что выбираются все поля таблицы. Выполнение такого оператора показано на рис. 3.4. Выборка производится из таблицы «СОТРУДНИК» из схемы БД приведенной в Приложении В. На таблицах из этой схемы будут базироваться дальнейшие примеры.

Worksheet | Query Builder

```
SELECT * FROM "СОТРУДНИКИ"
```

Script Output | Query Result

SQL | All Rows Fetched: 12 in 0 seconds

	НОМЕР_СТ	ФИО	ОТДЕЛ	ЗАРПЛАТА	РУКОВОД
1	1	Штейн А.Я.	1	112000	(null)
2	2	Штейн И.Я.	2	113000	(null)
3	3	Филькенштейн А.Я.	3	114000	(null)
4	4	Фомин М.М.	1	80000	1
5	5	Бауман Ю.И.	2	80000	2
6	6	Сусанин И.И.	1	60000	4
7	7	Грязнов Г.Г.	2	60000	5
8	8	Невструев Я.П.	3	75000	3
9	9	Камнеедов А.П.	3	80000	8
10	10	Ничушкина Т.Н.	2	60000	11
11	11	Иванова Г.С.	2	100000	(null)
12	12	Смирнова Е.В.	1	60000	11

Рис. 3.4 Простейший запрос

На рис 3.4 показана таблица, которая является результатом запроса видного в верхней части. Первая колонка — это псевдостолбец, которого нет в результирующей таблице и который просто показывает порядковый номер записи. Вторая колонка — это первичный ключ таблицы «СОТРУДНИК» (является суррогатным ключом), а далее идут поля таблицы в произвольном порядке, т.к. порядок следования полей в реляционной БД не определен. Поля ОТДЕЛ и РУКОВОДИТЕЛЬ являются внешними ключами, указывающими на отдел, в котором числится сотрудник и непосредственного руководителя, если он есть, если руководителя нету, это поле остается пустым, на что указывает слово ‘null’.

Следующий пример, представленный на рис. 3.5 такой же простой.

На рис. 3.5 результирующая таблица содержит одно поле – ИМЯ, причем результирующие записи никак не упорядочены, потому что записи в исходной таблице согласно реляционной модели не упорядочены. Имена повторяются, и мы видим одинаковые записи в результирующем наборе, причем непонятно к какой конкретно записи исходной таблицы они относятся.

Следующий пример показывает, как убрать повторяющиеся (одинаковые) записи. На рис. 3.6 в операторе SELECT используется фраза DISTINCT, которая указывает на необходимость убрать одинаковые записи, т.е. вместо всех одинаковых записей остается одна, опять же не привязанная ни к какой записи в исходной таблице «СОТРУДНИК». Обратите внимание – записи опять не упорядочены, и их расстановка не совпадает с предыдущим запросом (рис. 3.5).

```
SELECT ИМЯ FROM СОТРУДНИК
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 13 in 0 seconds

	ИМЯ
1	Дмитрий
2	Илья
3	Александр
4	Михаил
5	Юрий
6	Александр
7	Алексей
8	Алексей
9	Иван
10	Елена
11	Галина
12	Татьяна
13	Александр

Рис. 3.5 Выборка имен

Worksheet | Query Builder

```
SELECT DISTINCT ИМЯ FROM СОТРУДНИК
```

Script Output × | Query Result ×

SQL | All Rows Fetched: 10 in 0,011 seconds

	ИМЯ
1	Михаил
2	Юрий
3	Алексей
4	Дмитрий
5	Галина
6	Илья
7	Елена
8	Иван
9	Александр
10	Татьяна

Рис. 3.6 Выборка имен без повторений

Далее, на рис. 3.7 приведен пример с предложением WHERE, которое позволяет вывести только часть записей, а именно те записи, которые удовлетворяют условию, следующему за

ключевым словом WHERE. Условие отбора записей WHERE должно вычисляться в пределах одной записи т.е. все операнды условного выражения должны принадлежать одной записи таблицы полученной после выполнения предложения FROM.

Для текстов (а также для даты и времени) используются апострофы, для чисел они не нужны. В некоторых SQL-системах используются не апострофы, а двойные кавычки, или допустимо и то и другое.

В качестве знаков сравнения можно использовать =, <, >, <= (меньше или равно), >= (больше или равно), <> (не равно). Слева и справа от знака сравнения могут использоваться поля таблицы.

В условиях отбора возможно применение встроенных в SQL функций – в данном случае это функция выделения подстроки (подробней см. раздел 3.5).

На рис. 3.8 представлено сложное условие, состоящее из нескольких условных операций соединенных с помощью логической связки OR. Использование в условиях логических связок AND OR и NOT достаточно традиционно, поэтому нет смысла описывать их подробно. Понятно также, что для указания приоритета можно использовать круглые скобки. Особенно надо отметить, что результатом условных операций может быть не только TRUE или FOLSE, но и отсутствие значения, на которое указывает слово NULL (подробнее о NULL см. раздел 3.4.12)

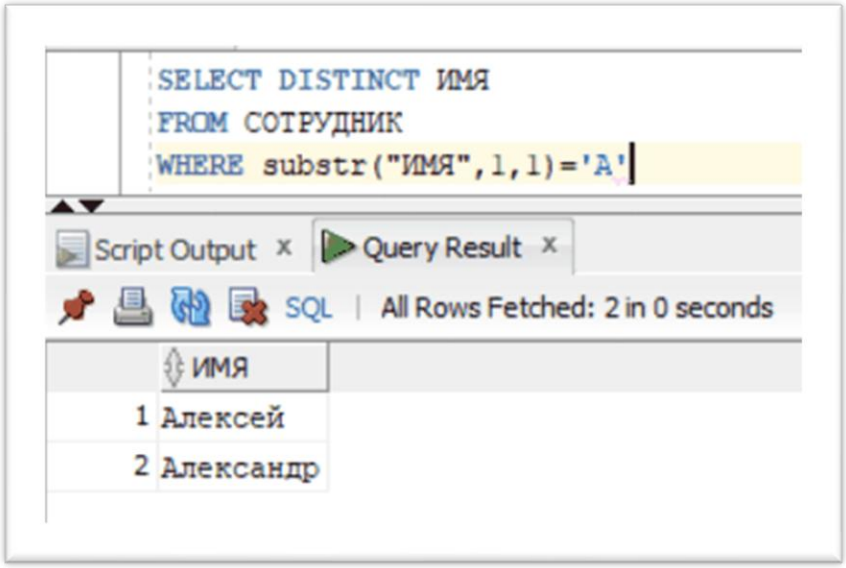

Рис. 3.7 Выборка некоторых имен

В общем, на рис. 3.8 приведен запрос количества упоминаний того или иного имени в таблице «СОТРУДНИК». Предложение GROUP BY позволяет получать агрегированные значения для одной или нескольких записей, получаемых при разбиения всего множества записей в таблице на группы по значениям указанных полей группировки.

Подсчет количества упоминаний имен осуществляется встроенной функцией COUNT(*), звездочка в параметре указывает на то, что надо посчитать все записи в группе. В данном случае группирование происходит по полю ИМЯ, т.е. все записи с одинаковым значением поля ИМЯ объединяются в группу. В результате получается таблица с именами сотрудников (по значению поля ИМЯ записи объединяются в группы) и количеством записей (количеству упоминаний имени) в каждой группе.

```
SELECT ИМЯ, COUNT(*)
FROM СОТРУДНИК
WHERE substr(ИМЯ,1,1) = 'А'
  OR  substr(ИМЯ,1,1) IN ('Г', 'Д')
GROUP BY "ИМЯ";
```

Script Output | Query Result

SQL | All Rows Fetched: 4 in 0 seconds

	ИМЯ	COUNT(*)
1	Алексей	2
2	Дмитрий	1
3	Галина	1
4	Александр	3

Рис. 3.8 Подсчет количества повторяющихся имен

Группы, полученные после выполнения предложения можно отфильтровать, и сделать это можно при помощи предложения HEVING. В предложении HEVING указывается условие отбора групп (также как в предложении WHERE указывается условие отбора записей таблицы) и в результирующую таблицу попадают только группы, удовлетворяющие этому условию. Предложение HAVING очень похожа на WHERE, и в ней можно использовать все те же функции и операции, что и в предложении WHERE. Действие предложения HEVING показано на рис. 3.9, в итоговую таблицу, как видно, попадают только группы, в которых число записей больше 1. Как и во всех предыдущих примерах итоговая таблица никак не отсортирована, записи в ней идут в произвольном порядке.

```
SELECT ИМЯ, COUNT(*)
FROM СОТРУДНИК
WHERE substr(ИМЯ,1,1) = 'А'
  OR  substr(ИМЯ,1,1) IN ('Г', 'Д')
GROUP BY "ИМЯ"
HAVING COUNT(*) > 1
```

Script Output | Query Result

SQL | All Rows Fetched: 2 in 0 seconds

	ИМЯ	COUNT(*)
1	Алексей	2
2	Александр	3

Рис. 3.9 Подсчет количества повторяющихся имен

Для сортировки итоговой таблицы используется предложение ORDER BY. Сортировка может проводиться по любому полю итоговой таблицы или по комбинации полей, так на рис. 3.10 сортировка проделывается сначала по второму полю в обратном порядке (от больших значений к меньшим, на это указывает ключевое слово **DESC**.), а затем, записи с одинаковыми значениями второго поля сортируются по значению первого поля. На рис. 3.10 в предложении ORDER BY используются номера полей, но можно указать и имена полей, используемых в предложении SELECT.

В принципе, в предложении ORDER BY можно использовать не только любое поле из предложения SELECT, но и любое выражение, составленное из этих полей. Более подробно о пространстве имен, разрешённых в предложении ORDER BY см. в разделе «Понимание SQL».

Оптимизатор запросов в современных СУБД определяет наиболее эффективный порядок сортировки записей при выполнении запроса. В частности, если для таблицы имеется подходящий индекс, то оптимизатор запроса будет его использовать.

Правила сортировки полей, в которых может содержаться значение NULL см. в разделе 3.4.12.

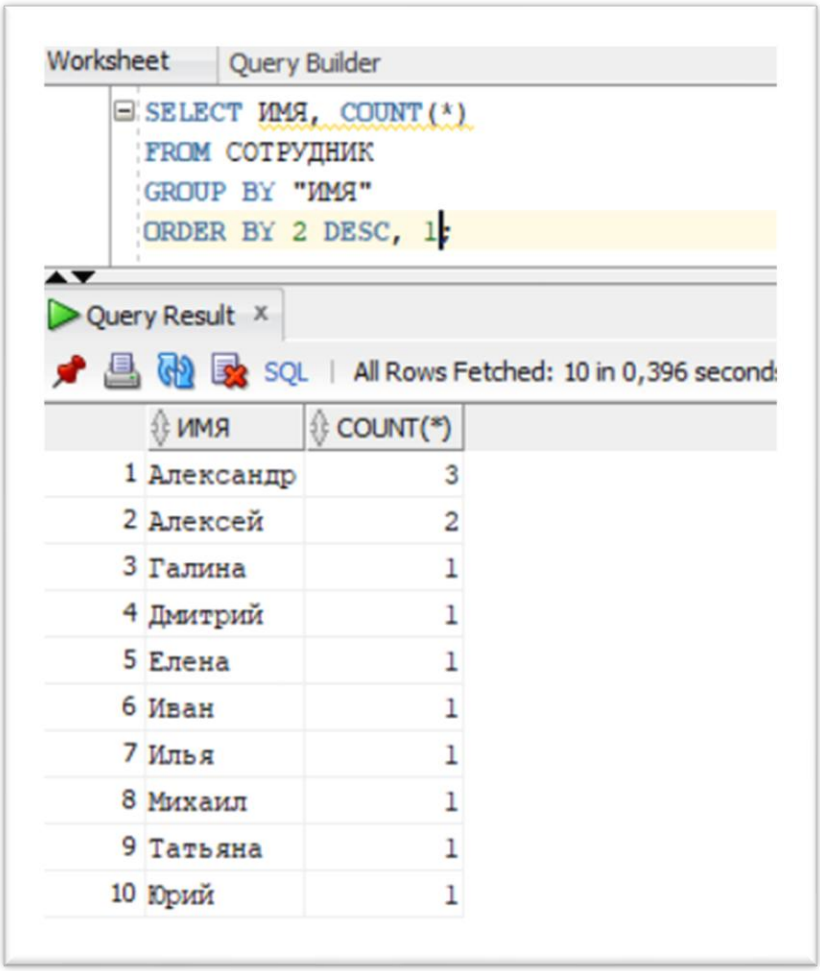

Рис. 3.10 Пример сортировки

3.4.4. Подзапросы

Везде, где по синтаксису языка SQL разрешены выражения или таблицы, возможно использование подзапроса (вложенного запроса), возвращающего отдельное значение, запись или таблицу. Вспомнив рис. 3.3. (синтаксис оператора SELECT), легко убедится, что подзапрос может

быть использован в любом предложении оператора. То же самое можно сказать и по поводу любого другого оператора группы манипулирования данными (см. раздел 3.4.6.).

Каждый подзапрос, в свою очередь, может содержать один или несколько вложенных подзапросов. Количество вложений подзапросов в операторе SQL не ограничено.

Подзапрос может содержать все стандартные предложения, разрешённые для использования в обычном SQL-запросе: DISTINCT, GROUP BY, ORDER BY, объединения таблиц, запросов и т.д. Надо, правда, заметить, что употребление предложения ORDER BY в подзапросах бессмысленно, т.к. сортировка итоговых строк подзапроса ни на что не влияет.

Подзапрос — это достаточно затратный (в смысле потребления ресурсов компьютера) механизм получения информации, и использовать его нужно обдуманно.

При использовании подзапросов надо помнить несколько простых правил:

- подзапрос должен быть заключен в круглые скобки;
- подзапрос должен указываться в правой части оператора сравнения;
- первым будет выполняться подзапрос с самым большим уровнем вложенности. После него будут выполнены подзапросы со вторым и первым уровнем вложенности, и, в заключение, когда вся необходимая информация будет собрана, выполнится главный (внешний) запрос.

В листинге 3.4.2. продемонстрирован вложенный подзапрос, который выдаёт список первичных ключей записей сотрудников, задействованных в работах, а внешний запрос выдает фамилии этих сотрудников (подробнее про операцию IN см. в разделе 3.4.11):

Листинг 3.4.2. Простой подзапрос. Сотрудники участвующие в работах

```
SELECT "ФИО"
FROM "СОТРУДНИКИ"
WHERE "НОМЕР_СТ" IN (SELECT DISTINCT "СОТР"
        FROM "СПИСКИ_СТ")
```

Часто используются коррелирующие подзапросы, которые отличается от обычных тем, что содержит в условии предложения WHERE параметр из внешнего запроса.

Пример использования коррелирующего подзапроса приведен на рис. 3.11. Этот запрос выдает сотрудников, с указанием отдела, при этом начальников отделов помечает символом «*». Внутренний подзапрос возвращает ту самую «звездочку», для каждого сотрудника, являющегося начальником отдела. поэтому в предложении WHERE внутреннего запроса используется поле из внешнего запроса. Внутренний запрос в данном случае выполняется для каждой записи таблицы внешнего запроса. Обратите внимание на сортировку итоговой таблицы, которая дает необходимую картину.

```
select o.НАЗВ_ОТ, с.ФИО,
     coalesce ((select '*'
                from  ОТДЕЛЫ o1
                where c.НОМЕР_СТ = o1.НАЧАЛЬНИК),'  ')
      as "Начальник"
from СОТРУДНИКИ с, ОТДЕЛЫ o
where с.ОТДЕЛ = o.НОМЕР_ОТ
order by 1,3 desc, 2;
```

ОТДЕЛЫ(+) 1

select o.НАЗВ_ОТ, с.ФИО, coalesce ((select '* | Введите SQL выражение чтобы

Таблица | Текст

НАЗВ_ОТ	ФИО	Начальник
Внешних коммуникаций	Штейн А.Я.	*
Внешних коммуникаций	Смирнова Е.В.	
Внешних коммуникаций	Сусанин И.И.	
Внешних коммуникаций	Фомин М.М.	
Внутренних инсинуаций	Штейн И.Я.	*
Внутренних инсинуаций	Бауман Ю.И.	
Внутренних инсинуаций	Грязнов Г.Г.	
Внутренних инсинуаций	Иванова Г.С.	
Внутренних инсинуаций	Ничушкина Т.Н.	
Протокольных мероприятий	Филькенштейн А.Я.	*
Протокольных мероприятий	Камнеедов А.П.	
Протокольных мероприятий	Невструев Я.П.	

Рис. 3.11. Коррелирующий подзапрос. Сотрудники и начальники отделов

При использовании в предложении WHERE имен полей из различных запросов и подзапросов (SELECT-ов) встает вопрос об области действия имен. Поскольку каждая таблица — это отдельное пространство имен, возможна ситуация, когда оптимизатор СУБД не может понять к какой таблице относится имя поля, более того, одна и та же таблица может использоваться несколько раз в одном запросе (см. рис. 3.11) и эти «экземпляры» таблиц могут порождать путаницу. Конечно, оптимизатор СУБД тотчас выдаст ошибку, но как таких ошибок избежать? Специально для разрешения подобных ситуаций был придуман механизм «псевдонимов» или «алиасов». Псевдоним (в дальнейшем будет употребляться именно этот термин) – это пространство имен, которое определяется в предложении FROM т.е. каждой таблице присваивается свой псевдоним - уникальное имя, действующее в пределах оператора SQL, включая все вложенные операторы. Полное имя поля любой таблицы в запросе получается сложением псевдонима таблицы и имени поля:

<псевдоним_таблицы>.<имя_поля>

например,

РУК."НОМЕР_СТ" или С."ОТДЕЛ" или С. "РУКОВОД"

Считается хорошим стилем программирования определять псевдонимы для всех таблиц, участвующих в запросе независимо от их повторяемости, т.к. никто не гарантирует, что со временем какое-нибудь поле не добавят в любую таблицу вашего запроса таким образом, что имена полей совпадут, а тогда ваш запрос просто станет неработоспособным.

Пример использования псевдонимов виден на рис. 3.11.

Разберем еще один пример – нахождение сотрудников с максимальной зарплатой в отделе, заметим, что их может быть несколько. Листинг 3.4.3 показывает, как подзапрос используется в

предложении WHERE, он возвращает одно значение и выполняется для каждой строки таблицы СОТРУДНИКИ.

Листинг 3.4.3. Подзапрос. Сотрудники с максимальной зарплатой в отделе

```
select o.НАЗВ_ОТ, c.ФИО, c.ЗАРПЛАТА
from СОТРУДНИКИ c, ОТДЕЛЫ o
where c.ОТДЕЛ = o.НОМЕР_ОТ
and c.ЗАРПЛАТА =
   (select max(св.ЗАРПЛАТА)
    from СОТРУДНИКИ св
    where св.ОТДЕЛ = c.ОТДЕЛ)
order by 1
```

А листинг 3.4.4. иллюстрирует вариант, когда подзапрос используется в предложении FROM, возвращает таблицу максимальных зарплат для каждого отдела и выполняется один раз.

Листинг 3.4.4. Подзапрос. Сотрудники с максимальной зарплатой в отделе

```
select o.НАЗВ_ОТ, c.ФИО, c.ЗАРПЛАТА
from СОТРУДНИКИ c, ОТДЕЛЫ o,
    (select max(св.ЗАРПЛАТА) as ЗП, св.ОТДЕЛ as отдел
     from СОТРУДНИКИ св
     group by ОТДЕЛ) МАХЗП
where c.ОТДЕЛ = o.НОМЕР_ОТ
and c.ЗАРПЛАТА = МАХЗП.ЗП
order by 1
```

Используя подзапросы, мы фактически выполняем выборку данных из нескольких таблиц, причем в коррелирующих подзапросах используются связи между таблицами. Связи между таблицами, в этом случае, реализуются условиями в предложении WHERE подзапроса. Более подробно выборка из нескольких таблиц обсуждается в следующих разделах.

3.4.5. Выборка из нескольких таблиц

В реляционных базах данных информацию часто приходится извлекать одновременно из большого количества таблиц, на практике в запросах иногда участвуют десятки таблиц. Похоже, 90% искусства программирования на языке SQL вытекает из таланта объединения множества таблиц и основывается на доскональном знании структуры объектов базы данных.

Результат объединения двух таблиц можно рассматривать как новую виртуальную таблицу, которая, в свою очередь, может быть объединена в запросе с другими таблицами, создавая новую виртуальную таблицу и т.д. Рассмотрим простейший пример объединения двух таблиц (см рис 3.12).

Worksheet | Query Builder

```
SELECT
ИГРУШКА, ЦИ.Цвет as "Цвет игрушки"
FROM ИГРУШКИ, Цвета ЦИ;
```

Query Result

SQL | All Rows Fetched: 20 in 0 seconds

	ИГРУШКА	Цвет игрушки
1	Мишка	Красный
2	Мишка	Синий
3	Мишка	Зеленый
4	Мишка	Желтый
5	Мишка	Оранжевый
6	Кукла	Красный
7	Кукла	Синий
8	Кукла	Зеленый
9	Кукла	Желтый
10	Кукла	Оранжевый
11	Зайка	Красный
12	Зайка	Синий
13	Зайка	Зеленый
14	Зайка	Желтый
15	Зайка	Оранжевый
16	Машинка	Красный
17	Машинка	Синий
18	Машинка	Зеленый
19	Машинка	Желтый
20	Машинка	Оранжевый

Worksheet | Query Builder

```
SELECT
ИГРУШКА, ЦИ.Цвет as "Цвет игрушки",
Цу.Цвет as "Цвет упаковки"
FROM ИГРУШКИ, Цвета ЦИ, Цвета Цу;
```

Query Result

SQL | All Rows Fetched: 100 in 0 seconds

	ИГРУШКА	Цвет игрушки	Цвет упаковки
1	Мишка	Красный	Красный
2	Мишка	Красный	Синий
3	Мишка	Красный	Зеленый
4	Мишка	Красный	Желтый
5	Мишка	Красный	Оранжевый
6	Мишка	Синий	Красный
7	Мишка	Синий	Синий
8	Мишка	Синий	Зеленый
.....			
91	Машинка	Желтый	Красный
92	Машинка	Желтый	Синий
93	Машинка	Желтый	Зеленый
94	Машинка	Желтый	Желтый
95	Машинка	Желтый	Оранжевый
96	Машинка	Оранжевый	Красный
97	Машинка	Оранжевый	Синий
98	Машинка	Оранжевый	Зеленый
99	Машинка	Оранжевый	Желтый
100	Машинка	Оранжевый	Оранжевый

Рис. 3.12 Простые объединения

Слева результат объединения двух таблиц, а справа трех, таблица – ЦВЕТА в этом соединении присутствует дважды: как «цвет игрушки» и как «цвет упаковки», при этом у таблицы ЦВЕТА появляются два псевдонима (про псевдонимы см. раздел 3.4.4) «ЦИ» и «ЦУ». В соответствии со стандартом SQL таблицы, упомянутые в предложении FROM, соединяются так, что в результирующей таблице записи представляют собой все возможные сочетания записей исходных таблиц. Такая операция над таблицами называется «декартово произведение». В результате количество записей результирующей таблицы равно произведению количества записей таблиц-операндов декартова произведения. В нашем примере таблица ИГРУШКИ содержит 4 записи, а таблица ЦВЕТА — 5. В результате получается 4 * 5 = 20 записей или 4 * 5 * 5 = 100 (см. рис. 3.12).

В чистом виде декартово произведение практически не используется как не используются в реляционной модели данных несвязанные таблицы. Все таблицы в БД какой-либо предметной области связаны между собой (см. раздел 1.1.5). Эти связи осуществляются при помощи первичных и внешних ключей, а в запросе – это условие равенства значений ключей в предложении WHERE, как на рис. 3.13. Такое соединение таблиц в операторе SELECT называется «внутренним», в отличии от «внешнего» соединения таблиц, которое осуществляется вне оператора SELECT при помощи специальных операторов (см. раздел 3.4.7.).

Worksheet | Query Builder

```
SELECT o."НАЗВ_ОТ", c."ФИО"
FROM "СОТРУДНИКИ" c, "ОТДЕЛЫ" o
WHERE c."НОМЕР_СТ" = o."НАЧАЛЬНИК";
```

Query Result

SQL | All Rows Fetched: 3 in 0 seconds

	НАЗВ_ОТ	ФИО
1	Внешних коммуникаций	Штейн А.Я.
2	Внутренних инсинуаций	Штейн И.Я.
3	Протокольных мероприятий	Фильkенштейн А.Я.

Рис. 3.13. Запрос «начальники отделов». ТЭТА - стиль

В этом запросе выбираются начальники отделов в соответствии со связью таблиц «СОТРУДНИКИ» и «ОТДЕЛЫ», т.е. выбираются только те записи декартова произведения таблиц, которые соответствуют равенству первичного ключа в таблице «СОТРУДНИКИ» и внешнего ключа в таблице «ОТДЕЛЫ». Такое соединение, выполненное при помощи предложения WHERE - называется ТЭТА-соединением (см. рис. 3.13). В противоположность, соединение таблиц при помощи фразы JOIN называется ANSI – соединением (см раздел 3.4.6.).

Описанный алгоритм действий (сначала декартово произведение – FROM, потом выборка записей - WHERE) является строго логическим, то есть он лишь объясняет результат, который должен получиться при выполнении операции, но не предписывает, чтобы конкретная СУБД выполняла соединение именно указанным образом. Существует несколько способов реализации операции соединения, например, соединение вложенными циклами (англ. *inner loops join*), соединение хешированием (англ. *hash join*), соединение слиянием (англ. *merge join*). Единственное требование состоит в том, чтобы любая реализация логически давала такой же результат, как при применении описанного алгоритма.

Далее на рис. 3.14 приведен пример соединения трех таблиц, причем две из них – таблица «СОТРУДНИКИ» и связанная с ними таблица «ОТДЕЛЫ». По сути этот запрос является усложненной версией оператора SELECT приведенного на рис. 3.13; добавление еще одной колонки (Количество сотрудников отдела) привело к появлению дополнительной таблицы в предложении FROM и предложения GROUP BY, которое позволяет произвести подсчет количества сотрудников в отделе. В предложении WHERE два связанных условия, каждое из которых связывает пару таблиц, в общем случае при использовании ТЭТА-соединения количество связующих условий должно быть на единицу меньше, чем количество таблиц в запросе. Функция MAX здесь необходима, в связи с тем, что при использовании предложения GROUP BY все извлекаемые элементы должны либо повторять поля группирования, либо быть агрегатными функциями (подробнее см. раздел 3.5.4). Поскольку во всех записях группы связанной с отделом значение поля НАЧ.”ФИО” одинаково, можно с таким же успехом употребить функцию MIN.

Worksheet | Query Builder

```
SELECT o."НАЗВ_ОТ", MAX(НАЧ."ФИО") "Начальник отд.", COUNT(*) "Кол. сотрудников"
FROM "СОТРУДНИКИ" С, "ОТДЕЛЫ" о, "СОТРУДНИКИ" НАЧ
WHERE НАЧ."НОМЕР_СТ" = о."НАЧАЛЬНИК"
      AND о."НОМЕР_ОТ" = С."ОТДЕЛ"
GROUP BY о."НАЗВ_ОТ";
```

Query Result | SQL | All Rows Fetched: 3 in 0 seconds

	НАЗВ_ОТ	Начальник отд.	Кол. сотрудников
1	Протокольных мероприятий	Фильценштейн А.Я.	3
2	Внешних коммуникаций	Штейн А.Я.	4
3	Внутренних инсинуаций	Штейн И.Я.	5

Рис. 3.14. Количественный состав отделов и их начальники

3.4.6. Конструкция JOIN. Внутренние соединения

В предыдущем разделе рассматривались внутренние соединения, записанные ТЭТА – стилем, а в этом разделе будет обсуждаться ANSI – стиль. Надо сразу отметить, что речь идет именно о разных стилях записи: результат выполнения операторов, записанных по-разному, совершенно одинаков. Конструкция JOIN появилась сравнительно недавно – новый “стандартный” синтаксис начал поддерживаться только ORACLE 9i и Microsoft SQL Server 7.0. Синтаксис внутреннего соединения предполагает конструкцию JOIN в предложении FROM:

FROM таблица [AS псевдоним]
[тип_соединения] JOIN соединяемая_таблица [[AS] псевдоним]
{ ON условие_соединения1 [{AND | OR} условие_соединения2] [...] |
USING (поле1[, ...])}
[...]

Рассмотрим ключевые слова, используемые в этой синтаксической конструкции.

FROM таблица

Указывает первую таблицу или представление, участвующую в соединении.

[тип_соединения] JOIN соединяемая_таблица

Указывает тип используемого соединения и вторую (или последующую) соединяемую таблицу или представление. Надо отметить, что при использовании нескольких конструкций JOIN происходит последовательное соединение таблиц, т.е. к полученной после первого соединения таблице происходит присоединение следующей и т.д. Для соединяемых таблиц можно указать псевдонимы. Возможны следующие типы соединений:

CROSS JOIN

Кросс соединение (также известное как декартово произведение) работает также как оператор, представленный на рис. 3.12.

[INNER] JOIN

При внутреннем соединении из каждой таблицы отсеиваются записи, к которым не присоединяются записи из другой таблицы. Этот тип соединения используется по умолчанию поэтому можно опустить ключевое слово INNER.

ON условие_соединения

Соединяет записи из таблицы, указанной во FROM, и таблицы, указанной в JOIN. Вы можете указывать несколько операторов JOIN, основанных на общем наборе значений. Обычно (но не всегда) ключ соединения является первичным ключом в одной таблице и внешним ключом в другой. Если данные в полях совпадают, то объединенные записи присутствуют в результирующей таблице.

USING (поле1[…])

Указывает, что условием соединения является равенство значений в указанных полях, имеющихся в каждой таблице. Написание фразы USING может быть и быстрее, чем ON таблица1.полеА=таблица2.полеА, но результат при этом одинаковый.

Простой пример использования конструкции JOIN показан на рис. 3.15. он повторяет запрос, приведенный на рис. 3.13.

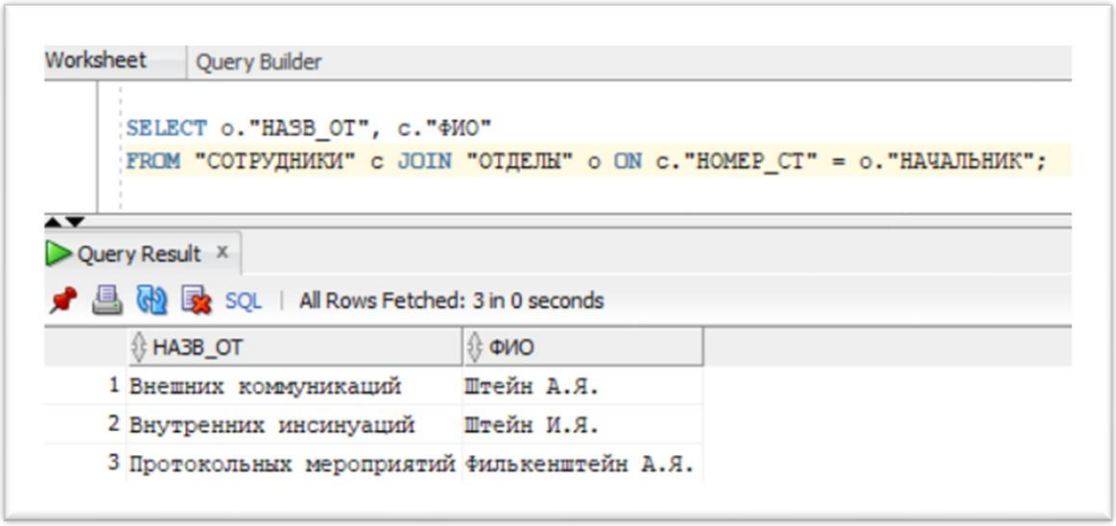

Рис. 3.15 Запрос «начальники отделов». ANSI – стиль

Надо повториться: ТЭТА-стиль и ANSI-стиль — это просто разная запись одних и тех же действий, которые выполняются СУБД совершенно одинаково, хотя во многих публикациях утверждается превосходство в быстродействии, то одного, то другого типа записи.

По рекомендациям ANSI следует использовать для соединений фразу JOIN, а не описывать условие соединения в WHERE. Это делает ваш код более простым и понятным, позволяя легко отличить условие соединения от условия поиска. Кроме того, это избавляет от неочевидных конструкций, используемых на некоторых платформах для записи с помощью WHERE внешних соединений.

С другой стороны, необходимо учитывать, что, если в операторе SELECT соединяются три и больше таблиц при помощи ANSI – стиля, надо очень внимательно смотреть за последовательностью соединений и учитывать результат каждого соединения, особенно если смешиваются внешние и внутренние соединения. При соединении большого количества таблиц очень трудно проследить к какому результату приведет масса последовательных JOIN-ов. Кроме того, присутствует соблазн записать в конструкцию JOIN и условия выборки, тогда операторы становятся совсем далекими от понимания.

И тот и другой тип записи (ТЭТА и ANSI) имеет своих сторонников и противников, которые готовы до хрипоты спорить о правильном написании многотабличных запросов. В больших корпорациях стиль написания чаще всего определяется внутренними стандартами, если же таких стандартов нет, то вам самим придется решать этот вопрос.

3.4.7. Объединение результатов запросов. Внешние соединения

Из результатов нескольких запросов можно сформировать единый набор данных при помощи оператора UNION. Синтаксис оператора UNION представлен ниже

<оператор_select1>
UNION [ALL | DISTINCT]
<оператор_select2>
UNION [ALL | DISTINCT]
<оператор_select3>
……

Следует отметить, что в списках извлекаемых элементов предложений SELECT содержащихся в операторе UNION, должно содержаться одинаковое количество элементов, причем соответствующие элементы должны иметь совместимые типы данных. К примеру, если в первом запросе выражение возвращает число, во втором запросе, объединенном оператором UNION, соответствующее выражение не должно возвращать символьную строку. Листинг 3.4.2. демонстрирует объединение трех запросов из разных таблиц, которые считают количество людей в контингенте университета.

Листинг 3.4.2. Объединение результатов запросов. Оператор UNION

```
select sum(cnt) -- подсчет контингента университета
from (            --результирующая таблица попадает в предложение from
select count(*) as cnt -- подсчет количества студентов
from "СТУДЕНТЫ"
UNION
select count(*)     -- подсчет количества сотрудников
from "СОТРУДНИКИ"
UNION
select count(*)     -- подсчет количества преподавателей
from "ПРЕПОДАВАТЕЛИ");
```

Объединенный результирующий набор данных можно упорядочить, однако следует отметить, что предложение ORDER BY может применяться только к общим результатам, хотя с точки зрения синтаксиса предложение относится к последнему запросу (см. листинг 3.4.3.).

Если запросы потенциально могут возвращать одинаковые записи, их можно как объединить, так и не объединять. По умолчанию оператор UNION объединяет одинаковые записи (по умолчанию UNION DISTINCT), однако если применить оператор UNION ALL, все записи, отобранные отдельными запросами, будут присутствовать в объединенном наборе данных.

Оператор UNION относится к классу операторов работы с множествами, куда также входят операторы EXCEPT и INTERSECT. Все эти операторы используются одинаковым образом для работы с множествами записей, полученных как результат работы нескольких операторов SELECT, отсюда и название - операторы работы с множествами.

Оператор EXCEPT формирует из результата двух или более операторов SELECT множество записей, включающее все записи из результата первого запроса, которые отсутствуют в результатах последующих запросов. EXCEPT фильтрует записи, оставляя те, которые есть только в первой из нескольких однотипных таблиц. При выполнении запроса из листинга 3.4.3. будут выданы данные по всем аспирантам, не работающим в качестве сотрудника (не

содержащиеся в таблице "СОТРУДНИКИ") и не преподающим (не содержащиеся в таблице "Преподаватели").

Листинг 3.4.3. Объединение результатов. Оператор EXCEPT

```
select А.ФИО, А.Паспорт        -- все аспиранты
from "Аспиранты" А
EXCEPT
select Сот.ФИО, Сот.Паспорт    -- все сотрудники
from "СОТРУДНИКИ" Сот
EXCEPT
select П.ФИО, П.Паспорт        -- все преподаватели
from "Преподаватели" П
order by 1;
```

Оператор INTERSECT возвращает из двух или более запросов записей, которые есть в каждом из запросов. При выполнении запроса из листинга 3.4.4. будут выданы данные по всем работающим студентам одновременно присутствующих в таблицах "СТУДЕНТЫ" и "СОТРУДНИКИ".

Листинг 3.4.4. Объединение результатов. Оператор INTERSECT

```
select Сту.ФИО, Сту.Паспорт    -- все студенты
from "СТУДЕНТЫ" Сту
INTERSECT
select Сот.ФИО, Сот.Паспорт    -- все сотрудники
from "СОТРУДНИКИ" Сот;
```

Исходя из вышеизложенного можно разделить все соединения на две категории: внутренние (см. раздел 3.4.5 – 3.4.6) – определяемые внутри оператора в предложении FROM и внешние (см. раздел 3.4.7) – объединяющие результаты нескольких предложений SELECT. Однако в стандарте SQL появилась странные конструкции – внешние соединения, определяемые в предложении FROM, об этих конструкциях и пойдёт речь в следующем разделе.

3.4.8. Конструкция JOIN. Внешние соединения

Рассмотрим простой запрос – «вывести всех сотрудников с их руководителями» (см. рис. 3.15)

Worksheet | Query Builder

```
SELECT C."ФИО" AS "Сотрудник", РУК."ФИО"  AS "Руководитель"
FROM "СОТРУДНИКИ" C, "СОТРУДНИКИ" РУК
WHERE C."РУКОВОД" = РУК."НОМЕР_СТ";
```

Explain Plan | Query Result

SQL | All Rows Fetched: 8 in 0 seconds

	Сотрудник	Руководитель
1	Смирнова Е.В.	Иванова Г.С.
2	Ничушкина Т.Н.	Иванова Г.С.
3	Фомин М.М.	Штейн А.Я.
4	Бауман Ю.И.	Штейн И.Я.
5	Невструев Я.П.	Фильkенштейн А.Я.
6	Сусанин И.И.	Фомин М.М.
7	Грязнов Г.Г.	Бауман Ю.И.
8	Камнеедов А.П.	Невструев Я.П.

Рис. 3.16 Запрос «Руководители сотрудников». ТЭТА – стиль

Запрос приведенный на рис. 3.16. Выдает всех сотрудников с руководителем каждого сотрудника, однако сотрудники, не имеющие руководителя, в результирующей таблице отсутствуют, ведь у таких сотрудников в поле РУКОВОД содержится NULL (см. рис. 3.4). Для того, чтобы получить список всех сотрудников, дополненный руководителями необходимо усложнить предыдущий запрос используя внешнее соединение (см. рис. 3.17).

Worksheet | Query Builder

```
SELECT C."ФИО" AS "Сотрудник", РУК."ФИО"  AS "Руководитель"
FROM "СОТРУДНИКИ" C, "СОТРУДНИКИ" РУК
WHERE C."РУКОВОД" = РУК."НОМЕР_СТ"
UNION
SELECT C1."ФИО" AS "Сотрудник", NULL
FROM "СОТРУДНИКИ" C1
WHERE C1."РУКОВОД" IS NULL;
```

Explain Plan | Query Result

SQL | All Rows Fetched: 12 in 0,016 seconds

	Сотрудник	Руководитель
1	Бауман Ю.И.	Штейн И.Я.
2	Грязнов Г.Г.	Бауман Ю.И.
3	Иванова Г.С.	(null)
4	Камнеедов А.П.	Невструев Я.П.
5	Невструев Я.П.	Фильkенштейн А.Я.
6	Ничушкина Т.Н.	Иванова Г.С.
7	Смирнова Е.В.	Иванова Г.С.
8	Сусанин И.И.	Фомин М.М.
9	Фильkенштейн А.Я.	(null)
10	Фомин М.М.	Штейн А.Я.
11	Штейн А.Я.	(null)
12	Штейн И.Я.	(null)

Рис. 3.17 Запрос «Все сотрудники с Руководителями». ТЭТА – стиль

Теперь в результирующей таблице присутствуют все сотрудники, с руководителями, а отсутствие руководителя отмечено словом (null). В стандарте SQL для подобных запросов придуман специальный тип соединения LEFT JOIN (левое внешнее). Левое внешнее соединение возвращает все записи из таблицы, указанной слева от слова JOIN. Если для некоторых записей из левой таблицы нет соответствующих записей в правой, то они все равно не отбрасываются, при этом в поля, соответствующие правой таблице, помещаются значения NULL. На рис. 3.18 представлен такой запрос написанный в соответствии с ANSI-стилем. Таким образом получилось «внешнее соединение», записанное как «внутреннее», т.е. внутри оператора SELECT.

Worksheet | Query Builder

```
SELECT C."ФИО" AS "Сотрудник", РУК."ФИО"  AS "Руководитель"
FROM "СОТРУДНИКИ" C LEFT JOIN "СОТРУДНИКИ" РУК ON C."РУКОВОД" = РУК."НОМЕР_СТ";
```

Explain Plan | Query Result

SQL | All Rows Fetched: 12 in 0 seconds

	Сотрудник	Руководитель
1	Смирнова Е.В.	Иванова Г.С.
2	Ничушкина Т.Н.	Иванова Г.С.
3	Фомин М.М.	Штейн А.Я.
4	Бауман Ю.И.	Штейн И.Я.
5	Невструев Я.П.	Фильkенштейн А.Я.
6	Сусанин И.И.	Фомин М.М.
7	Грязнов Г.Г.	Бауман Ю.И.
8	Камнеедов А.П.	Невструев Я.П.
9	Фильkенштейн А.Я.	(null)
10	Штейн И.Я.	(null)
11	Штейн А.Я.	(null)
12	Иванова Г.С.	(null)

Рис. 3.18 Запрос «Все сотрудники с Руководителями». ANSI – стиль

Приведем еще один запрос, иллюстрирующий использование LEFT JOIN (см. рис. 3.19) – «Вывести всех сотрудников и все выполняемые ими работы».

Если в предыдущем запросе внешнее соединение использовалось потому, что во внешнем ключе некоторых записей значения оказались не заполненными (содержат NULL), то в следующем запросе необходимость использования внешнего соединения диктуется отсутствием записей в таблице СПИСКИ_СТ связанных с некоторыми записями таблицы СОТРУДНИКИ, т.е. существуют сотрудники, не участвующие ни в одной из работ. В Листинге 3.4.5. приведен запрос, реализующий ту же выборку, что и на рис. 3.19.

Листинг 3.4.5. Запрос «Все сотрудники и выполняемые ими работы». ТЭТА – стиль

```
SELECT C."ФИО" AS "Сотрудник", P."НАЗВАНИЕ" AS "Работа"
FROM "СОТРУДНИКИ" C, "СПИСКИ_СТ" CCT, "РАБОТЫ" P
WHERE CCT."СОТР" = C."НОМЕР_СТ"
 AND CCT."РАБОТА" = P."НОМЕР_РБ"
UNION
SELECT C1."ФИО" AS "Сотрудник", NULL
FROM "СОТРУДНИКИ" C1
WHERE NOT EXISTS
   (SELECT C2.*
     FROM "СОТРУДНИКИ" C2, "СПИСКИ_СТ" CCT1
     WHERE CCT1."СОТР" = C1."НОМЕР_СТ"
   AND C2."НОМЕР_СТ" = C1."НОМЕР_СТ");
```

Worksheet | Query Builder

```
SELECT C."ФИО" AS "Сотрудник", P."НАЗВАНИЕ" AS "Работа"
FROM "СОТРУДНИКИ" C LEFT JOIN "СПИСКИ_СТ" CCT ON CCT."СОТР" = C."НОМЕР_СТ"
LEFT JOIN "РАБОТЫ" P ON CCT."РАБОТА" = P."НОМЕР_РБ"
```

Explain Plan | Query Result

SQL | All Rows Fetched: 17 in 0,016 seconds

	Сотрудник	Работа
1	Бауман Ю.И.	Встреча инопланетян
2	Бауман Ю.И.	Награждение непричасных
3	Бауман Ю.И.	Сбор объяснительных
4	Бауман Ю.И.	Наказание невиновных
5	Грязнов Г.Г.	Встреча инопланетян
6	Иванова Г.С.	(null)
7	Камнеедов А.П.	Создание кадавра
8	Камнеедов А.П.	Всеобщая вакцинация
9	Невструев Я.П.	Всеобщая вакцинация
10	Ничушкина Т.Н.	(null)
11	Смирнова Е.В.	(null)
12	Сусанин И.И.	Встреча инопланетян
13	Фильkенштейн А.Я.	Всеобщая вакцинация
14	Фомин М.М.	Установка рогаток
15	Фомин М.М.	Встреча инопланетян
16	Штейн А.Я.	(null)
17	Штейн И.Я.	(null)

Рис. 3.19 Запрос «Все сотрудники и выполняемые ими работы». ANSI – стиль

Если в операторе SELECT соединяются три и больше таблиц при помощи ANSI – стиля, надо внимательно смотреть за последовательностью соединений и учитывать результат каждого соединения, особенно если смешиваются внешние и внутренние соединения. В листинге 3.4.6. показан запрос, который «должен» работать так же, как и запрос на рис. 3.19, однако это не так: все записи со словом (null) здесь утеряны.

Листинг 3.4.6. Запрос «Все сотрудники и выполняемые ими работы». ANSI – стиль

```
SELECT C."ФИО" AS "Сотрудник", P."НАЗВАНИЕ" AS "Работа"
FROM "СОТРУДНИКИ" C LEFT JOIN "СПИСКИ_СТ" CCT ON CCT."СОТР" = C."НОМЕР_СТ"
JOIN "РАБОТЫ" P ON CCT."РАБОТА" = P."НОМЕР_РБ"
ORDER BY 1;
```

Во многих публикациях указывается, что нельзя смешивать внешние и внутренние соединения, хотя листинг 3.4.7. показывает такую возможность - результат этого запроса такой же как на рис. 3.19.

Листинг 3.4.7. Запрос «Все сотрудники и выполняемые ими работы». ANSI – стиль

```
SELECT C."ФИО" AS "Сотрудник", P."НАЗВАНИЕ" AS "Работа"
FROM "СПИСКИ_СТ" CCT JOIN "РАБОТЫ" P ON CCT."РАБОТА" = P."НОМЕР_РБ"
RIGHT JOIN "СОТРУДНИКИ" C ON CCT."СОТР" = C."НОМЕР_СТ"
```

```
ORDER BY 1;
```

Далее приводится синтаксис конструкции внешнего соединения, и объяснение работы ключевых слов, используемых в этой конструкции:

FROM таблица 1
{{LEFT | RIGHT | FULL} [OUTER]} JOIN таблица 2
[ON условие_соединения]

Рассмотрим ключевые слова, используемые в этой синтаксической конструкции.

LEFT [OUTER] JOIN

Левое внешнее соединение возвращает все записи из таблицы, указанной слева от слова JOIN. Если для некоторых записей из левой таблицы нет соответствующих записей в правой, то они все равно не отбрасываются, при этом в поля, соответствующие правой таблице, помещаются значения NULL. Хорошим тоном считается не смешивать использование правых и левых соединений, а использовать всегда только левое соединение.

RIGHT [OUTER] JOIN

Правое внешнее соединение возвращает все записи из таблицы, указанной справа от слова JOIN. Если для некоторых записей из правой таблицы нет соответствующих записей в левой, то они все равно не отбрасываются, при этом в поля, соответствующие левой таблице, помещаются значения NULL.

FULL [OUTER] JOIN

Полное внешнее соединение возвращает все записи из обеих таблиц, вне зависимости от того, есть ли для определенной записи соответствующая запись в противоположной таблице. Для записей, не соединенных со записями противоположной таблицы, полям противоположной таблицы присваиваются пустые значения.

3.4.9. Понимание SQL. Как выполняется оператор SELECT

Код SQL не является упорядоченным - элементы кода SQL не расположены в том порядке, в каком они выполняются. Лексический порядок выглядит так:

- SELECT [DISTINCT]
- FROM
- WHERE
- GROUP BY
- HAVING
- UNION
- ORDER BY

Для простоты перечислены только основные предложения SQL. Этот лексический порядок принципиально отличается от логического порядка выполнения предложений (который в свою очередь может незначительно отличаться от фактического порядка выполнения в зависимости от выбора оптимизатора):

- FROM
- WHERE
- GROUP BY
- SELECT
- DISTINCT

- HAVING
- UNION
- ORDER BY

Рассмотрим выполнение примера изложенному в Листинге 3.4. Оператор SELECT в этом листинге, впрочем, как и в остальных, работает со схемой данных учебного примера, описанного в Приложении В.

Описанный ниже механизм выполнения операторов SQL является строго логическим, то есть он лишь объясняет результат, который должен получиться при выполнении оператора, но не предписывает, чтобы конкретная СУБД обеспечивала именно такой механизм выполнения операторов.

Листинг 3.4.5. Нахождение самой большой зарплаты среди сотрудников занятых в одной работе.

```
SELECT p."НАЗВАНИЕ", max(c."ЗАРПЛАТА") as ЗП
FROM FMM."СПИСКИ_СТ" сст, FMM."РАБОТЫ" p, FMM."СОТРУДНИКИ" c
WHERE c."НОМЕР_СТ" = сст."СОТР"
 and сст."РАБОТА" = p."НОМЕР_РБ"
GROUP by p."НАЗВАНИЕ"
HAVING max(c."ЗАРПЛАТА") >= 85000
ORDER BY ЗП;
```

Первый шаг: Предложение FROM. Выполняется соединение всех таблиц упомянутых в предложении FROM методом «Декартово произведение». Результат приведен на рис. 3.20. В результирующей таблице присутствуют поля из всех таблиц упомянутых в предложении FROM, а записи в количестве 567 представляют собой множество всех возможных сочетаний строк из трех таблиц.

Второй шаг: Предложение WHERE. Из таблицы, которая является результатом слияния таблиц упомянутых в предложении FROM отбираются все записи, которые удовлетворяют условию, указанному в предложении WHERE т.е. условие применяется к каждой записи таблицы, и результатом является таблица, содержащая те и только те записи, для которых результатом вычисления условного выражения является TRUE (значения FALSE, UNKNOWN и NULL не являются разрешающими).

Результат приведен на рис. 3.21. Обратите внимание, что условие в предложении WHERE может включать любые поля всех таблиц упомянутых в предложении FROM, т.е. область имен возможных в предложении WHERE определяется таблицей, полученной на первом шаге.

Третий шаг: Предложение GROUP BY. Таблица, полученная на шаге 2 преобразуется коренным образом:

- происходит группирование записей по полям, указанным в предложении GROUP BY;
- таблица формируется из полей, указанных в предложении SELECT.

Результат этого преобразования приведен на рис. 3.22.

В результате выполнения раздела GROUP BY образуется сгруппированная таблица, в которой записи таблицы, полученной на втором этапе, расставлены в минимальное число групп, таких, что во всех записях одной группы значения полей, указанных в списке имен полей предложения GROUP BY (полей группировки), одинаковы.

Надо помнить, что при использовании предложения GROUP BY все извлекаемые элементы, упомянутые в предложении SELECT, должны либо повторять поля группировки, либо быть агрегатными функциями (подробнее см. раздел 3.5.4).

На третьем шаге результирующая таблица полностью обретает свою структуру, дальше может меняться только количество записей, а вот состав полей уже определился окончательно. Определилось и новое пространство имен, в котором будут работать остальные предложения оператора SELECT.

Обратите внимание: если в операторе нет предложения GROUP BY, то в третьем шаге выполняется операция проекции, т.е. из таблицы, полученной на шаге FROM выбираются или вычисляются поля упомянутые в предложении SELECT.

Четвертый шаг: Предикат DISTINCT удаляет повторяющиеся записи, оставляя по одной записи из каждой группы дубликатов. В противоположность DISTINCT предикат ALL не выполняет никаких действий.

Пятый шаг: Предложение HAVING. В предложении HAVING указывается условие выборки групп т.е. записей таблицы, полученной на четвертом шаге и в результате останутся только записи, удовлетворяющие этому условию. Условие в предложении HAVING может включать любые выражения, включающие в себя поля упомянутые в предложении SELECT и агрегатные функции, в параметрах которых могут быть использованы любые поля таблицы, полученной в шаге 2. В предложении HAVING отбираются все группы, которые удовлетворяют условию, т.е. условие применяется к каждой записи соответствующей группе, и результатом является таблица, содержащая те и только те записи, для которых результатом вычисления условного выражения является TRUE (значения FALSE, UNKNOWN и NULL не являются разрешающими).

Шестой шаг: Предложение UNION (EXCEPT/INTERSECT). Так называемое внешнее соединение таблиц, в отличии от внутреннего, которое производится в предложении FROM. На этом шаге происходит слияние результатов нескольких запросов в одну таблицу (подробнее см. раздел 3.4.6). в каждой из входящих таблиц должно содержаться одинаковое количество полей, причем соответствующие поля должны иметь совместимые типы данных.

Седьмой шаг: Предложение ORDER BY. Таблица, полученная на предыдущем шаге, может быть отсортирована с помощью фразы ORDER BY в соответствии со «списком сортировки». Каждый элемент списка сортировки представляет собой выражение, в котором могут быть использованы имена полей таблицы, полученной на предыдущем шаге. Вместо выражения допускается использование номера столбца сортируемой таблицы.

Выполнение раздела ORDER BY производится следующим образом. Выбирается первый элемент списка сортировки, и записи таблицы, полученной на предыдущем этапе, расставляются в порядке возрастания или в порядке убывания (при наличии спецификации DESC) в соответствии со значениями первого элемента списка сортировки, которые вычисляются для каждой записи таблицы. Далее выбирается второй элемент списка сортировки, и в соответствии со значениями заданного в нем выражения расставляются записи, которые после первого шага сортировки образовали группы с одинаковым значением выражения первого элемента списка сортировки. Операция продолжается до исчерпания списка элементов сортировки. Результирующий отсортированный список строк является окончательным результатом запроса.

Если ORDER BY не используется, то большинство платформ возвращают данные либо в порядке их физического хранения, либо в порядке, определяемом используемым в запросе индексом, однако какой-либо определенный порядок строк не гарантируется.

Необходимо указать, что предложение ORDER BY может быть одно в операторе и относится оно к итоговой таблице, хотя при использовании UNION, синтаксически кажется, что предложение ORDER BY относится к последнему оператору SELECT.

```
SELECT *
FROM FMM."СПИСКИ_СТ" сст, FMM."РАБОТЫ" р, FMM."СОТРУДНИКИ" с;
```

Script Output × | Query Result ×

SQL | Fetching next 50 rows in 0,092 seconds

	НОМЕР_СП	СОТР	РАБОТА	НОМЕР_РБ	НАЗВАНИЕ	ОБЪЕМ	ГЛАВНЫЙ	НОМЕР_СТ	ФИО	ОТДЕЛ	ЗАРПЛАТА	РУКОВОД
62	8	5	3	1	Награждение непричасных	133	5	7	Грязнов Г.Г.	2	60000	5
63	9	9	5	1	Награждение непричасных	133	5	7	Грязнов Г.Г.	2	60000	5
64	1	4	6	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
65	2	6	6	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
66	3	5	6	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
67	4	7	6	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
68	5	4	4	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
69	6	5	1	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
70	7	5	2	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
71	8	5	3	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
72	9	9	5	1	Награждение непричасных	133	5	8	Невструев Я.П.	3	75000	3
73	1	4	6	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
74	2	6	6	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
75	3	5	6	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
76	4	7	6	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
77	5	4	4	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
78	6	5	1	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
79	7	5	2	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
80	8	5	3	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
81	9	9	5	1	Награждение непричасных	133	5	9	Камнеедов А.П.	3	85000	8
82	1	4	6	2	Наказание невиновных	124	5	1	Штейн А.Я.	1	112000	(null)
83	2	6	6	2	Наказание невиновных	124	5	1	Штейн А.Я.	1	112000	(null)
84	3	5	6	2	Наказание невиновных	124	5	1	Штейн А.Я.	1	112000	(null)
85	4	7	6	2	Наказание невиновных	124	5	1	Штейн А.Я.	1	112000	(null)
86	5	4	4	2	Наказание невиновных	124	5	1	Штейн А.Я.	1	112000	(null)

Рис. 3.20. Таблица после предложения FROM

Worksheet | Query Builder

```
SELECT *
FROM FMM."СПИСКИ_СТ" сст, FMM."РАБОТЫ" р, FMM."СОТРУДНИКИ" с
WHERE с."НОМЕР_СТ" = сст."СОТР"
and сст."РАБОТА" = р."НОМЕР_РБ";
```

Script Output | Query Result

SQL | All Rows Fetched: 9 in 0,006 seconds

	НОМЕР_СП	СОТР	РАБОТА	НОМЕР_РБ	НАЗВАНИЕ	ОБЪЕМ	ГЛАВНЫЙ	НОМЕР_СТ	ФИО	ОТДЕЛ	ЗАРПЛАТА	РУКОВОД
1	6	5	1	1	Награждение непричасных	133	5	5	Бауман Ю.И.	2	80000	2
2	7	5	2	2	Наказание невиновных	124	5	5	Бауман Ю.И.	2	80000	2
3	8	5	3	3	Сбор объяснительных	12	1	5	Бауман Ю.И.	2	80000	2
4	5	4	4	4	Установка рогаток	15	4	4	Фомин М.М.	1	90000	1
5	9	9	5	5	Создание кадавра	333	9	9	Камнеедов А.П.	3	85000	8
6	4	7	6	6	Встреча инопланетян	500	4	7	Грязнов Г.Г.	2	60000	5
7	2	6	6	6	Встреча инопланетян	500	4	6	Сусанин И.И.	1	60000	4
8	3	5	6	6	Встреча инопланетян	500	4	5	Бауман Ю.И.	2	80000	2
9	1	4	6	6	Встреча инопланетян	500	4	4	Фомин М.М.	1	90000	1

Рис. 3.21. Таблица после предложения WHERE

Worksheet | Query Builder

```
SELECT p."НАЗВАНИЕ", max(c."ЗАРПЛАТА")
FROM FMM."СПИСКИ_СТ" сст, FMM."РАБОТЫ" p, FMM."СОТРУДНИКИ" c
WHERE c."НОМЕР_СТ" = сст."СОТР"
and сст."РАБОТА" = p."НОМЕР_РБ"
GROUP by p."НАЗВАНИЕ";
```

Script Output x | Autotrace x | Explain Plan x | Query Result x

SQL | All Rows Fetched: 6 in 0,007 seconds

	НАЗВАНИЕ	MAX(C."ЗАРПЛАТА")
1	Встреча инопланетян	90000
2	Награждение непричасных	80000
3	Установка рогаток	90000
4	Наказание невиновных	80000
5	Сбор объяснительных	80000
6	Создание кадавра	85000

Рис. 3.22. Таблица после предложения GROUP BY

3.4.10. Все операторы языка SQL

Одним из основных требований к системам управления базами данных является наличие высокоуровневых средств выполнения запросов. В системах, реализующих реляционную модель данных, в качестве такого средства используется язык SQL. Фактически этот язык содержит полный набор операций, необходимых для выполнения любых действий с базой данных. Стандарты SQL предусматривают подразделение средств языка на несколько категорий, отличающихся по их обязательности, для того чтобы реализация удовлетворяла требованиям стандарта. Далее представлены все операторы языка SQL, разбитые на группы в соответствии с современным стандартом:

- Операторы определения данных (*Data Definition Language,* ***DDL***) обеспечивают создание, модификацию и удаление элементов описания структуры базы данных, как логической, так и структуры хранения (см. разделы 3.4.17. и 3.4.18):
  - CREATE создает объект БД,
  - ALTER изменяет объект,
  - DROP удаляет объект;
- Операторы манипуляции данными (*Data Manipulation Language,* ***DML****)* обеспечивают выполнение поиска, извлечения, добавления, изменения и удаления данных, определенных в описании логической структуры базы данных, но не позволяют изменять эту структуру (см. разделы 3.4.3., 3.4.16., 3.4.17.):
  - SELECT выбирает данные, удовлетворяющие заданным условиям,
  - INSERT добавляет новые данные,
  - UPDATE изменяет существующие данные,
  - DELETE удаляет данные;

- Операторы определения доступа к данным (*Data Control Language,* ***DCL***) обеспечивают разграничение доступа пользователей к данным (см. раздел 4.7.):
  - GRANT предоставляет пользователю разрешения на определенные операции с объектом БД,
  - REVOKE отзывает ранее выданные разрешения,
- Операторы управления транзакциями (*Transaction Control Language,* ***TCL)*** обеспечивают определение транзакций (о транзакциях см. раздел 2.2.5):
  - START TRANSACTION отмечает начало транзакции,
  - COMMIT сохраняет все изменения текущей транзакции,
  - ROLLBACK откатывает все изменения текущей транзакции,
  - SAVEPOINT делит транзакцию на более мелкие участки.

В расширениях и диалектах языка SQL присутствуют дополнительные средства, обеспечивающие описание особенностей, специфических для конкретной СУБД. Чаще всего это определения конкретных структур хранений и их параметров.

3.4.11. Предложение WHERE

Назначение предложения WHERE - фильтрация ненужных данных в таблице, полученной в предложении FROM, после работы WHERE остаются только записи, удовлетворяющие заданному в предложении условному выражению. Условие отбора применяется к каждой записи таблицы, и отбираются те и только те записи, для которых результатом вычисления условного выражения является TRUE (значения FALSE, UNKNOWN и NULL не являются разрешающими).

Неправильно написанное предложение WHERE может похоронить производительность нормального в остальном оператора SELECT [1].

Изучение всех нюансов WHERE является задачей первостепенной важности.

В условии, следующем за ключевым словом WHERE можно употреблять следующие операции:

Логическая операция	**Значение**
AND	TRUE, если оба булевых выражения истинны
OR	TRUE, если хотя бы одно из двух булевых выражений истинно
BETWEEN	TRUE, если операнд попадает в заданный интервал
IN	TRUE, если операнд равен хотя бы одному выражению из списка или строке из результата подзапроса
LIKE	TRUE, если операнд удовлетворяет шаблону
EXISTS	TRUE, если подзапрос возвращает хотя бы одну строку
ALL	TRUE, если истинно каждое условие из множества
ANY / SOME	TRUE, если истинно хотя бы одно условие из множества
NOT	Заменяет булево значение на противоположное

Далее приведены основные возможности фразы WHERE:

Операторы сравнения

Вы можете сравнивать наборы значений, используя операторы <, >, <>, >=,

<=, и =. Например:

WHERE "СОТРУДНИКИ"."ФИО" = 'Фомин М.М.'

AND

Позволяет объединять несколько условий и отбирать только те записи, которые удовлетворяют всем условиям. Например:

WHERE ("РАБОТЫ"."НАЗВАНИЕ" LIKE 'На%') **AND** "ОБЪЕМ" > 300

OR

Позволяет объединять несколько условий и отбирать только те записи, которые удовлетворяют хотя бы одному условию. Например:

WHERE "НОМЕР_РБ" < '13' **OR** "ОБЪЕМ" > 300

LIKE

Позволяет реализовывать поиск по шаблонам, указываемым в кавычках. Групповые символы, поддерживаемые в шаблонах, зависят от платформы и описываются в разделах, посвященным этим платформам. Все платформы поддерживают групповые символы ‘%’ ‘_’ (процент и подчеркивание). Символ процента показывает возможность любого количества (в том числе и нулевое) любых символов, а подчеркивание – один обязательный символ. Например, чтобы найти все телефоны, код города которых начинается на 495, можно использовать следующее условие:

WHERE "ТЕЛЕФОН" **LIKE** '495%'

Не путайте шаблоны с регулярными выражениями – это разные вещи, правда надо сказать, что почти все СУБД поддерживают регулярные выражения на уровне встроенных функций.

BETWEEN

Проверяет попадание значения в диапазон, задаваемый двумя другими значениями (границы диапазона включаются), например:

WHERE "СОТРУДНИКИ"."ЗАРПЛАТА" **BETWEEN** 40000 AND 90000

Ниже приведено аналогичное условие, написанное по-другому:

WHERE "СОТРУДНИКИ"."ЗАРПЛАТА" >= 40000
  AND "СОТРУДНИКИ"."ЗАРПЛАТА" <= 90000

Операндами операции **BETWEEN** могут быть данные любых типов и поэтому надо помнить правила сравнения строк или дат.

IN

Проверяет вхождение значения в определенный список.

WHERE *код_региона* **IN** (‘77’, ‘777’, ‘799’, ‘99’)

Вместо перечисления значений вы можете использовать подзапрос, возвращающий одно или несколько значений подходящего типа. Здесь подзапрос выдаёт список первичных ключей сотрудников, задействованных в работах, а внешний запрос - фамилии этих сотрудников:

```
SELECT "ФИО"
FROM "СОТРУДНИКИ"
WHERE "НОМЕР_СТ" IN (SELECT DISTINCT "СОТР"
                     FROM "СПИСКИ_СТ")
```

EXISTS

Используется совместно с подзапросами для определения того, что подзапрос возвращает данные. Обычно это работает намного быстрее, чем подзапросы с условием WHERE IN. Следующий запрос, аналогичный предыдущему, находит всех сотрудников, задействованных в работах:

```
SELECT A."ФИО"
FROM "СОТРУДНИКИ" A
WHERE EXISTS (SELECT 'X'
              FROM "СПИСКИ_СТ" C
              WHERE A."НОМЕР_СТ" = C."СОТР")
```

SOME | ANY

Работает так же, как оператор EXISTS, но использует немного другой синтаксис. Следующий запрос возвращает сотрудников, участвующих в работах:

```
SELECT A."ФИО"
FROM "СОТРУДНИКИ" A
WHERE A."НОМЕР_СТ" =
                SOME (SELECT DISTINCT C."СОТР" FROM "СПИСКИ_СТ" C)
```

ALL

Проверяет, что все записи, возвращаемые подзапросом, удовлетворяют определенному условию. Если запрос не возвращает ни одной записи, то результат ALL будет TRUE. Например:

```
WHERE city =ALL (SELECT city FROM employees WHERE emp_id = 54123)
```

NOT

Инвертирует значение условия. В запросе вы можете написать

```
WHERE NOT LIKE... или WHERE NOT IN... или IS NOT NULL
```

IS NULL и IS NOT NULL

Позволяют искать пустые или непустые значения, используя в предложении WHERE выражение IS [NOT] NULL. Работа с NULL подробно обсуждается в следующем разделе.

3.4.12. NULL и работа с ним

Во всех реляционных СУБД любой тип данных может иметь неопределенное значение (NULL). Неопытные SQL-программисты и разработчики часто принимают NULL за ноль или пробел. Но NULL – это ни то, ни другое. В SQL3 NULL буквально означает, что значение неизвестно или не определено. Употребление NULL позволяет проектировщику базы данных отличать данные, сознательно введенные как ноль, и данные, значение которых не зафиксировано в системе или намеренно проставлено как неопределенное. На самом деле, NULL – указатель на отсутствие значения в поле!

Побочным эффектом такой неопределенной природы значения NULL является то, что NULL не используется в вычислениях и сравнениях. Любая арифметическая или логическая операция с полем, содержащим NULL дает в результате NULL.

Вот несколько коротких, но очень важных правил из стандарта ANSI, которые необходимо помнить при работе с неопределенными значениями в выражениях SQL:

• Значение NULL нельзя вставить в поле, определенный как NOT NULL.

• Значения NULL не равны между собой. Ожидать, что два значения NULL окажутся равны при сравнении двух полей – это очень частая ошибка. (Правильным способом определения NULL в логическом выражении или во фразе WHERE является использование специальных фраз «IS NULL» и «IS NOT ULL»).

• Значения NULL игнорируются при вычислении агрегированных значений, таких как COUNT, AVG, SUM или MAX.

• Если поле, содержащее NULL, используется в запросе во фразе GROUP BY, то результат запроса содержит одну строку для всех значений NULL. Стандарт ANSI предписывает, что все найденные неопределенные значения должны попасть в одну группу.

• Фразы DISTINCT и ORDER BY, как и GROUP BY, также не различают значения NULL друг от друга. Про сортировку с помощью фразу ORDER BY в случае пустых значений (NULL) см. раздел. 3.4.3.

Важным элементом в работе с значением NULL является встроенная функция COALESCE(list)

COALESCE(list) возвращает первый отличный от NULL элемент списка. Например:

COALESCE(NULL, 1, 2) вернет значение 1, а функция

COALESCE(АДРЕС, ГОРОД, ‘Москва’) если значение полей АДРЕС и ГОРОД – NULL, вернет значение ‘Москва’, если же поля АДРЕС и/или ГОРОД заполнены, то функция вернет значение первого заполненного поля.

Если в каком-либо из полей содержится неопределенное значение NULL, порядок сортировки, выполняемой предложением ORDER BY, зависит от конкретной СУБД. В стандарте SQL:2003 вообще не определено, как следует упорядочивать значения NULL. В ORACLE и DB2 эти значения рассматриваются как большие любого определенного значения, поскольку этот фиктивный символ по определению имеет больший код ACSII, чем любой печатный символ. В то же время в ORACLE этот порядок можно изменить с помощью параметров NULLS FIRST и NULLS LAST, заданных в предложении ORDER BY.

3.4.13. Создание объектов СУБД

Создание объектов базы данных осуществляется с помощью операторов языка определения данных (DDL).

Таблицы базы данных создаются с помощью команды CREATE TABLE. Эта команда создает пустую таблицу, то есть таблицу, не имеющую записей. Новые записи в таблицу вводятся с помощью команды INSERT. Команда CREATE TABLE определяет имя таблицы и множество поименованных полей. Для каждого поля должен быть определен тип и размер. Каждая создаваемая таблица должна иметь, по крайней мере, два поля. Синтаксис команды CREATE TABLE в простейшем случае имеет следующий вид:

CREATE [{LOCAL TEMPORARY| GLOBAL TEMPORARY}] TABLE
имя_таблицы
(имя_поля тип_данных атрибуты [, ...]) |
[имя_поля WITH OPTIONS опции] |
[CONSTRAINT тип_ограничения [имя_ограничения][, ...]]

Используемые в SQL типы данных как минимум поддерживают стандарты ANSI (см. раздел 3.3)

Создание других объектов БД будут рассмотрены в разделах, посвященных этим объектам:

- Представления – раздел 3.4.15, 4.6.2
- Индексы – раздел 4.10.1
- Триггеры – раздел 4.6.4

Примеры использования операторов создания объектов БД приведены в приложении В.

3.4.14. Изменение и удаление объектов СУБД

Оператор ALTER TABLE позволяет модифицировать таблицу без удаления существующих индексов, триггеров и разрешений. Для оператора ALTER TABLE используется следующий синтаксис:

ALTER TABLE имя_таблицы
[ADD [COLUMN] имя_поля тип_данных атрибуты]
| [ALTER [COLUMN] имя_поля SET DEFAULT значение]
| [ALTER [COLUMN] имя_поля DROP DEFAULT]
| [ALTER [COLUMN] имя_поля ADD SCOPE имя_таблицы]
| [ALTER [COLUMN] имя_поля DROP
SCOPE {RESTRICT | CASCADE}]
| [DROP [COLUMN] имя_поля {RESTRICT | CASCADE}]
| [ADD ограничение]
| [DROP CONSTRAINT имя_ограничения {RESTRICT | CASCADE}]

Чтобы удалить существующую таблицу, необходимо предварительно удалить все данные из этой таблицы, то есть сделать ее пустой. Таблица, имеющая записи, не может быть удалена.

Синтаксис команды, осуществляющей удаление пустой таблицы, имеет следующий вид:

DROP TABLE имя таблицы

Общий синтаксис оператора DROP

DROP [тип_объекта] имя_объекта {RESTRICT | CASCADE}

[тип_объекта] одно из перечисленных ключевых слов: DOMAIN, FUNCTION, METHOD, PROCEDURE, ROLE, SCHEMA, TABLE, TRIGGER, TYPE, VIEW

RESTRICT | CASCADE

Позволяет предотвратить выполнение оператора DROP при наличии зависимых объектов (RESTRICT), либо позволяет одновременно удалить как сам объект, так и все зависимые объекты (CASCADE). Эта опция недопустима для некоторых типов объектов, например, для DROP TRIGGER, а в некоторых случаях она обязательна, например, для DROP SCHEMA. В частности, DROP SCHEMA RESTRICT удалит только пустую схему, а если в схеме есть объекты, то оператор будет прерван. А DROP SCHEMA CASCADE удалит и схему, и все содержащиеся в ней объекты.

3.4.15. Представления в языке SQL

Представление – это виртуальная таблица, не хранящая данные физически, а создаваемая динамически из SQL запроса каждый раз, когда происходит обращение к представлению. Представления позволяют создавать несколько разных образов одних и тех же исходных данных для разных пользователей без необходимости модифицировать структуру хранения данных.

Некоторые производители СУБД поддерживают объекты, называемые материализованными представлениями. Но не попадитесь на схожесть названий: материализованные представления не подчиняются тем же правилам, что и стандартные ANSI представления. Здесь мы будем обсуждать только ANSI представления.

С точки зрения синтаксиса языка SQL представление не отличается от таблицы и может быть использовано везде, где используется таблица, за исключением изменения данных в представлении – оно невозможно. Пользователь чаще всего не знает, откуда извлекает значения: из таблицы или представления. Представления хранятся в СУБД в виде скомпилированных запросов, которые динамически наполняют данными виртуальные таблицы, находящиеся в оперативной памяти, в отличие от таблиц, представления не занимают физическое пространство диска.

Представления используются для разных целей. К примеру, в представлении можно скомбинировать данные из множества таблиц в удобном для пользователя виде. Также с их помощью можно реализовать определенные правила системы безопасности, открывая для пользователя только указанные горизонтальные и вертикальные срезы данных.

В стандарте SQL:2003 определен следующий синтаксис оператора CREATE VIEW, который создает представление.

CREATE [RECURSIVE] VIEW имя_представления [(поле [, ...])]

AS оператор_select [WITH [CASCADED|LOCAL] CHECK OPTION]

Ключевое слово RECURSIVE указывает, что представление является рекурсивным, то есть извлекает данные из самого себя.

Опции WITH [CASCADED | LOCAL] CHECK OPTION используются только с обновляемыми представлениями. Проверяет, что через представление могут быть вставлены, изменены или удалены только те данные, которые доступны представлению. В этой книге мы не будем рассматривать обновляемые представления. Листинг 3.4.3 демонстрирует создание представления которое выдает список отделов и их начальников.

Листинг 3.4.3. Создание представления

```
CREATE VIEW НАЧАЛЬНИКИ
```

```
AS SELECT С.ФИО, С."НОМЕР_СТ", О."НАЗВ_ОТ", О."НОМЕР_ОТ"
FROM "СОТРУДНИКИ" С, "ОТДЕЛЫ" О
WHERE О."НАЧАЛЬНИК" = С."НОМЕР_СТ"
```

Далее, на листинге 3.4.4, показаны различные варианты применения представления, т.е., как было сказано ранее, представление употребимо везде, где возможно употребление таблицы.

Листинг 3.4.4. Применение представления

```
SELECT "ФИО", "НАЗВ_ОТ"
FROM НАЧАЛЬНИКИ
WHERE "ФИО" = 'Штейн И.Я.';

SELECT "ФИО", "НАЗВ_ОТ"
FROM НАЧАЛЬНИКИ НЧ
WHERE НЧ."НОМЕР_ОТ" = 3;

SELECT С.ФИО, С."ОТДЕЛ", COALESCE((SELECT '*'
     FROM НАЧАЛЬНИКИ НЧ
     WHERE НЧ."НОМЕР_СТ" = С."НОМЕР_СТ"), ' ') AS "НАЧ"
FROM "СОТРУДНИКИ" С
ORDER BY 2, 1;
```

Последний запрос в листинге 3.4.4 требует комментария: в этом запросе выводятся все сотрудники по отделам, а начальники отделов помечены «звездочкой». Работа этого запроса представлена на рис. 3.23.

Worksheet | Query Builder

```
SELECT C.ФИО, C."ОТДЕЛ", COALESCE((SELECT '*'
                    FROM НАЧАЛЬНИКИ НЧ
                    WHERE НЧ."НОМЕР_СТ" = C."НОМЕР_СТ"), ' ') AS "НАЧ"
FROM "СОТРУДНИКИ" C
ORDER BY 2, 1;
```

Script Output | Query Result

SQL | All Rows Fetched: 9 in 0 seconds

	ФИО	ОТДЕЛ	НАЧ
1	Сусанин И.И.	1	
2	Фомин М.М.	1	
3	Штейн А.Я.	1	*
4	Бауман Ю.И.	2	
5	Грязнов Г.Г.	2	
6	Штейн И.Я.	2	*
7	Камнеедов А.П.	3	
8	Невструев Я.П.	3	
9	Фильkенштейн А.Я.	3	*

Рис. 3.23. Применение представлений

Представления можно создавать на основе других представлений, хотя такой подход не рекомендуется и считается плохой практикой. В зависимости от платформы такие представления могут дольше компилироваться, но производительность таких представлений будет такой же, как и при операциях над базовыми таблицами.

3.4.16. Добавление данных. Оператор INSERT

Для добавления в таблицу новых используют оператор INSERT. Синтаксис этого оператора несколько отличается в стандарте SQL:2003 и отдельных СУБД. В то же время можно привести такой вид синтаксиса, который будет работать во всех трех ведущих СУБД и не противоречить стандарту SQL:2003:

INSERT INTO имя_таблицы [(имя_поля[, ...])]
{{VALUES (<литерал> | <выражение> | NULL | DEFAULT [, ...]}) |
{<оператор_SELECT>}}

Оператор INSERT может добавлять, как одну запись, так и целую группу записей, например (см. Листинг 3.4.5):

Листинг 3.4.5. Добавление единичных записей в таблицу

```
INSERT INTO "ОТДЕЛЫ" ("НОМЕР_ОТ", "НАЗВ_ОТ","НАЧАЛЬНИК")
VALUES (4, 'Внутреннего контроля', NULL)
INSERT INTO "РАБОТЫ" ("НОМЕР_РБ", "НАЗВАНИЕ","ОБЪЕМ", "ГЛАВНЫЙ")
```

```
VALUES (8, 'Всеобщая вакцинация', 700, 4)
```

Здесь была добавлена одна запись в таблицу «ОТДЕЛЫ» и одна запись в таблицу «РАБОТЫ». В следующем примере (см. Листинг 3.4.6) добавляется сразу несколько записей:

Листинг 3.4.6. Добавление группы записей в таблицу

```
INSERT INTO "СПИСКИ_СТ" ("НОМЕР_СП", "СОТР", "РАБОТА")
SELECT NULL, "СОТРУДНИКИ"."НОМЕР_СТ",
 (SELECT "РАБОТЫ"."НОМЕР_РБ"
  FROM "РАБОТЫ"
  WHERE "РАБОТЫ"."НАЗВАНИЕ" = 'Всеобщая вакцинация')
FROM "СОТРУДНИКИ", "ОТДЕЛЫ"
WHERE "СОТРУДНИКИ"."ОТДЕЛ" = "ОТДЕЛЫ"."НОМЕР_ОТ"
  AND "ОТДЕЛЫ"."НАЗВ_ОТ" = 'Протокольных мероприятий';
```

В этом примере (см. Листинг 3.4.6) все сотрудники отдела «Протокольных мероприятий» записываются в исполняющие работу «Всеобщая вакцинация» посредством добавления соответствующих записей в таблицу «Списки сотрудников» (СПИСКИ_СТ). Необходимо уточнить, что для правильного выполнения этого оператора необходим триггер (см. Листинг 3.4.7.) заполняющий значение первичного ключа таблицы «СПИСКИ_СТ» (о триггерах см. раздел 4.6.4.) и последовательность (НОМЕР_СП_SEQ) для создания уникального суррогатного ключа.

Листинг 3.4.7. Триггер для заполнения поля первичного ключа

```
CREATE OR REPLACE TRIGGER IN_СПИСКИ BEFORE INSERT ON СПИСКИ_СТ
 FOR EACH ROW WHEN (NEW.НОМЕР_СП IS NULL)
BEGIN
  : NEW.НОМЕР_СП := НОМЕР_СП_SEQ.NEXTVAL;
END;
```

Примеры использования операторов вставки данных и триггеров заполнения первичных ключей см. в приложении В.

3.4.17. Изменение данных. Оператор UPDATE

Для изменения данных таблицы предназначен оператор UPDATE. Ниже приведен общий синтаксис оператора UPDATE, описанный в стандарте SQL:2003.

UPDATE имя_таблицы
SET имя_поля = {выражение | литерал | NULL | DEFAULT}[, ...].
[WHERE условие_отбора_изменяемых_записей]

Предложение WHERE очень важное, и относиться к нему нужно с предельным вниманием. Если случайно его опустить или указать в нем ошибочное условие, то указанные поля во многих или во всех записях таблицы могут быть ошибочно заполнены одним и тем же значением. Перед изменением выполните оператор SELECT с тем же фильтром в WHERE, это позволит вам убедиться, что вы собираетесь изменить именно те записи, которые нужно.

Пример изменения единичной записи на листинге 3.4.8. В этом примере отделу «Внутреннего контроля» назначается начальник.

Листинг 3.4.8. Изменение одной записи

```
UPDATE "ОТДЕЛЫ"
SET "НАЧАЛЬНИК" = (SELECT "СОТРУДНИКИ"."НОМЕР_СТ"
           FROM "СОТРУДНИКИ"
           WHERE "СОТРУДНИКИ"."ФИО" = 'Фомин М.М.')
WHERE "НАЗВ_ОТ" = 'Внутреннего контроля'
```

На следующем листинге (3.4.9.) показано изменение группы записей, здесь все сотрудники отдела «Протокольных мероприятий» переводятся в отдел «Внутреннего контроля»

Листинг 3.4.9. Изменение нескольких записей

```
UPDATE "СОТРУДНИКИ"
SET "ОТДЕЛ" = (SELECT "ОТДЕЛЫ"."НОМЕР_ОТ"
      FROM "ОТДЕЛЫ"
      WHERE "ОТДЕЛЫ"."НАЗВ_ОТ" = 'Внутреннего контроля')
WHERE "ОТДЕЛ" = (SELECT "ОТДЕЛЫ1"."НОМЕР_ОТ"
      FROM "ОТДЕЛЫ" "ОТДЕЛЫ1"
      WHERE "ОТДЕЛЫ1"."НАЗВ_ОТ" = 'Протокольных мероприятий')
```

Обратите внимание, что во всех вышеприведенных листингах производится изменение значений вторичных ключей и это выполняется копированием нужного значения первичного ключа при этом само значение первичного ключа не упоминается, т.к. оно неизвестно.

3.4.18. Удаление данных. Оператор DELETE

Оператор DELETE можно по праву назвать самым простым в языке DML. В ней нужно указать только таблицу, в которой будут удаляться записи, и (что необязательно) критерий отбора записей для удаления. Однако пусть простота синтаксиса оператора не вводит вас в заблуждение. Оператор DELETE при неуважительном к ней отношении способен вызвать большие разрушения в базе данных. К примеру, если опустить в операторе предложение WHERE, вся доступная в таблице информация будет удалена. Если условию WHERE не соответствует ни одна запись таблицы, ничего удалено не будет, при этом сообщение об ошибке не генерируется. Очень часто некорректно сформированное условие отбора удаляемых записей приводит к необходимости их восстановления, что требует больших затрат времени. Итак, синтаксис оператора DELETE:

DELETE FROM имя_таблицы
[WHERE условие_отбора_удаляемых_записей]

Оператор DELETE достаточно редко выполняется без фразы WHERE, так как в этом случае удаляются все записи таблицы. Перед удалением выполните оператор SELECT с тем же фильтром в WHERE - это позволит вам убедиться, что вы удаляете те записи, которые нужно.

При удалении записей следует внимательно относиться к ограничениям ссылочной целостности (см. раздел 4.8.1). К примеру, вам не удастся удалить запись, на которую ссылаются внешние ключи других таблиц, если, конечно, при создании таблицы не был применен параметр ON DELETE CASCADE. В последнем случае вместе с удаляемой из главной таблицы записью будут удалены и все ссылающиеся на нее записи других таблиц. Такой режим работы опасен,

особенно при некорректно сформированном условии WHERE, так что относитесь к параметру ON DELETE CASCADE с осторожностью. Еще об удалении данных см. в главе «Проектирование БД», раздел 4.8.3.

3.4.19. Защита данных в SQL

Как уже отмечалось в разделе 3.2.7. в язык SQL сразу были встроены средства для разграничения доступа пользователей БД.

Любая СУБД фактически является набором объектов: схем, таблиц, представлений, процедур и т.д., а также процессов, управляющих этими объектами. Ограничение доступа к этим объектам — важный инструмент защиты, реализуемый на уровне SQL с помощью системы привилегий.

Привилегиями называют права конкретного пользователя на доступ, создание, редактирование и удаление объектов базы данных, а также на администрирование базы данных. Привилегии назначаются пользователю и/или роли (концепция роли будет описана далее).

В основе применяемого разграничения доступа пользователей к объектам БД лежит модель дискреционного управления доступом (discretionary access control) — разграничение доступа между поименованными субъектами и поименованными объектами. На языке SQL это выглядит так - оператор GRANT предоставляет поименованному субъекту (пользователю БД) привилегии для работы с объектами БД (таблицы, представления, встроенные процедуры и т.д.):

GRANT {{ALL [PRIVILEGES] | объектная_привилегия[, ...]}| роль [, ...]}
[ON объект_базы_данных]
[TO пользователь [, ...]]
[WITH GRANT OPTION]
[WITH ADMIN OPTION]

Последние две строчки уточняют предоставленные привилегии:

WITH GRANT OPTION

Позволяет пользователю передавать дальше полученные привилегии.

WITH ADMIN OPTION

Дает пользователю возможность назначать полученную роль другим пользователям. Среди привилегий могут быть: SELECT, INSERT, DELETE, UPDATE, ALTER, EXECUTE и т.д.

Выполнение некоторых действий не требует явного разрешения и доступно по умолчанию. Эти действия могут быть выполнены только членами ролей сервера или владельцами объектов в базе данных. Неявные права не предоставляются пользователю непосредственно, он получает их лишь при определенных обстоятельствах. Например, пользователь может стать владельцем объекта базы данных, только если сам создаст объект, либо если кто-то другой передаст ему право владения своим объектом. Таким образом, владелец объекта автоматически получит права на выполнение любых действий с объектом, в том числе и на предоставление доступа к объекту другим пользователям. Эти права нигде не указываются, выполнять любые действия позволяет только факт владения объектом.

Если возникает необходимость, у пользователя можно отозвать доступ к объектам БД, тогда аннулируются все разрешения на доступ, полученные им на любом уровне иерархии. При этом гарантируется, что доступ останется запрещенным независимо от разрешений, предоставленных на более высоком уровне. Для отзыва доступа к объектам БД используется команда:

REVOKE [GRANT OPTION FOR]
{ALL [PRIVILEGES] | привилегия [, ...]}
ON объект_базы_данных
TO | FROM {пользователь | роль} [, ...]
[CASCADE]

Ключевое слово CASCADE указывает на необходимость отозвать права не только у конкретного пользователя, но также и у всех тех, кому он предоставил аналогичные права.

Более подробно о доступе пользователей к объектам БД и ролях пользователей см. в разделе 4.7.

3.5. Функции в SQL

Функции SQL серьезно облегчают жизнь программистов и в некоторой степени вносят элемент процедурности в изначально основанный на множествах язык SQL. В SQL функция – это частный случай команды. В каждом диалекте SQL реализован свой набор таких команд. По существу, функция представляет собой команду, название которой состоит из одного слова и которая возвращает единственное значение. Значение функции зависит от входных параметров. Например, функция может вычислять среднее арифметическое для списка значений, выбранных из базы данных. Однако у многих функций входных параметров нет вообще. Примером может служить функция CURRENT_TIME возвращающая текущее время. Сам запрос предназначен для извлечения информации, а функции, в нем задействованные, для проверки и преобразования данных, выполнения вычислений, получения системной информации и других целей. Функции SQL лучше всего рассматривать как средство выполнения отдельных узкоспециализированных задач, и к примеру вычисления квадратного корня, или преобразования строчных букв в прописные. Функции как правило вызываются в запросе для каждой записи результирующего набора данных. От прочих выполняемых модулей СУБД функции отличает то, что они всегда возвращают значение.

В стандарте ANSI описан целый ряд полезных функций. В этой главе мы рассмотрим некоторые из них, наиболее часто употребимые. Дополнительно любая СУБД поддерживает множество собственных функций, не вошедших в стандарт.

3.5.1. Классификация функций

Можно выделить следующие группы функций упомянутых в стандарте ANSI:

- Строковые функции. Используются для преобразования текстовых строк, например, для обрезания или заполнения значений.
- Математические функции. Используются для выполнения математических операций над числовыми данными. К математическим функциям относятся функции возвращающие абсолютные значения, синусы и косинусы углов, квадратный корень числа и т.д. Используются они только для алгебраических, тригонометрических и геометрических вычислений.
- Агрегатные функции. Используются для получения итоговых данных по таблицам, например, когда надо просуммировать какие-либо данные без их выборки.
- Функции даты и времени. Используются для преобразования значений даты и времени, и для арифметических действий с датами, например, для возвращения разницы между датами.
- Системные функции. Возвращают служебную информацию СУБД.

Надо отметить, что это деление условно, и есть еще несколько различных критериев по которым можно провести классификацию, например, см. раздел 3.5.7. Кроме того некоторые функции работают с данными разных типов и поэтому попадают в различные группы функций, например, функция TRANC(D,F), кроме работы с числами, усекает значение даты D согласно заданному шаблону - F.

3.5.2. Строковые функции

SQL предоставляет вам широкий набор функций для манипулирования со строковыми данными (конкатенация строк, CHR, LENGTH, INSTR и другие). Список основных функций для обработки строковых данных приведен в таблице 3.2.

Таблица 3.2. Функции SQL для обработки строк

Функция	Описание
CHR(N)	Возвращает символ ASCII кода для десятичного кода N
ASCII(S)	Возвращает десятичный ASCII код первого символа строки
INSTR(S1,S2,pos[,N])	Возвращает позицию строки S2 в строке S1 большую или равную pos. N - число вхождений
LENGHT(S)	Возвращает длину строки
LOWER(S)	Заменяет все символы строки на прописные символы
INITCAP(S)	Устанавливает первый символ каждого слова в строке на заглавный, а остальные символы каждого слова на прописные
SUBSTR(S,pos,[,len])	Выделяет в строке S подстроку длиной len, начиная с позиции pos
UPPER(S)	Преобразует прописные буквы в строке на заглавные буквы
LPAD(S,N[,A])	Возвращает строку S, дополненную слева символами A до числа символов N. Символ - наполнитель по умолчанию - пробел
RPAD(S,N[,A])	Возвращает строку S, дополненную справа символами A до числа символов N. Символ - наполнитель по умолчанию - пробел
LTRIM(S,[S1])	Возвращает усеченную слева строку S. Символы удаляются до тех пор, пока удаляемый символ входит в строку - шаблон S1 (по умолчанию - пробел)
RTRIM(S,[S1])	Возвращает усеченную справа строку S. Символы удаляются до тех пор, пока удаляемый символ входит в строку - шаблон S1 (по умолчанию - пробел
TRANSLATE(S,S1,S2)	Возвращает строку S, в которой все вхождения строки S1 замещены строкой S2. Если S1 <> S2, то символы, которым нет соответствия, исключаются из результирующей строки
REPLACE(S,S1,[,S2])	Возвращает строку S, для которой все вхождения строки S1 замещены на подстроку S2. Если S2 не указано, то все вхождения подстроки S1 удаляются из результирующей строки
NVL(X,Y)	Если X есть NULL, то возвращает в Y либо строку, либо число, либо дату в зависимости от исходного типа Y

Названия одних и тех же функций могут отличаться в различных СУБД. Так, например, функция СУБД ORACLE SUBSTR(S, pos, [, len]) в СУБД SQLBase называется @SUBSTRING(S, pos, len)

3.5.3. Математические функции

SQL поддерживает полный набор арифметических операций и математических функций для построения арифметических выражений над колонками базы данных (+, -, *, /, ABS, LN, SQRT и т.д.). Список основных встроенных математических функций дан ниже в таблице 3.3.

Таблица 3.3. Математические функции SQL

Функция	Описание
ABS(X)	Возвращает абсолютное значение числа X
ACOS(X)	Возвращает арккосинус числа X
ASIN(X)	Возвращает арксинус числа X
ATAN(X)	Возвращает арктангенс числа X
COS(X)	Возвращает косинус числа X
EXP(X)	Возвращает экспоненту числа X

SIGN(X)	Возвращает -1, если X<0,0, если X=0, +1, если X>0
LN(X)	Возвращает натуральный логарифм числа X
MOD(X,Y)	Возвращает остаток от деления X на Y
CEIL(X)	Возвращает наименьшее целое, большее или равное X
ROUND(X,n)	Округляет число X до числа с n знаками после десятичной точки
SIN(X)	Возвращает синус числа X
SQRT(X)	Возвращает квадратный корень числа X
TAN(X)	Возвращает тангенс числа X
FLOOR(X)	Возвращает наибольшее целое, меньшее или равное X
LOG(a,X)	Возвращает логарифм числа X по основанию A
SINH(X)	Возвращает гиперболический синус числа X
COSH(X)	Возвращает гиперболический косинус числа X
TANH(X)	Возвращает гиперболический тангенс числа X
TRANC(X,n)	Усекает число X до числа с n знаками после десятичной точки
POWER(A,X)	Возвращает значение A, возведенное в степень X

Набор встроенных функций может изменяться в зависимости от версии СУБД одного производителя и также в СУБД различных производителей. Так, например, в СУБД SQLBase, есть функция @ATAN2(X,Y), которая возвращает арктангенс Y/X, но отсутствует функция SIGN(X).

3.5.4. Агрегатные функции

В различных СУБД их называют также итоговыми, аналитическими и табличными, что порождает путаницу в некоторых публикациях. В отличии от остальных, упомянутых в разделе 3.5 функций, агрегатные функции обрабатывают таблицы или группы записей целиком, т.е. на входе функции – выражение, содержащее имена полей таблицы и вычисляемое для каждой записи таблицы, а на выходе - значение, подсчитанное на основании всех записей таблицы или группы записей.

Таблица 3.4. Агрегатные функции

Функция	Описание
AVG(X) = AVG(ALL X) AVG(DISTINCT X)	**Вычисляет среднее значение выражения X, вычисляемого для каждой записи таблицы. Указанное выражение может содержать любое поле таблицы любого типа. Нуль-значения игнорируются, ключевое слово DISTINCT подавляет дубликаты**
COUNT(*) COUNT(X) = COUNT(ALL X) COUNT(DISTINCT X)	**Вычисляет числа отдельных значений в столбце, при этом нуль-значения игнорируются. При указании * всегда возвращается число строк в таблице. Указание DISTINCT подавляет дубликаты**
MAX(X) = MAX(ALL X) MAX (DISTINCT X)	**Вычисляет максимальное значение выражения X, вычисляемого для каждой записи таблицы. Указанное выражение может содержать любое поле таблицы любого типа. Нуль-значения игнорируются, ключевое слово DISTINCT подавляет дубликаты**
MIN(X) = MIN(ALL X) MIN (DISTINCT X)	**Вычисляет минимальное значение выражения X, вычисляемого для каждой записи таблицы. Указанное выражение может содержать любое поле таблицы любого типа. Нуль-значения игнорируются, ключевое слово DISTINCT подавляет дубликаты**
SUM(X) = SUM(ALL X) SUM (DISTINCT X)	**Вычисляет сумму значений выражения X, вычисляемого для каждой записи таблицы. Указанное выражение может содержать любое поле таблицы любого типа. Нуль-значения игнорируются, ключевое слово DISTINCT подавляет дубликаты**
STDDEV([DISTINCT|ALL]X)	**Вычисляет стандартное отклонение на множестве значений выражения X, вычисляемого для каждой записи таблицы. Указанное выражение может содержать любое поле таблицы любого типа. Нуль-значения игнорируются, ключевое слово DISTINCT подавляет дубликаты**
VAR_POP ([DISTINCT|ALL]X)	**Вычисляет дисперсию генеральной совокупности, состоящей из набора чисел, представленных выражением X, вычисляемого для каждой записи таблицы. Указанное выражение может содержать любое поле таблицы любого типа. Нуль-значения игнорируются, ключевое слово DISTINCT подавляет дубликаты**

Агрегатные функции возвращают единственное значение, основанное на некотором вычислении в группе значений, определяемых предложением GROUP BY оператора SELECT, при отсутствии предложения GROUP BY в качестве группы берется вся таблица (см. рис. 3.24).

Worksheet Query Builder

```
SELECT  НАЗВ_ОТ Отдел,COUNT(*) "Кол. Сотр", ROUND(AVG(зарплата),0) Средняя,
MAX(зарплата) Максимальная, MIN(зарплата)Минимальная
FROM отделы, сотрудники
WHERE сотрудники.отдел = отделы.номер_от
GROUP BY назв_от;
```

Query Result

All Rows Fetched: 3 in 0,006 seconds

	ОТДЕЛ	Кол. Сотр	СРЕДНЯЯ	МАКСИМАЛЬНАЯ	МИНИМАЛЬНАЯ
1	Протокольных мероприятий	3	89667	114000	75000
2	Внешних коммуникаций	3	84000	112000	60000
3	Внутренних инсинуаций	3	95333	113000	60000

Worksheet Query Builder

```
SELECT COUNT(*) "Кол. Сотр", ROUND(AVG(зарплата),0) Средняя,
MAX(зарплата) Максимальная, MIN(зарплата) Минимальная
FROM сотрудники;
```

Query Result

All Rows Fetched: 1 in 0 seconds

	Кол. Сотр	СРЕДНЯЯ	МАКСИМАЛЬНАЯ	МИНИМАЛЬНАЯ
1	12	82833	114000	60000

Рис. 3.26. Использование агрегатных функций

3.5.5. Функции даты и времени

В различных СУБД функции для работы с датами имеют мало общего. Самый минимум функций в СУБД Oracle (см. таблицу 3.5).

Таблица 3.5. Функции для работы с датами

Функция	Описание
CURRENT_DATE	Возвращает текущую дату
TRUNC(D, F)	Возвращает дату **D**, усеченную до единицы измерения времени, заданной параметром **F**. Если параметр **F** опущен, производится усечение до ближайшего дня
ADD_MONTHS(D,M)	Возвращает результат прибавления **M** месяцев к дате **D**
LAST_DAY(D)	Возвращает дату последнего дня месяца, содержащего дату **D**
MONTHS_BETWEEN(D1,D2)	Возвращает количество месяцев между датами **D1** и **D2**. Если **D1** позже **D2**, то результат положителен, иначе отрицателен

В других СУБД есть аналогичные функции, порой отличающиеся синтаксисом, например, для функции trunc(d,f) в PostgreSQL есть аналогичная функция date_trunc(f,d), а в MySQL для этих целей может использоваться функция date_format(d,f), но она возвращает результат в виде строки.

3.5.6. Системные функции.

Системные функции, как было сказано выше, предоставляют доступ к некоторой информации связанной с функционированием СУБД, например, функция UID в СУБД ORACLE возвращает целое число, уникально идентифицирующее пользователя, владеющего текущим сеансом. Пример использования:

SELECT UID FROM DUAL _> 47

Естественно, системные функции не стандартизированы и зависят от конкретной СУБД, с которой вы работаете, и описаны в документации на конкретную платформу.

3.5.7. Детерминированные и недетерминированные функции

Функция может быть детерминированной или недетерминированной. Детерминированная функция возвращает один и тот же результат при вызове с одинаковыми значениями входных параметров. Недетерминированная функция может возвращать разные результаты при каждом вызове, даже если на вход подаются одни и те же значения.

Почему так важно, что при одинаковых входных параметрах возвращается один и тот же результат? Это важно, чтобы понимать, как использовать функции в представлениях, внутри определяемых пользователем функций или в хранимых процедурах. Ограничения на использование недетерминированных функций различны в разных СУБД. Например, СУБД SQL Server позволяет строить индекс по столбцу выражению только в случае, когда выражение не содержит недетерминированных функций. При репликации используемые в транзакциях функции должны быть детерминированными, иначе состояние реплики разойдется с состоянием основного сервера. Запомните - Все функции конфигурации, курсоров, метаданных, безопасности и системные функции являются недетерминированными. Правила работы с функциями, действующие на конкретной платформе, должны быть описаны в прилагаемой к ней документации.

4. Проектирование БД

В настоящее время нельзя представить себе информационную систему (ИС), в ядре которой нет базы данных (БД), встроенной или внешней, содержащей все необходимые данные для работы ИС и сохраняющую все результаты работы. От того как спроектирована эта БД во многом зависят все важные параметры ИС, начиная с быстродействия и кончая надежностью. В свете этого, особенно важно уметь проектировать БД таким образом, чтобы обеспечить максимальные характеристики ИС.

Опытные разработчики хорошо знают, что в начале каждого проекта по созданию приложения встает выбор между следующими альтернативами:

- Тщательное проектирование структуры базы данных и запросов на основе анализа требований;
- Использование программных средств, обеспечивающих быструю разработку приложений.

Известно, что первый вариант обеспечивает высокое качество и высокую производительность, однако требует дополнительных временных затрат. Второй вариант позволяет достичь быстрого получения результатов и низкую стоимость, достигаемые за счет многочисленных компромиссов, ухудшающих качество и производительность проектируемой ИС. В подавляющем большинстве случаев, к сожалению, выбирается второй вариант.

Проектирование БД всегда было и остается творческой задачей, скорее ремеслом, искусством, нежели строго алгоритмизированной работой.

По сути проектирование БД — это создание информационной модели предметной области на необходимом уровне абстракции в терминах сущностей (таблиц) и их экземпляров, связей между ними и правил согласованности данных.

В этой главе мы обсудим основные принципы и приемы проектирования БД, которые практикуются в настоящее время.

4.1. Исторический экскурс

Почти 50 лет прошло с тех пор, как Эдгар Кодд опубликовал свою статью, положившую начало реляционным БД. Он показал, что любое представление данных может сводится к совокупности двумерных таблиц, которые он назвал отношениями - relation (англ.). В настоящее время реляционные базы данных являются самыми распространенными, а реляционные СУБД проникли во все сферы жизни человечества. Мы будем далее говорить именно о реляционных базах данных, хотя многие положения справедливы и для других моделей данных.

Вместе с развитием СУБД развивались и изменялись подходы к проектированию БД: сначала методы проектирования базировались на представлении, что СУБД сама по себе является информационной системой, а пользователь обращается к ней на языке SQL, самостоятельно формируя необходимые запросы. Именно поэтому язык SQL проектировался как непроцедурный, декларативный и максимально приближенный к простому английскому.

В большинстве случаев проектирование рассматривается независимо от приложения. Иначе говоря, интерес представляют сами данные, а не то, как они будут использоваться. Независимость от приложения в этом смысле желательна по той простой причине, что в момент проектирования базы данных обычно еще неизвестны все возможные способы использования ее данных. [1]

При таком подходе БД должна содержать как можно больше данных, относящихся к какой-либо предметной области, при этом структура БД с точки зрения работы пользователя не оптимизируется, т.к. нет практических критериев оптимизации – нет ясных алгоритмов работы пользователя, нет предпочтительных запросов, неизвестна цель пользователя при работе с данными. Можно предложить только абстрактные, теоретические методы проектирования БД,

которые, будут приводить БД к виду, обеспечивающему минимальную логическую избыточность, позволяющую избежать некоторых аномалий обновления.

Нормализация — это разбиение таблицы на две или более, обладающих лучшими свойствами при включении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором каждый факт появляется лишь в одном месте, т. е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных. [2]

Когда заходит разговор о разбиении таблицы, сразу встает вопрос – откуда она берется, эта самая первая ненормализованная таблица? Вопрос не праздный и имеет вполне определенный ответ – в далекие времена зарождения реляционных БД, проектирование начиналось с определения всех атрибутов (данных) предметной области и из этих атрибутов и получалась первая таблица. Работать с такой таблицей, естественно, было нельзя, и вот тогда вступали в действия правила нормализации или нормальные формы, при помощи которых таблица разбивалась на части (на связанные между собой таблицы), с которыми уже можно было работать.

В реальном проектировании структуры базы данных применяется другой метод – так называемое семантическое моделирование. Семантическое моделирование представляет собой моделирование структуры данных, опираясь на смысл этих данных. В качестве инструмента семантического моделирования используются различные варианты диаграмм «сущность-связь» или ER-диаграмм (ER – Entity-Relationship).

Первый вариант модели сущность-связь был предложен в 1976 г. П. Ченом. В дальнейшем многими авторами были разработаны свои варианты подобных моделей (нотация Мартина, нотация IDEF1X, нотация Бейкера и др.). Различные программные средства, реализующие одну и ту же нотацию, могут отличаться своими возможностями, но все они используют графическое изображение сущностей предметной области, их свойств (атрибутов) и взаимосвязей между сущностями. Получаемые графические изображения называются диаграммами «сущность-связь». Мы будем использовать нотацию и методологию IDEF [19]. По диаграммам «сущность-связь» современные технологические средства могут автоматически генерировать программы, создающие схему БД для конкретной СУБД. При построении диаграмм «сущность-связь», приведенных в этой книге использовалась система ORACLE SQL Developer Data Modeler.

Если раньше СУБД были дорогими, редкими, сложными и громоздкими, то сейчас это не так. СУБД стали широко распространены, среди них есть системы с открытым кодом, такие как MySQL и PostgreSQL, так что цена тоже перестала быть значимым фактором. Теперь можно использовать промышленную СУБД в любой ИС, тем самым упрощая работу с данными. В настоящее время процесс проектирования БД не рассматривается в отрыве от процесса проектирования ИС, т.е. проектирование БД — это лишь часть проекта создания ИС. Как только БД стала частью ИС появились и ясные алгоритмы работы и цели пользователя, стали понятны основные запросы к БД, появились критерии объёмных и скоростных характеристик конкретной БД. Вместе с изменением идеологии использования, естественно изменились и принципы проектирования БД, о них мы и будем говорить в дальнейшем.

4.2. Простой пример. Фразеологический словарь

В разделе 3.4.1. мы создали первую таблицу «Фразы», попробуем создать теперь простейшую БД. Вспомним определение БД из раздела 1.1.2:

База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

И, поскольку таблица «Фразы» у нас уже есть, создадим БД для фразеологического словаря – БД должна хранить актуальные данные о фразеологизмах и поддерживать поиск и выдачу

фразеологизмов пользователю системы. Сначала ознакомимся с предметной областью – полистаем фразеологический словарь русского языка [20].

После изучения словаря сделаем некоторые выводы относительно содержимого БД:

1. Сам фразеологизм
2. Ко всем фразеологизмам в словаре дано подробное толкование.
3. Для некоторых фразеологизмов дается эмоционально-экспрессивная окраска, а также область применения (*шутл., неодобр., разг., книжн.* и пр.).
4. Для некоторых фразеологизмов приведены примеры употребления.
5. Для некоторых фразеологизмов указана история их происхождения.

Таким образом мы получим следующие поля в таблице «Фразы» (см. рис. 4.1.)

COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
ID	NUMBER	No	(null)	1	Первичный ключ
ФРАЗА	VARCHAR2(40 BYTE)	No	(null)	2	(null)
ЗНАЧЕНИЕ	VARCHAR2(256 BYTE)	No	(null)	3	(null)
ТЕГИ	VARCHAR2(20 BYTE)	Yes	(null)	4	(null)
ИСТОРИЯ	VARCHAR2(1024 BYTE)	Yes	(null)	5	(null)
ПРИМЕРЫ_ПРИМ	VARCHAR2(1024 BYTE)	Yes	(null)	6	(null)

Рис. 4.1. Поля таблицы «Фразы» из БД «Фразеологический словарь»

А саму таблицу вы можете видеть на рис. 4.2.

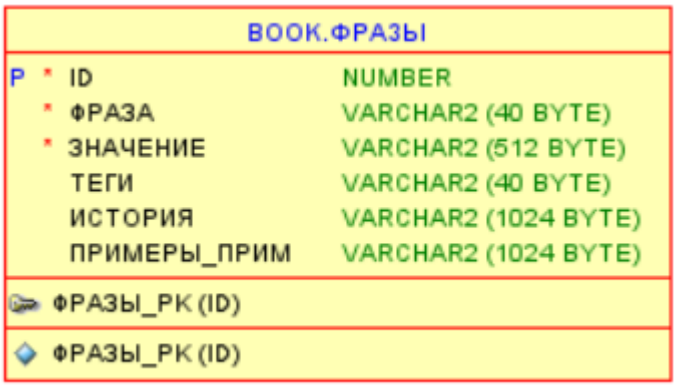

Рис. 4.2. Таблица «Фразы» из БД «Фразеологический словарь»

Еще одна функция фразеологического словаря – поиск фразеологизмов с похожим и\или противоположным смыслом. Для реализации этой функции нужна еще одна таблица – «список фраз с похожим смыслом». Теперь схема нашей БД будет такой как показано на рис. 4.3.

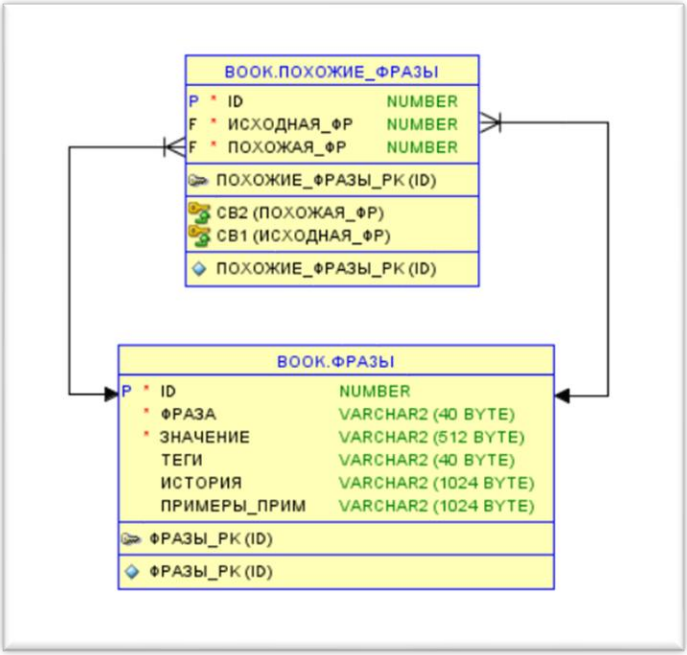

Рис. 4.3. БД «Фразеологический словарь»

Далее на рис. 4.4. показан простейший запрос, обращающийся к БД «Фразеологический словарь» и результат его выполнения.

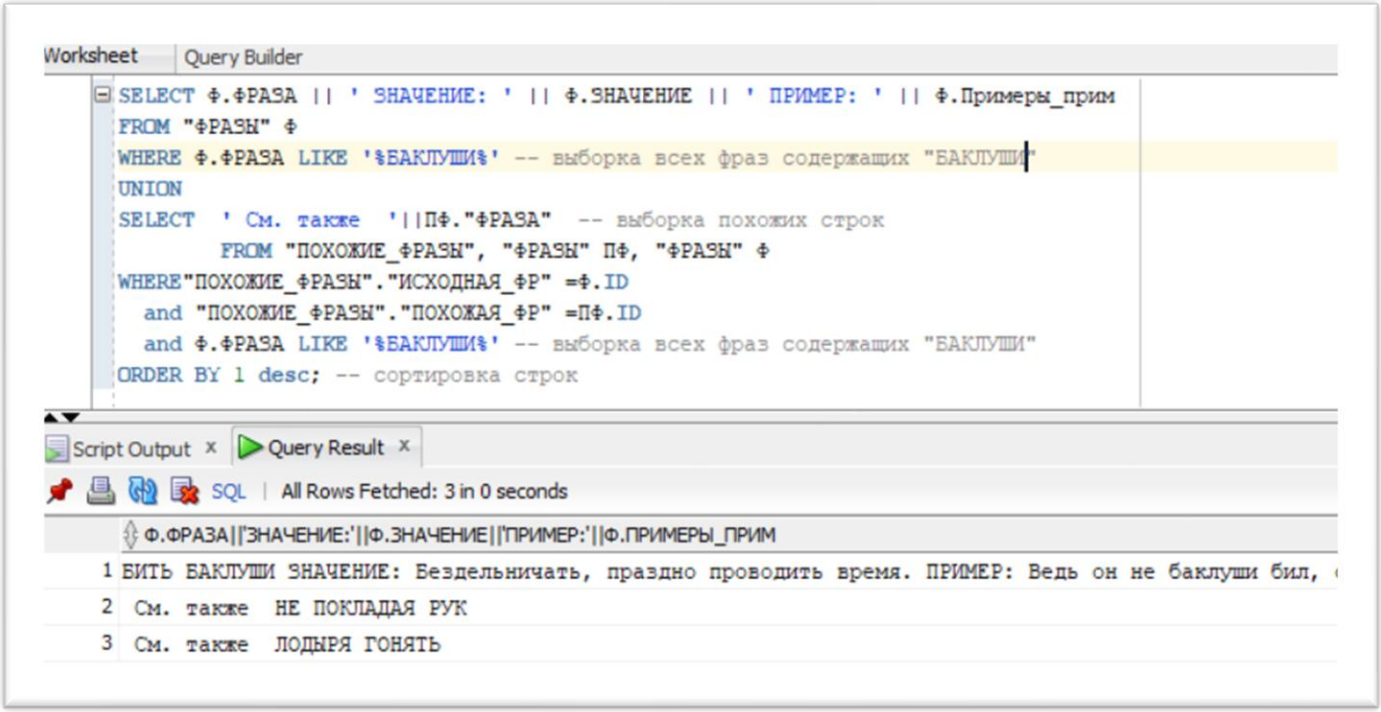

Рис. 4.4. Запрос к БД «Фразеологический словарь»

Конечно, в реальном приложении вместо строки '%БАКЛУШИ%' будет подставлено значение поля, в котором пользователь введет строку поиска нужной ему фразы, а в строку выдачи будут вставлены спец. символы форматирования, однако текст запроса останется примерно такой же как на рис. 4.4. Работу системы фразеологического словаря, построенного на такой схеме данных, можно посмотреть по ссылке:

http://rus-yaz.niv.ru/doc/phraseological-dictionary/articles/19/bit-baklushi.htm

4.3. Основные цели проектирования БД

Прежде чем начать проектирование надо определить основные цели, к которым разработчик должен стремиться в процессе проектирования. Обратите внимание, мы говорим здесь о проектировании БД, как о подзадаче в процессе проектирования ИС и часть требований к ИС непременно оказывает влияние на цели, методы и подходы к проектированию БД, например, проблемы согласованности данных сильно связаны с требуемым уровнем отказоустойчивости и быстродействия ИС.

Итак, основные цели проектирования БД:

1. Обеспечение хранения в БД всей необходимой информации.
2. Обеспечение возможности получения данных по всем необходимым запросам.
3. Сокращение избыточности и дублирования данных.
4. Обеспечение согласованности данных.

Надо понимать, что цель №1 должна быть достигнута непременно, а вот достижение остальных целей — это путь из компромиссов между достижением требуемых характеристик БД, ИС, мощностью аппаратуры, стоимостью, быстродействием, защищенностью и т.д.

Необходимо несколько слов сказать по каждой из целей:

С первой целью все ясно – старайтесь не упустить никаких сущностей и никаких атрибутов из предметной области – изучайте предметную область, станьте специалистом по предметной области, станьте продвинутым пользователем проектируемой ИС, станьте, наконец, администратором проектируемой ИС и тогда, может быть, вы ничего не упустите.

Вторая цель, на самом деле, о связях в базе данных – все сущности (таблицы) в БД должны быть связаны между собой так, чтобы понадобился всего один запрос, при помощи которого можно было достать любые данные необходимые для функционирования ИС.

Третья цель совсем не об экономии памяти, а о том, что *«каждая часть данных должна иметь единственное, непротиворечивое и авторитетное представление в рамках системы»* [5]. Принцип DRY. (Don’t repeat yourself (рус. *не повторяйся*) — это принцип разработки программного обеспечения, нацеленный на снижение повторения информации различного рода. Если какая-либо информация в базе данных будет дублироваться, будьте уверены: содержимое этих дублированных полей разойдутся в соответствии с законом Мерфи [6], а вы получите неработоспособную базу данных.

Четвертая цель – это соблюдение ограничений, накладываемых на данные в БД. Согласованность (целостность) БД — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. «Явно заданные правила» это и есть ограничения, которые накладываются на значения полей в БД. В большинстве случаев ограничения согласованности определяются особенностями предметной области. Механизмы обеспечения согласованности закладываются при проектировании базы данных, и мы обсудим их позднее (см. раздел 4.8).

Одновременно с вышеизложенными целями необходимо помнить, что, как правило, база данных является информационной основой не одного, а нескольких приложений, часть из которых появится в будущем.

Только достигнув вышеперечисленные цели можно построить «хорошую» базу данных… Запомните, что плохой проект базы данных не может быть исправлен с помощью любых (даже самых изощренных) приложений.

Перечисленные цели кажутся тривиальными, но без постоянной «оглядки» на них создать хорошую БД не удается.

4.4. Основные задачи и результат проектирования БД

Вот примерный перечень задач, решаемых на этапе создания БД:

1. Анализ предметной области: описание предметной области, выявление ограничений целостности, определение статуса конфиденциальности данных, определение ролей пользователей, определение разграничения прав пользователей при доступе к данным, определение временных характеристик обработки данных.
2. Задание ограничений целостности, присущих предметной области, разработка процедур обеспечения целостности БД при вводе и изменении данных, обеспечение ограничений целостности при параллельной работе пользователей.
3. Работа с пользователями: сбор информации о бизнес-процессах в предметной области, об оценке пользователями работы существующей базы данных, определение регламента работы пользователей с базой данных, обучение и консультирование пользователей.
4. Проектирование структуры базы данных: определение состава и структуры информационных единиц, составляющих базу данных, задание связей между ними, выбор методов упорядочения данных и методов доступа к информации, описание структуры БД на языке SQL.
5. Определение целей и направлений развития системы, планирование этапов развития базы данных, разработка и выпуск организационно-методических материалов.

Конечно, при проектировании большой системы эти работы проводятся командой разработчиков, среди которых есть и аналитики, и программисты, и администраторы БД. Некоторые из этих задач решаются в рамках разработки требований к ИС, частью которой является БД.

В данной книге рассмотрены только основные задачи проектирования БД.

4.4.1. Основные задачи

В процессе проектирования БД разработчик должен решить следующие задачи:

1. Разработка бизнес-процессов;
2. Определение ролей пользователей;
3. Определение состояний и переходов;
4. Определение сущностей;
5. Создание схемы сущностей и их связей;
6. Создание таблиц и связей в рамках конкретной СУБД;
7. Создание объектов БД;
8. Разработка скриптов создания БД.

Далее мы подробно обсудим каждую из вышеперечисленных задач, а сейчас необходимо сделать несколько важных замечаний.

Первые 5 задач относятся к логическому проектированию БД т.е. здесь проектировщик решает общие задачи организации БД, не привязываясь к конкретной СУБД, хотя чаще всего уже на ранних стадиях разработки ИС принимается решение о применяемой СУБД. Поэтому в реальной жизни все решения принимаются с учетом особенностей конкретной СУБД. Выбор СУБД почти всегда диктуется не техническими, а экономическими или конъюнктурными соображениями (см. раздел 4.4.3).

Последние три задачи относятся к техническому проектированию, на этом этапе логические схемы и правила выливаются в конкретные объекты выбранной СУБД. Конструкция таблиц, ограничений, представлений, и других объектов БД оформляется в виде операторов языка SQL или характерного для выбранной СУБД расширения языка SQL. Полученная таким образом программа и является результатом проектирования БД. Необходимо отметить, что все таблицы и другие объекты БД, которые нужны приложению, должны быть созданы до начала работы приложения, и в дальнейшем изменения схемы производятся относительно редко (например, при изменении версии приложения или появлении новых функций группы приложений, работающих с базой данных). Очевидно, что все эти изменения должны отражаться в скрипте создания БД. Это особенно актуально для тиражируемых систем, которые разворачиваются на различных объектах: при каждой инсталляции системы БД создается заново.

4.4.2. Результат проектирования БД

Результатом проектирования БД является программа (скрипт) на SQL (или его расширении), который содержит:

1. Операторы создания таблиц;
2. Операторы создания представлений;
3. Операторы создания индексов;
4. Операторы создания ролей пользователей;
5. Операторы создания хранимых процедур и других объектов БД.

Запустив этот скрипт в конкретной СУБД, мы получим полную схему всей БД, роли и механизмы разграничения прав пользователей, механизмы обеспечения согласованности, безопасности и быстродействия БД. Пример такого скрипта можно увидеть в Листинге В.1.

4.4.3. Выбор СУБД

Несколько слов надо сказать о выборе СУБД. Чаще всего выбор СУБД не является технической задачей, но представляет собой сложный компромисс между множеством факторов не являющихся техническими ограничениями. Например, в банке должна быть промышленная СУБД сопровождаемая разработчиком или всемирно известной компанией, иначе банк не пройдет международный аудит и его действия на зарубежных рынках будут ограничены, или в большой компании есть специалисты, подготовленные к работе с ORACLE и есть ряд соответствующих лицензий, поэтому вопрос о выборе СУБД не стоит. Надо запомнить одно правило:

«Если проектируемая БД небольшая, то любая известная СУБД подойдет, если же БД большая и критична для работы предприятия, то выбор СУБД чаще всего не связан с техническими вопросами».

4.5. Логическое проектирование БД

В настоящее время разработка БД неотделима от разработки ИС, с которой эта база будет работать, поэтому проектирование БД начинается вместе с проектированием ИС. В подавляющем большинстве проектов по разработке ИС осуществляется автоматизация бизнес-процессов (БП) [8], поэтому основная работа по анализу и разработке требований сводится к изучению существующих и созданию новых БП.

В этом документе термин «бизнес-процесс» понимается как «совокупность взаимосвязанных действий пользователей и самой ИС, направленных на создание определённого продукта или услуги для потребителей».

При проектировании ИС создаются описания всех бизнес-процессов, которые задействованы в процессе функционирования автоматизируемого предприятия.

Описание бизнес-процесса – это описание последовательности действий сотрудников при выполнении определенных действий в графическом и текстовом виде с целью регламентации действий в коллективе, анализа и оптимизации их последовательности.

Описание бизнес-процессов – работа творческая. Даже если вы описываете «то, что есть», все равно допускаются некоторые неточности, «сглаживаются» углы, какие-то действия упускаются для простоты восприятия. А если описывается «то, что должно быть», то здесь на основе существующего процесса создается нечто новое, при этом бизнес-аналитик работает в строгих рамках правил, синтаксиса, логических ограничений.

Запомните, что ни один бизнес-процесс не может быть совершенным и на 100% соответствовать реальности. Всегда есть место каким-то упрощениям и допущениям, где-то при реализации даже самого строгого регламента свои коррективы вносит человеческий фактор.

Упрощенный бизнес-процесс логического проектирования БД в нотации IDEF0 представлен на рис. 4.5.

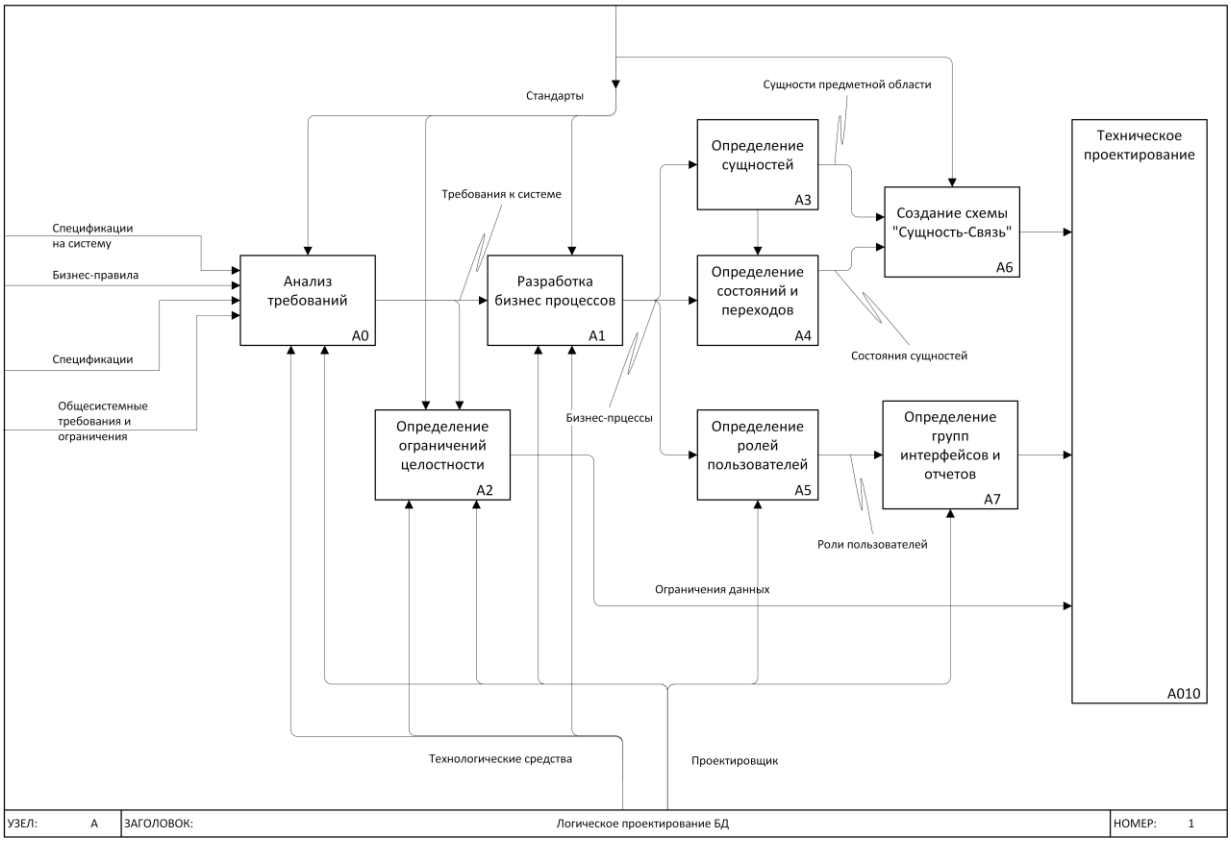

Рис.4.5. Бизнес-процесс логического проектирования БД

Сбор информации о предметной области, а также анализ и разработку требований к ИС мы оставим за границами этого документа, тем более что этот процесс подробно и исчерпывающе раскрыт в литературе [8]. Оставим мы также в стороне разработку требований к аппаратной базе, сетевой архитектуре и планирование необходимых ресурсов для осуществления проекта. Собственно, проектирование БД следует начать с анализа БП, полученных после разработки требований к ИС.

4.5.1. Анализ бизнес-процессов

Поскольку БП это действия пользователей в ИС и действия самой ИС, нашим первым вопросом будет: с какими объектами (или описаниями объектов) работает пользователь, какие документы поступают на вход БП и какие формируются на выходе? Это и будут сущности нашей ИС. (см. раздел «Определение сущностей»)

Второй, не менее важный вопрос: какие пользователи выполняют те или иные действия в БП? (см. раздел «Создание ролей пользователей»)

И, наконец, третий вопрос: каковы свойства и возможные состояния объектов, с которыми работает пользователь в этом бизнес-процессе (см. раздел «Создание атрибутов») и как эти свойства и состояния изменяются?

4.5.2. Определение сущностей

На рис. 4.6. приведен фрагмент действующего документа, описывающего БП «Выдача кредита безналичными в другой банк» из проекта автоматизированной банковской системы (нотация BPMN). Посмотрим с какими объектами работают сотрудники банка при выполнении своих должностных обязанностей.

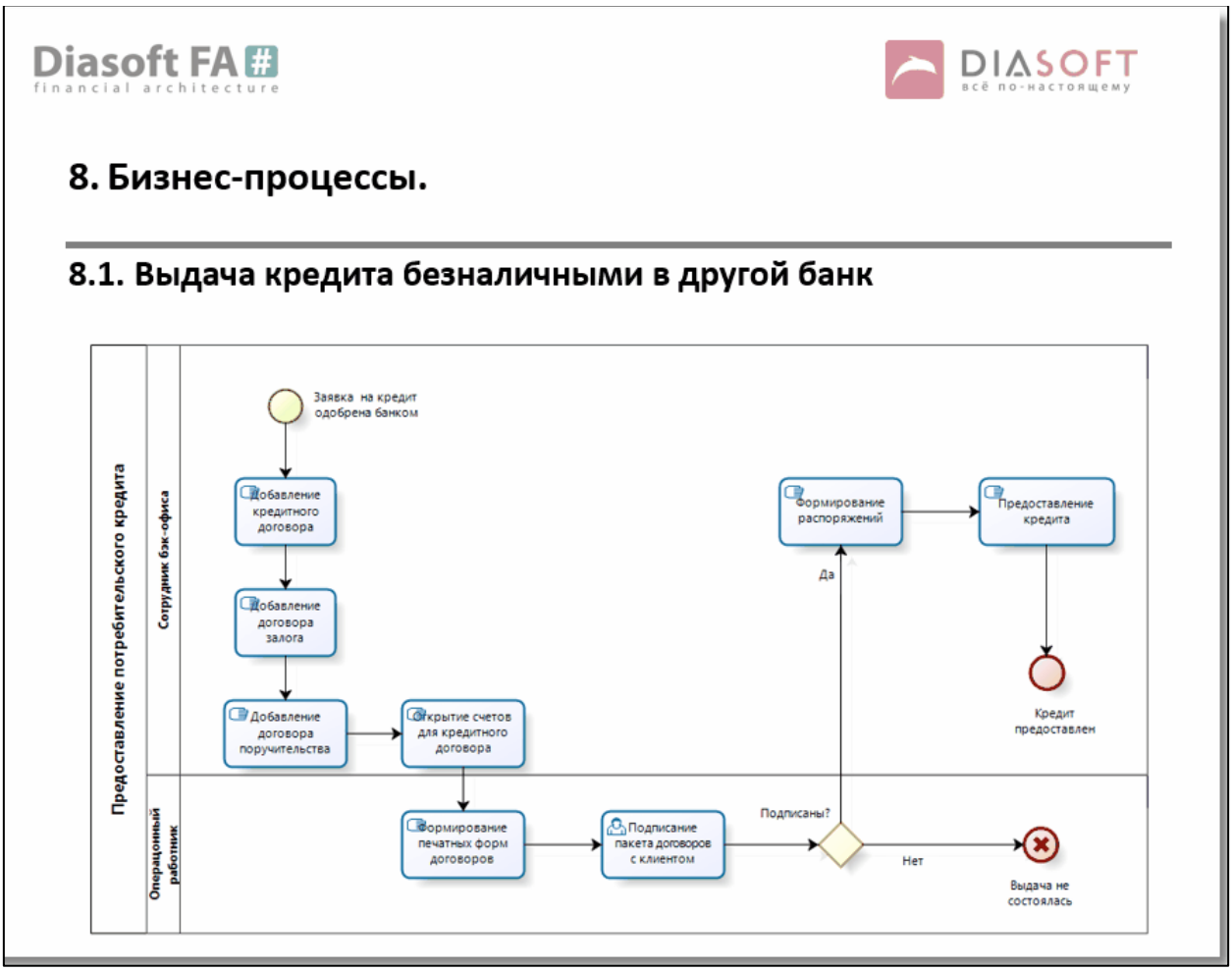

Рис. 4.6. Бизнес-процесс «Выдача кредита безналичными в другой банк»

Из этих объектов и родятся сущности будущей БД:

- Кредитный договор

  Мы изучили предметную область и знаем, что в любом договоре есть как минимум 2 стороны, эти стороны и будут следующими двумя сущностями:
- Банк (кредитор)
- Клиент (заемщик)
- Договор залога
- Счета кредитного договора
- Платежные распоряжения

Рассмотрим еще один пример – БП записи на прием в поликлинике (см. приложение Б). Этот процесс представлен на рис. 4.7.

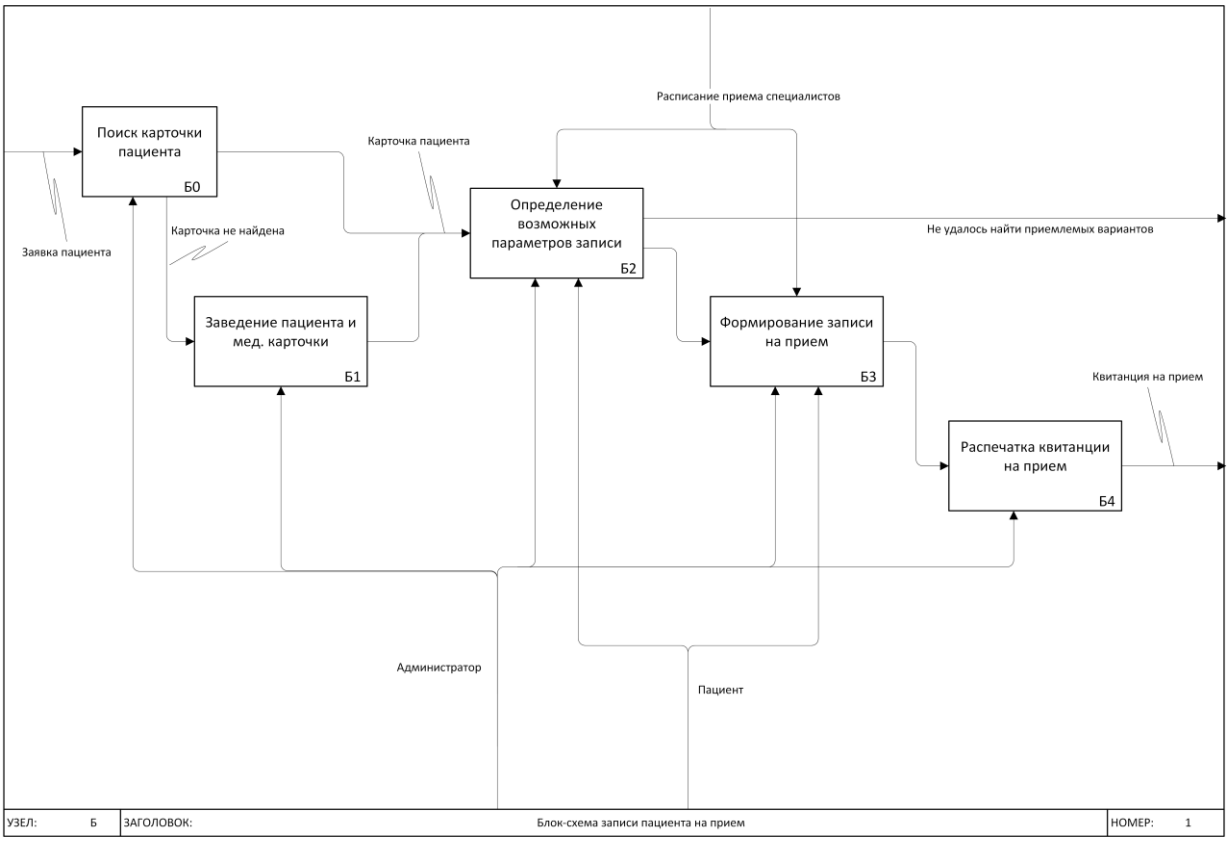

Рис. 4.7. БП «Запись на прием в поликлинике»

Здесь можно выделить следующие сущности, с которыми работает администратор и пациент:

- *Карточка пациента*
  Здесь содержаться результаты приема пациента у различных врачей
- *Пациент*
  Данные о пациенте
- *Расписание приема специалистов*
  Эта сущность пришла в БП извне… наверно есть другой БП где она формируется
- *Специалист*
  Данные о враче-специалисте
- *Запись на прием*
  Данные о резервировании времени приема пациента

И, наконец, последний пример – БП приема у врача в поликлинике на рис. 4.8.

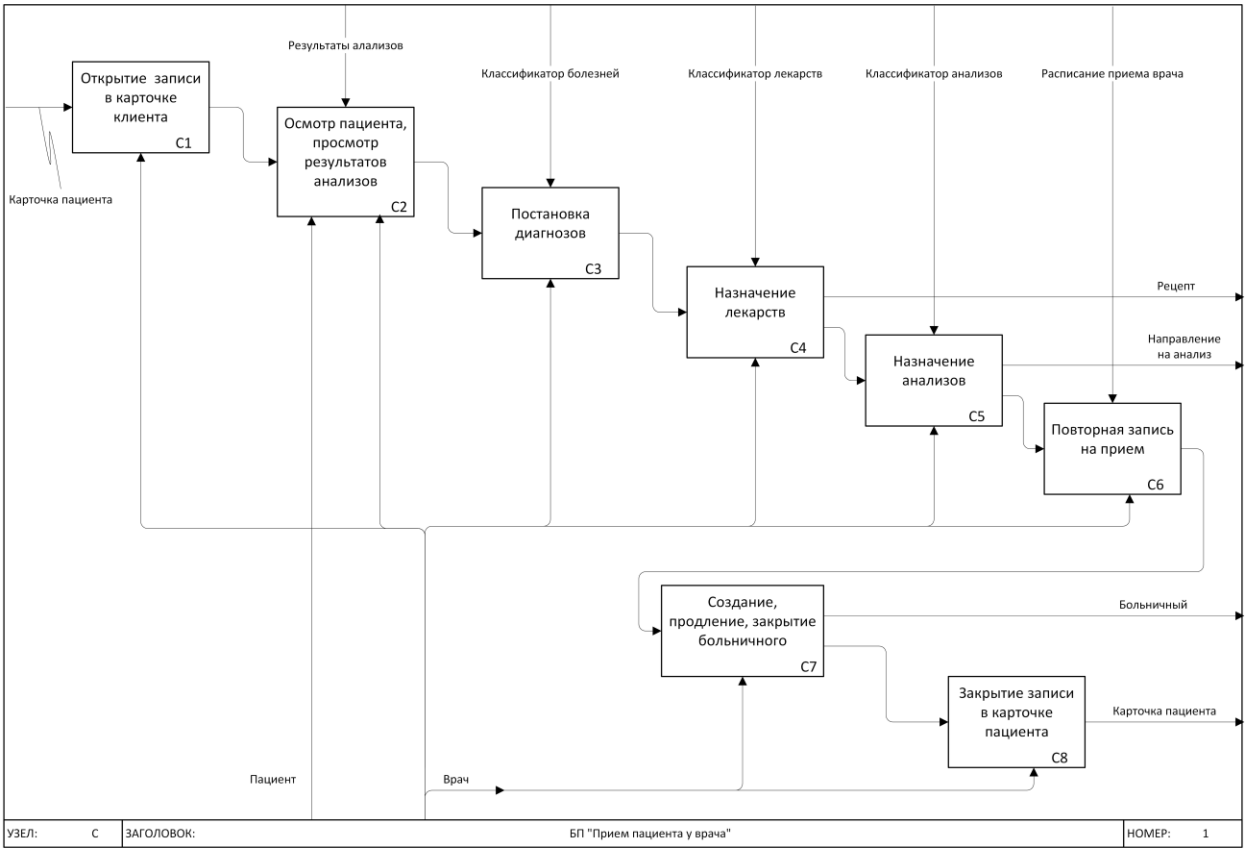

Рис. 4.8. БП «Прием пациента врачом»

Выделим и из этого БП сущности, с которыми работает врач:

- *Запись в карточке пациента*
- *Запись на прием*
  Данные о резервировании времени приема пациента.
- *Классификатор (справочник) болезней*
  Эта сущность пришла в БП извне, также, как и справочник анализов и лекарственных препаратов… изучая предметную область выясняем, что сейчас действует «Международная классификация болезней десятого пересмотра МКБ-10 (принята 43-ей Всемирной Ассамблеей Здравоохранения)».
- *Поставленный (уточненный) диагноз*
  В соответствии со справочником болезней, анализируя результаты анализов и симптомы, врач ставит диагноз и записывает его в Карточку пациента.
- *Классификатор (справочник) анализов*
- *Классификатор (справочник) лекарственных препаратов*
  Аналогично классификатору болезней существуют и другие справочники предметной области.
- *Назначенные анализы*
  В соответствии со справочником анализов, врач назначает необходимые анализы и записывает их в Карточку пациента.
- *Результаты анализов*
  После выполнения анализов их результаты поступают в систему и привязываются к карточке пациента.
- *Назначенные лекарственные препараты (Рецепт)*
  Документ, регламентированный законом РФ, удостоверяет возможность платного или бесплатного получения лекарств из перечня Минздрава.

- *Больничный лист*
  Документ, регламентированный законом РФ, удостоверяет временную неработоспособность пациента.

Из полученного перечня сущностей мы можем заключить, что сущностями чаще всего бывают:

a. **объекты** (Кабинет врача, Назначенный лекарственный препарат) - предметы окружающего мира;
b. **субъекты** (Пациент, Специалист) - группа людей, роль;
c. **события** (Запись в карточке пациента) - сущность фиксирующая событие, в данном случае это прием врачом пациента;
d. **документы** (Больничный лист, Рецепт) - материальный носитель с зафиксированной на нём информацией, который имеет реквизиты, и предназначен для общественного использования;
e. **справочники** (Справочник диагнозов) - перечни возможных типов сущностей или значений атрибутов.

4.5.3. Верификация сущностей

После того, как определились сущности предметной области, надо проверить их корректность, для чего существуют специальные вопросы, ответив на которые вы убедитесь в том, что сущности на первой стадии проработки определены правильно.

1. Отражает ли имя сущности суть данного объекта?
2. Есть ли синонимы/омонимы данной сущности?
3. Нет ли уже такой сущности, возможно, под другим именем?
4. Нет ли пересечения с другими сущностями?
5. Имеются ли хотя бы два атрибута?
6. Сущность определена полностью?
7. Имеется ли у сущности хотя бы одна связь?
8. Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, и использованию значения сущности?
9. Ведется ли история изменений?
10. Не имеет ли сущность слишком общий смысл?
11. Достаточен ли уровень обобщения, воплощенный в ней?

Разберем эти вопросы более подробно.

1. и 2. Вопросы помогут правильно назвать сущность, чтобы коды ИС и запросы к БД выглядели осмысленно и логично – это очень поможет при верификации и отладке кода программ.
3. Поинтересуйтесь, возможно такая же сущность уже была выделена в результате анализа другого БП, подумайте, полностью ли они идентичны.
4. Не повторяются ли группы атрибутов в разных сущностях и не вынести ли эти группы в отдельные сущности. Например, в сущностях «Студент», «Преподаватель», «Сотрудник» существует группа одинаковых полей: Фамилия, Имя, Отчество, Дата рождения, номер паспорта, адрес и т.д. (то, что называется «Личные данные»). Хорошо бы выделить их в отдельную сущность – «Физическое лицо», тем более что студент может быть и сотрудником кафедры, а его личные данные будут храниться в одной таблице, а не в двух. Вспомните про принцип DRY и закон Мерфи.

5. Один обязательный атрибут – первичный ключ и нужен еще хотя бы один, для характеристики экземпляра сущности.
6. Подумайте, все ли атрибуты сущности упомянуты и определены.
7. Связана ли сущность с другими сущностями, а если не связана, может быть она просто лишняя?
8. А используется ли вообще данная сущность в проектируемой ИС, и, если нет ни одной функции работающей с ней, может эта сущность не нужна?
9. Нужно ли сохранять данные о предыдущих значениях атрибутов сущности, например, человек, на протяжении всей жизни, меняет несколько раз паспорт, надо ли хранить параметры предыдущих паспортов и даты их замены? В каких-то системах да, в каких-то нет, а если да, то где хранить предыдущие значения?
10. Может в этой сущности заложено слишком много смыслов и надо разбить ее на несколько более простых? Например, стоит разбить сущность «Транспортное средство», на «Гужевой транспорт», «Железнодорожный транспорт», «Автомобильный транспорт» и т.д.?
11. Может в этой сущности заложено слишком узкое понятие? Это вопрос обратный предыдущему - может быть стоит объединить сущности «Гвоздь», «Шуруп», «Саморез» в сущность - «Крепеж»?

4.5.4. От автоматизации процессов к автоматизации компании

Процесс проектирования ИС - это не процесс автоматизации одного БП, автоматизировать иногда приходится десятки, а то и сотни БП, составляющих ландшафт предприятия. Сущности, выделенные в процессе анализа различных БП, необходимо объединить в одну или несколько схем, руководствуясь принципом DRY. Объединение необходимо начинать с поиска одинаковых (общих) сущностей присутствующих в разных БП и выстраивать общую схему связывая новые сущности с найденными общими сущностями. Обычно общая схема БД предприятия распадается на несколько «кустов», что связано с административным делением предприятия: понятно, что в одном подразделении решаются схожие, смежные задачи, сущности которых тесно связаны между собой. Связи между «кустами» обычно осуществляются через небольшое количество общих сущностей, пронизывающих всю ИС и создающих информационный каркас предприятия. После формирования единого перечня сущностей, необходимо снова проверить каждую при помощи вышеприведенных вопросов.

Определение сущностей завершает первый этап разработки БД, но надо помнить, что на дальнейших этапах количество и состав сущностей будет уточнятся и, возможно, дополняться.

4.5.5. Определение атрибутов сущностей

Следующим важным этапом проектирования является определение атрибутов сущностей.

Атрибут – это свойство сущности в предметной области. Его наименование должно быть уникальным для конкретного типа сущности. Например, для сущности студент могут быть использованы следующие атрибуты: фамилия, имя, отчество, дата и место рождения, паспортные данные и т.д. В реляционной БД атрибуты хранятся в полях таблиц. Совокупность значений атрибутов определяет конкретный экземпляр сущности.

В дальнейшем изложении термина атрибут практически совпадает с термином поле, т.к. каждому атрибуту будет соответствовать поле в таблице, отображающей сущность, связанную с этим атрибутом.

4.5.6. Первичный ключ – важный атрибут

Одним из атрибутов сущности должен быть первичный ключ – это атрибут однозначно определяющий экземпляр сущности, например, первичным ключом сущности «Студент» может быть номер паспорта, а первичным ключом сущности «Врач специалист» - табельный номер.

Ясно, что значения первичного ключа должны быть уникальными и неизменными, иначе с экземплярами сущностей произойдет путаница, но это значит, что номер паспорта как ключ не подойдет - ведь он может меняться при замене паспорта (например, при утрате), а табельный номер при слиянии компаний становится неуникальным. Можно привести огромное количество примеров, которые показывают, что реальные атрибуты не подходят на роль первичного ключа и, хотя теоретики утверждают обратное – «первичным ключом может быть только реальные атрибуты или их суперпозиция» [3], в настоящее время при проектировании БД первичным ключом сущности всегда служит специальный атрибут, не несущей никакой информации об экземпляре сущности, а только идентифицирующий ее. Такой ключ называется суррогатным – его значение генерируется СУБД при добавлении экземпляра сущности (записи в таблицу).

Все размышления и утверждения, относящиеся к ключам сущностей переносимы на ключи таблиц реляционной БД, ведь каждой отдельной сущности соответствует отдельная таблица в реляционной базе данных, что следует из определения самой таблицы (см. приложение А).

4.5.7. Вычисляемые атрибуты (поля)

Частая ошибка проектировщиков – использование вычисляемых полей т.е. полей, содержащих значение некоторых выражений, состоящих из значений других полей таблицы. Пример – поле возраст пациента, которое можно определить, как «год текущей даты» минус «год даты рождения». Хотя практически все СУБД поддерживают вычислимые поля, перед тем как вставить такое поле в таблицу, подумайте о том, когда оно будет вычисляться. Обычно вычислимые поля заполняются в момент создания записи, а потом не изменяются, поэтому данные о возрасте пациента из нашего примера устареют уже через год и БД станет неактуальной. Если есть необходимость в вычислимых полях создайте преставление (см. раздел 4.6.2) и тогда значение этих полей всегда будет актуальным, ведь оно будет вычислено в момент обращения к представлению.

4.5.8. Верификация атрибутов

После того, как определились атрибуты сущностей предметной области, надо проверить их корректность, для чего существуют специальные вопросы, ответив на которые вы убедитесь в том, что атрибуты определены правильно.

1. Является ли наименование атрибута существительным единственного числа, отражающим суть обозначаемого атрибутом свойства?
2. Имеет ли атрибут только одно значение в каждый момент времени?
3. Есть ли необходимость в истории изменений?
4. Все ли ограничения, накладываемые на данный атрибут известны и описаны?
5. Не может ли этот атрибут быть пропущенной сущностью?
6. Не может ли атрибут быть пропущенной связью?
7. Не является ли его значение вычислимым т.е. производным от других атрибутов?
8. Зависит ли его значение только от данной сущности?
9. Зависит ли его значение только от уникального ключа?
10. Зависит ли его значение от значений некоторых атрибутов, не включенных в уникальный ключ?

Разберем эти вопросы более подробно.

1. Вопрос поможет правильно назвать атрибут сущности, чтобы коды ИС и запросы к БД выглядели осмысленно и логично – это очень поможет при верификации и отладке кода программ.

2. Этот вопрос предполагает, что если атрибут называется «свойство», то он не может иметь одновременно два значения, например, «мокрый» и «красный».
3. Этот вопрос о том, нужно ли сохранять предыдущие значения атрибута при его изменении, например, нужно ли нам знать размер предыдущих окладов сотрудника и данные о том, когда они изменялись, а если надо, то где хранить эти значения?
4. Проверьте описаны ли формат, длина атрибута, допустимые значения, значение по умолчанию, алгоритм получения и т.п.?
5. Проверьте, не является ли атрибут ссылкой на некую сущность, еще не определенную в данном проекте. Например, атрибут «диагноз» сущности «Запись в карточке пациента» оказывается фактически ссылкой на не выявленную сущность – «Классификатор болезней».
6. Проверьте, нет ли уже в схеме сущности с таким же названием или с таким же смыслом, как у атрибута, если есть, то скорей всего этот атрибут является пропущенной связью.
7. Про вычислимые атрибуты (поля) см раздел 4.5.7.
8. Вопросы 8 – 10 связаны с особенностями использования атрибутов в качестве первичного ключа. Если вы пользуетесь суррогатными ключами, то эти вопросы можно не рассматривать.

После проверки правильности атрибутов, т.е. после ответов на все вопросы и внесения корректив, необходимо вернуться к разделу 4.5.3. и снова верифицировать все сущности. Любой процесс проектирования – это итерационный процесс, постепенно приводящий разработчика к оптимальным решениям.

4.5.9. Виды связей между сущностями

Объекты реального мира находятся друг с другом в сложных взаимоотношениях. В БД такие взаимоотношения представляются как связи между сущностями. Под понятием «связь между сущностями А и В» мы будем понимать соответствие одному или нескольким экземплярам сущности А одного или нескольким экземплярам сущности В, например,

- Много студентов учатся в одной группе (Студенты – Группы);
- Студент изучает некоторые предметы (Студенты – Предметы);
- Врач является пациентом своей поликлиники (Врач – Пациент);
- Пациенты записываются на прием к врачам (Пациент – Врач).

Можно различать связи между двумя сущностями по количеству экземпляров присутствующих в связи с разных сторон:

1. Связь много к одному - **много студентов** учатся в **одной группе;**
2. Связь один ко многим – **один студент** изучает **много предметов;**
3. Связь один к одному - **врач** является **пациентом**;
4. Связь много ко многим - **пациенты** ходят на прием к **врачам,**
   т.е. один пациент может ходить на прием ко многим врачам, а к одному врачу может ходить на прием много пациентов.

Первый и второй варианты равнозначны (сравните *«**много студентов** учатся в **одной группе»*** и *«**в одной группе** учатся **много студентов**»*), а третий вариант является вырожденным случаем предыдущих. Рассмотрим, как осуществляются связи первых трех типов между двумя сущностями. В связанных сущностях присутствуют особые атрибуты, при помощи которых можно отыскать связанные экземпляры сущностей. Например, в сущности «Студент» есть атрибут содержащий значение определяющее группу, к которой принадлежит студент (этот атрибут содержит номер группы). В общем случае связывающий атрибут первой сущности

должен содержать значение однозначно определяющее экземпляр второй сущности. Лучше всего для такого однозначного определения подходит первичный ключ.

4.5.10. Первичные и внешние ключи

Ключи, существующие в таблицах (а каждая таблица соответствует сущности), однозначно определяют экземпляры сущностей:

- первичный ключ — это поле таблицы (атрибут сущности) однозначно определяющий запись таблицы (экземпляр сущности), напомним, что первичным ключом может быть только суррогатный ключ;
- вторичный ключ — это поле таблицы однозначно определяющее запись таблицы, но не являющееся первичным ключом, например, студента можно однозначно определить по номеру паспорта, номеру зачетки, номеру студенческого билета, номеру пропуска - все эти поля являются вторичными ключами (заметьте, все эти поля уникальны);
- внешний ключ — это поле дочерней таблицы содержащее значение первичного ключа внешней, родительской таблицы и, следовательно, однозначно определяющее запись связанной родительской таблицы.

Основное предназначение ключей - связывание сущностей. Пример на рис. 4.9.

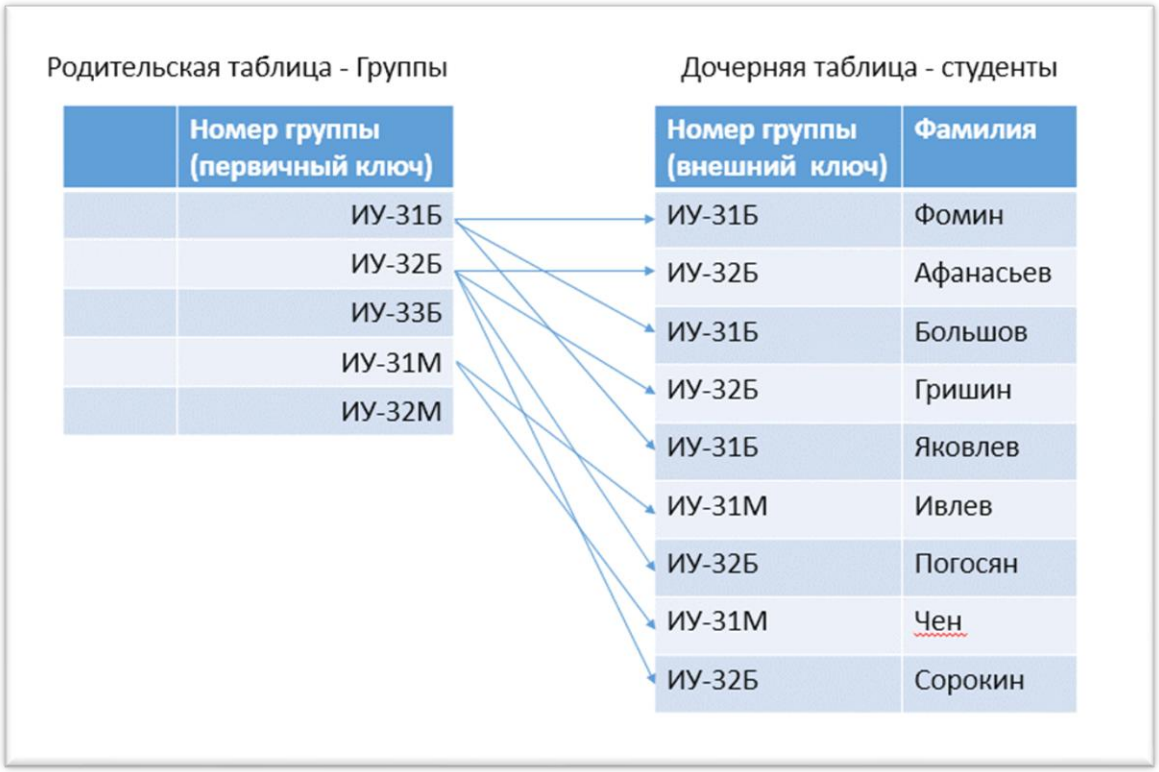

Рис. 4.9. Связь между таблицами

В таблицах на рис. 4.9. присутствуют поле «Номер группы» в таблице «Группа» — это первичный ключ, а в таблице «Студент» поле «Номер группы» — это внешний ключ, определяющий группу, в которой учится студент. Зная экземпляр группы и, следовательно, значение первичного ключа – «Номер группы» (НГ) можно по значениям внешнего ключа равным НГ выбрать всех студентов этой группы и наоборот, по значению внешнего ключа (НГ конкретного студента) можно найти нужный экземпляр сущности «Группа».

Так работают связи типа «один ко многим» для таблиц (сущностей) в реляционной БД.

Но ведь номер группы меняется каждый семестр и, следовательно, связывать таблицы по этому полю нельзя, иначе связи в вашей БД будут становиться некорректными каждый семестр.

Чтобы такого не случилось, первичными ключами служат только суррогатные ключи (см. раздел 4.5.6.), при этом связи между таблицами получатся как на рис. 4.10.

Рис. 4.10. Связь между таблицами при помощи суррогатного ключа

Таким образом, номер группы меняется каждый семестр, но это нисколько не затрагивает связи в базе данных, поскольку суррогатный первичный ключ группы не меняется на протяжении всего жизненного цикла группы с момента ее создания и до перемещения данных о группе в архив. Этот суррогатный первичный номер группы уникален во всем вузе и на все времена.

4.5.11. «Один к одному» - редкий тип связи

Для иллюстрации и понимания сути связи «один к одному» рассмотрим пример связи «Физическое лицо – Паспорт». Конечно, здесь все кажется понятным: паспортные данные – атрибут сущности «Физическое лицо», тут и связи-то никакой нет. Однако, рассмотрим следующие предметные области:

Пример 1. ИС Банка.

Процесс банковского обслуживания можно коротко описать так: «Операционист по предъявленному паспорту удостоверяет личность клиента и оказывает ему требуемую услугу». Здесь явно выделяются сущности – клиент, паспорт и услуга. Первые две сущности связаны между собой ОБЯЗАТЕЛЬНОЙ связью «один к одному». Обязательность этой связи заключается в том, что у каждого клиента обязательно есть паспорт, причем только один. Такая связь — это прямое указание, на то, что одна из сущностей является всего лишь атрибутом другой, т.е. паспортные данные – атрибут сущности «Физическое лицо».

Пример 2. ИС Паспортного стола.

В этой системе тоже есть связь «Физическое лицо – Паспорт», однако, здесь она не обязательная, потому как в паспортном столе есть еще не выданные паспорта (чистые бланки) и есть физические лица, которые еще не имеют паспортов, но уже зарегистрированы, как подавшие заявление на получение паспорта. В этой системе «Физическое лицо» и «паспорт» – две отдельные сущности, связанные между собой НЕОБЯЗАТЕЛЬНОЙ связью «один к одному». Необязательная связь «один к одному» говорит о том, что мы имеем дело с разными сущностями.

Другой пример - в базе данных ВУЗа есть сущности: «физические лица», «служащие», «преподаватели», «студенты». Каждая из последних трех сущностей связана с сущностью «физические лица» связью «один к одному», причем внешние ключи находятся в таблицах «служащие», «преподаватели» и «студенты». Обратите внимание – три внешних ключа содержат значения одного первичного. Это говорит о том, что одно и то же физическое лицо может

выступать в нескольких ролях, например, студенты могут работать в вузе в роли служащего или преподавателя.

Для реализации необязательной связи «один к одному» необходимо на внешний ключ наложить ограничение - уникальный (UNIQUE), тогда каждому значению первичного ключа будет соответствовать одно значение внешнего ключа.

4.5.12. «Много ко многим» - несуществующий тип связи

Еще раз посмотрев на рис. 9 и 10 можно понять, что связи между таблицами в реляционной БД бывают только одного типа – «один ко многим», это связано с единственным механизмом реализации связи через первичный и внешний ключ. Между сущностями могут быть и другие связи, так как механизмы связей между сущностями не регламентированы. Однако, с другой стороны, таблицы — это отражение сущностей и, следовательно, сущности тоже не могут иметь никаких других связей, кроме «один ко многим».

В реальной жизни (в реальной предметной области) не существует связей «много ко многим»! Но что делать если проектировщик все-же наталкивается на необходимость связи отличающейся от «много к одному»? Это значит он пропустил какую-то важную, для данной предметной области сущность, которая разрешает «странный» тип связи и преобразует ее в несколько связей «один ко многим». Вот хрестоматийный пример связи «много ко многим» – авторы и книги: каждый автор написал одну или более книг и у каждой книги один или более авторов. В реальной же системе, например, библиотечной ИС, присутствует сущность «Библиография авторов» и таким образом мы получаем связи «один ко многим» (см. рис. 4.11).

Другой классический пример – преподаватель преподает несколько предметов, а один и тот же предмет может вести несколько преподавателей. Однако, в реальной БД ВУЗА обязательно присутствует сущность «Расписание занятий», связывающая преподавателей и предметы. Ведь только из расписания можно, понять кто и что читает. Таким образом мы опять получаем связи «один ко многим» (см. рис. 4.11).

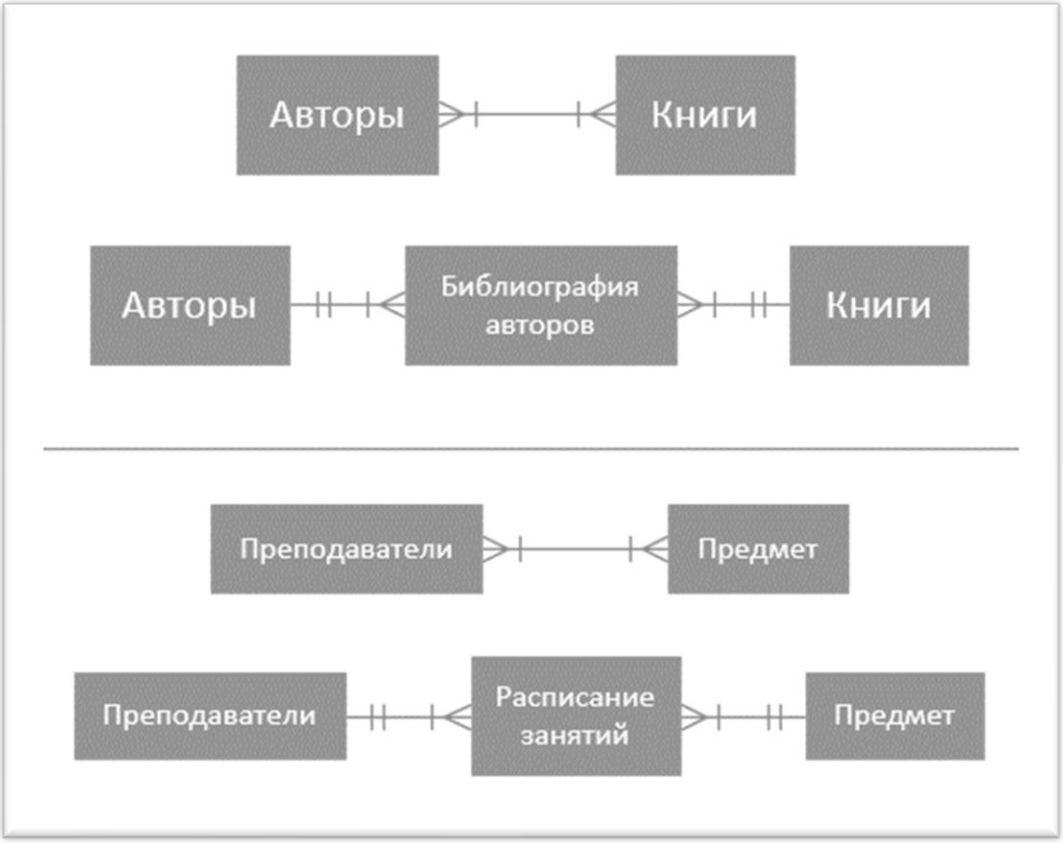

Рис. 4.11. Примеры разрешения связей «много ко многим»

4.5.13. Экзотические связи

В некоторых источниках упоминаются совсем уж экзотические «тринарные» связи, которые связывают между собой сразу три сущности, например, Продавцы, Покупатели и Товары. Однако, после анализа предметной области становится понятно, что эта связь всего лишь пропущенная сущность «Договор купли-продажи» (см. рис. 4.12.).

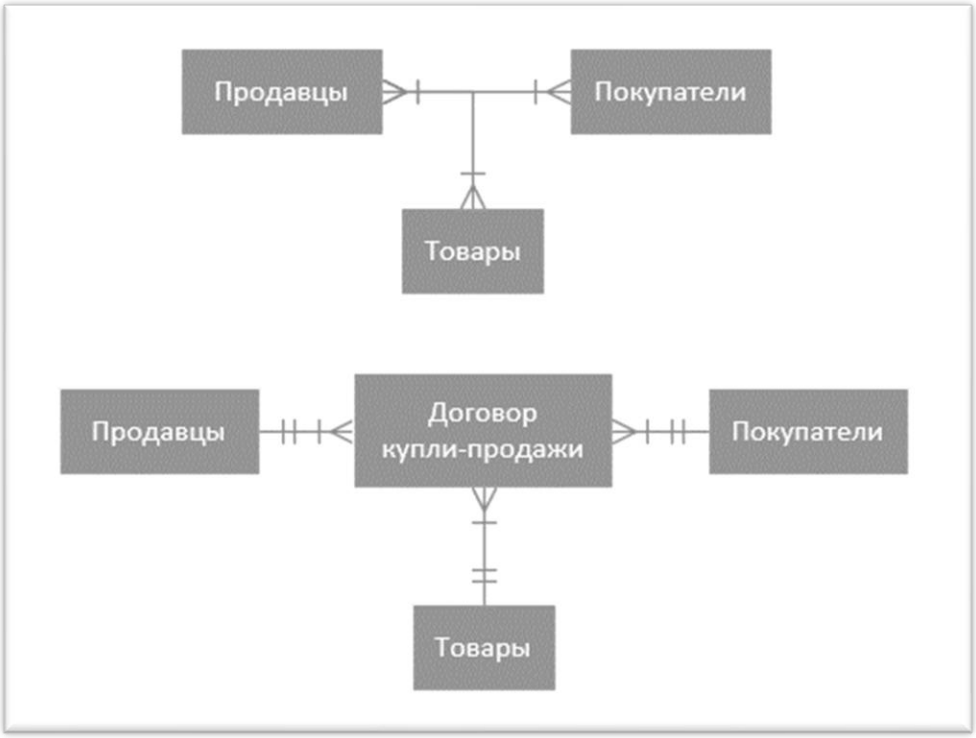

Рис. 4.12. Пример экзотической связи

Помните – если у вас на схеме появилась связь отличная от типов «один к одному» и «один ко многим», значит вы упустили какую-то важную сущность.

4.5.14. Верификация связей

После того, как определились связи между сущностями предметной области, надо проверить их корректность, для чего существуют специальные вопросы, ответив на которые вы убедитесь в том, что связи определены правильно.

1. Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи?
2. Допустима ли конструкция связи?
3. Не является ли связь пропущенной сущностью?
4. Необходима ли обязательность для каждой стороны связи?
5. Не является ли она избыточной?
6. Не изменяется ли связь с течением времени?

Разберем эти вопросы более подробно:

1. Полное ли описание связи, проверьте все ли вы знаете об этой связи.
2. Не отличается ли конструкция связи от «один ко многим» или «один к одному»

3. Если тип связи не относится к тем, что перечислены в предыдущем пункте, может быть это пропущенная сущность (см. раздел 4.5.8)
4. По сути это ответ на вопрос должен ли каждый экземпляр сущности иметь связь с экземпляром другой сущности.
5. В идеальном случае цепочка связей, ведущая от одной сущности к другой должна быть единственной, однако в реальной жизни таких путей может быть несколько. Каждую такую ситуацию необходимо внимательно рассмотреть и по возможности убрать лишние связи для уменьшения рисков несогласованности БД.
6. Пример: сущность «физ. лицо» связано с сущностью «студент», однако после выпуска, это же физ. лицо должно быть связано с другой сущностью – «выпускник». Как видим эта связь с течением времени меняется и надо предусмотреть специальные механизмы реализующие эту смену.

После проверки корректности связей, т.е. после ответов на все вышеприведенные вопросы и внесения корректив, необходимо вернуться к разделу 4.5.3. и снова верифицировать все сущности. Любой процесс проектирования – это итерационный процесс, постепенно приводящий разработчика к оптимальным решениям

4.5.15. Диаграмма «Сущность-связь»

Одновременно с определением связей разрабатывается диаграмма «Сущность-связь», которая содержит все сущности и все связи. Начинать разработку этой диаграммы надо с основной сущности, той, которую вы считаете самой главной, той, которая имеет больше всего связей. В случае с системой «Кафедра» эта сущность – «Учебный план», а в системе «Поликлиника» – «Карточка пациента». Определяя связи и отображая новые сущности, проектировщик постепенно получает целостную картину БД, выраженную в диаграмме «Сущность-связь». Несколько следующих рисунков показывают процесс построения диаграммы.

На рис. 4.13. представлена первая итерация разработки схемы «Сущность-связь» БД «Поликлиника», здесь видно только начало бизнес-процесса: Врач осматривает клиента, записывает в БД симптомы и ставит диагноз. Центральная сущность в схеме «Запись в медкнижке» соответствует «Карточке пациента» - дело в том, что «Карточка пациента» по сути просто набор записей о посещениях пациентом врачей, набор страниц в медкнижке. Надо отметить, что, говоря о сущности, надо точно понимать, что является экземпляром сущности - в данном случае экземпляр сущности – запись (страничка) в медицинской книжке. При этом на каждой страничке отмечен пациент и врач, ведущий прием.

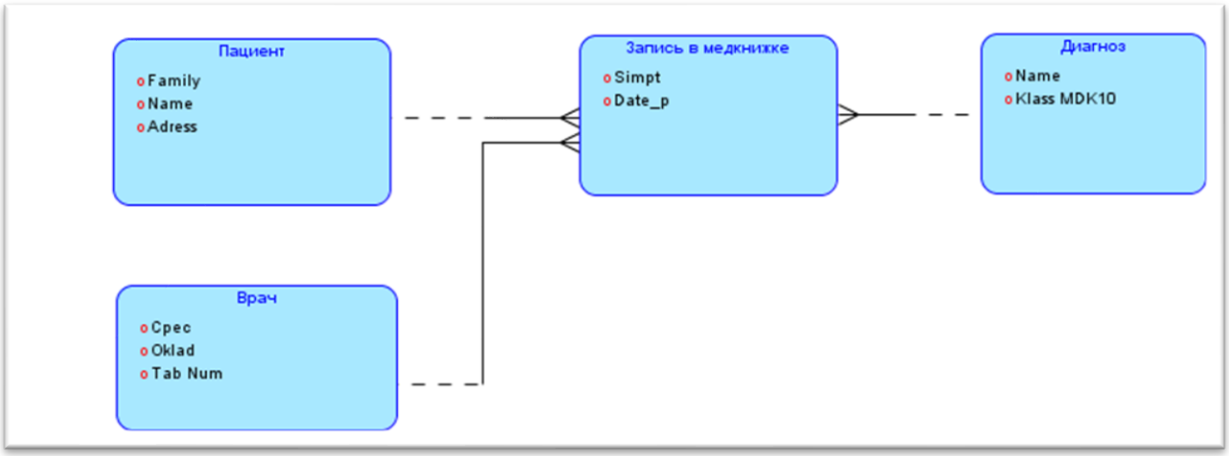

Рис. 4.13. Первый вариант схемы «Сущность-связь» БД «Поликлиника»

Доработаем схему последовательно присоединяя сущности БД

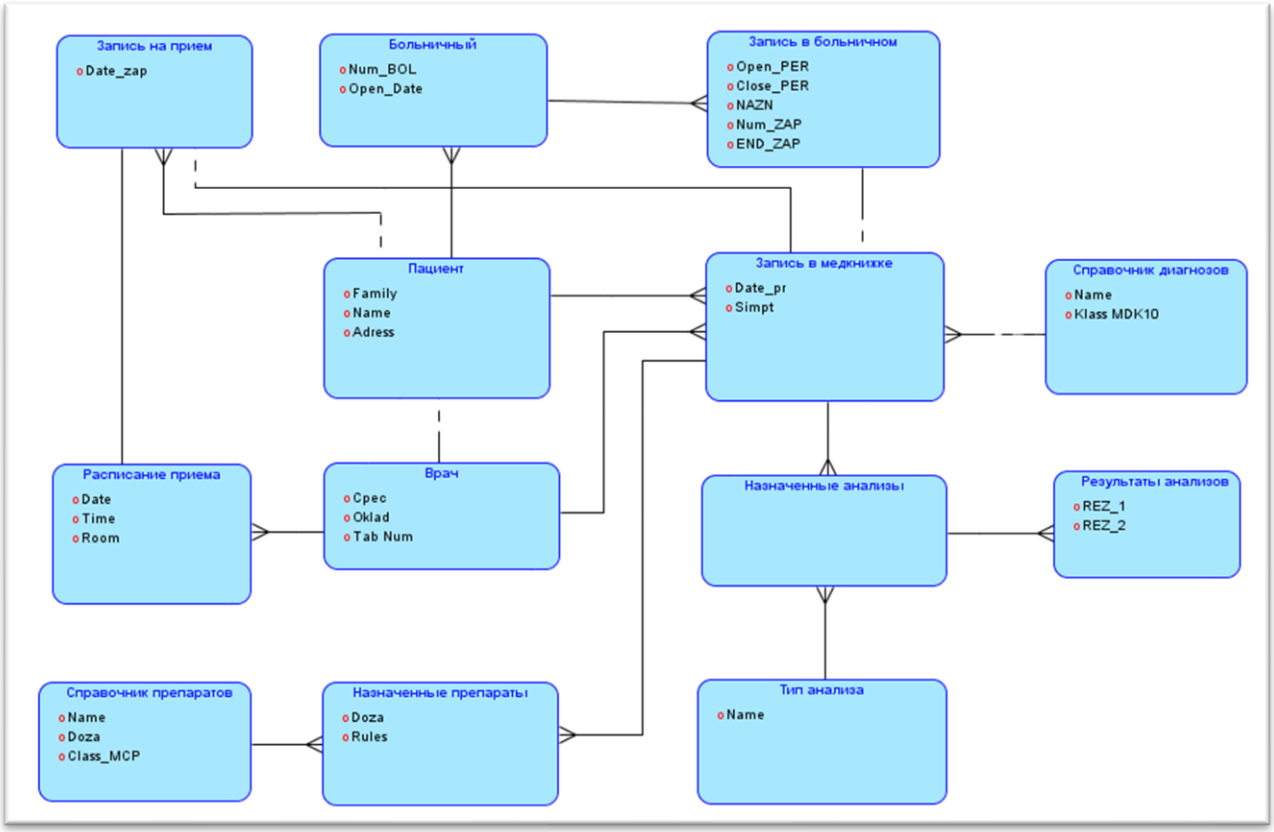

Рис. 4.14. Полная Схема «Сущность-связь» для БП «Прием пациента врачом»

Внимательно изучите рис. 4.14. и вспомните БП «Прием пациента врачом», приведенный на рис. 4.4, посмотрите еще раз: как мы выделяли из этого БП сущности и как определяли связи между этими сущностями. Разберитесь, каким образом получили свое отображение на схеме вышеупомянутые сущности и связи. Обратите внимание на атрибуты сущностей, убедитесь, что здесь нет первичных и внешних ключей (они «родятся» потом, в процессе формирования таблиц для конкретной СУБД»). Обратите внимание, что сущность «Назначенные анализы» совсем не содержит атрибутов - она предназначена для связей.

4.6. Физическое проектирование базы данных

4.6.1. Формирование схемы таблиц

После проработки схемы «сущность-связь» можно переходить к физическому проектированию БД, т.е. к созданию схемы таблиц и связей между ними. Эта схема уже привязана к конкретной СУБД и создается в ее логике и терминах. Если логическое проектирование было корректным, эта схема будет очень похожа на схему «сущность-связь», ведь каждой сущности соответствует таблица (сравните рис. 4.14. и рис. 4.15.). Схема таблиц реляционной базы данных сгенерирована автоматически программой SQL Data Modeler фирмы ORACLE, она уже зависит от реализации (марки) СУБД, ведь здесь реализованы таблицы и поля с типами и названиями, созданными по правилам конкретной СУБД: появились таблицы, первичные и внешние ключи, при помощи которых осуществляются связи, добавлены ограничения уникальности, индексы и т.д. Часто опытные разработчики минуют фазу составления схемы «сущность-связь» и сразу делают схему таблиц. Такое возможно только при неукоснительном следовании принципу – «Одна сущность – одна таблица» и при большом опыте ручного создания элементов связей.

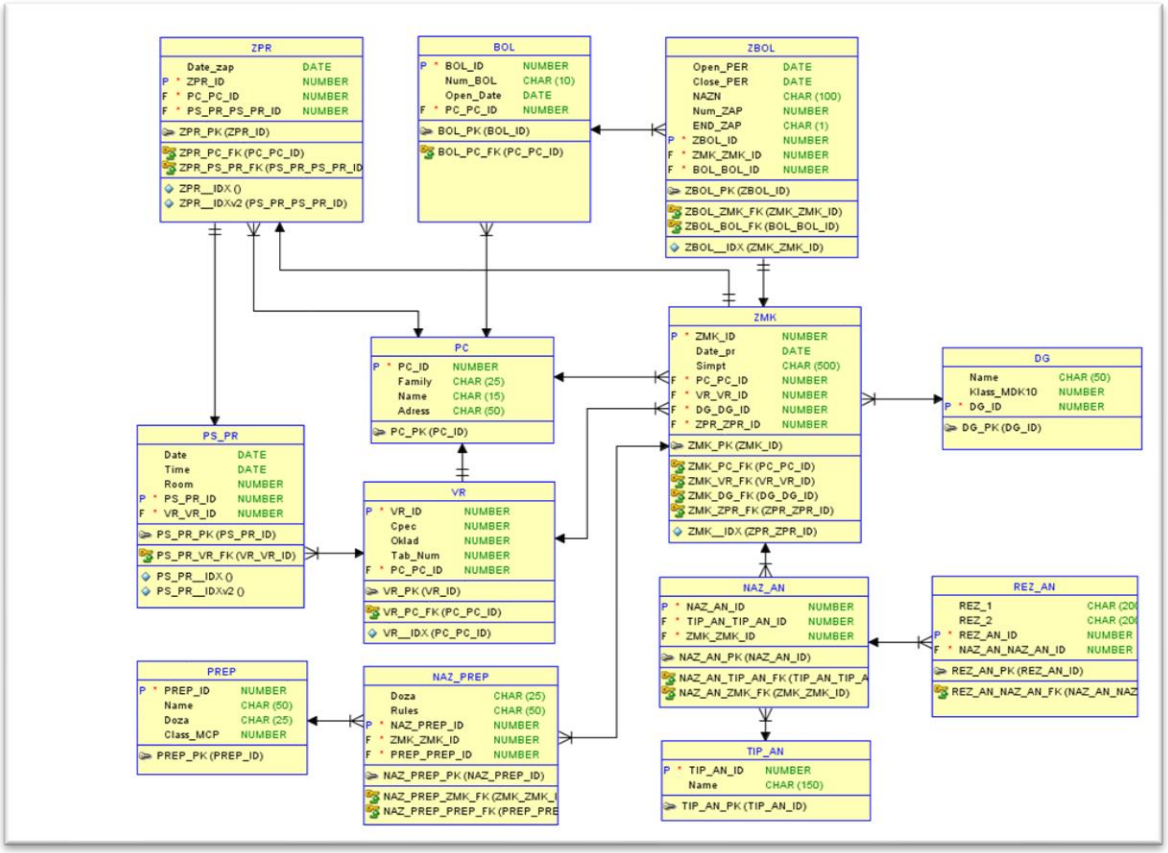

Рис. 4.15. Полная схема таблиц БД для БП «Прием у врача»

Дальше генератор текста языка SQL программы «ORACLE SQL Developer Data Modeler» по этой схеме сформирует скрипт создания БД, этот скрипт уже привязан к конкретной СУБД, и написан на соответствующем диалекте SQL. Итак, мы получили первую итерацию скрипта результирующего создание БД (см. раздел 3.2.), но, это далеко не все, что должно быть в этом скрипте.

4.6.2. Создание представлений

На вышеприведенной схеме мы видим только таблицы, однако у разработчика реляционных БД есть еще один мощный инструмент, позволяющий улучшить характеристики БД, это – представления.

Представление (англ. view,) — виртуальная (логическая) таблица, представляющая собой поименованный запрос (синоним к запросу), который будет подставлен как подзапрос при использовании представления. К представлению обращаются как к обычной таблице, и многие разработчики ИС вообще не знают, что стоит за конкретным именем – таблица или представление. Представления скрывают от ИС сложность запросов и саму структуру таблиц БД.

База данных может использоваться разными приложениями, каждое из которых может иметь свой «взгляд» на предметную область, и этот «взгляд» обеспечивается представлениями, показывающими БД в разных разрезах, скрывая физическую сложность как самой структуры, так и преобразований к различным разрезам предметной области. Вот что дает применение представлений:

1. Представления позволяют использовать запрещенные в обычных таблицах элементы, например, вычислимые поля.
2. Использование представлений позволяет отделить прикладную схему представления данных от схемы хранения.

В словаре СУБД, например, очень сложная структура хранения данных, она оптимизирована на скорость выполнения запросов ядра СУБД, а вот для пользователей доступ к словарю осуществляется на основе представлений, чтобы человек легко мог найти необходимые ему данные.

3. С помощью представлений обеспечивается ещё один уровень защиты данных.

   С одной стороны, мы закрываем таблицы и данные в них от случайных изменений (по умолчанию данные в представлении нельзя изменить, ведь это не настоящая таблица, а результат выполнения запроса), с другой стороны, мы можем закрыть для некоторых пользователей часть таблицы, разрешив доступ к этой таблице через ограничивающее представление. Например, словарь СУБД изменить вручную практически невозможно, т.к. пользователь видит только представления, которые изменить нельзя.

4. СУБД получает возможность применить к запросу, на котором строится представление, оптимизацию или предварительную компиляцию, а значит представления работают намного быстрее, чем обычные запросы.

   Примеры представлений

Листинг 1. Создание представления – Средний оклад по возрастным категориям сотрудников

```
CREATE VIEW CP_Оклад as
SELECT ROUND(EXTRACT(Year from SYSDATE)- EXTRACT(Year from ДАТА_РОЖД),-1) возраст,
-- Вычисляется возраст пациента и округляется до десятков лет
COUNT(*) кол, ROUND(avg(оклад)) "средний оклад"
FROM "СОТРУДНИКИ"
GROUP BY ROUND(EXTRACT(Year from SYSDATE)- EXTRACT(Year from ДАТА_РОЖД),-1)
ORDER BY 1;
```

Листинг 2. Создание представления – Курсы валют на текущую дату

```
CREATE VIEW Курсы as
SELECT КВ."КУРС", ' ', СП."КРАТКОЕ_ИМЯ" Валюта, ' за ',
  КВ."МНОЖИТЕЛЬ" Кол_единиц, ' ', СП2."КРАТКОЕ_ИМЯ" Валюта2, КВ."ДАТА"
FROM "КУРСЫ_ВАЛЮТ" КВ, "СПР_ВАЛЮТ" СП, "СПР_ВАЛЮТ" СП2
-- участвуют две таблицы "КУРСЫ_ВАЛЮТ" "СПР_ВАЛЮТ" связанные по коду валюты
WHERE КВ."ДАТА" =
  (SELECT MAX(КВ2."ДАТА")
  FROM "КУРСЫ_ВАЛЮТ" КВ2
  WHERE КВ."ВАЛ1_ID" = КВ2."ВАЛ1_ID"
   and КВ2."ДАТА" < TO_date('19.09.2017','DD.MM.YYYY'))
-- подзапрос возвращает дату последнего изменения валюты перед заданной датой
 AND КВ."ВАЛ1_ID" = СП."ВАЛ_ID"
 AND КВ."ВАЛ2_ID" = СП2."ВАЛ_ID";
-- таблицы связываются, чтобы найти наименования валют по коду
```

Запрос представленный в листинге 2 подробно разбирается в разделе 4.9.

Работа с представлениями ничем не отличается от работы с таблицами. Примеры можно увидеть в разделе 3.4.15.

4.6.3. Создание хранимых процедур

Хранимая процедура — объект БД, представляющий собой набор инструкций, который компилируется один раз и хранится на сервере в словаре БД. Хранимые процедуры пишутся на SQL или на расширении языка SQL специфичного для каждой отдельной СУБД. Эти процедуры могут иметь входные и выходные параметры, локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения. Все последние версии СУБД для создания таких модулей допускают использование языков программирования общего назначения, таких как Java или C#.

Хранимые процедуры похожи на определяемые пользователем функции. Основное различие заключается в том, что пользовательские функции можно использовать, как и любое другое выражение в SQL запросе, в то время как хранимые процедуры должны быть вызваны с помощью функции CALL или EXECUTE.

Хранимые процедуры используются для обработки ошибок, обеспечения согласованности БД, реализации особо сложных запросов. Особой разновидностью хранимых процедур являются триггеры, отличающиеся способом вызова. Более подробно триггеры рассмотрены в следующем разделе.

4.6.4. Триггеры

Триггер - хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой инициируется СУБД при попытке изменения данных в таблице, с которой он связан. Триггеры применяются для обеспечения согласованности данных и реализации сложной бизнес-логики. Все модификации данных производимые в триггере рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера, соответственно при возникновении ошибки в триггере все вызвавшие его транзакции завершаются неудачно. Надо отметить, что триггеры, также, как и другие встроенные процедуры, пишутся на процедурном расширении SQL принятом в конкретной СУБД или на внешних языках, таких как Java или C#.

Оператор определения триггера в синтаксисе SQL2003:

CREATE TRIGGER имя_триггера
{BEFORE | AFTER} {DELETE | INSERT | UPDATE [OF столбец[, ...]]}
ON имя_таблицы
[**REFERENCING** {OLD {[ROW] | TABLE} [AS] старое_имя | NEW
{ROW | TABLE} [AS] новое_имя}]
[**FOR EACH** {ROW | STATEMENT}]
[**WHEN** (условие)]
[**BEGIN ATOMIC**] блок_кода
[**END**]

BEFORE | AFTER

Указывает, что действия триггера выполняются либо до (BEFORE), либо после (AFTER) DML операции, вызывающей выполнение триггера. Триггер типа BEFORE выполняется до

операторов INSERT, UPDATE и DELETE и позволяет выполнить любые действия, вплоть до полной подмены DML операции.

DELETE | INSERT | UPDATE [OF столбец[, ...]]

Определяет тип операции, вызывающей выполнение триггера: операторы DELETE, операторы INSERT или операторы UPDATE. Также вы можете указать список столбцов для оператора UPDATE, обновление значений которых вызывает триггер. Если обновляется столбец не из указанного списка, то триггер не выполняется.

ON имя_таблицы

Указывает таблицу, при изменении которой будет срабатывать триггер.

REFERENCING {OLD{[ROW] | TABLE} [AS] старое_имя | NEW {ROW | TABLE}

[AS] новое_имя}

Используется для создания псевдонимов для старых (OLD) и новых (NEW) версий строки (ROW) или таблицы (TABLE). Если во фразе REFERENCING используется OLD ROW или NEW ROW, то обязательно должна использоваться фраза FOR EACH ROW.

FOR EACH {ROW | STATEMENT}

Определяет способ вызова триггера: либо один раз для каждой измененной записи (ROW), либо один раз для каждого оператора. Представьте оператор UPDATE, который изменяет зарплаты 100 сотрудников. Триггер FOR EACH ROW в таком случае будет выполнен 100 раз, а триггер FOR EACH STATEMENT будет выполнен один раз.

WHEN (условие)

Позволяет указать условие на запуск триггера. Если при вызове триггера условие при вычислении возвращает TRUE, то выполнение триггера продолжается. В противном случае триггер не запускается.

BEGIN ATOMIC | блок_кода | END

Стандарт ANSI требует, чтобы тело триггера состояло либо из одного SQL оператора, либо было заключено в BEGIN…END.

В некоторых диалектах SQL (например, PostgreSQL) можно создать триггер, подменяющий то действие, которое привело к его срабатыванию. В следующем примере для SQL Server мы создаем триггер INSTEAD OF INSERT для подмены операции вставки в таблицу employee. В триггере в таблицу employee вставляются только сотрудники, находящиеся в штате, а сотрудники, работающие по контракту, вставляются в отдельную таблицу contractors:

Листинг 1. Триггер – Подмена действия (диалект Transact-SQL)

```
CREATE TRIGGER if_emp_is_contractor
INSTEAD OF INSERT ON employee
BEGIN
INSERT INTO contractor
SELECT * FROM inserted WHERE status = 'CON'
INSERT INTO employee
SELECT * FROM inserted WHERE status = 'FTE'
END
GO
```

Триггеры используются в следующих случаях:

1. Для автоматической генерации значений полей, например, для присвоения значения суррогатного ключа при создании записи (см. листинг 2 и 3).
2. Для логирования при ограничении доступа.
3. Для предотвращения DML операций записи (см. листинги 4 и 6).
4. Для реализации сложных ограничений согласованности данных (см. листинг 4).
5. Для организации всевозможных видов аудита и сбора статистики записи (см. листинг 5).
6. Для оповещения других модулей о том, что делать в случае изменения информации в БД, так как передача параметров невозможна, производится запись во временные таблицы данных управляющих последующим выполнением программы.
7. Для реализации бизнес логики записи (см. листинг 6).
8. Для организации каскадных воздействий на таблицы БД записи (см. листинг 7).

Листинг 2. Триггер – Заполнение первичного ключа (диалект ORACLE)

```
CREATE OR REPLACE TRIGGER VR_ID_TRG
BEFORE INSERT ON VR FOR EACH ROW
WHEN (:new.VR_ID IS NULL)
BEGIN
:new.VR_ID := VR_ID_SEQ.NEXTVAL;
END;
```

Листинг 3. Триггер – Заполнение первичного ключа (диалект PostgreSQL)

```
-- Создание последовательности
CREATE SEQUENCE pk_seq INCREMENT BY 1 START WITH 25;



-- Создание триггерной функции
CREATE FUNCTION pk_wr () RETURNS trigger AS
$$
BEGIN
NEW.pk := nextval('pk_seq');
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

-- Создание триггера
CREATE TRIGGER pk_w
BEFORE INSERT ON s FOR EACH ROW
WHEN (NEW.pk IS NULL)
EXECUTE PROCEDURE pk_wr ();
```

Листинг 4. Триггер – Запрет записи пациента не в рабочее время

```
CREATE OR REPLACE TRIGGER Огр_записи
BEFORE INSERT ON ZPR
BEGIN
 IF TO_CHAR (SYSDATE, 'HH24:MI') NOT BETWEEN '08:00' AND '18:00'
  OR TO_CHAR (SYSDATE, 'DY') IN ('SAT', 'SUN') THEN
     RAISE_APPLICATION_ERROR (-20205,
     'Запись производится только с 8:00 до 18:00');
 END IF;
END Огр_записи;
```

Листинг 5. Триггер – Сохранение истории записи врача

```
CREATE OR REPLACE TRIGGER Ист_вр
AFTER UPDATE OF Oklad, Cpec ON VR
FOR EACH ROW
BEGIN
 INSERT INTO job_history (VR_id, date_upd, Oklad, Cpec)
  VALUES(:old.VR_id, sysdate, :old.Oklad, :old.Cpec);
END Ист_вр;
```

Листинг 6. Триггер – Запрет удаления записи врача

```
CREATE OR REPLACE TRIGGER Огр_удал
BEFORE DELETE ON VR
BEGIN
 RAISE_APPLICATION_ERROR (-20205,
   'Удаление записи о враче невозможно');
END Огр_удал;
```

Листинг 7. Триггер – Каскадное удаление записей из таблицы «Расписание»

```
CREATE OR REPLACE TRIGGER Каск_удал
BEFORE DELETE ON VR
BEGIN
 DELETE FROM ZPR
WHERE :old.VR_id = ZPR.VR_VR_id;
END Каск_удал;
```

Обратите внимание на то, что при необходимости прервать действие, которое привело к срабатыванию триггера производится генерация внутренней ошибки записи (см. листинги 2 и 4) и все внешние транзакции завершаются неудачно, а управление передается обработчику ошибок.

Триггеры могут использовать две псевдотаблицы. (Псевдотаблицами они называются, так как не создаются явно с помощью оператора CREATE TABLE, но тем не менее, логически существуют в базе данных.) Эти псевдотаблицы могут называть в разных платформах по разному, здесь мы их будем именовать Before и After. Они имеют в точности такую же структуру, что и таблица, с которой связан триггер, но содержат моментальные копии содержимого таблицы: Before содержит копию всех строк таблицы, какими они были до выполнения оператора, вызвавшего триггер, After содержит копию всех строк таблицы после выполнения оператора. Вы можете использовать данные этих псевдотаблиц в триггере для определения того, что вы хотите сделать. Именно эти псевдотаблицы присутствуют в листингах 1-5: в полях с префиксом «:old» содержатся значения до изменения записи (Before), тогда как в полях с префиксом «:new» содержатся значения после изменения (After).

4.7. Разграничение доступа и создание ролей пользователей

4.7.1. Что такое разграничение доступа

Любые действия в СУБД производятся от имени какого-нибудь пользователя и во время входа система должна его аутентифицировать и предоставить возможность работы в соответствии с выделенными правами. Средства аутентификации, как и средства разграничения доступа могут различаться в разных СУБД, мы рассмотрим только наиболее распространенные.

В современных СУБД достаточно развиты средства дискреционной защиты. Дискреционное управление доступом (discretionary access control) — разграничение доступа между поименованными субъектами и поименованными объектами. Под *субъектами* здесь понимаются пользователи СУБД, а под *объектами* – объекты БД, такие как таблицы, представления, индексы и т.д., под *доступом* понимается возможность использования данных из объектов БД (содержимое таблиц), так и использование самих объектив (индексы, хранимые процедуры и т.д.). Т.е. каждый пользователь имеет конкретные права на работу с конкретными объектами БД, например, доступ только по чтению к таблицам «Страны», «Города» и «Улицы» или доступ к созданию, изменению и удалению всех объектов БД «Адреса».

В настоящем документе разграничение доступа рассматривается только с точки зрения дискреционной защиты, реализуемой средствами СУБД. В настоящее время все СУБД поддерживают разграничение прав доступа хотя бы на уровне стандарта языка SQL, в который встроены средства разграничения доступа (см. раздел 3.4.19.), однако, все они делают это по-разному – всеохватывающую систему дискреционной защиты имеет только ORACLE, поэтому мы рассмотрим проектирование системы доступа на примере СУБД ORACLE.

4.7.2. Пользователи, их схемы и роли

При создании пользователя СУБД создается и его схема – именованный набор объектов и данных, принадлежащих конкретному пользователю. Понятие схемы примерно совпадает с понятием базы данных. Сразу после создания схема пустая и создать в этой схеме объекты может только хозяин схемы или тот пользователь, которому будут даны права на объекты этой схемы. Права на объекты или данные схемы может дать либо владелец схемы, либо администратор СУБД. При развертывании и эксплуатации большой БД приходится раздавать десятки прав на сотни, а то и тысячи объектов, сотням пользователей.

Надо отметить, что схема пользователя – это обособленное пространство имен, которое имеет свое наименование, совпадающее с именем пользователя. Такое поименованное пространство имен имеет сходство с псевдонимами таблиц в запросах и распространяется на все объекты схемы пользователя, так полное имя таблицы «СОТРУДНИКИ» будет «BOOK.СОТРУДНИКИ», где префикс BOOK указывает на принадлежность таблицы к схеме пользователя BOOK от чьего имени и были созданы все примеры в этой книге. Другие пользователи, зарегистрированные в СУБД, тоже могут создать в своей схеме таблицу «СОТРУДНИКИ», но полное имя этой таблицы будет другое.

Чтобы упростить и упорядочить процедуру раздачи прав создаются роли пользователей. По сути, роль — это набор прав на конкретные объекты БД, позволяющие выполнять некий набор бизнес - процессов. Например, все преподаватели–пользователи ИС ВУЗа обладают одинаковыми правами и им присваивается роль – «преподаватель», а некоторые еще и являются кураторами групп и им присваивается роль – «куратор», в результате права этих ролей складываются. Аналогично, некоторые студенты кроме роли «студент» обладают еще и ролью «староста». Таким образом не нужно раздавать права на конкретные объекты БД конечным пользователям, достаточно назначить пользователям набор ролей для выполнения их должностных обязанностей.

4.7.3. Предопределённые роли пользователей СУБД

В СУБД ORACLE, сразу после ее развертывания автоматически создаются три предопределённые роли:

- CONNECT — работает с данными (не может создавать и изменять объекты БД).
- RESOURCE — работает с объектами БД (может их создавать, модифицировать, удалять и раздавать на них права).
- DBA — работает с объектами СУБД (пользователями, их схемами, табличными пространствами, устройствами хранения данных и т.д.).

Здесь мы видим четкое распределение прав по работе с данными, объектами БД и объектами СУБД.

Конечный пользователь ИС, работающий с данными и имеющий все права на эти данные, обладает правами роли CONNECT, эта роль и дается ему при создании.

Пользователь, обладающий правами RESOURCE, это обычно владелец схемы БД, разработчик ИС, имеет все права на все объекты БД. Он также обладает всеми правами роли CONNECT, хотя никогда не правит данные в работающей системе, работать с данными он может только при отладке системы и при устранении сбойных ситуаций.

Пользователь, обладающий правами DBA, это владелец всей схемы СУБД, и имеет все права на все объекты всех схем всех пользователей и, соответственно обладает всеми правами ролей CONNECT и RESOURCE. Пользователь DBA работает в основном с объектами СУБД – создает пользователей, схемы, рабочие пространства и их копии, следит за аппаратурой на, которой работает СУБД. Обычно пользователь DBA не пользуется своими правами, для действий, связанных с производственными задачами других пользователей, единственная ситуация, когда это возможно – устранение последствий сбоев системы.

При создании нового пользователя он наделяется одной из вышеупомянутых ролей.

На самом деле в момент инсталляции создается большое количество предопределенных ролей, которые используются для определения прав пользователей, вот например некоторые предопределенные роли в PostgreSQL:

Таблица 4.1. Некоторые предопределенные роли пользователей в PostgreSQL

Роль	Разрешаемый доступ
pg_read_all_data	Позволяет читать все данные (таблицы, представления, последовательности), как будто роль имеет права SELECT на эти объекты и права USAGE на все схемы, но при этом явным образом такие права ей не назначены.
pg_write_all_data	Позволяет записывать все данные (таблицы, представления, последовательности), как будто роль имеет права INSERT, UPDATE и DELETE на эти объекты и права USAGE на все схемы, но при этом явным образом такие права ей не назначены.
pg_read_all_settings	Позволяет читать все конфигурационные переменные, даже те, что обычно видны только суперпользователям.

Роль	Разрешаемый доступ
pg_read_all_stats	Позволяет читать все представления pg_stat_* и использовать различные расширения, связанные со статистикой, даже те, что обычно видны только суперпользователям.

4.7.4. Создание ролей пользователей

Присвоив пользователю одну из ролей, описанных в п 10.3. необходимо дать ему права, определяющие его роль в ИС.

На основании анализа БП мы можем определить какие роли пользователей нам нужны для обеспечения работы ИС.

Таблица 4.2. Роли пользователей в проекте банковской БД

Роль	Зона ответственности (разрешения доступа)
Операционный работник / клиентский менеджер	Оформление договора потребительского кредита, предоставление кредита, прием и обработка обращений клиентов
Сотрудник бэк-офиса	Выполнение плановых операций по договорам потребительских кредитов (начисление процентов, погашения, оформление досрочных погашений, оформление и погашение просроченных задолженностей, завершение договоров)
Кассир	Прием и выдача наличных денежных средств, связанных с предоставлением или погашением потребительского кредита

Для проекта БД поликлинике также можно выделить роли пользователей исходя из описанных БП (см. рис. 3,4).

Таблица 4.3. Роли пользователей в проекте БД поликлиники

Роль	Зона ответственности (разрешения доступа)
Администратор	Регистрация пациентов, заведение карточки пациента, поддержание в актуальном состоянии справочников ИС
Врач специалист	Прием больных, заполнение карточки пациента, оформление больничных
Главврач	Составление расписаний, графиков дежурств и отпусков, работа с врачами

На рис. 4.16. представлено распределение доступа в соответствии с ролями, перечисленными в таблице 3. Розовым фоном выделены таблицы, к которым имеет полный доступ «Врач специалист», салатовым фоном – «Администратор», без фона - «Главврач»,

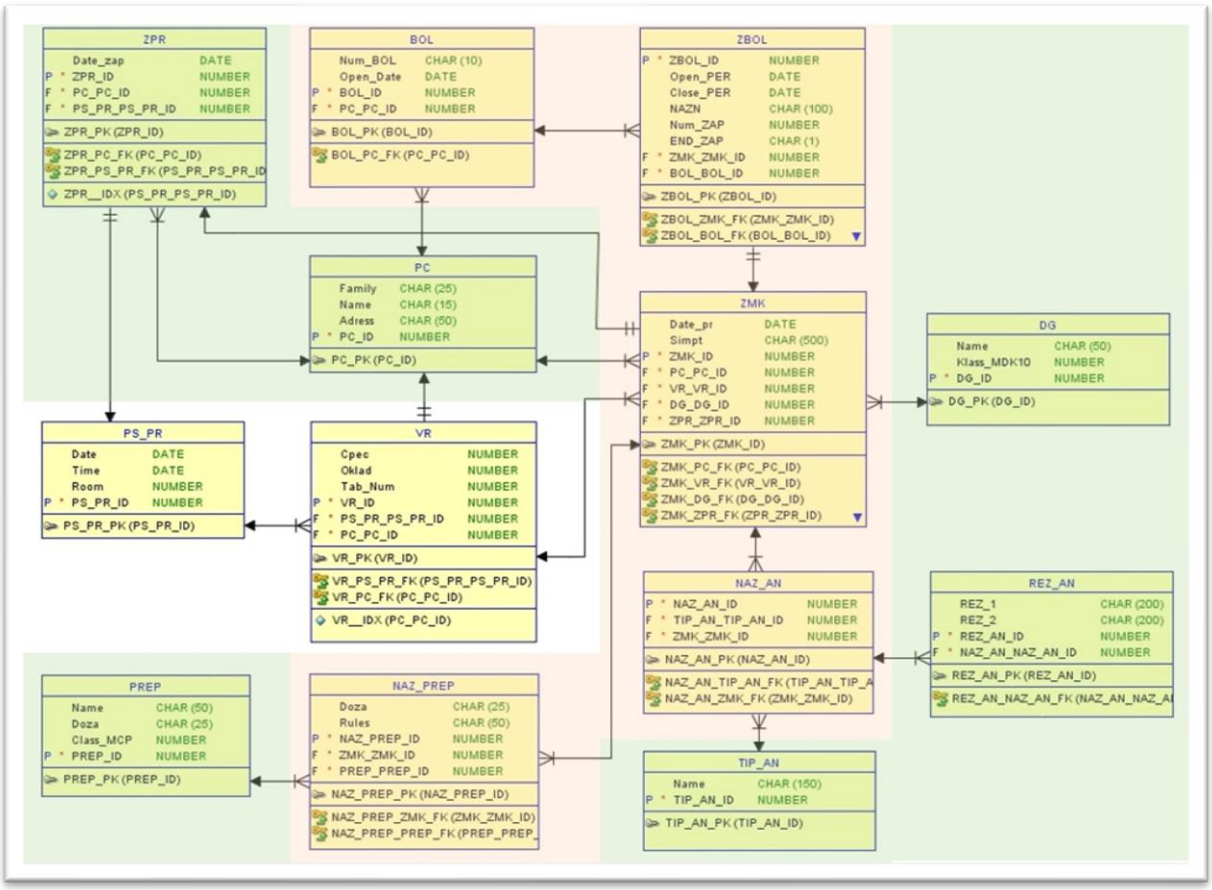

Рис. 4.16. Полная схема таблиц БД для БП «Прием у врача» с распределением доступа

4.7.5. Операторы SQL для определения доступа к данным

Для того чтобы предоставить те или иные привилегии пользователям или ролям пользователей используется оператор GRANT. Ниже представлен его синтаксис:

GRANT { {объектная_привилегия[, ...] | роль[, ...]} }
[ON объект_базы_данных]
[TO получатель[, ...]]
[WITH HIERARCHY OPTION] [WITH GRANT OPTION]
[WITH ADMIN OPTION] [FROM {CURRENT_USER | CURRENT_ROLE}]

Предоставляет «объектную_привилегию» на выполнение различных SQL операторов над упомянутым «объектом_базы_данных» «получателю» (пользователю или роли). Может также присваивать указанную «роль» «получателю».

WITH HIERARCHY OPTION

Позволяет выполнять SELECT не только для указанной таблицы, но и для всех ее подтаблиц. Используется только при выдаче привилегий.

WITH GRANT OPTION

Позволяет пользователю передавать дальше полученные привилегии. Используется только при выдаче привилегий.

WITH ADMIN OPTION

Дает пользователю возможность назначать полученную роль другим пользователям.

FROM {CURRENT_USER | CURRENT_ROLE}

Указывает, что привилегии выдаются либо от имени текущего пользователя (CURRENT_USER), либо от имени текущей роли (CURRENT_ROLE).

Работа с ролями пользователей, указанными в таблице 3 представлена в листинге 1.

Листинг 1. Создание ролей в проекте БД поликлиники

```
CREATE ROLE ГлВрач; -- Создание роли «ГлВрач»
GRANT ALL ON VR TO ГлВрач; -- Полные права на таблицу «VR»
GRANT ALL ON PS_PR TO ГлВрач;
GRANT SELECT ON ZMK TO ГлВрач; -- Права на чтение таблицы «ZMK»
… здесь перечисляются все таблицы ИС, которые доступны роли «ГлВрач» по чтению
GRANT SELECT ON ZMK TO ГлВрач; -- Права на чтение таблицы «TIP_AN»
GRANT ГлВрач TO smirnova; -- роль «ГлВрач» назначена пользователю «smirnova»
CREATE ROLE Врач;
GRANT ALL ON BOL TO Врач; -- Создание роли «Врач»
GRANT ALL ON ZBOL TO Врач; -- Полные права на таблицу «ZBOL»
GRANT ALL ON ZMK TO Врач;
GRANT ALL ON NAZ_AN TO Врач;
…
```

4.8. Обеспечение согласованности БД

Согласованность базы данных (как уже указывалось в разделе 2.2.4.)— соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам и ограничениям. Эти правила и ограничения определяются особенностями предметной области.

Очевидно, что эти правила и ограничения должны быть формально объявлены для СУБД, после чего СУБД должна отслеживать их выполнение. Объявление ограничений сводится просто к использованию соответствующих средств языка SQL, а отслеживание ограничений осуществляется СУБД с помощью контроля операций над данными, которые могут нарушить эти ограничения, и запрещения тех операций, которые их действительно нарушают.

4.8.1. Виды согласованности БД

Разделяют несколько видов согласованности БД:

1. Сущностная согласованность

   Поддержание сущностной согласованности состоит в недопущении многократного попадания данных об одной сущности в таблицу БД. Если использовать «естественные» уникальные ключи, такие как номер паспорта или СНИЛС то сущностная согласованность поддерживается автоматически. При использовании суррогатных первичных ключей добиться этого можно, используя уникальные

«естественные» вторичные ключи, такие как, уже упомянутые, номер паспорта или номер зачетки и т.д.

2. Доменная согласованность

   Доменная согласованность — это достоверность данных в конкретном поле. Она включает в себя ограничения типа данных, ограничения формата при помощи ограничений CHECK и правил, а также ограничения диапазона возможных значений при помощи ограничений FOREIGN KEY, CHECK, DEFAULT, определений NOT NULL и т.д. После того как ограничения объявлены их выполнение автоматически отслеживает СУБД.

3. Ссылочная согласованность

   Ссылочная согласованность гарантирует согласованность значений первичных и внешних ключей во всех таблицах. Обычно ссылочная согласованность поддерживается триггерами, автоматически заполняющими значение ключей или обеспечивающие невозможность их изменения. Необходимо помнить, что первичные суррогатные ключи заполняются автоматически СУБД, а внешние ключи заполняются только копированием значений первичных ключей.

4. Пользовательская согласованность

   Пользовательская согласованность определяется бизнес-правилами, не входящие ни в одну из первых трех категорий согласованности, эти правила определяются и отслеживаются не на уровне СУБД, а на уровне ИС, работающей с БД. Например, баланс банка должен сходится или число проданных и непроданных билетов на авиарейс должно в сумме равняться вместимости самолета. Пользовательская согласованность поддерживается при помощи транзакций и использования триггеров, при этом нужно помнить, что отслеживание пользовательской согласованности БД целиком ложится на плечи разработчика ИС.

4.8.2. Механизмы обеспечения согласованности БД

В обеспечение согласованности важную роль играет использование транзакций (см. раздел 2.2.5), триггеров (см. раздел 4.6.4) и хранимых процедур (см. раздел 4.6.3).

Транзакция — логическая единица работы, состоящая из одного или нескольких операторов манипулирования данными, которую СУБД рассматривает и обрабатывает как неделимое действие, переводящее БД из одного согласованного состояния в другое согласованное состояние.

Из этого определения видно, что механизм транзакций надо использовать везде, где необходимо обеспечить согласованность БД.

Транзакции должны выполняться максимально быстро. Долгие транзакции увеличивают вероятность, что пользователи не получат доступ к заблокированным данным.

Следующие рекомендации позволят вам сделать транзакции как можно короче:

- Без особой надобности не выполняйте в транзакции запросы SELECT. Выборка данных должна происходить вне транзакции;
- Для уменьшения времени транзакции, будьте внимательны, когда используете долго выполняемые операторы процедурных расширений SQL, такие как циклы и создание объектов базы данных;
- Не требуйте от пользователя ввода данных во время выполнения транзакции. Делайте ввод данных до начала выполнения транзакции.
- Операторы INSERT, UPDATE и DELETE должны быть главными в транзакции, и они должны быть написаны так, чтобы получать минимальный набор записей, что позволяет повысить скорость работы любого запроса.

- Все проверки данных и подготовительные расчеты необходимо произвести до начала транзакции. После оператора BEGIN TRANSACTION должно выполняться только изменение данных.

Еще одним механизмом обеспечения согласованности БД являются хранимые процедуры и триггеры, предназначенные для поддержания безопасности и согласованности на уровне объектов. Процедуры и триггеры представляют собой скомпилированные модули (в стандарте SQL:2003 определены процедуры уровня схемы и уровня модуля), реализованные на некотором языке программирования, внутреннем или внешнем, и обычно хранимые на сервере СУБД. Все последние версии СУБД для создания таких модулей допускают использование языков программирования общего назначения, таких как Java или C#.

Основной задачей хранимых процедур в области обеспечения согласованности является внедрение в модули, хранимые на сервере, определенных правил бизнес-логики, и обеспечение доступа пользователя к данным только посредством этих процедур. К примеру, можно создать набор хранимых процедур, в которых реализованы операции вставки, обновления и удаления данных. При этом пользователям предоставляются разрешения на использование этих процедур и запрещается непосредственное обращение к соответствующим таблицам.

4.8.3. Удаление данных

В различной литературе довольно большое место уделяется процессу удаления данных и возникающим вследствие этого нарушениям ссылочной согласованности БД. Рассматриваются многочисленные методы и способы приведения БД к согласованному состоянию после удаления данных.

В противовес этим источникам необходимо со всей категоричностью заявить, что из БД ничего удалять нельзя. Любое удаление данных из БД приводит к невозможности получения аналитических отчетов за период, в котором были удалены данные. Если из базы данных кафедры ВУЗА удалить, например, отчисленных студентов, то нельзя будет получить списки групп, статистику посещаемости и успеваемости за прошлые годы. Исходя из невозможности удаления данных, надо предусмотреть методы поддержания изменений состояния сущностей, атрибутов и даже связей между сущностями в процессе жизни и развития ИС (см. раздел «Поддержка историчности»), а также методы ускорения работы БД при постоянном росте ее объемов (см. разделы «Секционирование таблиц» и «Кластеризация таблиц»). В редких случаях в процессе эксплуатации ИС приходится удалять некоторые данные, но это делается вне ИС, администратором БД и последствия этих действий целиком ложатся на его плечи.

Иногда записи таблицы удаляют, если они были вставлены по ошибке. К примеру, сведения о заказчике могут быть по ошибке введены дважды. Такая ситуация типична в крупных компаниях, где вводом данных о заказах одновременно занимается множество операторов. Удаление таких записей, чаще всего возникающих из-за нарушения технологии ввода данных, должны выполнять лица занимающие должности начальников операторов ввода, призванные не допускать нарушения технологии и разбирающиеся в каждом отдельном случае нарушений.

Конечно, правило «из БД ничего удалять нельзя» не относится к процессу разработки и отладки ИС, в эти периоды жизненного цикла удаление данных из БД – обычное дело, только надо помнить о согласованности данных.

4.9. Поддержка историчности

В этом разделе мы обсудим каким образом осуществляется сохранение истории изменений данных. Сохранение истории изменения данных очень часто встречающаяся задача, например, необходимо хранить все изменения зарплат сотрудников, чтобы получать правильные справки для налоговых органов за длительные промежутки времени, одновременно с этим возникает необходимость получения зарплаты сотрудника на конкретную дату. Итак, у нас есть поле Oklad

содержащее размер оклада врача на текущий момент. При изменении значения этого поля необходимо сохранить предыдущее значение в отдельной таблице (в данном случае «job_history»), которая и будет хранилищем истории изменения данных о врачах. Для осуществления такого сохранения идеально подойдет механизм триггеров. Триггер запоминающий предыдущее значение полей таблицы «VR» приведен на листинге 4.9.1.

Листинг 4.9.1. Триггер – Сохранение истории зарплат

```
CREATE OR REPLACE TRIGGER Ист_вр
AFTER UPDATE OF Oklad ON VR
FOR EACH ROW
BEGIN
 INSERT INTO job_history (VR_id, date_upd, Oklad, Cpec)
-- сохранение значений таблицы «VR» в таблице истории «Ист_вр»
  VALUES(:old.VR_id, sysdate, :old.Oklad, :old.Cpec);
END Ист_вр;
```

Листинг 4.9.2. Запрос – Нахождение зарплат всех специалистов на заданную дату

```
SELECT JH.Oklad, JH.Cpec
FROM job_history JH
WHERE JH.date_upd =
  (SELECT MAX(JH2.date_upd)
  FROM job_history JH2
  WHERE JH2.Cpec = JH.Cpec
    and JH2.date_upd < TO_DATE('19.09.2017','DD.MM.YYYY'))
-- подзапрос возвращает дату последнего изменения валюты перед заданной датой
```

Более сложный пример – нахождение курсов валют на заданную дату приведен в листинге

Листинг 4.9.3. Запрос – Курсы валют на заданную дату

```
SELECT КВ."КУРС", ' ', СП."КРАТКОЕ_ИМЯ" Валюта, ' за ',
  КВ."МНОЖИТЕЛЬ" Кол_единиц, ' ', СП2."КРАТКОЕ_ИМЯ" Валюта2, КВ."ДАТА"
FROM "КУРСЫ_ВАЛЮТ" КВ, "СПР_ВАЛЮТ" СП, "СПР_ВАЛЮТ" СП2
-- участвуют две таблицы "КУРСЫ_ВАЛЮТ" "СПР_ВАЛЮТ" связанные по коду валюты
WHERE КВ."ДАТА" =
  (SELECT MAX(КВ2."ДАТА")
  FROM "КУРСЫ_ВАЛЮТ" КВ2
  WHERE КВ."ВАЛ1_ID" = КВ2."ВАЛ1_ID"
   and КВ2."ДАТА" < TO_DATE('19.09.2017','DD.MM.YYYY'))
-- подзапрос возвращает дату последнего изменения валюты перед заданной датой
 AND КВ."ВАЛ1_ID" = СП."ВАЛ_ID"
 AND КВ."ВАЛ2_ID" = СП2."ВАЛ_ID";
-- таблицы связываются, чтобы найти наименования валют по коду
```

Конечно, в реальном приложении вместо константы даты (выделено жирным шрифтом) будет подставлено значение поля, в котором пользователь введет нужную ему дату, а в выходную таблицу будут вставлены спец. символы форматирования, однако текст запроса останется такой же как в листингах 4.9.2, 4.9.3.

Результат работы запроса из листинга 4.9.3 представлен на рис. 4.15.

	КУРС	"	ВАЛЮТА	'ЗА'	КОЛ_ЕДИНИЦ	"_1	ВАЛЮТА2	ДАТА
1	58,22		RUB	за	1		USD	18.09.17
2	1,14		EUR	за	1		USD	15.09.17
3	0,92		USD	за	100		JPY	18.09.17

Рис. 4.15. Курсы валют на заданную (19.09.2017) дату

4.10. Обеспечение быстродействия БД

4.10.1. Создание индексов

Что такое индексы

Таблицы в базе данных могут иметь большое количество записей, которые хранятся в произвольном порядке, и поиск нужной записи путём последовательного перебора занимает много времени. Чтобы ускорить поиск записей используются индексы. Индекс — это вспомогательная таблица, которая формируется из значений одного или нескольких полей индексируемой таблицы и указателей на соответствующие записи. Простейший индекс показан на рис 4.16.

Рис. 4.16. Простейший индекс

В данном случае индекс представляет собой таблицу, которая состоит из полей индекса – т.е. копии индексируемого поля таблицы и поля, указывающего на запись в индексируемой таблице. Записи в таблице индекса упорядочены по значению поля «фамилия» в порядке

возрастания. Здесь ускорение работы происходит за счет того, что не надо перебирать большую таблицу, при поиске нужной фамилии, а поскольку записи упорядочены по алфавиту можно применить, например, дихотомический поиск. А если требуется просто выдать все фамилии на букву «Ж», то и обращаться к основной таблице не надо, достаточно обращение к индексу.

Но чаще всего используются индексы, построенные на основе B-дерева (балансированное дерево). Ускорение работы здесь достигается за счёт того, что индексная таблица имеет структуру, оптимизированную под поиск. На рис. 4.17 показана структура индексной таблицы, построенной на основе B-дерева (балансированное дерево).

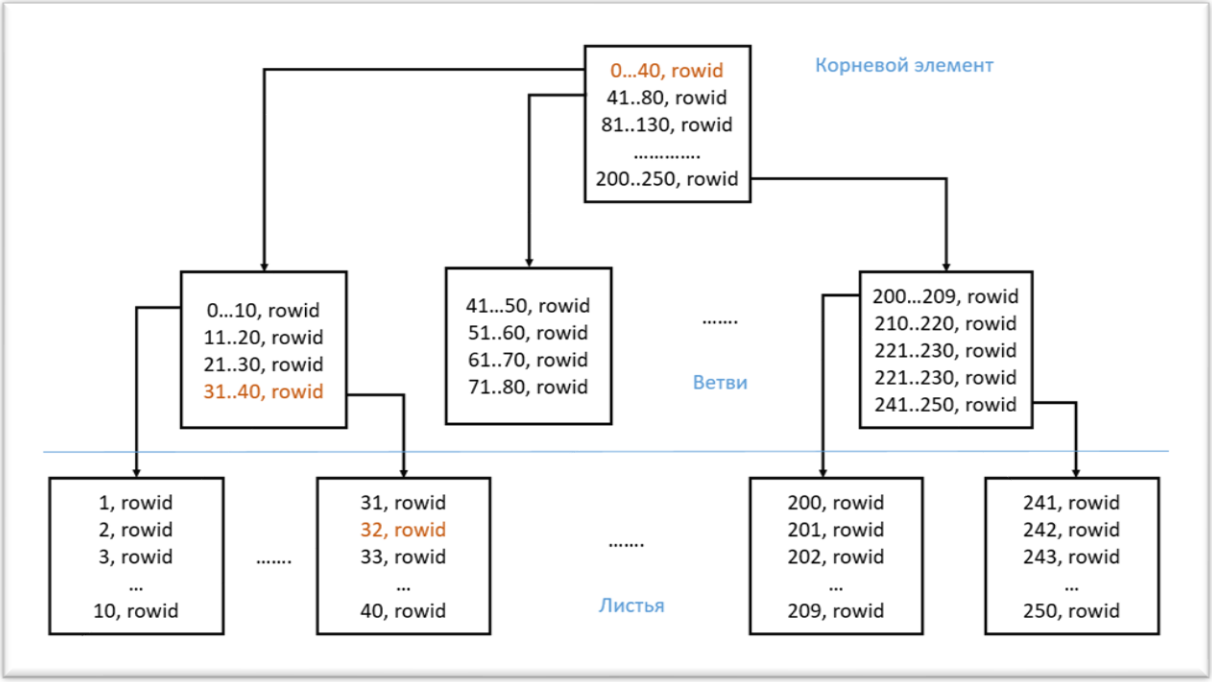

Рис. 4.17. Простейший индекс на основе B-дерева.

Каждый элемент представляет собой запись в таблице, независимо от того относится он к корню, веткам или листьям. Эти элементы состоят из диапазонов индекса и уникальному ID записи (rowid, подробнее см. раздел 3.3) следующего элемента, раскрывающего вышеупомянутый диапазон индекса. Поиск по такому индексу всегда начинается с корневого элемента, и ищется диапазон, который содержит искомое значение, а затем необходимо перейти к следующему элементу в соответствии с rowid и повторить поиск диапазона. Этот процесс происходит до тех пор, пока не будет найдено искомое значение индекса и ссылка на соответствующую запись в индексируемой таблице. Например, нам нужно найти запись, соответствующую индексу «32» (см. рис. 17): начинается просмотр корневого элемента и первый же диапазон содержит искомый индекс (помечено красным), далее осуществляется переход по rowid к элементу, раскрывающему диапазон индекса от 0 до 40. После перебора диапазонов этого элемента, находится rowid лист дерева, в котором уже можно найти rowid указывающий на запись индексируемой таблицы, соответствующую индексу «32».

При разработке БД необходимо выделить часто исполняемые запросы и проанализировав таблицы, используемые в запросах, выделить поля, по которым чаще всего осуществляется поиск или сортировка. Внимания так же заслуживают поля, по которым производится связывание таблиц – первичные и внешние индексы. Именно по этим полям многие СУБД автоматически создают индексы (см. листинг 4.10.1.). Некоторые СУБД позволяют создавать индексы по суперпозиции полей или по выражению, содержащему поля таблицы. Составные индексы создаются тогда, когда необходимо осуществлять поиск в таблице по нескольким полям, например, запись в медкнижке ищется обычно по пациенту и дате приема (см. листинг 4.10.2).

Ваша задача — спроектировать и создать индексы, которые лучше всего подходят для работы с конкретной БД, чтобы оптимизатор запросов мог выбирать из нескольких индексов наиболее эффективный для конкретного запроса.

Листинг 4.10.1. Создание индекса по первичному ключу таблицы «Врачи»

```
CREATE UNIQUE INDEX VR_IN_id
ON VR (VR_id)
```

Листинг 4.10.2. Создание индекса таблицы «Запись в медицинской книжке»

```
CREATE UNIQUE INDEX ZMK_IN_DP
ON ZMK (PC_PC_ID, Date_pr) -- индекс для поиска записи в медкнижке по
                           -- пациенту и дате приема
```

Что нужно знать об индексах

Необходимо запомнить ряд правил, которые обязательны к исполнению при работе с индексами:

1. Индексы создаются для ускорения поиска и выборки.
2. Индексы замедляют все операции кроме выборки.

   Все операции с таблицами, кроме выборки, изменяют информацию и, следовательно, ведут к перестроению всех индексов, связанных с этими таблицами, что может занять достаточно большое время.
3. Индексы занимают место.

   Для больших таблиц индексы могут занимать много памяти, а если индексов много, то они могут занять места больше, чем сама таблица.
4. Для маленьких таблиц индексы не нужны.

   Простой перебор для небольших таблиц (менее 1000 записей) эффективней, чем работа с индексом, а значит индексы для таких таблиц не нужны.
5. Для таблиц, которые часто обновляются используйте как можно меньше индексов.

   Частое обновление таблицы ведет к частому перестроению индексов и замедляет работу системы.
6. Для временных таблиц индексы не нужны.

   Временные таблицы часто изменяются и часто очищаются – это ведет к огромной работе по обновлению индексов, которая сводит на нет все преимущества индексации.
7. Для составного индекса НЕ важен порядок полей в индексе.

   В СУБД начала века, действительно, можно было получить выигрыш при правильной комбинации полей в составном индексе и это упоминается в разной литературе. В настоящее время все СУБД оптимизированы так, что порядок полей в составном индексе не важен.

4.10.2. Кластеризация таблиц

Как известно записи в реляционной таблице не упорядочены, т.е. хранятся в виде кучи, однако реальные таблицы в современных СУБД содержат записи, упорядоченные по каким-либо правилам. Одним из таких видов упорядочивания и является кластеризация таблиц.

Кластеризация— это попытка разместить рядом в одном физическом блоке данные тех записей, доступ к которым осуществляется при помощи одинаковых значений индекса.

Документы

Блок памяти	Первичный ключ	Дата документа	№	Сумма	...
		21.12.2015	1		
		...	...		
		21.12.2015	137		
		20.12.2015	1		
		...	...		
		20.12.2015	184		
		19.12.2015	1		
		...	...		

Рис. 4.18. Кластеризованная таблица.

На рис. 4.18. представлена таблица «Документы», записи в которой упорядочены по дате и по номеру документа (как известно, согласно правилам документооборота, нумерация документов каждый новый день начинается с 1). Документы чаще всего запрашиваются для отчетов в некотором временном диапазоне – за день, за неделю, за квартал, за год и т.д., поэтому при такой упорядоченности достигается максимальная скорость чтения данных. По упорядоченным таким способом записям можно построить кластеризованной индекс, при работе с которым, осуществляется наиболее высокая скорость выборки данных. Так как записи в таблице могут храниться в единственном порядке, значит и кластеризованной индекс может быть только один на таблицу. Заметим, что кластеризованной индекс строится чаще всего не по суррогатным ключам, а по действительным атрибутам сущности или их сочетанию, по которым чаще всего ведется выборка данных (см. рис. 4.18). Впрочем, если нет явных предпочтений по правилам выборки данных, правильным будет построения кластеризованного индекса по первичному или внешнему ключу, что ускорит соединение таблиц в запросах Кластеризация, также, как и кластеризованные индексы, по-разному реализуются и поддерживаются в различных СУБД, поэтому ознакомление с документацией обязательно перед тем, как использовать кластеризацию. Кластеризация, и создание кластеризованных индексов обычно проводится в момент создания таблицы т.к. динамическое создание кластеризованных индексов (особенно для больших таблиц) может привести к длительной блокировке и затруднению работе с системой.

4.10.3. Секционирование таблиц

Еще одной методикой по повышению производительности работы SQL-запросов по выборке и модификации является секционирование таблиц. Увеличение производительности достигается за счет того, что поиск и модификация записей идут не по всей таблице, а только в ее части (в одной или нескольких секциях). Эти секции могут храниться на разных носителях, отличающихся как быстродействием, так и стоимостью. Важным шагом в создании секционированной таблицы является определение ключа секционирования. В качестве ключа секционирования может выступить поле или несколько полей, относительно значений которых, будет выполняться разнесение таблицы на секции. Например, документы, датируемые текущим и предыдущим годом, составляют одну секцию, а документы, датируемые более ранними годами – другую секцию таблицы, т.к. система чаще всего работает с текущей информацией, и аналитические документы чаще всего не собираются на глубину более года. Можно

секционировать таблицу по полю, содержащему дату последнего изменения записи, тогда работа с последними изменениями будет быстрее.

Ни одна из СУБД не может выполнить секционирование таблицы в процессе работы, т.е. секционирование должно быть задано при создании таблицы, поэтому принципы разделения на секции необходимо продумать и предусмотреть при проектировании БД. Далее в листинге 4.10.3. приведен пример создания таблицы с несколькими секциями.

Листинг 4.10.3. Создание секционированной таблицы товарных чеков

```
CREATE TABLE sales_data
(ticket_no NUMBER,
sale_year INT NOT NULL,
sale_month INT NOT NULL,
sale_day INT NOT NULL)
PARTITION BY RANGE (sale_year, sale_month, sale_day)
-- таблица делится на секции в зависимости от даты товарного чека
(PARTITION sales_q1 VALUES LESS THAN (2008, 04, 01)
TABLESPACE ts1,
-- в первой секции документы до 01.04.2008
PARTITION sales_q2 VALUES LESS THAN (2008, 07, 01)
TABLESPACE ts2,
-- во второй секции документы с 01.04.2008 до 01.10.2008  и т.д.
PARTITION sales_q3 VALUES LESS THAN (2008, 10, 01)
TABLESPACE ts3,
PARTITION sales_q4 VALUES LESS THAN (2009, 01, 01)
TABLESPACE ts4);
```

Более подробно про секционирование можно прочитать

https://oracle-patches.com/блоги/70-базы-данных-oracle/3006-секционированные-таблицы

https://www.oracle.com/technetwork/ru/database/partitioning-wp-12c-1896137-ru.pdf

4.11. Нормализация базы данных

Во многих источниках большое значение придается нормализации, она объявляется чуть ли не краеугольным камнем проектирования БД, однако в настоящее время разработчики БД обычно и не вспоминают про нормальные формы и сам процесс нормализации. Почему? Давайте разберем это подробно:

1. Понятия нормальных форм относятся к отношениям, переменным отношений, кортежам и другим понятиям реляционной алгебры… переложить их на язык СУБД весьма проблематично, например, первая нормальная форма запрещает в отношении

одинаковые записи, но ни одна СУБД этого не поддерживает (если мы говорим о таблицах и записях).

2. Мнение многих авторов-теоретиков расходятся по поводу «атомарности» полей таблиц, но тогда понятие первой нормальной формы становится неопределенным.

   Понятие атомарности *вообще не имеет смысла*; оно зависит от того, что мы собираемся делать с данными. Иногда мы хотим обращаться с множеством номеров деталей, как с единым целым, а иногда рассматривать отдельные номера деталей из этого множества – но в этом случае мы опускаемся на более низкий уровень детализации, или абстракции [2].

3. Для проведения нормализации необходимо выделить зависимости атрибутов, а это не формализованная и сложная процедура, основанная на анализе семантических связей между полями таблиц. Т.е. глядя на схему БД нельзя «с ходу» понять в какой нормальной форме она находится, более того, нет формального алгоритма для определения формы, в которой находится БД.
4. При использовании таких простых понятий, как суррогатный первичный ключ, мы гарантированно получаем таблицы, которые находятся во второй нормальной форме.
5. Если в качестве единственного потенциального ключа рассматривать первичный суррогатный ключ, то мы получим таблицу в усиленной третьей нормальной форме

В настоящее время проектирование БД основывается на семантическом моделировании структуры данных, опираясь на смысл этих данных. В качестве инструмента семантического моделирования используются различные варианты диаграмм бизнес-процессов и диаграмм «сущность-связь» с использованием методологии IDEF.

При использовании правил и приемов, содержащемся в этом руководстве проектируемая база данных автоматически получается в третьей нормальной форме, а даже теоретики утверждают, что дальнейшая нормализация излишня. С другой стороны, справедливо утверждение:

Если база данных нуждается в нормализации – значит она неправильно спроектирована.

Если вы научитесь правильно проектировать базы данных, то нормализовать их не надо.

5. Администрирование БД

Функционирование БД и СУБД невозможно без участия специалистов, обеспечивающих создание, функционирование и развитие базы данных. Такая группа специалистов называется администраторами базы данных (АБД), в некоторой литературе применяется аббревиатура DBA (Database administrator).

Администраторы базы данных выполняют работы по созданию и обеспечению функционирования БД на протяжении всех этапов жизненного цикла системы см. рисунок 5.1. Численность группы администрации, и выполняемые ими функции, будут в значительной степени зависеть от масштаба БД, специфики хранимой в ней информации, типа и особенностей используемых программных средств и некоторых других факторов.

5.1. Этапы администрирования

Администрирование БД фактически разделяется на два этапа: «создание БД» и «сопровождение БД». Цели и задачи администрирования на этих этапах кардинально отличаются, так же как отличаются пользователи БД. На этапе создания БД – это программисты, разработчики БД и ИС, работающие с объектами БД, на этапе сопровождения — это пользователи ИС, работающие с данными в БД.

Создание БД – процесс создания схемы базы данных для конкретной СУБД, а также создание всех объектов необходимых для полноценного функционирования БД, начиная от таблиц, индексов, ограничений согласованности (целостности БД) и кончая триггерами и другими встроенными процедурами.

Сопровождение (поддержка) программного обеспечения — процесс улучшения, оптимизации и устранения дефектов информационной системы (ИС) после передачи в эксплуатацию. В ходе сопровождения в БД вносятся изменения, чтобы исправить обнаруженные в процессе использования дефекты и недоработки, а также для добавления новой функциональности, чтобы повысить удобство использования (юзабилити) и применимость ИС.

В свою очередь, опираясь на схему жизненного цикла ПО (см рисунок 5.1.), можно разбить эти два больших этапа на более мелкие:

Создание

- Проектирование
- Разработка и тестирование

Сопровождение

- Внедрение
- Эксплуатация

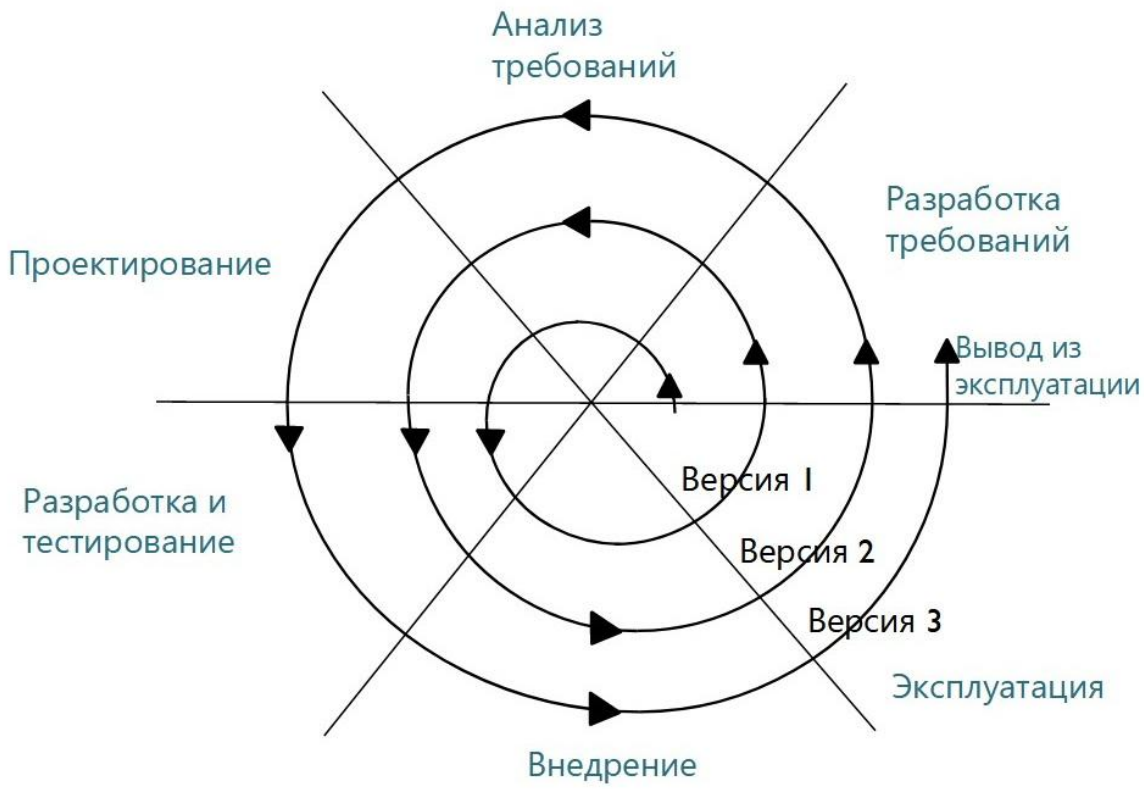

Рис. 5.1. Жизненный цикл ПО.

5.2. Задачи Администратора БД

Администраторы базы данных выполняют большой круг разнообразных функций, определенных в профессиональном стандарте 06.011 (см. Приложение Д):

I. На этапе проектирования

Разработка архитектуры системы

Выбор СУБД (хранилищ данных)

Разработка рекомендаций по построению аппаратной части проекта

Определение перечня ролей пользователей системы

Определение необходимых мероприятий для защиты данных

II. На этапе разработки и тестирования ИС

Инсталляция и первоначальная настройка СУБД.

Создание тестовых версий БД

Заполнение БД тестовыми данными. Подготовка исходных данных, ввод и контроль ввода тестовой информации

Поддержание среды разработки и отладки

Разработка регламентов копирования и восстановления данных

Разработка политики информационной безопасности на уровне БД

1. Защита данных разработки от несанкционированного доступа:
   - обеспечение парольного входа в систему: регистрация пользователей, назначение и изменение паролей;

- определение допустимых операций над объектами БД для отдельных пользователей, выбор/создание программно-технологических средств защиты данных; шифрование информации с целью защиты данных от несанкционированного использования;
- тестирование средств защиты данных;
- фиксация попыток несанкционированного доступа к информации разработчиков;
- исследование возникающих случаев нарушения защиты данных и проведение мероприятий по их предотвращению.

2. Обеспечение защиты БД от сбоев:
   - разработка программно-технологических средств копирования и восстановления БД
   - организация ведения системных журналов.

Подготовка и поддержание системных программных средств: сбор и анализ информации о СУБД и других прикладных программ, приобретение и приемка программных средств, их установка, проверка работоспособности, поддержание системных библиотек, развитие программных средств.

III. На этапе внедрения

Установка и программного обеспечения СУБД и прикладных средств.

Распределение внешней памяти системы

Создание первичных структур хранения базы данных (табличных пространств)

Развертывание внедряемой системы на СУБД

Регистрация пользователей и назначение им ролей

Разработка методов и функций защиты информации на этапе внедрения

Мониторинг и оптимизация производительности базы данных

Резервное копирование и восстановление базы данных

IV. На этапе эксплуатации (сопровождения) ИС

Управление внешней памятью системы и планирование дальнейших потребностей в ней

Регистрация пользователей и обеспечение защиты системы

Управление и мониторинг доступа пользователей

Мониторинг и оптимизация производительности базы данных

Анализ обращений пользователей к БД: сбор статистики обращений пользователей к БД, ее хранение и анализ (кто из пользователей, к какой информации, как часто обращался, какие выполнял операции, время выполнения запросов, анализ причин безуспешных (в т.ч. и аварийных) обращений к БД).

Резервное копирование и восстановление базы данных

Подготовка и поддержание системных программных средств: сбор и анализ информации о СУБД и других прикладных программ, приобретение программных средств, их установка, проверка работоспособности, поддержание системных библиотек, развитие программных средств.

Анализ эффективности функционирования базы данных и развитие системы: анализ показателей функционирования системы (время обработки, объем памяти, стоимостные показатели); реорганизация и реструктуризация баз данных, изменение состава баз данных; развитие программных и технических средств.

V. На этапе вывода из эксплуатации

Подготовка данных к переносу в новую ИС.

Разработка регламентов по миграции БД на новые платформы и новые версии ПО

Определение перечня переносимых данных

Определение глубины переносимых данных

Архивация

Обеспечение надежного хранения

Консервация

Обеспечение получения отчетности за периоды работы БД, выводимой из эксплуатации.

6. Сопровождение БД

7. Обеспечение безопасности данных

7.1. Защита данных от несанкционированного доступа:

• обеспечение парольного входа в систему - регистрация пользователей, назначение и изменение паролей;

• обеспечение защиты конкретных данных: определение прав доступа групп пользователей и отдельных пользователей; определение допустимых операций над данными для отдельных пользователей; выбор/создание программно-технологических средств защиты данных; шифрование информации с целью защиты данных от несанкционированного использования;

• тестирование средств защиты данных;

• фиксация попыток несанкционированного доступа к информации;

• исследование возникающих случаев нарушения защиты данных и проведение мероприятий по их предотвращению.

7.2. Копирование и восстановление БД:

разработка программно-технологических средств восстановления БД, организация ведения системных журналов.

7.3.

Заключение

Уже 50 лет идет развитие реляционных баз данных и 50 лет они по праву возглавляют рейтинги по количеству установок и по объему продаж. Несмотря на обилие в современном мире разнообразных моделей данных, СУБД и хранилищ, их поддерживающих [26], по-прежнему реляционные СУБД работают во всех отраслях человеческой деятельности, занимая основные, лидерские позиции. К сожалению, небольшой объем этого учебного пособия не позволяет осветить все аспекты разработки и применения реляционных БД, которые могут потребоваться специалисту в современном мире. Однако тех базовых знаний, которые можно почерпнуть из глав этой книги вполне достаточно для дальнейшего самообразования с помощью приведенной литературы и многочисленных информационных ресурсов доступных на просторах интернета.

Принимая во внимание высокий темп развития информационных технологий и в том числе развитие технологий хранения о обработки больших данных, можно считать данный учебник всего лишь стартовым материалом, на основе которого будут формироваться новые знания на протяжении всей профессиональной карьеры выпускника технического вуза, работающего в сфере информационных технологий.

Список литературы

1. Кляйн К., Кляйн Д., Хант Б. SQL. Справочник, 3_е издание. Пер. с англ. СПб: Символ_Плюс, 2010. 656 с., ил. ISBN 978-5-93286-165-3
2. Дейт, К. Дж. Введение в системы баз данных, 8-е издание. Пер. с англ. — М.: Издательский дом "Вильямс", 2005. 1328 с.: ил. ISBN 5-8459-0788-8
3. Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. – Пер. с англ. –СПб.: Символ-Плюс, 2010. 480 с., ил. ISBN 978-5-93286-173-8
4. Кайт Т., Кун Д. ORACLE для профессионалов: архитектура и методики программирования, 3-е изд.: Пер. с англ. М.: Издательский дом "Вильямс", 2016. - 960 с.: ил.- I SBN 978-5-8459-2042-3
5. Кириллов В.В., Громов Г.Ю. Введение в реляционные базы данных. — СПб.: БХВ-Петербург, 2009. — 464 с.: ил. ISBN 978-5-94157-770-5
6. Туманов В.Е. Основы проектирования реляционных баз данных. - М.: Национальный Открытый Университет "ИНТУИТ", 2016 (Основы информационных технологий) ISBN 978-5-94774-713-3
7. Электронный ресурс: «97 вещей, которые должен знать каждый программист» https://www.gitbook.com/book/97-things-every-x-should-know/97-things-every-programmer-should-know/details/ru
8. Электронный ресурс: «Закон Мерфи» - http://murphy-law.net.ru/catalog.html
9. Вигерс Карл, Битти Джой Разработка требований к программному обеспечению. 3-е изд., дополненное Пер. с англ. М.: Издательство «Русская редакция»; СПб.: БХВ-Петербург, 2014. 736 стр.: ил. ISBN 978-5-7502-0433-5
10. Райордан Р. Основы реляционных баз данных Пер. с англ. М.: Издательско-торговый дом ≪Русская Редакция≫, 2012. 384 с.: ил. ISBN 5-7502-0150-3
11. Коннолли Томас, Бегг Каролина Базы данных. Проектирование, реализация и сопровождение. Теория и практика, Издательский дом "Вильямс", 2003, ISBN 5-8459-0527-3
12. Кузнецов С.Д. Базы данных: учебник для студ. учреждений высшего проф. образования М. Издательский центр «Академия», 2012. 496 с. (Университетский учебник. Сер. Прикладная математика и информатика). ISBN 978-5-7695-8430-5
13. Кузнецов С. Д. Базы данных. Модели и языки, М.: Бином-Пресс, 2008, ISBN: 978-5-9518-0132-6
14. Когаловский М. Р. Энциклопедия технологий баз данных. М.: Финансы и статистика, 2002. ISBN 5-279-02276-4
15. SQL. Библия пользователя, 2-е издание. : Пер. с англ. М. : Издательский дом "Вильямс", 2010. — 752 с. : ил. ISBN 978-5-8459-1546-7 (рус.)
16. Codd, Edgar F. The relational model for database management: Version 2. Addison-Wesley, 1990. — ISBN 9-780-201-14192-4.
17. Фейерштейн С., Прибыл Б., ORACLE PL/SQL. Для профессионалов. 6-е изд. СПб.: Питер, 2015. 1024 с. ил., ISBN 978-5-496-01152-5.
18. Маркин А. В. Программирование на SQL в 2 ч. учебник и практикум для вузов. 2-е изд., М.: Издательство Юрайт, 2020. 340 с. ISBN 978-5-534-12258-9.
19. Шамаев Иван Электронный ресурс: «Основы PL/SQL» -https://ivan-shamaev.ru/pl_sql_functions_procedures_variables_cursors_cycles/
20. Reinhard Pichler Электронный ресурс «Database TheoryVU. Codd’s Theorem» - https://www.dbai.tuwien.ac.at/staff/pichler/dbt/slides/dbt03.pdf/

21. Электронный ресурс «МЕТОДОЛОГИЯ ФУНКЦИОНАЛЬНОГО МОДЕЛИРОВАНИЯ IDEF0» - https://nsu.ru/smk/files/idef.pdf
22. Фразеологический словарь современного русского языка. — М.:«Аделант»,2014.— 512с. ISBN 978-5-93642-359-8
23. Архитектура центров обработки данных Под ред. профессора В. А. Докучаева, Докучаев В.А., Кальфа А.А., Маклачкова В.В. – М,: «Горячая линия – Телеком», 2020г. а. с., ISBN 978-5-9912-0849-9
24. Информационная безопасность. Защита и нападение, Андрей Бирюков М,: «ДМК-Пресс», 2017 г., 434с., ISBN: 978-5-97060-435-9
25. Волк В.К. Базы данных. Проектирование, программирование, управление и администрирование: учебник для вузов / В. К. Волк. — 2-е изд., стер. — Санкт-Петербург: Лань, 2021. — ISBN 978-5-8114-8412-6.
26. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. Под редакцией Жаклин Картер / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2013. – 384с.: ил. ISBN 978-5-94074-866-3
27. Стоунз Р., Мэтью Н. PostgreSQL. Основы. – Пер. с англ. – СПб: Символ_Плюс, 2002. – 640 с., ил. ISBN 5_93286_043_X
28. Болье, А. Изучаем SQL. Генерация, выборка и обработка данных, 3-е изд./ пер. с англ. И.В. Красикова. — Киев.: “Диалектика”,2021. — 402 с.: ил. ISBN 978-1-492-05761-1
29. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. - СПб.: БХВ-Петербург, 2018. - 65-68 с, 68-72 с.
30. Рогов, Е. В. PostgreSQL 15 изнутри: руководство — Москва: ДМК Пресс, 2023. — 662 с. — ISBN 978-5-93700-178-8.
31. Комаров В. И. Путеводитель по базам данных. — М.: ДМК-Пресс, 2024. — 520 с. ISBN 978-5-93700-287-7

Приложение А. Сокращения, принятые в этой книге

БД — база данных

БП — бизнес - процесс

ДЛМ — даталогическая модель базы данных

ИЛМ — инфологическая модель предметной области

ИМ — иерархическая модель данных

НФ — нормальная форма отношения

ПО — программное обеспечение

РБД — реляционная база данных

РМ — реляционная модель данных

СМ — сетевая модель данных

СУБД — система управления базами данных

ФЗ — функциональная зависимость

ФМ — физическая модель базы данных

ЯОД — язык описания данных

ANSI — American National Standards Institute (Американский институт национальных стандартов)

API — Application Programming Interface (интерфейс прикладного программирования)

CASE — Computer Aided Software Engineering

DDL — Data Definition Language (язык определения данных)

DML — Data Manipulation Language (язык манипулирования данными)

ER — Entity-Relationship (сущность-связь)

GUID — Globally Unique Identifier (уникальный идентификационный номер)

HTML — Hypertext Markup Language (язык разметки гипертекста)

HTTP — Hypertext Transfer Protocol (сетевой протокол передачи гипертекста)

IDEF — методология ICAM DEFinition

ISO — International Organization for Standardization (Международная организация по стандартам)

MSDE — машина баз данных Microsoft Data Engine

OLE — Object Linking and Embedding (технология связывания и внедрения объектов и протокол разработанные компанией «Майкрософт»)

OLTP — Online Transaction Processing (оперативная обработка транзакций)

OLAP — Online Analitic Processing (оперативная обработка аналитических запросов)

PHP — Hypertext Preprocessor (Препроцессор Гипертекста — скриптовый язык программирования)

QBE — Query-By-Example (язык запросов по образцу)

SPARC — Standards Planning and Requirements Committee (подкомитет Американского института национальных стандартов)

SQL — Structured Query Language (структурированный язык запросов)

UML — Unified Modeling Language (унифицированный язык моделирования)

WWW — World Wide Web (Всемирная паутина)

XML — eXtensible Markup Language (расширяемый язык разметки)

Приложение Б. Основные термины и понятия

Предметная область базы данных - это та часть реального мира, о которой база данных хранит, собирает и обрабатывает информацию.

База данных (БД) - организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

Бизнес-процесс - совокупность взаимосвязанных действий пользователей и ИС, направленных на создание определённого продукта или услуги для потребителей.

Система управления базами данных (СУБД) - совокупность программных, лингвистических и методических средств общего или специального назначения, обеспечивающих управление созданием и использованием БД.

OLTP - *(Online Transaction Processing)*, Способ организации БД, при котором система работает с небольшими по размерам транзакциями, но идущими большим потоком, и при этом клиенту требуется от системы минимальное время отклика.

OLAP (*Online Analytical Processing)* - Способ организации БД, заключающийся в подготовке суммарной (агрегированной) информации на основе больших массивов данных, структурированных по многомерному принципу. Эта БД предназначена для быстрой обработки аналитических запросов.

Приложение - комплекс программ, автоматизирующий обработку данных для прикладной задачи.

Сущность (*объект)* - предмет, человек или событие, о котором собирается и хранится информация.

Запись – данные об одном конкретном экземпляре сущности - предмете, человеке или событии. Запись состоит из полей.

Пример. Сущность “СОТРУДНИК”, экземпляр сущности - данные об Иванове И. составляют одну запись таблицы «Сотрудники»

Поле записи (*атрибут, столбец таблицы****)*** - отдельная характеристика (свойство) сущности.

Домен - область значений одного или нескольких атрибутов.

Таблица - совокупность записей с фиксированным числом полей. Таблица содержит данные о всех экземплярах сущности. Таблица идентифицируется именем.

Далее приведено соответствие терминов в различных областях знаний.

База данных	Реляционная алгебра	ГОСТ (EXCEL)
Таблица	Отношение	Таблица
Запись	Кортеж	Строка
Поле	Атрибут	Столбец

Ключ - поле или суперпозиция несколько полей и\или их фрагментов (называемых *ключевыми*), определяющая запись или группу записей таблицы.

Первичный ключ – поле, однозначно определяющее запись (экземпляр сущности), используется для связи с другими таблицами. Значения этого поля должны быть уникальны в

пределах таблицы. Такой ключ может быть только один на таблицу. В реальных БД в качестве первичных используются суррогатные ключи.

Вторичный ключ (*кандидат****)*** – поле, содержащее уникальные значения. Это поле может стать первичным ключом, но не является оным, поскольку первичный ключ может быть только один в таблице.

Внешний ключ - поле дочерней таблицы, которое содержит значение первичного ключа родительской таблицы (поле “Код подразделения” таблицы “СОТРУДНИК” является внешним ключом, т.к. оно содержит значение первичного ключа в родительской таблице “ПОДРАЗДЕЛЕНИЕ”).

Родительская таблица *(сущность)* – таблица, содержащая первичный ключ используемой в *связи* (см.), значение которого записывается в поле внешнего ключа дочерней таблицы.

Дочерняя таблица *(сущность)* - таблица, в которой присутствует внешний ключ используемой в *связи* (см.) и содержащий значение первичного ключа родительской таблицы.

Связь — это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой сущностью или сама с собою. Связи позволяют по одной сущности (экземпляру сущности) находить другие сущности (или их экземпляры), связанные с нею.

Связь между родительской и дочерней таблицами (сущностями) это сопоставление записей, которое производится по равенству значений внешнего и первичного ключей в обеих таблицах. Различают связи следующих типов:

Связь типа “Один-к-одному”, или бинарная связь (1:1). Одной записи (экземпляру) родительской таблицы (сущности) соответствует одна запись (экземпляр) дочерней таблицы (сущности).

Связь типа “Один-ко-многим” (1:M). Одной записи родительской таблицы соответствует несколько записей дочерней таблицы. При этом внешний ключ участвующий в связи не уникален.

Связь типа “Многие-ко-многим” (M:M). Одному экземпляру родительской сущности соответствуют несколько экземпляров дочерней сущности и наоборот одному экземпляру дочерней сущности соответствуют несколько экземпляров родительской сущности. Наличие такой связи между сущностями говорит о том, что, какая-то значимая сущность еще не выявлена. Именно эта сущность и преобразует связь типа “Многие-ко-многим” к двум связям типа “Один-ко-многим”.

Индекс **–** объект БД, в простейшем случае, *таблица*, содержащая записи, каждая из которых содержит два поля: индекса (ключа индексируемой таблицы) и адреса записи индексируемой таблицы со значением данного индекса.

Словарь - централизованное хранилище сведений об объектах БД: таблицах, связях, форматах и диапазонах значений полей, представлениях, хранимых процедурах и т.д. Словарь — это БД, предметной областью которой является служебная информация СУБД.

Запрос - выражение, определяющее выборку нужных данных из одной или нескольких таблиц базы, удаление или изменение этих данных, создание новых таблиц. Все реляционные СУБД понимают структурированный язык запросов SQL.

Согласованность (целостность) базы данных - соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным ограничениям предметной области. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением согласованности. Такие ограничения должны быть формально объявлены в СУБД, после чего СУБД должна контролировать их выполнение. Некоторые ограничения не могут отслеживаться СУБД, они отслеживаются в информационной системе «окружающей» СУБД.

Примеры правил: каждый студент должен состоять в учебной группе; количество знаков в телефонном номере не должно превышать 15; баланс банка должен сходится; число проданных билетов не превышает число посадочных мест и т. д.

Транзакция - процесс, который переводит базу данных из одного согласованного состояния, в другое согласованное состояние. Допускается, что в процессе работы транзакции согласованность может нарушаться, но извне транзакции этого не видно.

По сути, транзакция — это группа последовательных операций с базой данных, обычно записанная на языке SQL, которая представляет собой логическую неделимую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая согласованность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта. Транзакции обрабатываются СУБД и другими транзакционными системами, в процессе работы которых создаётся история транзакций (журналы транзакций). Транзакции необходимы для поддержания согласованности базы данных.

Блокировка записей **–** процесс предотвращения доступа к данным, изменяемым транзакцией, пока она не будет закончена. Механизмы блокировки обеспечивают безопасность данных в случае параллельной обработки данных.

Триггер **-** хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением `INSERT`, удалением `DELETE` записи в заданной таблице, или изменением `UPDATE` данных в определенном поле заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения согласованности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения согласованности данных в результате работы триггера может произойти откат этой транзакции.

Нормализация – это разбиение таблицы на две или более, обладающие лучшими свойствами при включении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором каждый атрибут появляется лишь в одном месте, т.е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных и предсказуемости поведения системы во время эксплуатации. Нормализация применялась в прошлом веке при проектировании БД, в настоящее время не используется.

Реляционная база данных представляет собой набор взаимосвязанных двухмерных таблиц. Эта модель предложена сотрудником фирмы “IBM” Эдгаром Коддом в 1970 году.

Каждая таблица базы данных соответствует одной сущности предметной области и состоит из фиксированного числа полей (атрибутов сущности), собранных в записи, каждая запись соответствует экземпляру сущности.

Большинство современных СУБД являются реляционными.

Все реляционные СУБД поддерживают язык SQL.

SQL (*structured query language* - «язык структурированных запросов») - формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД).

Постреляционная модель - это реляционная модель, допускающая многозначные поля (атрибуты), т.е. само поле может быть таблицей.

Многомерная модель - узкоспециализированная модель, предназначенная для хранения данных в виде многомерного массива (гиперкуба), используемых *OLAP* (см.).

Декомпозиция — разделение целого на части. Также декомпозиция — это научный метод, использующий структуру задачи и позволяющий заменить решение одной большой задачи решением серии меньших задач, пусть и взаимосвязанных, но более простых.

Декомпозиция, как процесс расчленения, позволяет рассматривать любую исследуемую систему как сложную, состоящую из отдельных взаимосвязанных подсистем, которые, в свою

очередь, также могут быть расчленены на части. В качестве систем могут выступать не только материальные объекты, но и процессы, явления и понятия.

Агрегируемость данных наличие различных уровней обобщения информации, подразумевает привязку данных к какому-либо параметру (например, ко времени, географическому положению, структурным единицам и т.д.).

Срез - подмножество *гиперкуба* (см.), полученное в результате фиксации одного или нескольких измерений.

Вращение - вращение *гиперкуба* (см.) при этом местоположение отдельных осей меняются местами.

Агрегация/детализация - переход к более общему/детальному представлению информации по параметрам *агрегируемости* (см.).

Распределенная база данных — это совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети. Распределенная база данных может объединять базы данных, поддерживающие любые модели (иерархические, сетевые, реляционные и объектно-ориентированные базы данных) в рамках единой глобальной схемы. Подобная конфигурация должна обеспечивать для всех приложений прозрачный доступ к любым данным независимо от их местоположения и формата.

Репликация (replication) — это процесс, под которым понимается копирование данных из одной БД в другую и поддержание копий в актуальном и согласованном состоянии. При репликации изменения, сделанные в одной копии БД, распространяются в другие копии (реплики).

Тиражирование данных предполагает создание копий (реплик) фрагментов базы в узлах сети. Синхронизация копий и базы осуществляется специальной программой-репликатором, при этом передаются только изменения или транзакции.

Монопольный доступ используется обычно при массовых операциях с базой (реорганизация или восстановление базы и др.) и блокирует доступ ко всей БД другим пользователям.

Коллективный (многопользовательский, многозадачный) доступ дает возможность одновременной работы с БД многим пользователям. Для корректной работы при одновременном изменении отдельных записей возможны блокировки записей.

Аудит – это наблюдение за выбранными действиями пользователей БД. Его обычное применение - контроль подозрительных операций и сбор информации об отдельных действиях пользователей в базе данных.

Администратор базы данных - лицо или группа лиц, ответственная за проектирование и эффективное использование БД. Полный перечень задач см. в Приложении Д.

Администратор СУБД - лицо или группа лиц, ответственная за развертывание, бесперебойную эксплуатацию и эффективное использование СУБД.

Концептуальная модель БД (*концептуальная схема*) — это совокупность сущностей и их взаимосвязей вне зависимости от конкретной СУБД.

Логическая модель БД (*логическая схема*) – это конкретизация концептуальной модели в терминах конкретной СУБД.

Локальная архитектура СУБД - приложение и база находятся на одном компьютере.

Файл-серверная архитектура СУБД – приложение находятся на одном компьютере, а база данных - на другом. Приложение обращается за информацией к файлу-серверу, который возвращает приложению файлы, содержащие необходимые данные. В настоящее время не применяется.

Клиент-серверная архитектура СУБД - характеризуется тем, что запросы от приложения выполняются на сервере БД, а приложению (клиенту) передаются выбранные данные.

Трехзвенная архитектура ИС – архитектура, в которой явно выделены три слоя:

- слой клиента;
- серверы приложений;
- серверы баз данных.

Клиент формирует исходную информацию для расчета, посылает запрос на сервер приложения, где этот запрос и выполняется. При необходимости сервер приложения формирует запросы к серверу/серверам БД, для получения необходимой информации.

Тонкий клиент - клиентская часть, реализуемая на браузере.

Толстый клиент- клиентская часть, реализуемая в виде приложения.

Интерфейс ODBC (Open Database Connectivity) является посредником между приложением и СУБД; обеспечивает доступ из приложения к базам с различными СУБД. В состав ODBC входят драйверы (для каждой СУБД один драйвер, который преобразует форматы данных и команды приложения в форматы и команды СУБД и обратно) и диспетчер драйверов, который подключает нужный драйвер. Разработан и поддерживается компанией Microsoft.

Конфиденциальная информация (sensitive information) — информация, которая требует защиты.

Доступ к информации (access to information) — ознакомление с информацией, ее обработка (в частности, копирование), модификация, уничтожение.

Субъект доступа (access subject) — лицо или процесс, действия которого регламентируются правилами разграничения доступа.

Объект доступа (access object) — единица информации автоматизированной системы, доступ к которой регламентируется правилами разграничения доступа. Объектами доступа (контроля) в СУБД является практически все, что содержит конечную информацию: таблицы (базовые или виртуальные), представления, а также более мелкие элементы данных: поля и записи таблиц. Таблицы базы данных и представления всегда имеют владельца или создателя.

Правила разграничения доступа (security policy) — совокупность правил, регламентирующих права субъектов доступа по отношению к объектам доступа.

Санкционированный доступ (authorized access to information) — доступ к информации, который не нарушает правил разграничения доступа.

Несанкционированный доступ (unauthorized access to information) — доступ к информации, который нарушает правила разграничения доступа с использованием штатных средств, предоставляемых средствами вычислительной техники или автоматизированными системами.

Идентификатор доступа (access identifier) — уникальный признак объекта или субъекта доступа.

Идентификация (identification) — присвоение объектам и субъектам доступа идентификатора и (или) сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов.

Пароль (password) — это набор знаков (букв, чисел и других символов), предназначенный для подтверждения личности или полномочий.

В компьютерных системах служит для получения доступа к данным и сервисам. Пароль является средством защиты от несанкционированного доступа.

Аутентификация (authentification) — процесс проверки подлинности пользователя, удостоверяющий, что он является тем, за кого себя выдает. Для аутентификации пользователи могут вводить логин и пароль, биометрические данные или применять другие методы идентификации.

Авторизация — это процесс проверки прав доступа пользователя к определенным ресурсам или функциям. Он проводится после успешной аутентификации.

Дискреционное управление доступам (discretionary access control) — разграничение доступа между поименованными субъектами и поименованными объектами. Субъект с определенным правом доступа может передать это право любому другому субъекту.

Приложение В. База данных примера

Данная схема (рис. В.1) показывает взаимосвязь следующих сущностей:

- Сотрудники
- Отделы
- Работы, которые выполняют сотрудники

Каждая сущность реализована в данной схеме отдельной таблицей.

Между «Сотрудниками» и «Работами» существует связь типа «Много во Много». Эта связь реализована при помощи сущности «Списки сотрудников».

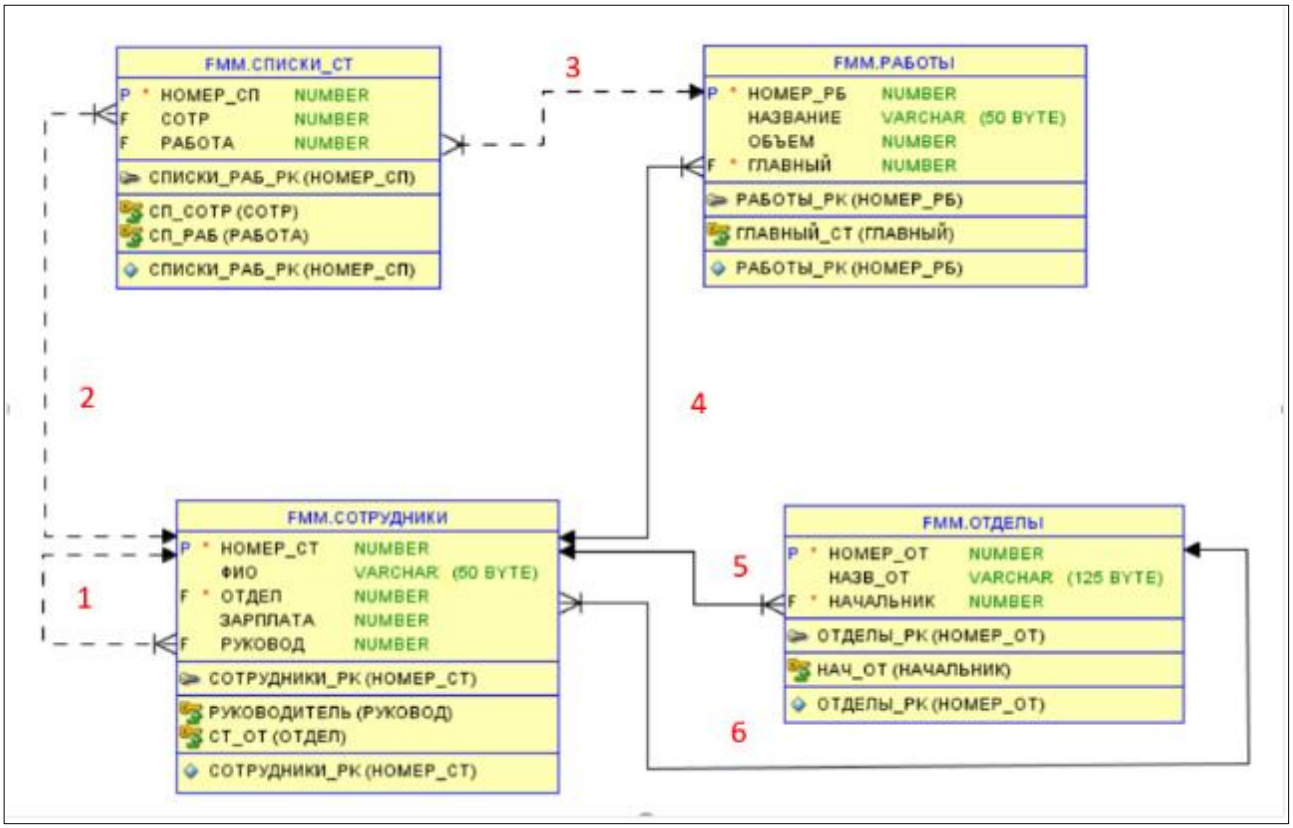

Рис. В.1 Полная схема таблиц примера для изучения SQL.

Между таблицами существуют связи (по часовой стрелке):

1. Каждый сотрудник может иметь руководителя из числа сотрудников.
2. Каждый сотрудник может состоять в нескольких списках по работам.
3. Каждая работа может быть связана с несколькими списками сотрудников.
4. Каждая работа имеет «Главного» из числа сотрудников.
5. Каждый отдел имеет начальника из числа сотрудников.
6. Каждый сотрудник состоит в отделе.

Ниже приведен листинг (Листинг В.1) для создания и заполнения тестовыми данными учебной БД в среде PostgreSQL. Запустив этот скрипт в своей СУБД, вы можете повторить и исследовать все примеры, приведенные в этой книге.

Листинг В.1. Создание учебной БД для СУБД PostgreSQL

```
-------------------------------------------------------
-- DDL for Table ОТДЕЛЫ
-------------------------------------------------------

 CREATE TABLE "ОТДЕЛЫ"
 (    "НОМЕР_ОТ" INTEGER PRIMARY KEY,
      "НАЗВ_ОТ" VARCHAR(125),
      "НАЧАЛЬНИК" INTEGER
 );
-------------------------------------------------------
-- DDL for Table РАБОТЫ
-------------------------------------------------------

 CREATE TABLE "РАБОТЫ"
 (    "НОМЕР_РБ" INTEGER PRIMARY KEY,
      "НАЗВАНИЕ" VARCHAR(50),
      "ОБЪЕМ" INTEGER,
      "ГЛАВНЫЙ" INTEGER
 );
-------------------------------------------------------
-- DDL for Table СОТРУДНИКИ
-------------------------------------------------------

 CREATE TABLE "СОТРУДНИКИ"
 (    "НОМЕР_СТ" INTEGER PRIMARY KEY,
      "ФИО" VARCHAR(50),
      "ОТДЕЛ" INTEGER,
      "ЗАРПЛАТА" INTEGER,
      "РУКОВОД" INTEGER
 );
-------------------------------------------------------
-- DDL for Table СПИСКИ_СТ
-------------------------------------------------------

 CREATE TABLE "СПИСКИ_СТ"
 (    "НОМЕР_СП" INTEGER PRIMARY KEY,
      "СОТР" INTEGER,
      "РАБОТА" INTEGER
 );
-- INSERTING into "ОТДЕЛЫ"

Insert into "ОТДЕЛЫ" ("НОМЕР_ОТ","НАЗВ_ОТ","НАЧАЛЬНИК") values ('1','Внешних коммуникаций','1');
Insert into "ОТДЕЛЫ" ("НОМЕР_ОТ","НАЗВ_ОТ","НАЧАЛЬНИК") values ('2','Внутренних инсинуаций','2');
Insert into "ОТДЕЛЫ" ("НОМЕР_ОТ","НАЗВ_ОТ","НАЧАЛЬНИК") values ('3','Протокольных мероприятий','3');
Insert into "ОТДЕЛЫ" ("НОМЕР_ОТ","НАЗВ_ОТ","НАЧАЛЬНИК") values ('4','Внутреннего контроля', null);

-- INSERTING into "РАБОТЫ"
```

```
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('1','Награждение непричасных','133','5');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('2','Наказание невиновных','124','5');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('3','Сбор объяснительных','12','1');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('4','Установка рогаток','15','4');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('5','Создание кадавра','333','9');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('6','Встреча инопланетян','500','4');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('7','Проводы инопланетян','600','2');
Insert into "РАБОТЫ" ("НОМЕР_РБ","НАЗВАНИЕ","ОБЪЕМ","ГЛАВНЫЙ") values ('8','Всеобщая вакцинация','700','4');

-- INSERTING into "СОТРУДНИКИ"

Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('10','Ничушкина Т.Н.','2','60000','11');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('11','Иванова Г.С.','2','100000', null);
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('12','Смирнова Е.В.','1','60000','11');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('1','Штейн А.Я.','1','112000', null);
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('2','Штейн И.Я.','2','113000', null);
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('3','Фильkенштейн А.Я.','3','114000', null);
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('4','Фомин М.М.','1','80000','1');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('5','Бауман Ю.И.','2','80000','2');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('6','Сусанин И.И.','1','60000','4');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('7','Грязнов Г.Г.','2','60000','5');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('8','Невструев Я.П.','3','75000','3');
Insert into "СОТРУДНИКИ" ("НОМЕР_СТ","ФИО","ОТДЕЛ","ЗАРПЛАТА","РУКОВОД") values ('9','Камнеедов А.П.','3','80000','8');

-- INSERTING into "СПИСКИ_СТ"

Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('1','4','6');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('2','6','6');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('3','5','6');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('4','7','6');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('5','4','4');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('6','5','1');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('7','5','2');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('8','5','3');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('9','9','5');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('10','8','8');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('11','3','8');
Insert into "СПИСКИ_СТ" ("НОМЕР_СП","СОТР","РАБОТА") values ('12','9','8');
```

```
--------------------------------------------------------
-- DDL for Index ОТДЕЛЫ_PK
--------------------------------------------------------

 CREATE UNIQUE INDEX "ОТДЕЛЫ_PK" ON "ОТДЕЛЫ" ("НОМЕР_ОТ");
--------------------------------------------------------
-- DDL for Index РАБОТЫ_PK
--------------------------------------------------------

 CREATE UNIQUE INDEX "РАБОТЫ_PK" ON "РАБОТЫ" ("НОМЕР_РБ");
--------------------------------------------------------
-- DDL for Index СОТРУДНИКИ_PK
--------------------------------------------------------

 CREATE UNIQUE INDEX "СОТРУДНИКИ_PK" ON "СОТРУДНИКИ" ("НОМЕР_СТ");
--------------------------------------------------------
-- DDL for Index СПИСКИ_РАБ_PK
--------------------------------------------------------

 CREATE UNIQUE INDEX "СПИСКИ_РАБ_PK" ON "СПИСКИ_СТ" ("НОМЕР_СП");

-- Ref Constraints for Table ОТДЕЛЫ
--------------------------------------------------------
 ALTER TABLE "ОТДЕЛЫ" ADD CONSTRAINT "НАЧ_ОТ" FOREIGN KEY ("НАЧАЛЬНИК")
       REFERENCES "СОТРУДНИКИ" ("НОМЕР_СТ");
--------------------------------------------------------
-- Ref Constraints for Table РАБОТЫ
--------------------------------------------------------
 ALTER TABLE "РАБОТЫ" ADD CONSTRAINT "ГЛАВНЫЙ_СТ" FOREIGN KEY ("ГЛАВНЫЙ")
       REFERENCES "СОТРУДНИКИ" ("НОМЕР_СТ");
--------------------------------------------------------
-- Ref Constraints for Table СОТРУДНИКИ
--------------------------------------------------------
 ALTER TABLE "СОТРУДНИКИ" ADD CONSTRAINT "РУКОВОДИТЕЛЬ" FOREIGN KEY ("РУКОВОД")
       REFERENCES "СОТРУДНИКИ" ("НОМЕР_СТ");
 ALTER TABLE "СОТРУДНИКИ" ADD CONSTRAINT "СТ_ОТ" FOREIGN KEY ("ОТДЕЛ")
       REFERENCES "ОТДЕЛЫ" ("НОМЕР_ОТ");
--------------------------------------------------------
-- Ref Constraints for Table СПИСКИ_СТ
--------------------------------------------------------

 ALTER TABLE "СПИСКИ_СТ" ADD CONSTRAINT "СП_РАБ" FOREIGN KEY ("РАБОТА")
       REFERENCES "РАБОТЫ" ("НОМЕР_РБ");
 ALTER TABLE "СПИСКИ_СТ" ADD CONSTRAINT "СП_СОТР" FOREIGN KEY ("СОТР")
       REFERENCES "СОТРУДНИКИ" ("НОМЕР_СТ");
```

Приложение Г. 12 правил Кодда

12 правил Кодда (англ. *Codd's 12 rules*) — 13 правил (в данном случае исчисление начинается с 0), которым должна удовлетворять каждая система управления реляционными базами данных. Предложены английским математиком Эдгаром Коддом (Edgar Codd) в 1985 году в статьях в журнале *ComputerWorld*. В действительности правила столь строги, что все современные так называемые «реляционные» СУБД полностью не соответствуют многим правилам.

Правило 0: Основное правило (*Foundation Rule*):

Система, которая рекламируется или позиционируется как реляционная система управления базами данных, должна быть способна управлять базами данных, используя исключительно свои реляционные возможности.

Правило 1: Информационное правило (*The Information Rule*):

Вся информация в реляционной базе данных на логическом уровне должна быть явно представлена единственным способом: значениями в таблицах.

Правило 2: Гарантированный доступ к данным (*Guaranteed Access Rule*):

В реляционной базе данных каждое отдельное (атомарное) значение данных должно быть логически доступно с помощью комбинации имени таблицы, значения первичного ключа и имени поля.

Правило 3: Систематическая поддержка отсутствующих значений (*Systematic Treatment of Null Values*):

Неизвестные, или отсутствующие значения NULL, отличные от любого известного значения, должны поддерживаться для всех типов данных при выполнении любых операций. Например, для числовых данных неизвестные значения не должны рассматриваться как нули, а для символьных данных — как пустые записи.

Правило 4: Доступ к словарю данных в терминах реляционной модели (*Active On-Line Catalog Based on the Relational Model*):

Словарь данных должен сохраняться в форме реляционных таблиц, и СУБД должна поддерживать доступ к нему при помощи стандартных языковых средств, тех же самых, которые используются для работы с реляционными таблицами, содержащими пользовательские данные.

Правило 5: Полнота подмножества языка (*Comprehensive Data Sublanguage Rule*):

Система управления реляционными базами данных должна поддерживать хотя бы один реляционный язык, который

(а) имеет линейный синтаксис,

(б) может использоваться как интерактивно, так и в прикладных программах,

(в) поддерживает операции определения данных, определения представлений, манипулирования данными (интерактивные и программные), ограничители согласованности, управления доступом и операции управления транзакциями (begin, commit и rollback).

Правило 6: Возможность изменения представлений (*View Updating Rule*):

Каждое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции выборки, вставки, изменения и удаления данных.

Правило 7: Наличие высокоуровневых операций управления данными (*High-Level Insert, Update, and Delete*):

Операции вставки, изменения и удаления данных должны поддерживаться не только по отношению к одной записи реляционной таблицы, но и по отношению к любому множеству записей.

Правило 8: Физическая независимость данных (*Physical Data Independence*):

Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютеров, на которых находится реляционная база данных.

Правило 9: Логическая независимость данных (*Logical Data Independence*):

Представление данных в приложении не должно зависеть от структуры реляционных таблиц. Если в процессе нормализации одна реляционная таблица разделяется на две, представление должно обеспечить объединение этих данных, чтобы изменение структуры реляционных таблиц не сказывалось на работе приложений.

Правило 10: Независимость контроля согласованности (*Integrity Independence*):

Вся информация, необходимая для поддержания согласованности, должна находиться в словаре данных. Язык для работы с данными должен выполнять проверку входных данных и автоматически поддерживать согласованность данных.

Правило 11: Независимость от расположения (*Distribution Independence*):

База данных может быть распределённой, может находиться на нескольких компьютерах, и это не должно оказывать влияния на приложения. Перенос базы данных на другой компьютер не должен оказывать влияния на приложения.

Правило 12: Согласование языковых уровней (*The Nonsubversion Rule*):

Если используется низкоуровневый язык доступа к данным, он не должен игнорировать правила безопасности и правила согласованности, которые поддерживаются языком более высокого уровня.

Приложение Д. Задачи и должностные обязанности Администратора Базы данных

Код специальности Администратора БД по «Общероссийскому классификатору профессий рабочих, должностей служащих и тарифных разрядов (ОКПДТР) — 40064.

По действующему профессиональному стандарту 06.011, для Администратора БД задачи и должностные обязанности определяются в зависимости от уровня квалификации из следующего списка:

- **Обеспечение функционирования БД**
  - Резервное копирование БД
  - Восстановление БД
  - Управление доступом к БД
  - Установка и настройка программного обеспечения (ПО) для обеспечения работы пользователей с БД
  - Установка и настройка ПО для администрирования БД
  - Мониторинг событий, возникающих в процессе работы БД
  - Протоколирование событий, возникающих в процессе работы БД

- **Оптимизация функционирования БД**
  - Мониторинг работы БД, сбор статистической информации о работе БД
  - Оптимизация распределения вычислительных ресурсов, взаимодействующих с БД
  - Оптимизация производительности БД
  - Оптимизация компонентов вычислительной сети, взаимодействующих с БД
  - Оптимизация выполнения запросов к БД
  - Оптимизация управления жизненным циклом данных, хранящихся в БД

- **Предотвращение потерь и повреждений данных**
  - Разработка регламентов резервного копирования БД
  - Контроль выполнения регламента резервного копирования
  - Разработка стратегии резервного копирования БД
  - Разработка регламентов восстановления БД
  - Разработка автоматических процедур для создания резервных копий БД
  - Проведение процедуры восстановления данных после сбоя
  - Контроль соблюдения регламента восстановления
  - Анализ сбоев в работе БД и выявление их причин
  - Разработка методических инструкций по сопровождению БД
  - Мониторинг работы программно-аппаратного обеспечения БД
  - Настройка работы программно-аппаратного обеспечения БД
  - Подготовка предложений по модернизации программно-аппаратных средств поддержки БД
  - Прогнозирование и оценка рисков сбоев в работе БД
  - Разработка автоматических процедур для горячего резервирования БД
  - Выполнение процедур по вводу в рабочий режим ресурсов горячей замены
  - Подготовка отчетов о функционировании БД
  - Консультирование пользователей в процессе эксплуатации БД
  - Подготовка предложений по повышению квалификации сотрудников

- **Обеспечение информационной безопасности на уровне БД**
  - Разработка политики информационной безопасности на уровне БД
  - Контроль соблюдения регламентов по обеспечению безопасности на уровне БД

- Оптимизация работы систем безопасности с целью уменьшения нагрузки на работу БД
- Разработка регламентов и аудит системы безопасности данных
- Подготовка отчетов о состоянии и эффективности системы безопасности на уровне БД
- Разработка автоматизированных процедур выявления попыток несанкционированного доступа к данным

- **Управление развитием БД**
  - Анализ системных проблем обработки информации на уровне БД, подготовка предложений по перспективному развитию БД
  - Разработка регламентов обновления версий программного обеспечения БД
  - Разработка регламентов по миграции БД на новые платформы и новые версии ПО
  - Изучение, освоение и внедрение в практику администрирования новых технологий работы с БД
  - Контроль обновления версий БД
  - Контроль миграции БД на новые платформы и новые версии ПО
  - Планирование организационной структуры подразделения и развития кадрового потенциала

Приложение Е. Перечень профессиональных стандартов

1. Standard Occupational Classification [Electronic resource] / U. S. Bureau of Labor Statistics. — Electronic data (1 file: 974848 bytes).

URL: https://www.bls. gov/ soc/2018/soc_structure_2018.pdf, free.

2. ГОСТ Р 56413-2015. Информационные технологии. Европейские профили профессий ИКТ-сектора / CWA 16458:2012 Information technologies. European ICT professional profiles. — Введ. 01.06.2016 приказом Федерального агентства по техническому регулированию и метрологии от 29 мая 2015 г. № 465-ст.

3. Профессиональный стандарт 06.011 «Администратор баз данных». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 27.04.2023 г.

4. Профессиональный стандарт 06.015 «Специалист по информационным системам». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 13 июля 2023 г. N 586н.

5. Профессиональный стандарт 06.026 «Системный администратор информационно-коммуникационных систем». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 29 сентября 2020 г. N 680н.

6. Профессиональный стандарт 06.033 «Специалист по защите информации в автоматизированных системах». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 14 сентября 2022г. № 525н.»

Приложение Ж. Стандартные формы Бэкуса — Наура (BNF)

Форма Бэкуса — Наура (сокр. **БНФ**, **Бэкуса — Наура форма**) — формальная система описания синтаксиса (метаязык), в которой одни синтаксические категории последовательно определяются через другие категории. БНФ используется для описания контекстно-свободных формальных грамматик. Используется для описания синтаксиса языков программирования, данных, протоколов (например, в документах RFC) и т. д.

В BNF-обозначениях используются следующие элементы:

– символ « **::=** » означает равенство по определению. Слева от знака стоит определяемое понятие, справа — собственно определение понятия;

– КЛЮЧЕВЫЕ СЛОВА (зарезервированные слова, составляющие часть оператора) записываются прописными буквами;

– заполнители конкретных значений элементов и переменных записываются строчными буквами;

– круглые скобки **( )** являются элементом оператора;

– фигурные скобки **{ }** указывают на то, что все, находящееся внутри них, является единым целым;

– в квадратные скобки **[ ]** заключаются необязательные элементы оператора;

– вертикальная черта | указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка, записанным после этой черты;

– многоточие **«…»** означает, что предшествующая часть оператора может быть повторена любое количество раз;

– многоточие после запятой «**, ...**», указывает на то, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений.

Например, синтаксис предложения FROM:

FROM таблица [AS псевдоним]
[тип_соединения] JOIN соединяемая_таблица [[AS] псевдоним]
{ ON условие_соединения1 [{AND | OR} условие_соединения2] [...] |
USING (поле1[, ...])}
[...]

Приложение 3. Семейство стандартов IDEF

В настоящий момент к семейству IDEF можно отнести следующие стандарты:

IDEFO - Function Modeling. Методология функционального моделирования. С помощью наглядного графического языка IDEFO изучаемая система предстает перед разработчиками и аналитиками в виде набора взаимосвязанных функций (функциональных блоков в терминах IDEFO). Как правило, моделирование средствами IDEFO является первым этапомизучения любой системы. Методологию IDEFO можно считать следующим этапом развития хорошо известного графического языка описания функциональных систем SADT (Structured Analysis and Design Teqnique);

IDEFl - lnformation Modeling. Методология моделирования информационных потоков внутри системы, позволяющая отображать и анализировать их структуру и взаимосвязи;

IDEFlX (IDEFl Extended) - Data Modeling. Методология построения реляционных структур (баз данных), относится к типу методологий «Сущность-связь» (ER-EntityRelationship) и, как правило, используется для моделирования реляционных баз данных, имеющих отношение к рассматриваемой системе;

IDEF2 – Simulation Model Design. Методология динамического моделирования развития систем. В связи с весьма серьезными сложностями анализа динамических систем от этого стандарта практически отказались, и его развитие приостановилось на самом начальном этапе. В настоящее время присутствуют алгоритмы и их компьютерные реализации, позволяющие превращать набор статических диаграмм IDEFO в динамические модели, построенные на базе «раскрашенных сетей Петри» (CPN – Color Petri Nets);

IDEF3 - Process Description Capture. Документирование технологических

процессов, IDEF3 - методология документирования процессов, происходящих в системе (например, на предприятии), описываются сценарий и последовательность операций для каждого процесса. IDEF3 имеет прямую взаимосвязь с методологией IDEFO - каждая функция (функциональный блок) может быть представлена в виде отдельного процесса средствами ID EF3;

IDEF4 - Object Oriented Design. Методология построения объектно – ориентированных систем, позволяет отображать структуру объектов и заложенные принципы их взаимодействия, тем самым позволяя анализировать и оптимизировать сложные объектно-ориентированные системы;

IDEF5 - Ontology Description Capture. Стандарт онтологического исследования сложных систем. С помощью методологии IDEF5 онтология системы может быть описана при помощи определенного словаря терминов и правил, на основании которых могут быть сформированы достоверные утверждения о состоянии рассматриваемой системы в некоторый момент времени. На основе этих утверждений формируются выводы о дальнейшем развитии системы и производится ее оптимизация;

IDEF6 - Design Rationale Capture. Обоснование проектных действий. Назначение IDEF6 состоит в облегчении получения «знаний о способе» моделирования, их представления и использования при разработке систем управления предприятиями. Под «знаниями о способе» понимаются причины, обстоятельства, скрытые мотивы, которые обусловливают выбранные методы моделирования. Проще говоря, «знания о способе» интерпретируются как ответ на вопрос: «Почему модель получилась такой, какой получилась?» Большинство методов моделирования фокусируются на собственно получаемых моделях, а не на процессе их создания. Метод IDEF6 акцентирует внимание именно на процессе создания модели;

IDEF7 - lnformation System Auditing. Аудит информационных систем. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF8 - User Interface Modeling. Метод разработки интерфейсов взаимодействия оператора и системы (пользовательских интерфейсов). Современные среды разработки пользовательских интерфейсов в большей степени создают внешний вид интерфейса. IDFE8 фокусирует внимание разработчиков интерфейса на программировании желаемого взаимного поведения интерфейса и пользователя на трех уровнях: выполняемой операции (что это за операция); сценарии взаимодействия, определяемом специфической ролью пользователя (по какому сценарию она должна выполняться тем или иным пользователем); и, наконец, на деталях интерфейса (какие элементы управления предлагает интерфейс для выполнения операции);

IDEF9 - ScenarioDriven IS Design (Business Constraint Discovery method). Метод исследования бизнес-ограничений был разработан для облегчения обнаружения и анализа ограничений, в условиях которых действует предприятие. Обычно при построении моделей описанию ограничений, оказывающих влияние на протекание процессов на предприятии, уделяется недостаточное внимание. Знания об основных ограничениях и характере их влияния, закладываемые в модели, в лучшем случае остаются неполными, несогласованными, распределенными нерационально, но часто их вовсе нет. Это не обязательно приводит к тому, что построенные модели нежизнеспособны, просто их реализация столкнется с непредвиденными трудностями, в результате чего их потенциал будет не реализован. Тем не менее в случаях, когда речь идет именно о совершенствовании структур или адаптации к предсказываемым изменениям, знания о существующих ограничениях имеют критическое значение;

IDEF10 - lmplementation Architecture Modeling. Моделирование архитектуры выполнения. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF11 - Information Artifact Modeling. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF12 - Organization Modeling. Организационное моделирование. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF13 - Three Schema Mapping Design. Трехсхемное проектирование преобразования данных. Этот метод определен как востребованный, однако так и не был полностью разработан;

IDEF14 - Network Design. Метод проектирования компьютерных сетей, основанный на анализе требований, специфических сетевых компонентов, существующих конфигураций сетей. Также он обеспечивает поддержку решений, связанных с рациональным управлением материальными ресурсами, что позволяет достичь существенной экономии.