|  |  |
| --- | --- |
|  |  |
|  |  |

|  |
| --- |
| **Реляционные базы данных.**  **Учебник для бакалавров** |
| **Версия: 1.0** |
|  |
|  |
| **Фомин М.М.** |

Оглавление

[Оглавление 2](#_Toc24312219)

[Предисловие 5](#_Toc24312220)

[1. Базы данных 7](#_Toc24312221)

[1.1. Появление БД 7](#_Toc24312222)

[1.1.1. Исторические предпосылки 7](#_Toc24312223)

[1.1.2. Концепции БД 7](#_Toc24312224)

[1.1.3. Первые базы данных 8](#_Toc24312225)

[1.1.4. Реляционная модель БД 11](#_Toc24312226)

[2. СУБД 13](#_Toc24312227)

[2.1. Архитектура СУБД 13](#_Toc24312228)

[2.1.1. Трехуровневая архитектура систем баз данных ANSI/SPARC 13](#_Toc24312229)

[2.1.2. Файл серверная архитектура 15](#_Toc24312230)

[2.1.3. Встроенные СУБД 16](#_Toc24312231)

[2.1.4. Клиент серверная архитектура 16](#_Toc24312232)

[2.1.5. Трехзвенная архитектура 18](#_Toc24312233)

[2.2. Функции СУБД 21](#_Toc24312234)

[2.2.1. Управление данными во внешней памяти 21](#_Toc24312235)

[2.2.2. Управление буферами оперативной памяти 23](#_Toc24312236)

[2.2.3. Поддержка языков БД 25](#_Toc24312237)

[2.2.4. Обеспечение целостности БД 25](#_Toc24312238)

[2.2.5. Управление транзакциями 26](#_Toc24312239)

[2.2.6. Управление блокировками и клинчами 30](#_Toc24312240)

[2.2.7. Ведение журнала изменений БД 34](#_Toc24312241)

[2.2.8. Ведение словаря БД 34](#_Toc24312242)

[2.2.9. Обеспечение безопасности БД 36](#_Toc24312243)

[2.3. Развитие реляционных СУБД 36](#_Toc24312244)

[3. Язык SQL 39](#_Toc24312245)

[3.1. Становление языка 39](#_Toc24312246)

[3.1.1. История возникновения 39](#_Toc24312247)

[3.1.2. Стандарт языка SQL 40](#_Toc24312248)

[3.2. Концепции языка SQL 41](#_Toc24312249)

[3.2.1. Декларативность и простота 41](#_Toc24312250)

[3.2.2. Непроцедурность и расширения языка 42](#_Toc24312251)

[3.2.3. Замкнутость и стиль мышления 42](#_Toc24312252)

[3.2.4. Зависимость от СУБД и независимость от платформы 42](#_Toc24312253)

[3.2.5. Равенство формализмов и отход от реляционной теории 43](#_Toc24312254)

[3.2.6. Срезы и консолидация 43](#_Toc24312255)

[3.2.7. Защита данных и привилегии пользователей 44](#_Toc24312256)

[3.3. Диалекты языка SQL 44](#_Toc24312257)

[3.4. Понимание SQL 44](#_Toc24312258)

[3.5. Типы данных SQL 44](#_Toc24312259)

[3.6. Операторы SQL 46](#_Toc24312260)

[3.6.1. Простые примеры 46](#_Toc24312261)

[3.6.2. Общая характеристика и схема выполнения 46](#_Toc24312262)

[3.6.3. Создание объектов СУБД 47](#_Toc24312263)

[3.6.4. Изменение и удаление объектов СУБД 47](#_Toc24312264)

[3.6.5. Инструкция SELECT 47](#_Toc24312265)

[3.6.6. Представления в языке SQL 47](#_Toc24312266)

[3.6.7. Защита данных в SQL 47](#_Toc24312267)

[3.7. Функции SQL 47](#_Toc24312268)

[4. Проектирование БД 47](#_Toc24312269)

[4.1. Исторический экскурс 47](#_Toc24312270)

[4.2. Основные цели проектирования БД 48](#_Toc24312271)

[4.3. Основные задачи и результат проектирования БД 49](#_Toc24312272)

[4.3.1. Основные задачи 49](#_Toc24312273)

[4.3.2. Результат проектирования БД 50](#_Toc24312274)

[4.3.3. Выбор СУБД 50](#_Toc24312275)

[4.4. Логическое проектирование БД 50](#_Toc24312276)

[4.4.1. Анализ бизнес процессов 51](#_Toc24312277)

[4.4.2. Определение сущностей 51](#_Toc24312278)

[4.4.3. Верификация сущностей 55](#_Toc24312279)

[4.4.4. От автоматизации процессов к автоматизации компании 56](#_Toc24312280)

[4.4.5. Определение атрибутов сущностей 56](#_Toc24312281)

[4.4.6. Первичный ключ – важный атрибут 56](#_Toc24312282)

[4.4.7. Вычисляемые атрибуты (поля) 57](#_Toc24312283)

[4.4.8. Верификация атрибутов 57](#_Toc24312284)

[4.4.9. Виды связей между сущностями 58](#_Toc24312285)

[4.4.10. Первичные и внешние ключи 59](#_Toc24312286)

[4.4.11. «Один к одному» - редкий тип связи 60](#_Toc24312287)

[4.4.12. «Много ко многим» - несуществующий тип связи 61](#_Toc24312288)

[4.4.13. Экзотические связи 62](#_Toc24312289)

[4.4.14. Реализация иерархических структур 63](#_Toc24312290)

[4.4.15. Верификация связей 63](#_Toc24312291)

[4.4.16. Диаграмма «Сущность-связь» 64](#_Toc24312292)

[4.5. Физическое проектирование Базы данных 65](#_Toc24312293)

[4.5.1. Формирование схемы таблиц 65](#_Toc24312294)

[4.5.2. Создание представлений 66](#_Toc24312295)

[4.5.3. Создание хранимых процедур 68](#_Toc24312296)

[4.5.4. Типы встроенных процедур 68](#_Toc24312297)

[4.5.5. Триггеры 68](#_Toc24312298)

[4.5.6. Создание индексов 70](#_Toc24312299)

[4.6. Разграничение доступа и создание ролей пользователей 71](#_Toc24312300)

[4.6.1. Что такое разграничение доступа 71](#_Toc24312301)

[4.6.2. Пользователи, их схемы и роли 72](#_Toc24312302)

[4.6.3. Предопределённые роли пользователей СУБД 72](#_Toc24312303)

[4.6.4. Создание ролей пользователей 72](#_Toc24312304)

[4.7. Обеспечение целостности БД 75](#_Toc24312305)

[4.7.1. Обеспечение целостности БД 75](#_Toc24312306)

[4.7.2. Использование транзакций – путь к обеспечению целостности БД. 75](#_Toc24312307)

[4.7.3. Использование триггеров – путь к обеспечению целостности БД. 76](#_Toc24312308)

[4.7.4. Удаление данных 76](#_Toc24312309)

[4.8. Поддержка историчности 76](#_Toc24312310)

[4.9. Обеспечение быстродействия БД 77](#_Toc24312311)

[4.9.1. Секционирование таблиц 77](#_Toc24312312)

[4.9.2. Кластеризация таблиц 78](#_Toc24312313)

[4.10. Нормализация базы данных 78](#_Toc24312314)

[Список литературы 80](#_Toc24312315)

[Заключение 81](#_Toc24312316)

[Приложение A. Сокращения принятые в этой книге 82](#_Toc24312317)

[Приложение Б. Основные термины и понятия 83](#_Toc24312318)

[Приложение В. База данных примера 89](#_Toc24312319)

[Приложение Г. 12 правил Кодда 90](#_Toc24312320)

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 теоретических основ. 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 надежной работы с базами данных. Рассматриваются современные архитектуры СУБД. Разумеется, в небольшом пособии невозможно охватить все аспекты теории и практики баз данных. Пособие отражает скорее опыт автора в работе с большими базами данных в крупных корпорациях. Приведенный список литературы поможет значительно расширить кругозор знаний о базах данных.

Термины и сокращения

|  |  |
| --- | --- |
| Термин | Определение |
| Программа | Последовательность инструкций, предназначенных для исполнения устройством управления вычислительной машины. Программа — один из компонентов программного обеспечения. |
| 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, сокр. **ОС** (англ. operating system, OS) — 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, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают сбор, обработку и распространение информации. |
| Сущность | Предмет, человек или событие, о котором собирается и хранится информация в БД. |
| Запись | Данные об одном конкретном экземпляре сущности - предмете, человеке или событии. Запись состоит из полей. |
| Поле записи (атрибут, столбец таблицы) | Отдельная характеристика (свойство) объекта. |
| Таблица | Объект базы данных хранящий данные об одной сущности: столбцы таблицы (поля) - это атрибуты сущности, а строки таблицы (записи) - это данные о конкретном экземпляре сущности. |
| 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, в первую очередь студентов высших учебных заведений, обучающихся по направлениям подготовки бакалавров "Информатика и вычислительная техника" и "Прикладная информатика". Может быть полезен 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 четырех глав и последовательно знакомит читателя с основными понятиями баз данных: сначала обсуждаются концепции БД, затем происходит знакомство с СУБД, поддерживающими БД, язык SQL базирующийся на понятиях реляционных БД и выполняемый в СУБД предваряет описание процесса проектирования 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.

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 (отступления оговариваются отдельно), а схемы сущностей и таблиц БД в виде, принятом в 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 литературе и документации на системы управления базами данных (СУБД), хотя в настоящее время все чаще применяется терминология ГОСТ, несмотря на то, что эта терминология появилась в результате небрежного 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».

# Базы данных

## 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.

### 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 (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, дело-производство. 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. Таким образом появилась необходимость в надежном хранении данных независимых от программ с одновременным доступом из нескольких программ. Эта необходимость и послужила толчком к разработке концепций баз данных, которым и будет посвящен следующий раздел. С историей развития БД тесно связана история развития СУБД, которая обсуждается в разделе 2.3.

### Концепции БД

В середине 60-х годов прошлого столетия начали вырабатываться основные принципы, по которым должны быть построены базы данных (Напомним, 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.) 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 для их освещения. 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 по студентам 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.

* 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 целостном (согласованном) состоянии.

Целостность БД (database integrity) — соответствие имеющейся в [базе данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации её внутренней логике, структуре и всем явно заданным правилам. В большинстве случаев ограничения целостности определяются особенностями предметной области. Одна из важнейших задач БД - поддержание в любой момент времени взаимной непротиворечивости, правильности и точности хранящихся данных.

* 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 многопользовательской (многозадачной) работы.

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

### Первые базы данных

Иерархическая модель

Концепция иерархических баз данных преследовала нас с 1960-х годов и (в это трудно поверить!) 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, организационную диаграмму или пирамиду. 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.

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

Заказы

Клиенты

Товары

Рис. 1.1. Пример иерархической базы данных

Так как записи дочерней таблицы доступны путем раскрытия иерархии уровней, некоторые из ее записей могут не иметь непосредственных связей с соответствующими элементами родительской таблицы (доступ к этим записям следует осуществлять, начиная с корневого элемента иерархии). Эту организацию 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».

Заказы

Посредники

Товары

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 реплицировать в несколько таблиц. 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 1.14.). 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 продуктов. 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 Adabas (впоследствии преобразованная в реляционнуюмодель).

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).

Заказы

Клиенты

Товары

Продавцы

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.

### Реляционная модель БД

Впервые термин "реляционная модель данных" появился в статье сотрудника фирмы IBM д-ра Кодда опубликованной 6 июня 1970г. 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.

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 (напр., Иванове И.И.), а каждое поле в этой записи содержит данные об Иванове И.И. - дата рождения, номер студенческого билета и т.д. Каждая таблица, также, как и поля идентифицируются своими именами («Студенты», «ФИО», «Дата рождения»), 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». И 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 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 в разделах 4.4.9. – 4.4.13.

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).

# 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 – «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.

Нужно ли программисту прикладных приложений понимать как работает БД? Том Кайт, признанный специалист Oracle, в своей книге «Oracle для профессионалов» (4) настаивает, что это просто необходимо.

## 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 архитектура ANSI/SPARC, впервые представленная в 1975 г.

### Трехуровневая архитектура систем баз данных ANSI/SPARC

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

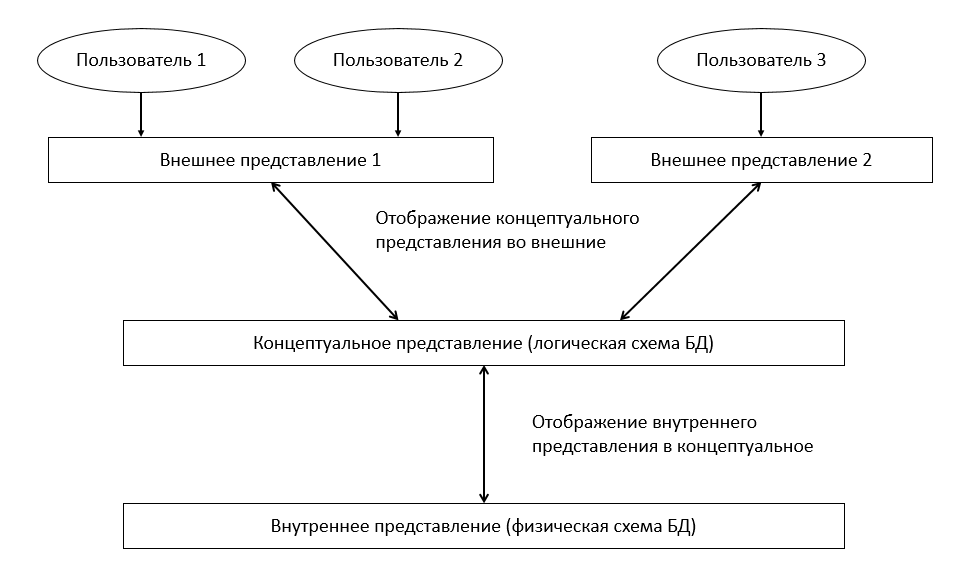
* внутренний (физический);
* промежуточный (концептуальный);
* внешний (пользовательский).

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 представления. 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.). 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 3.5.5. и 4.5.2.).

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 целостности 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 пространства в байтах.

Внутренний уровень.

**Внутренний уровень** – это физическое представление БД в памяти компьютера. Этот уровень описывает, как информация хранится в базе данных. Он содержит описание структур данных и организации отдельных файлов или 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. 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.

### Файл серверная архитектура

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. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок.

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. Применяются чаще всего в локальных приложениях, которые используют 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. Практически все эти системы не являются полноценными СУБД.

### Встроенные СУБД

Встраиваемая СУБД — СУБД, 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 через SQL либо через специальные программные интерфейсы.

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

### Клиент серверная архитектура

Суть этой архитектуры в том, что система разбивается на две части, которые могут выполняться в 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 клиенту. Клиентское приложение отображает полученные данные в удобной для пользователя форме.

*Клиент-серверная СУБД* 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 СУБД* гораздо менее требовательна к пропускной способности компьютерной сети, чем файл-серверная СУБД, 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 локальной 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.

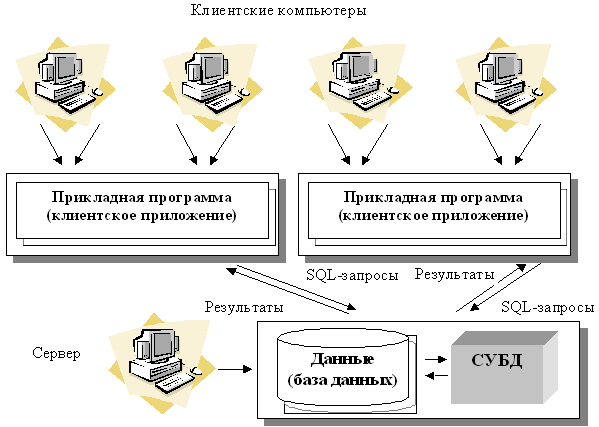


Рис 2.2 Клиент серверная архитектура

**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 слоя и три задачи, выполняемые системой клиент-сервер:

* 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 («а» и «б») используют «толстый» клиент.

**Толстый клиент** (англ.  *rich client; heavy client;* также, **Rich-клиент**) — 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.

**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. 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.

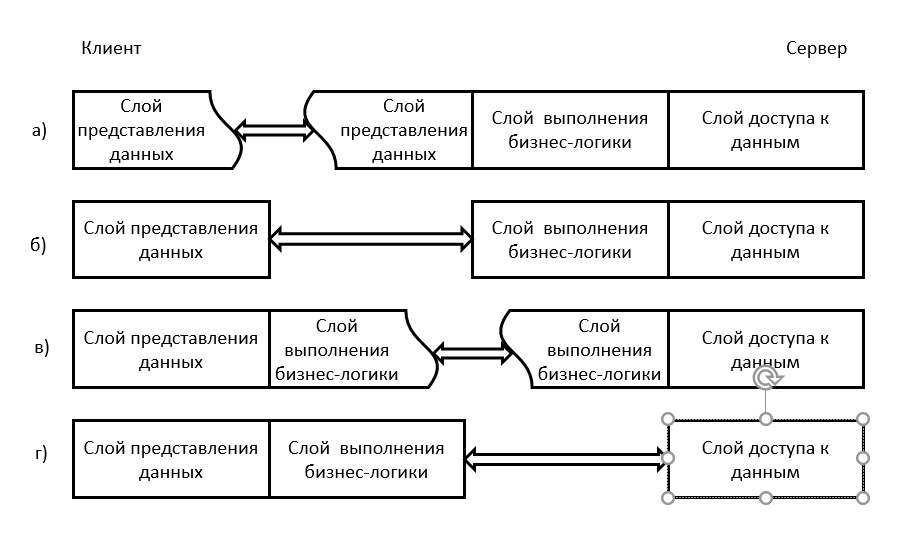


Рис 2.3 Варианты клиент серверной архитектуры

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.

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

### 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 двухзвенной архитектуре «клиент-сервер» (см раздел 2.1.4.). Каждый из уровней 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 целостности данных (например, протокола двухфазного подтверждения). Соответственно, на уровне баз данных высокой масштабируемости и отказоустойчивости достичь сложнее.

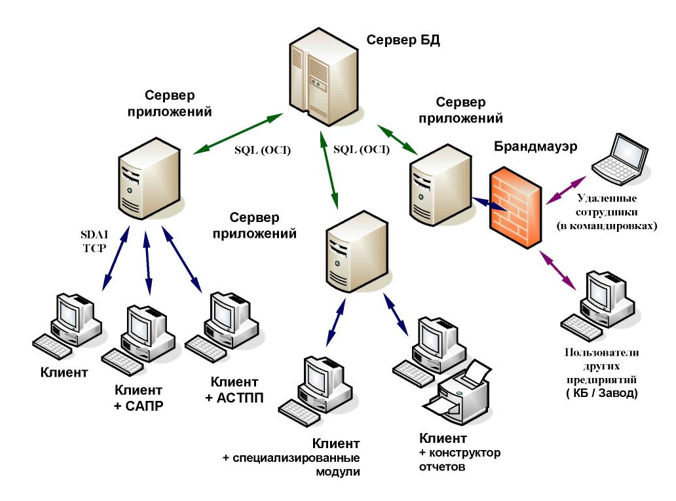


Рис 2.4 Трехзвенная архитектура

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 серверов. 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 дополнительного промежуточного слоя (обработки бизнес-логики) обеспечивает дополнительную защищенность данных. Каждый из слоев обеспечивает дополнительный уровень отказоустойчивости путём включения резервных серверов, которые принимают на себя нагрузку при отказах отдельных звеньев системы. Такое резервирование также повышает возможности модернизации системы путем вывода отдельных звеньев из эксплуатации для обновления и отладки. Необходимо отметить, что в подавляющем числе внедрений в трехзвенной архитектуре применяется тонкий клиент. Итак, подытожим преимущества и недостатки трехзвенной архитектуры.

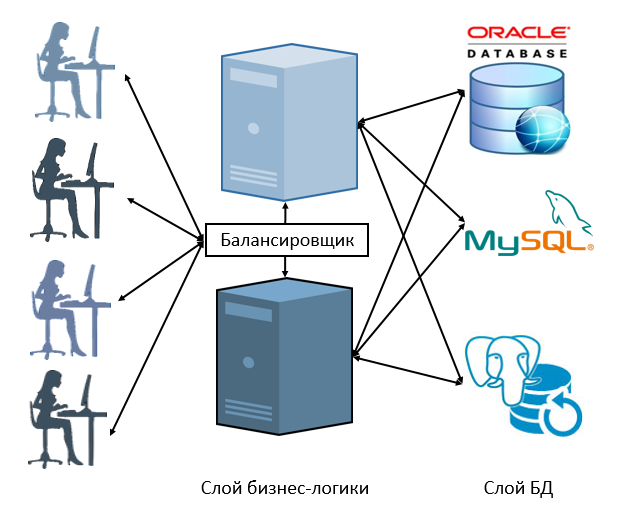


Рис. 2.5 Трехзвенная архитектура

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 разработки трехзвенной архитектуры быстро уменьшаются, и эта архитектура все больше и больше завоевывает пространство IT решений.

## Функции СУБД

Основные функции СУБД связаны с концепциями БД, которые СУБД должны поддерживать.

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 |
| Хранение описания данных вместе с самими данными | Ведение словаря БД |
| Поддержание БД в целостном (согласованном) состоянии. | Обеспечение целостности БД  Управление транзакциями  Управление блокировками и клинчами |
| Защита информации | Ведение журнала изменений в БД  Управление транзакциями  Обеспечение безопасности БД |
| Поддержка многопользовательской (многозадачной) работы | Управление транзакциями  Управление блокировками и клинчами |

### Управление данными во внешней памяти

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

Основные табличные пространства в 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.

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 места 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 одн6ому 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.

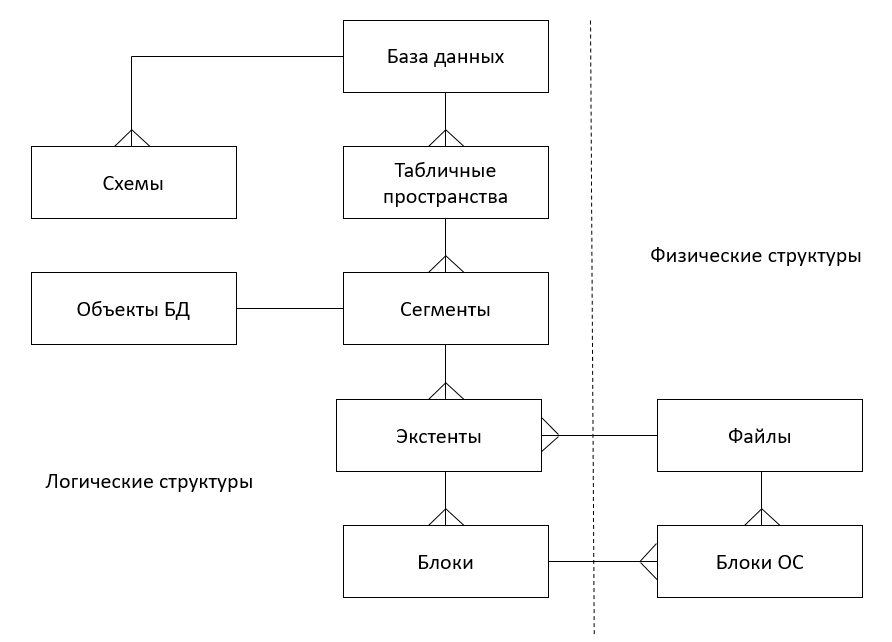


Рис. 2.6 Иерархия структура логических единиц хранения БД

Файлы, в которых размещаются экстенты могут быть файлами ОС, чистыми (RAW) разделами, или файлами в кластеризованной файловой системе.

Чистые (RAW) разделы — это не файлы, это диски без файловой системы. Их нельзя просматривать с помощью команды ls или проводника Windows. Они представляют собой всего лишь большие области диска без какой-либо файловой системы внутри. Для Oracle весь чистый раздел выглядит как одиночный крупный файл.

Кластеризованной файловая система предназначена специально для среды RAC и выглядит подобно готовой файловой системе, которая совместно используется многими узлам и (компьютерами) в кластеризованной среде.

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.

### 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 замены 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.

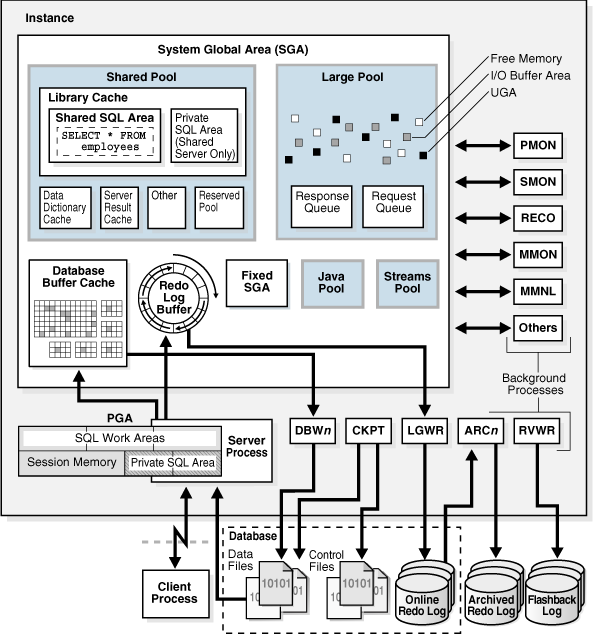


Рисунок 2.6 Общая схема буферов СУБД ORACLE.

* **Пул Java** (Java pool). Это фиксированный объем памяти, выделенной для виртуальной машины Java (Virtua\Java Machine - JVМ), 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 RМAN 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 модель согласованного чтения. Все пользователи, которые одновременно подключены к СУБД, совместно используют буферный кэш базы данных. Измененные блоки сбрасываются обратно на внешние носители.
* **Буфера журнала повторов** (redo log buffer)– этоциклический 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 в случае необходимости.
* **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 ваша СУБД в памяти по сравнению с обращениями к дискам, тем быстрее она реагирует на запросы. Конечно, по мере сокращения ввода-вывода загрузка процессора также сокращается, что повышает общую эффективность системы.

### Поддержка языков БД

В современных 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).

**процессор языка базы данных**, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода

### Обеспечение целостности БД

Целостность базы данных (database integrity) — соответствие имеющейся в [базе данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется [ограничением целостности](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1) (integrity constraint). В большинстве случаев ограничения целостности 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.

Разделяют несколько видов целостности БД:

1. Сущностная целостность

Поддержание сущностной целостности состоит в недопущении многократного попадания данных об одной сущности в таблицу БД.

1. Доменная целостность

Доменная целостность - это достоверность записей в конкретном столбце. Она включает в себя ограничения типа данных, ограничения формата, а также ограничения диапазона возможных значений.

1. Ссылочная целостность

Ссылочная целостность гарантирует согласованность значений первичных и внешних ключей во всех таблицах.

1. Пользовательская целостность

Определяется бизнес-правилами, не входящие ни в одну из приведенных категорий целостности, эти правила определяются и отслеживаются не на уровне СУБД, а на уровне ИС. Например, баланс банка должен сходится или число проданных и непроданных билетов на авиарейс должно в сумме равняться вместимости самолета.

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 целостности базы данных.

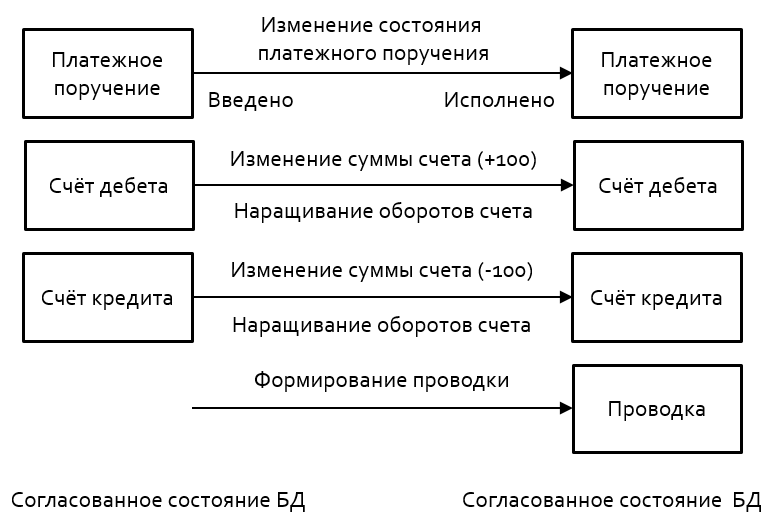
Очевидно, что ограничения должны быть формально объявлены для [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), после чего СУБД должна 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 обновления, которые могут нарушить эти ограничения, и запрещения тех операций, которые их действительно нарушают.

### Управление транзакциями

Если внимательно посмотреть на таблицу 2.1, станет ясно, что управление транзакциями - одна из основных функций СУБД. Поддержка транзакций является основой обеспечения целостности 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, при прямом переводе, означает «обмен» или «соглашение». В широком, общеупотребимом смысле термин используется для обозначения любой операции по обмену данными, в следствие которого в систему вносятся изменения. Сегодня это слово чаще всего 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

**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 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 идентификатор отдела из таблицы «Отделы», то база данных не должна допустить такого действия, если в таблице «Сотрудники» имеются записи сотрудников, относящиеся к удаляемому отделу.
* **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 она выполняется. 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.
* **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. Этот механизм связан с созданием мгновенных снимков – снапшотов.

**Снапшот** (от англ. *snapshot* — мгновенный снимок), — моментальный снимок, 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 Т1) идет через новую «таблицу блоков снапшота».

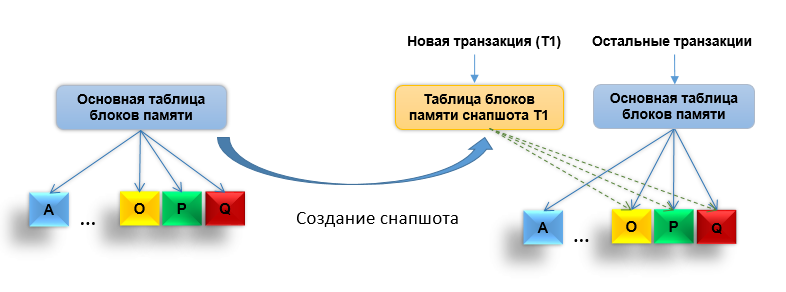


Рис. 2.11 Создание снапшота транзакции

При добавлении данных СУБД выделяет транзакции Т1 новый блок **R** куда и записывается новая информация. Обратите внимание, что информация об этом блоке содержится только в «таблице блоков снапшота» (рис. 2.12), связанной с транзакцией Т1, а остальные транзакции этот блок просто не видят, т.к. его нет в «основной таблице блоков» памяти.

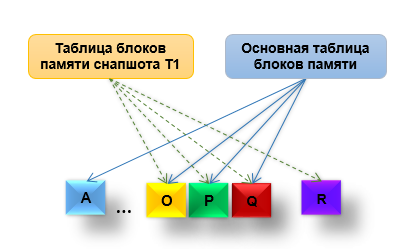


Рис. 2.12 Добавление данных

При необходимости изменения данных СУБД опять выделяет транзакции Т1 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». Теперь «таблица блоков снапшота Т1» ссылается на блок **Q1**, а «основная таблица блоков» ссылается на блок **Q.**

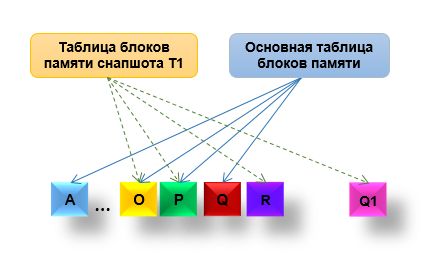


Рис. 2.13 Изменение данных

Таким образом достигается изолированность транзакций, ведь все остальные транзакции полностью независимы от изменений, порождаемых транзакцией Т1. Так, например, транзакция чтения, создав свой, скриншот просто читает данные из блоков памяти своего скриншота, не замечая, что происходит с блоками изменяемыми другими транзакциями.

Рассмотрим теперь завершение транзакции: если завершение транзакции успешно, то «таблица блоков снапшота Т1» копируется в «основную таблицу блоков», а все ненужные блоки памяти, в данном случае это блок **Q**, освобождаются (рис 2.14).

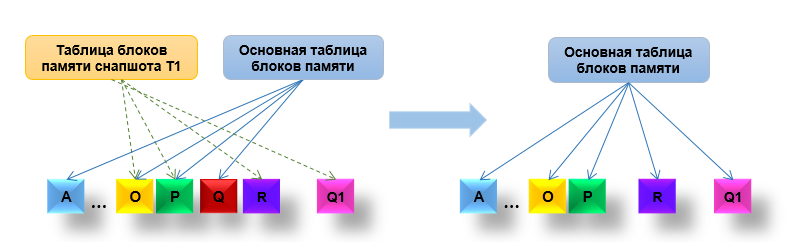


Рис. 2.14 Успешное завершение транзакции

Если во время обработки транзакции происходит сбой, транзакция откатывается - восстанавливается состояние БД на момент начала транзакции, в этом случае нужно просто стереть «таблицу блоков снапшота Т1» и освободить блоки, связанные только с транзакцией Т1, при этом состояние БД, определяемое «основной таблицей блоков» не меняется (рис. 2.15).

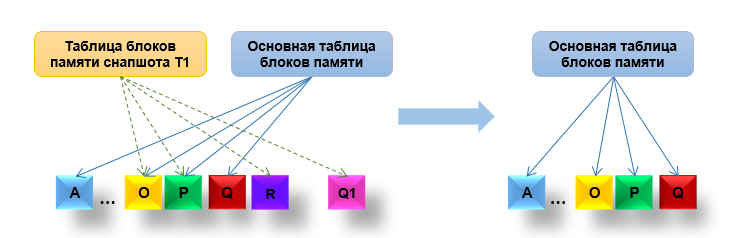


Рис. 2.15 Ошибочное завершение транзакции

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

### 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, все операция изменения блокируются. Блокирование операций чтения выражается в том, что транзакции, пытающиеся изменить заблокированный блок, переводятся в режим ожидания, а как только блокировка будет снята (т.е. закончится блокирующая транзакция1) приостановленная транзакция 2 будет продолжена. Таким образом ситуация, показанная на рис. 2.16 становится невозможной и блок **Q2** не будет создан.

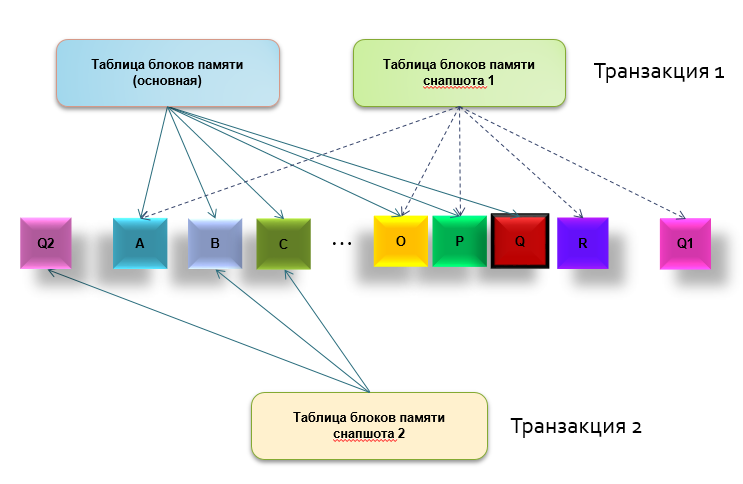


Рис. 2.16 Изменение одного блока двумя транзакциями

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. Все эти блокировки живут недолго и быстро разрешаются.

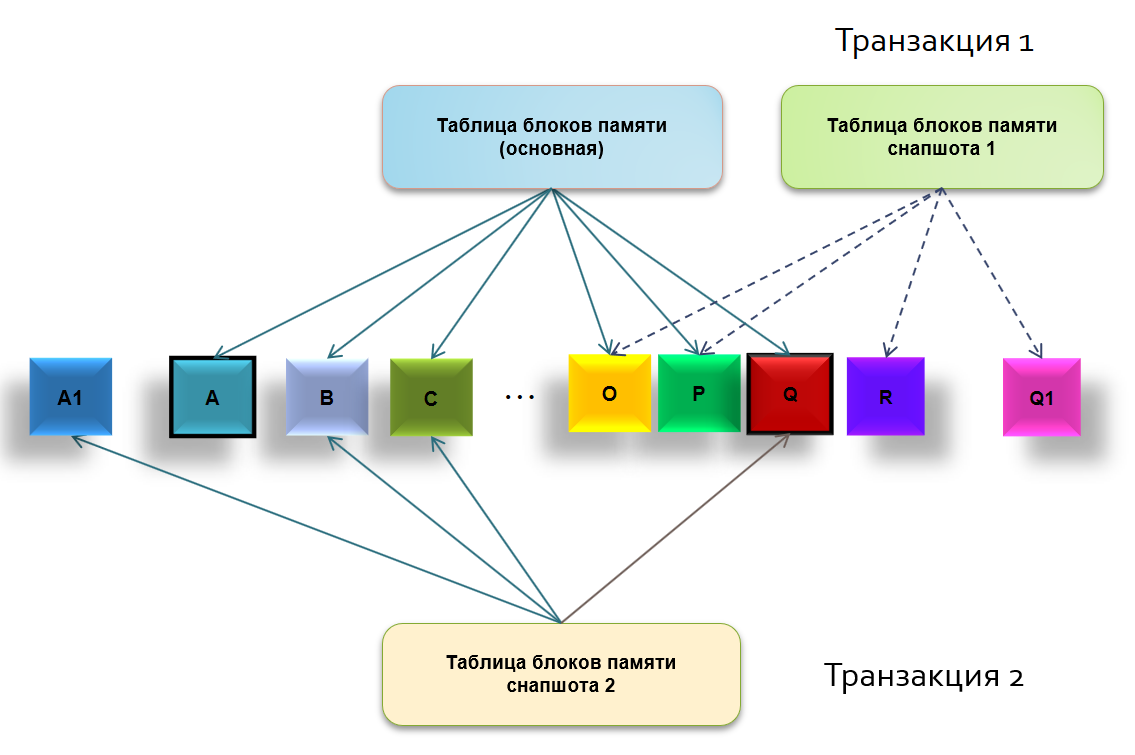


Рис. 2.17 Блокировка транзакции

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 **А** 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 откатывает транзакцию 2, все заблокированные блоки освобождаются и, после окончания работы транзакции 1, СУБД снова запускает транзакцию 2.

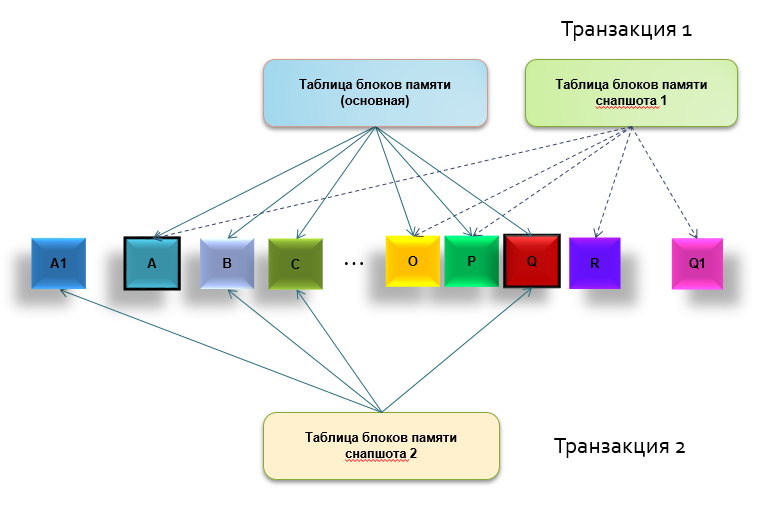


Рис. 2.18 Обоюдная блокировка двух транзакций

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 откатив 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 принудительно 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.

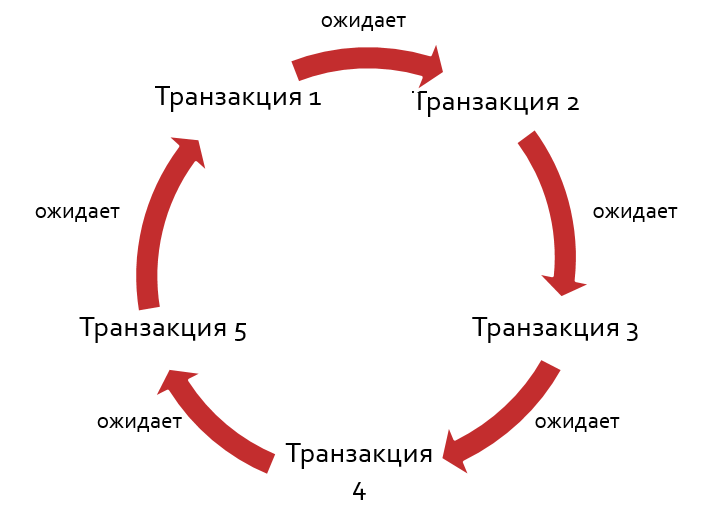


Рис. 2.19 Множественные клинчи

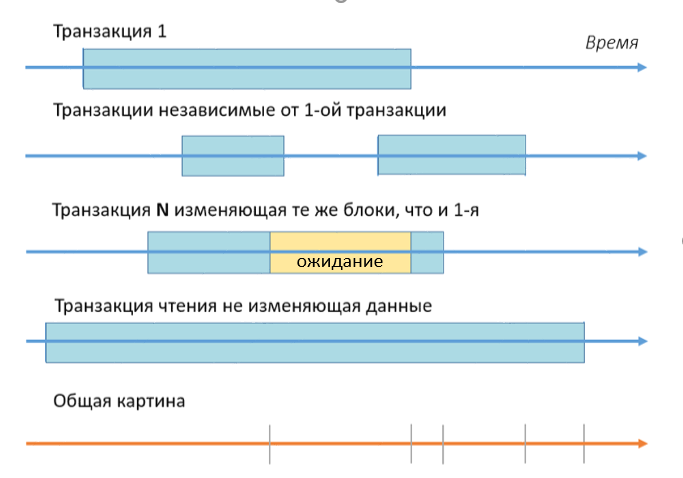


Рис. 2.19 Последовательное выполнение транзакций

### Ведение журнала 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 информация хранится в журналах изменений СУБД, т.е. при любом изменении данных сведения об этих изменениях заносятся в специальные журналы изменений. Все изменения в реляционной БД происходит в рамках транзакций, поэтому **журнал изменений** часто называют **журналом транзакций**.

**Журнализация транзакций -** функция СУБД, которая сохраняет информацию, необходимую для восстановления базы данных в предыдущее согласованное состояние в случае логических или физических отказов.

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 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 БД терпит крах, все транзакции из журнала, которые имеют отметки о фиксации, должны быть повторно выполнены, а все транзакции, имеющие отметку начала, но не имеют отметки о фиксации — отменены.

Функция журнализации транзакций позволяют решать следующие задачи:

* восстановление данных после сбоев (см. раздел 2.2.9);
* анализ работы СУБД и действий пользователей;
* репликация данных.

### Ведение словаря БД

Каждая база данных Oracle 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.

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, а СУБД работает со словарем данных, как с обычной БД.

Таблицы словаря данных описывают все базы данных 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.

Структура таблиц словаря данных оптимизирована для ускорения работы ядра СУБД и для человека практически не 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.

На рисунке 2.11 представлен небольшой фрагмент схемы словаря ORACLE, обратите внимание, что все объекты на схеме — представления.

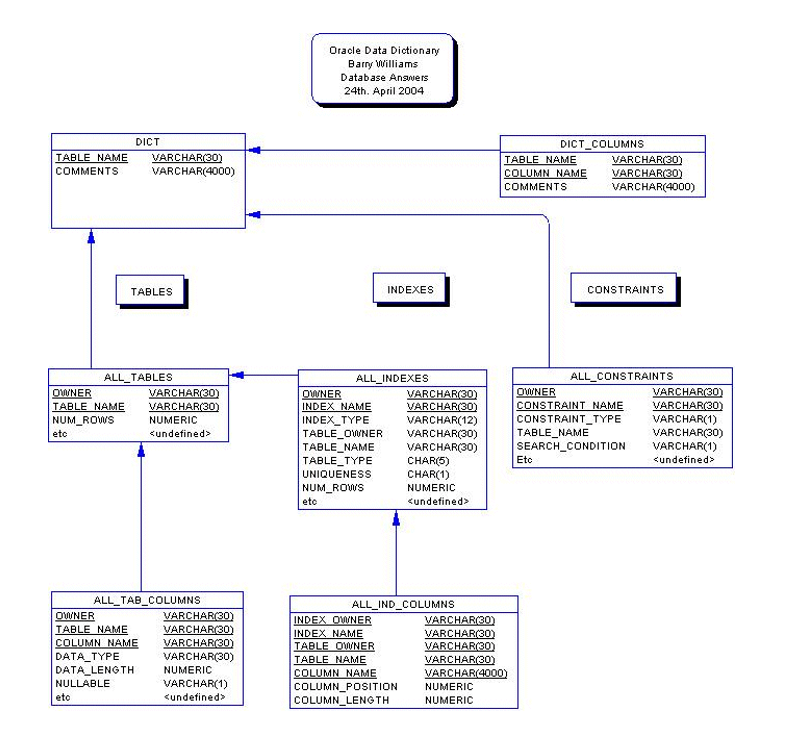


Рис 2.11. Фрагмент схемы словаря данных ORACLE

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 (16).

### 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, целостность 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 затрагивающий устройства долговременного хранения. При таком сбое данные, хранящиеся на дисках, остаются неповрежденными, но утрачивается содержимое буферов базы данных в оперативной памяти.

В журнале транзакций отмечаются точки физической согласованности базы данных, так называемые *контрольные точки* – моменты времени, в которые в долговременной памяти содержатся согласованные результаты операций, завершившихся до соответствующего момента времени, и отсутствуют результаты операций, которые не завершились, и при этом во внешнюю память полностью вытолкнут буфер журнала транзакций.

Жесткий сбой

**Жесткий сбой системы** возникает из-за разрушения структуры 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.

Катастрофа

Ошибки неизвестного происхождения

## 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.

Однако 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 - PostgresSQL. 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 фирмы SUN вместе с легендарным языком JAWA, а в 2010 перешла к ORACLE вместе с поглощенной SUN.

Дерево развития РСУБД приведено на рис 2.10.



Рис 2.10 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 в случае, когда данные распределены по нескольким сервисам. 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. 2.11.

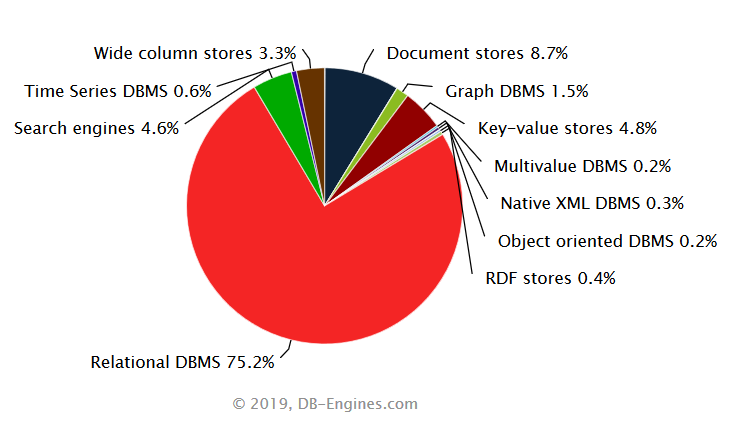


Рис 2.10 Распределение БД по типам используемой модели данных.

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

# Язык SQL

## Становление языка

### 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 1074 г.) и был разработан в компании IBM в рамках проекта экспериментальной реляционной СУБД System R. 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 компанией. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционным БД. Именно поэтому он был приближен к обычному английскому и был задуман как декларативный и непроцедурный.

Несмотря 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 и Microsoft SQL Server поддерживаются очень мощные диалекты SQL.

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.

### Стандарт языка SQL

С тех пор как был принят первый стандарт языка SQL — SQL-89, — он 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. Основные вехи стандартизации языка SQL можно видеть на рис. 3.1

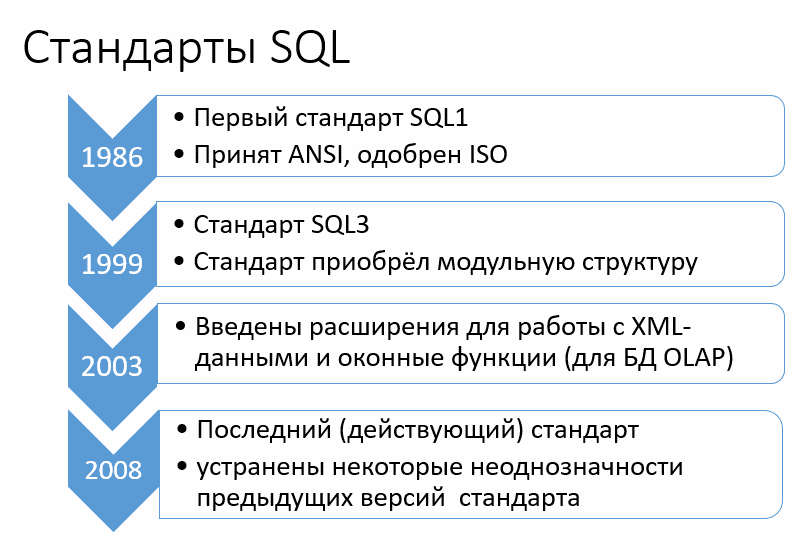


Рис 3.1. Основные вехи стандартизации языка SQL

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 содержи 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](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) 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.

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.

## 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.

### 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 – С, Java), вы указываете, *какого* результата надо достичь и на основе каких *входных данных*. Всю остальную работу выполняет компонент СУБД, 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.

### 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. Гораздо проще расширить язык процедурными операторами, так появились процедурные расширения PL/SQL (ORACLE), Transact-SQL (SYBASE, MS SQL SERVER) и т.д. Все эти языки специфичны для конкретной СУБД, а значит программы, написанные на них непереносимы между СУБД.

### Замкнутость и стиль мышления

Реляционная алгебра 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 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.

### 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 инструкции 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 более подробно будет рассказано в разделе 3.3.)

В некотором смысле 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).

### Равенство формализмов и отход от реляционной теории

Размышляя над языком SQL ка средством общения с реляционными БД 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).

### 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.

### 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 строки из некоторой таблицы и добавлять их в неё, но запрещено удалять или обновлять строки этой таблицы. У другого пользователя может быть другой набор привилегий.

## Диалекты языка SQL

Вот несколько распространенных диалектов:

*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;

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

Пользователи, планирующие плотно работать с одной СУБД, должны изучить все особенности диалекта, используемого этой платформой.

## Понимание SQL

## Типы данных SQL

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) | 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 OB0  JECT 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 |

ROWID (тип ORACLE, в SQL2003 отсутствует) - уникальный идентификатор любой записи в базе данных, это тип псевдостолбца, который называется также - ROWID. А вот другие псевдоcтолбцы - ROWNUM и LEVEL имеют тип NUMERIC.

## Операторы SQL

### Простые примеры

### Общая характеристика и схема выполнения

* Выполняется соединение всех таблиц упомянутых в предложении FROM методом «Декартово произведение»
* Отбираются все записи, которые удовлетворяют условию, указанному в предложении WHERE
* Выполнение действий с данными в зависимости от оператора (SELECT, DELETE, UPDATE, INSERT)
* Операторы определения данных (*Data Definition Language,* ***DDL***):
  + CREATE создает объект БД,
  + ALTER изменяет объект,
  + DROP удаляет объект;
* Операторы манипуляции данными (*Data Manipulation Language,* ***DML****):*
  + 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***):
  + GRANT предоставляет пользователю разрешения на определенные операции с объектом БД,
  + REVOKE отзывает ранее выданные разрешения,
* Операторы управления транзакциями (*Transaction Control Language,* ***TCL****):*
  + START TRANSACTION отмечает начало транзакции,
  + COMMIT сохраняет все изменения текущей транзакции,
  + ROLLBACK откатывает все изменения текущей транзакции,
  + SAVEPOINT делит транзакцию на более мелкие участки.

### Создание объектов СУБД

### Изменение и удаление объектов СУБД

### Инструкция SELECT

### Представления в языке SQL

### Защита данных в SQL

## Функции SQL

# Проектирование БД

В настоящее время нельзя представить вебе информационную систему (ИС) в ядре которой нет базы данных (БД), встроенной или внешней, содержащей все необходимые данные для работы приложения 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. Проектирование БД всегда было и остается творческой задачей, скорее ремеслом, искусством, нежели строго алгоритмизированной работой. В этой главе мы обсудим основные принципы и приемы проектирования БД, которые практикуются в настоящее время

## 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.

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 и 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.

## 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 целостности 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 целостности 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.

Еще несколько слов необходимо 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.

Второй пункт, 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.

Третий пункт 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. Целостность (согласованность) 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 целостности определяются особенностями предметной области. Механизмы обеспечения целостности закладываются при проектировании базы данных, и мы обсудим их позднее (см. раздел 11).

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

помнить, что, как правило, база данных является информационной основой не одного, а нескольких приложений, часть из которых появится в будущем, плохой проект базы данных не может быть исправлен с помощью любых (даже самых изощренных) приложений.

Основные задачи и результат проектирования БД

### 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 вышеприведенных 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 уже все решения принимаются с учетом особенностей конкретной СУБД. Выбор СУБД почти всегда диктуется не техническими, а экономическими или конъюнктурными соображениями (см. раздел «Выбор СУБД»).

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.

### 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 целостности, безопасности и быстродействия БД.

### 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 критичная для работы предприятия, то выбор СУБД чаще всего не связан с техническими вопросами».

## 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».

Упрощенный бизнес процесс логического проектирования БД в нотации IDEF0 представлен на рис.1.

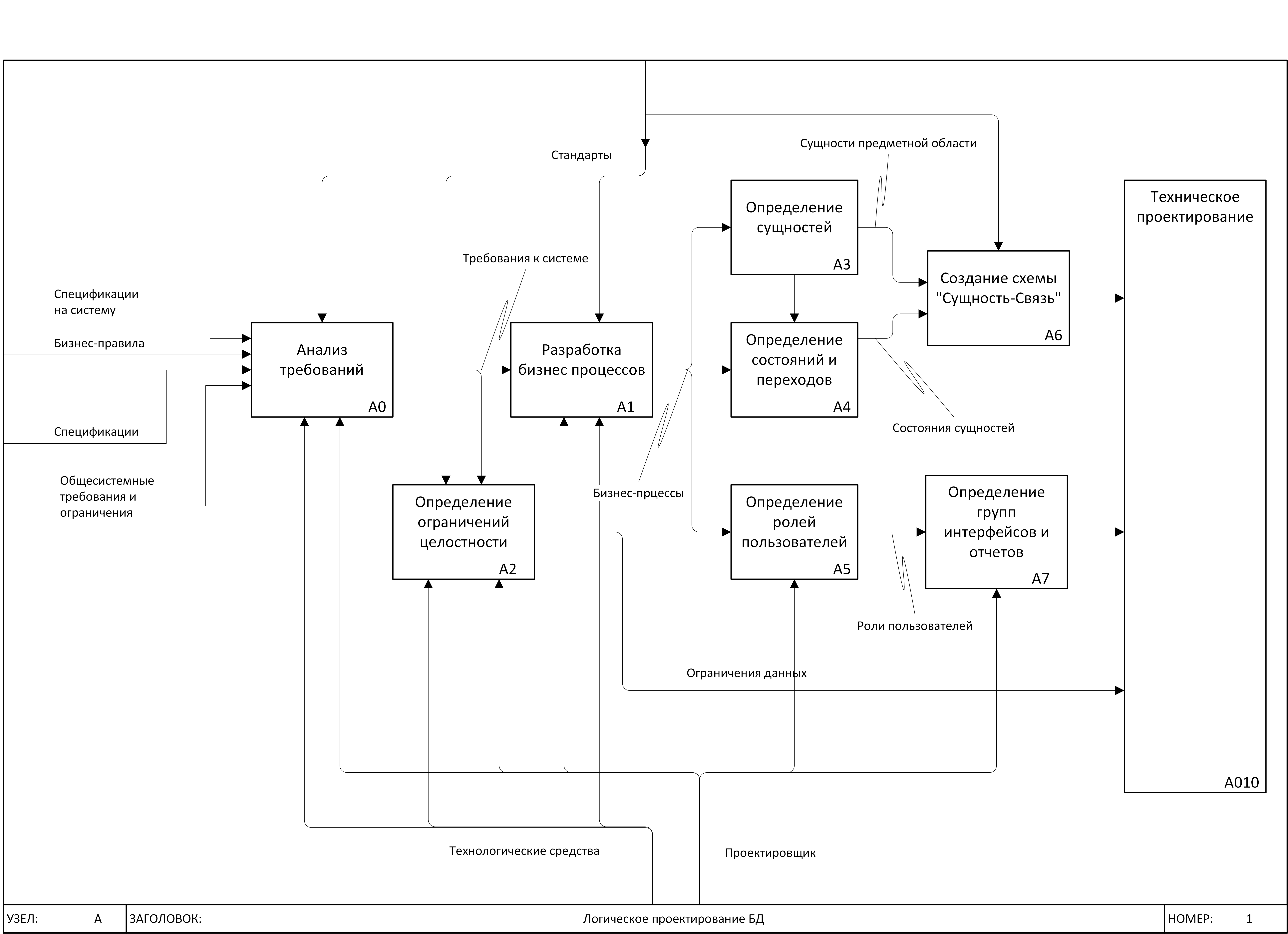
******

Рис.1. 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]. Собственно, проектирование БД лучше всего начать с анализа БП, полученных после разработки требований к ИС.

### 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 другие действия в БП? (см. раздел «Создание ролей пользователей»)

И, наконец, третий вопрос – в каких состояниях могут находиться объекты, с которыми работает пользователь в этом бизнес процессе (см. раздел «Создание атрибутов») и как эти состояния изменяются?

### Определение сущностей

На рис 2. 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.

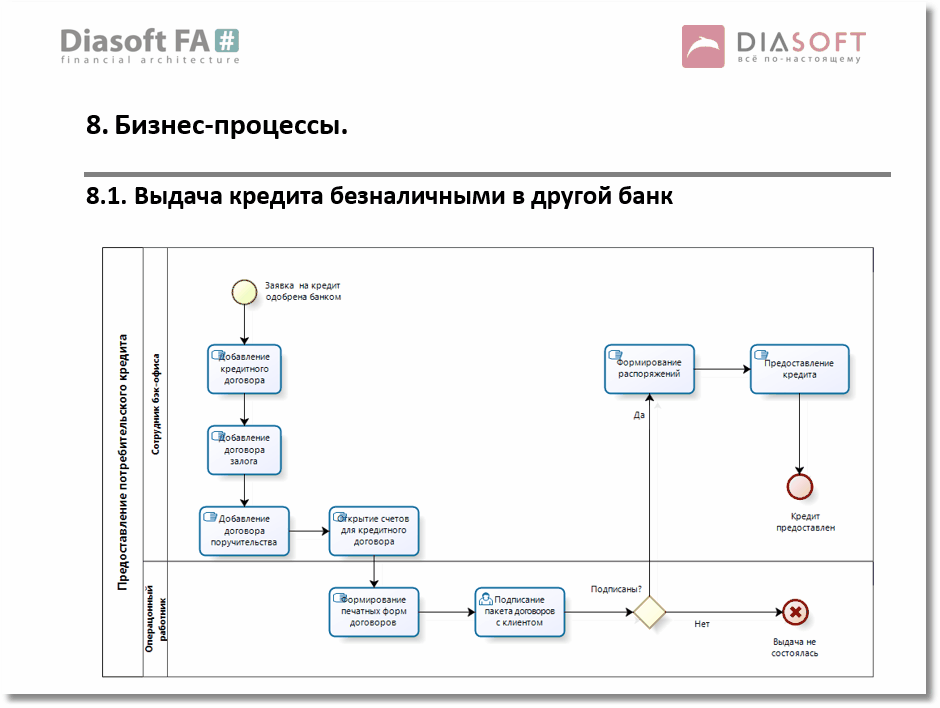
****

Рис.2. 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. 3.

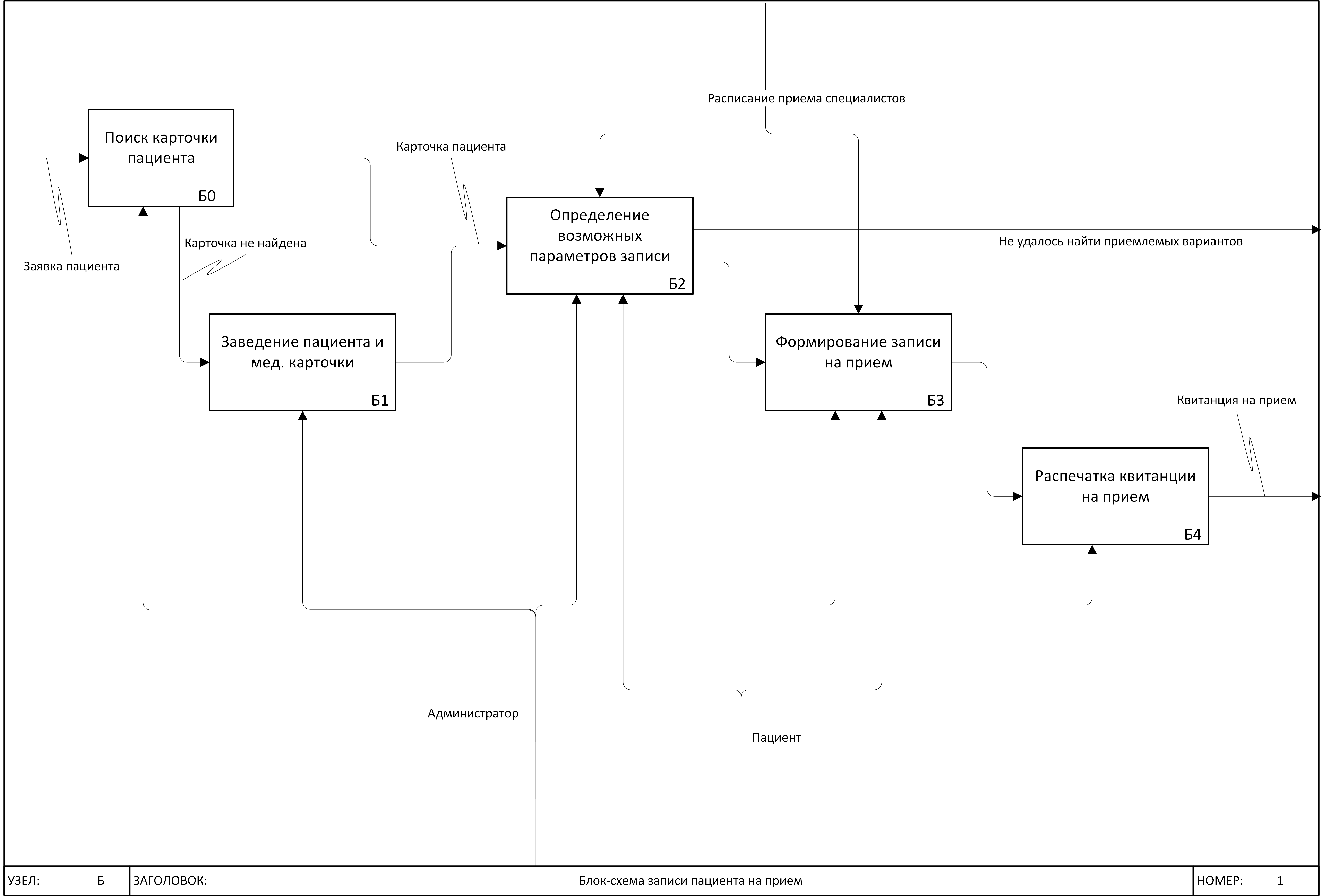


Рис. 3. БП «Запись на прием в поликлинике»

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.

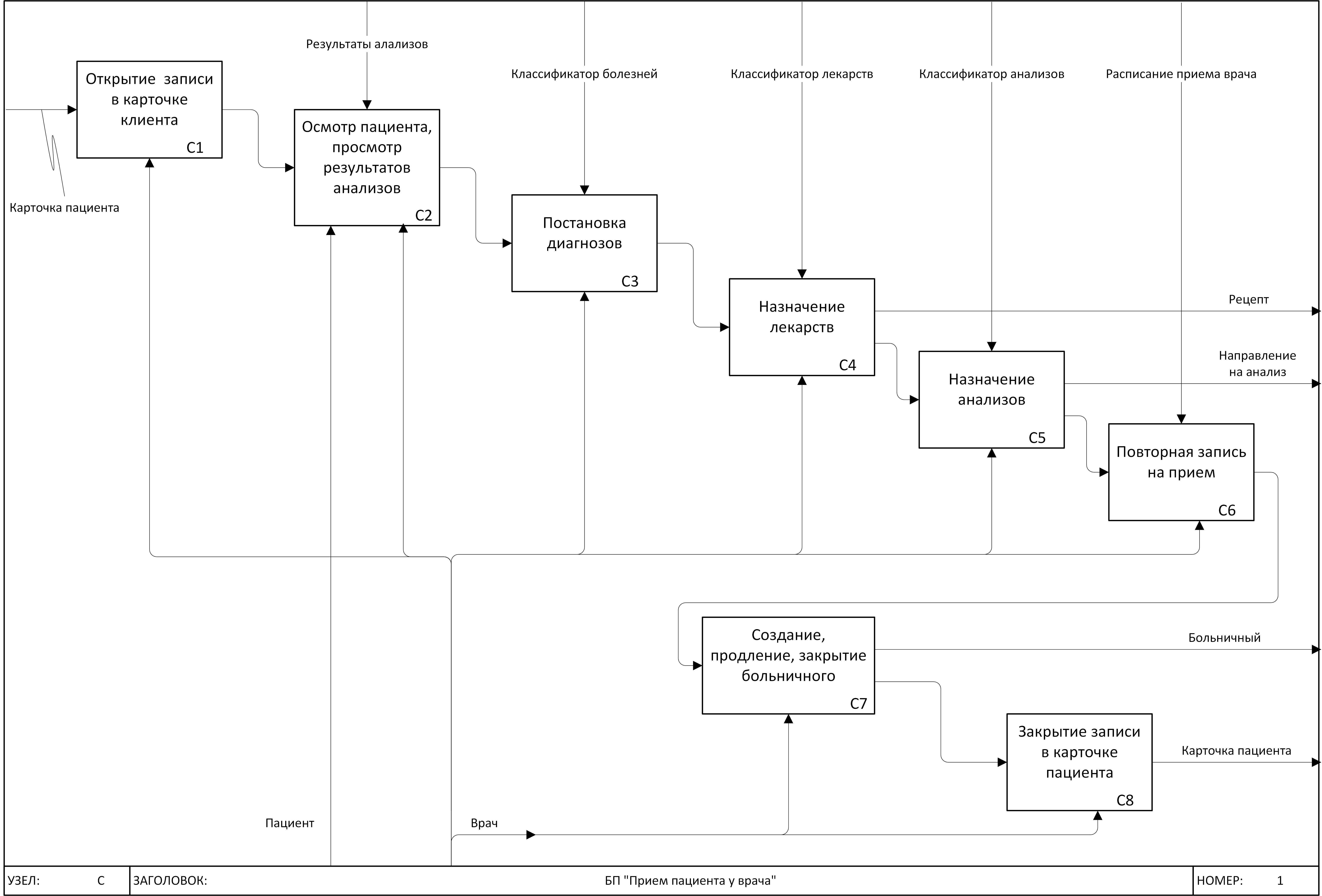


Рис. 4. БП «Прием пациента врачом»

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:

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

### 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.
2. Поинтересуйтесь, возможно такая же сущность уже была выделена в результате анализа другого БП, подумайте, полностью ли они идентичны.
3. 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.
4. Один обязательный атрибут – первичный ключ и нужен еще хотя бы один, для характеристики экземпляра сущности.
5. Подумайте, все ли атрибуты сущности упомянуты и определены.
6. Связана ли сущность с другими сущностями, а если не связана, может быть она просто лишняя?
7. А используется ли вообще данная сущность в проектируемой ИС, и, если нет ни одной функции работающей с ней, может эта сущность не нужна?
8. 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? В каких-то системах да, в каких-то нет, а если да, то где хранить предыдущие значения?
9. 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.?
10. Может в этой сущности заложено слишком узкое понятие? Это вопрос обратный предыдущему - может быть стоит объединить сущности «Гвоздь», «Шуруп», «Саморез» в сущность - «Крепеж»?

### 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.

### Определение атрибутов сущностей

Следующим важным этапом проектирования является определение атрибутов сущностей. 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.

### 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).

### 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 7) и тогда значение этих полей всегда будет актуальным, ведь оно будет вычислено в момент обращения к представлению.

### 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 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 5.3. и снова верифицировать все сущности. Любой процесс проектирования – это итерационный процесс, постепенно приводящий разработчика к оптимальным решениям.

### 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.

### 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. 9.

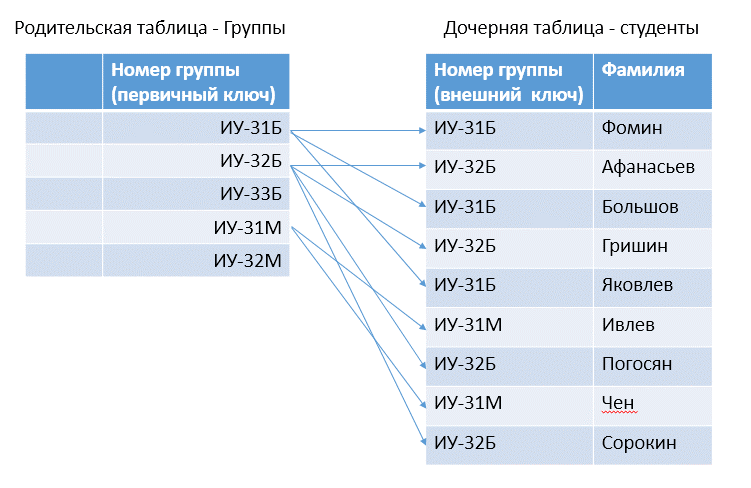


Рис. 9. Связь между таблицами

В таблицах на рис. 9 присутствуют «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 вторичного ключа равным НГ выбрать всех студентов этой группы и наоборот, по значению вторичного 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 связывать таблицы по этому полю нельзя, иначе связи в вашей БД будут становиться неправильными каждый семестр. Чтобы такого не случилось, первичными ключами служат только суррогатные ключи (см. раздел 5.6.), при этом связи между таблицами получатся как на рис. 10.

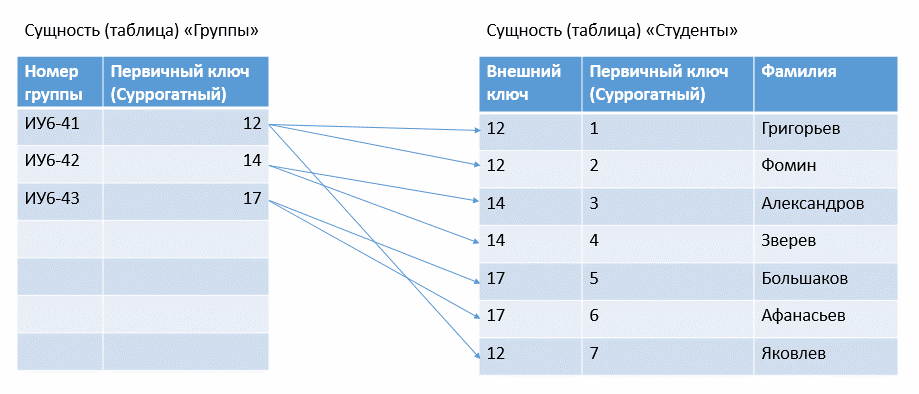


Рис. 10. Связь между таблицами при помощи суррогатного ключа

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.

### «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 примеры.

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 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.

Разделение пространства ключа (дополнительная информация в первичном ключе)

### «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. 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. 11).

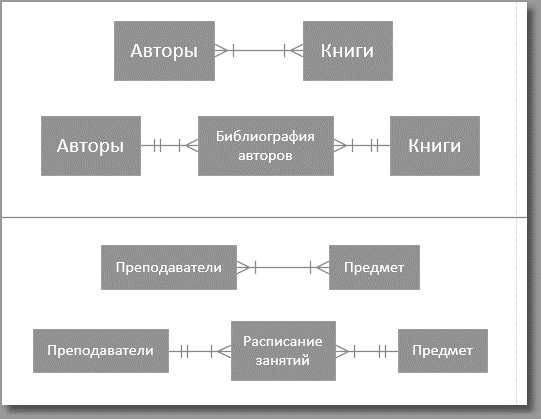


Рис. 11. Примеры разрешения связей «много ко многим»

### 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 11).

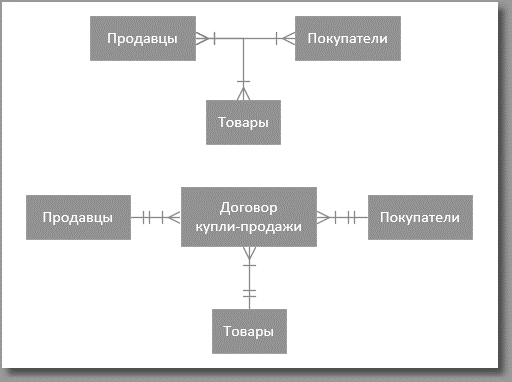


Рис. 12. Пример экзотической связи

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.

### Реализация иерархических структур

### 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? (циклы связей)
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. разделы 6.4, 6.5)

Необходимо раскрыть каждый вопрос и привести примеры

После проверки корректности связей, т.е. после ответов на все вышеприведенные вопросы и внесения корректив, необходимо вернуться к разделу 5.3. и снова верифицировать все сущности. Любой процесс проектирования – это итерационный процесс, постепенно приводящий разработчика к оптимальным решениям

### 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. 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»



Рис. 13. Схема «Сущность-связь» БД «Поликлиника»

Доработаем схему последовательно присоединяя сущности БД

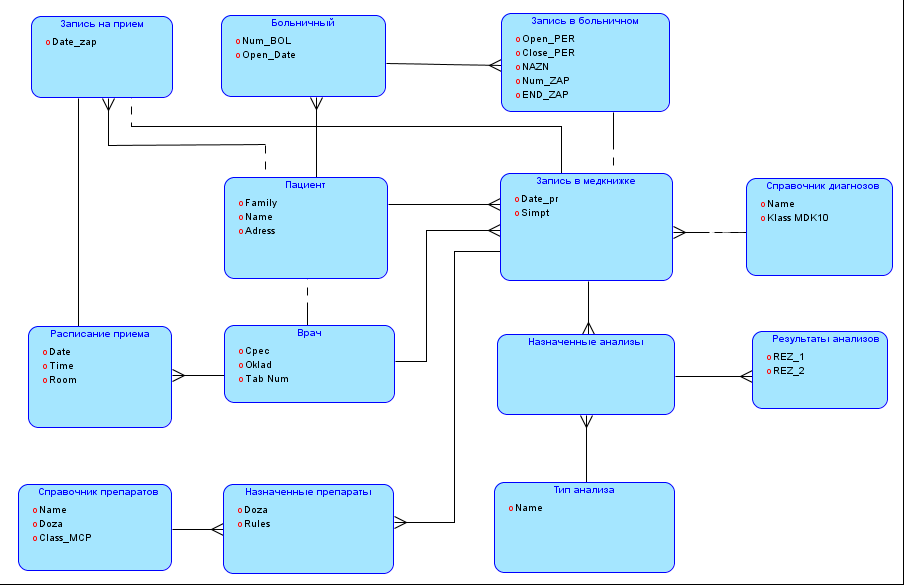


Рис. 14. Полная Схема «Сущность-связь» для БП «Прием пациента врачом»

Внимательно изучите рис. 14 и вспомните БП «Прием пациента врачом», приведенный на рис. 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.

## Физическое проектирование Базы данных

### 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. 14 и рис 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.

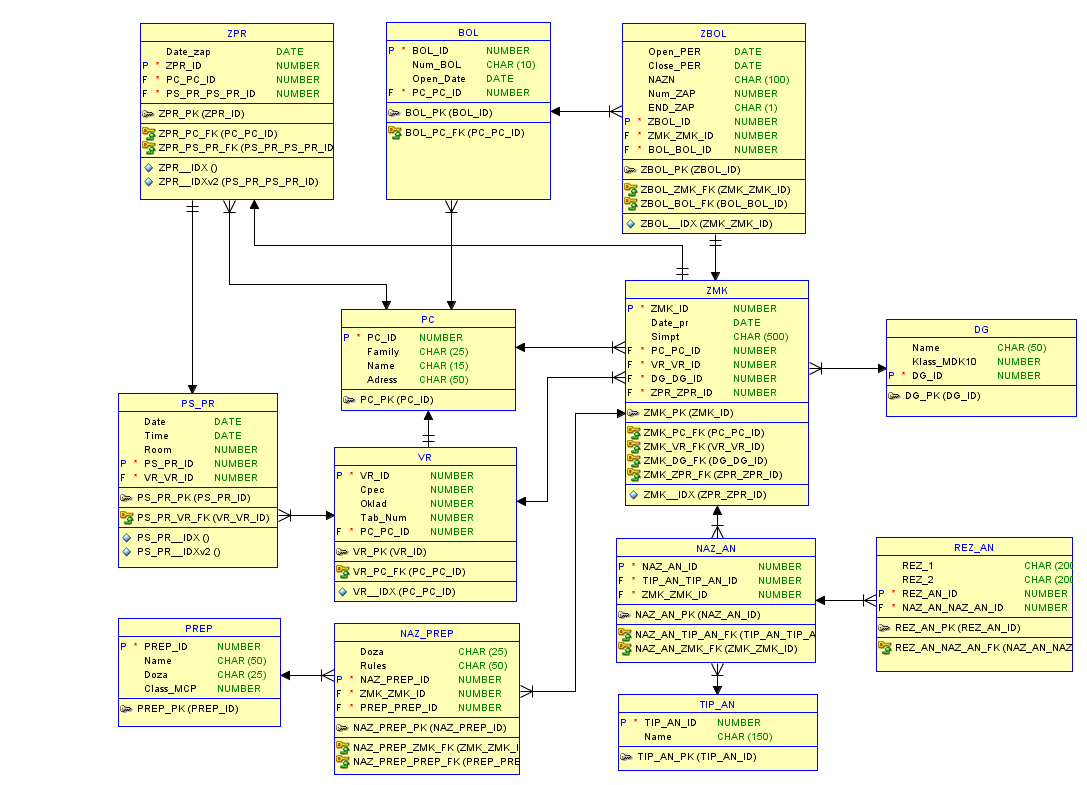


Рис. 15. Полная схема таблиц БД для БП «Прием у врача»

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.

### 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.

1. 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.

1. 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 СР\_Оклад as

SELECT ROUND(EXTRACT(Year from SYSDATE)- EXTRACT(Year from ДATA\_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 ДATA\_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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 3. Работа с представлениями

……..

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Запрос представленный в листинге 2 подробно разбирается в разделе 12.3.

### Создание хранимых процедур

Хранимая процедура — объект БД, представляющий собой набор [SQL](https://ru.wikipedia.org/wiki/SQL)-инструкций, который компилируется один раз и хранится на сервере в словаре БД. Хранимые процедуры пишутся на расширении языка SQL специфичного для каждой отдельной СУБД. Эти процедуры могут иметь входные и выходные параметры, локальные [переменные](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), 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](https://ru.wikipedia.org/wiki/DDL), так и [DML](https://ru.wikipedia.org/wiki/DML)). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры похожи на определяемые пользователем функции (UDF). 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 (для ORACLE)

### Типы встроенных процедур

1. Процедуры обработки ошибок
2. Процедуры, обеспечивающие целостность и согласованность БД
3. Процедуры реализующие особо сложные запросы
4. Триггеры

Заметьте, что встроенные процедуры пишутся не на чистом SQL (ведь он непроцедурный), а на процедурном расширении SQL принятом в конкретной СУБД.

Привести примеры процедур. Сделать акцент на том, что процедуры пишутся не на SQL а на расширениях SQL, специфичных для каждой СУБД (PL SQL)

### 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 целостности 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.

Триггеры используются в следующих случаях:

1. Для автоматической генерации значений полей, например, для присвоения значения суррогатного ключа при создании записи (см. листинг 1).
2. Для логирования при ограничении доступа.
3. Для предотвращения dml операций записи (см. листинг 4).
4. Для реализации сложных ограничений целостности данных записи (см. листинг 4).
5. Для организации всевозможных видов аудита и сбора статистики записи (см. листинг 3).
6. Для оповещения других модулей о том, что делать в случае изменения информации в БД, например, запись во временные таблицы данных управляющих последующим выполнением программы.
7. Для реализации бизнес логики записи (см. листинг 3).
8. Для организации каскадных воздействий на таблицы БД записи (см. листинг 5).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 1. Триггер – Заполнение первичного ключа

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 2. 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 3. Триггер – Сохранение истории записи врача

CREATE OR REPLACE TRIGGER Ист\_вр

AFTER UPDATE OF job\_id, department\_id 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 4. 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 5. 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.

### Создание индексов

Что такое индексы

Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и поиск нужной строки путём последовательного перебора строк таблицы занимает много времени. Чтобы ускорить поиск строк используются индексы. Индекс — это вспомогательная таблица, которая формируется из значений одного или нескольких столбцов индексируемой таблицы и указателей на соответствующие строки. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индексная таблица имеет структуру, оптимизированную под поиск — например, сбалансированного дерева. 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. Именно по этим полям и создаются индексы. 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 7). Заметим, что СУБД обычно автоматически создает индексы по первичным ключам, а по наиболее часто используемым внешним ключам индексы должен создать разработчик БД.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 6. Создание индекса по первичному ключу таблицы «Врачи»

CREATE UNIQUE INDEX VR\_IN\_id

ON VR (VR\_id)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 7. 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.

1. Индексы занимают место.

Для больших таблиц индексы могут занимать много памяти, а если индексов много, то они могут занять места больше, чем сама таблица.

1. Для маленьких таблиц индексы не нужны.

Простой перебор для небольших таблиц (менее 100 записей) эффективней, чем работа с индексом, а значит индексы для таких таблиц не нужны.

1. Для таблиц, которые часто обновляются используйте как можно меньше индексов.

Частое обновление таблицы ведет к частому перестроению индексов и замедляет работу системы.

1. 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.

1. 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.

## Разграничение доступа и создание ролей пользователей

### Что такое разграничение доступа

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, однако, все они делают это по-разному – всеохватывающую систему дискреционной защиты имеет только ORACLE, поэтому мы рассмотрим проектирование системы доступа на примере СУБД ORACLE.

### 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.

Чтобы упростить и упорядочить процедуру раздачи прав создаются роли пользователей. По сути, роль - это набор прав на конкретные объекты БД, позволяющие выполнять некий набор БП. 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.

### 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.

### 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 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 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. 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»,

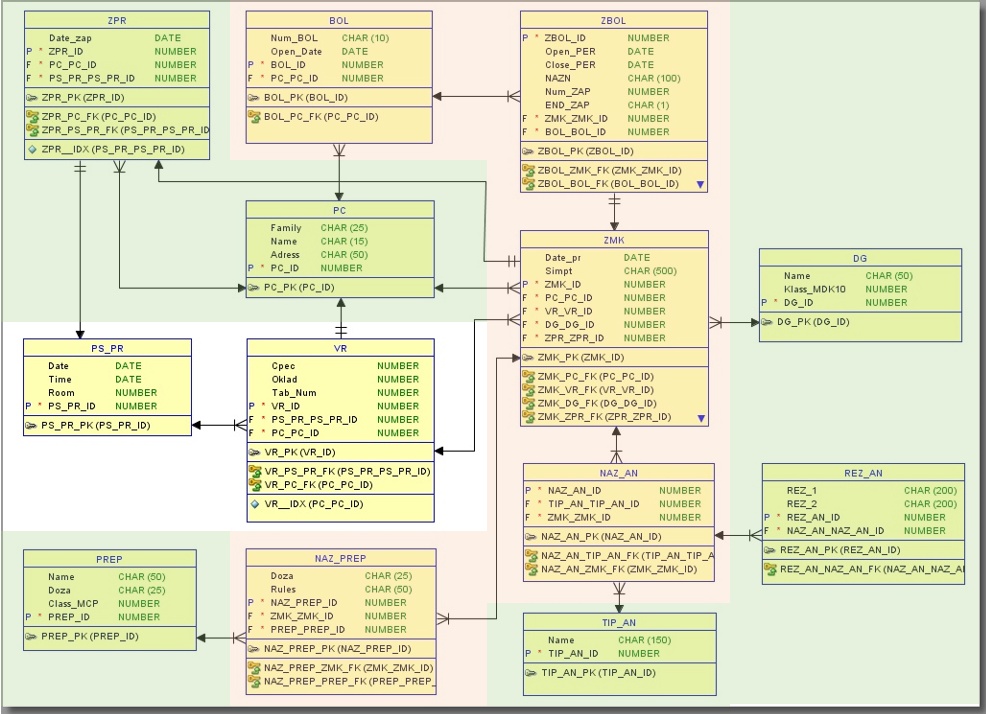


Рис. 16. Полная схема таблиц БД для БП «Прием у врача» с распределением доступа

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;

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Обеспечение целостности БД

### Обеспечение целостности БД

Целостность базы данных (database integrity) — соответствие имеющейся в [базе данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации её внутренней логике, структуре и всем явно заданным правилам.

Очевидно, что ограничения должны быть формально объявлены для [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), после чего СУБД должна отслеживать их выполнение. Объявление ограничений сводится просто к использованию соответствующих средств языка SQL, а соблюдение ограничений осуществляется с помощью контроля со стороны СУБД операций , которые могут нарушить эти ограничения, и запрещения тех операций, которые их действительно нарушают.

Разделяют несколько видов целостности БД:

1. Сущностная целостность

Поддержание сущностной целостности 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 целостность 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.

1. Доменная целостность

Доменная целостность -- это достоверность записей в конкретном столбце. 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.

1. Ссылочная целостность

Ссылочная целостность гарантирует согласованность значений первичных и внешних ключей во всех таблицах. Обычно ссылочная целостность поддерживается триггерами, автоматически заполняющими значение ключей или обеспечивающие невозможность их изменения.

1. Пользовательская целостность

Определяется бизнес-правилами, не входящие ни в одну из категорий целостности, эти правила определяются и отслеживаются не на уровне СУБД, а на уровне ИС. Например, баланс банка должен сходится или число проданных и непроданных билетов на авиарейс должно в сумме равняться вместимости самолета. Пользовательская целостность поддерживается при помощи транзакций и использования триггеров.

### Использование транзакций – путь к обеспечению целостности БД.

**Транзакция** — логическая единица работы, состоящая из одного или нескольких операторов манипулирования данными, которую СУБД рассматривает и обрабатывает как неделимое действие, переводящее БД из одного целостного (согласованного) состояния в другое целостное состояние.

Из этого определения видно, что механизм транзакций надо использовать везде, где необходимо обеспечить целостность БД.

Постой пример – добавление записи в таблицы – родительскую и дочернюю

### Использование триггеров – путь к обеспечению целостности БД.

Использование триггеров – это реализация [ограничений целостности](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1)

Ссылки на п 8.3 – примеры триггеров

### Удаление данных

В различной литературе довольно большое место уделяется процессу удаления данных и возникающим вследствие этого нарушениям ссылочной целостности базы данных. 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.

Конечно, тезис «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.

## Поддержка историчности

В этом разделе мы обсудим каким образом осуществляется сохранение истории изменений данных. 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, 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 2.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 2. Представление – Сохранение истории зарплат

CREATE OR REPLACE TRIGGER Ист\_вр

AFTER UPDATE OF job\_id, department\_id 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;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нахождение зарплаты на дату

Более сложный пример – нахождение курсов валют на заданную дату приведен в листинге

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 2. Представление – 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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Результат работы запроса представлен на рис. 15

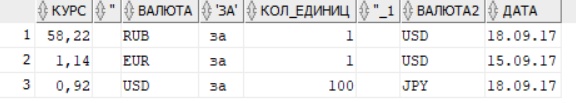


Рис. 15. Курсы валют на заданную (19.09.2017) дату

## Обеспечение быстродействия БД

### Секционирование таблиц

Наиболее **часто решаемой задачей** при секционировании является повышение производительности работы SQL-запросов по выборке и модификации строк таблицы. Это достигается за счет того, что поиск и модификация строк в таблице 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 столбец или несколько столбцов, относительно значений которых, будет делаться 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.

Секционирование необходимо указывать при создании таблицы, поэтому принципы разделения на секции необходимо продумать и предусмотреть при проектировании БД.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Листинг 1. 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)

(PARTITION sales\_q1 VALUES LESS THAN (2008, 04, 01)

TABLESPACE ts1,

PARTITION sales\_q2 VALUES LESS THAN (2008, 07, 01)

TABLESPACE ts2,

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>

### Кластеризация таблиц

Еще один способ увеличения быстродействия БД — это кластеризация таблиц…

Кластеризация— это попытка разместить рядом в одном физическом блоке данные тех записей, доступ к которым осуществляется при помощи одинаковых значений индекса.

## 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 таблиц и СУБД весьма проблематично, например, первая нормальная форма запрещает в отношении одинаковые строки, но не одна СУБД этого не поддерживает (если мы говорим о таблицах).
2. Мнение многих авторов-теоретиков расходятся по поводу «атомарности» полей таблиц, но тогда понятие первой нормальной формы становится неопределенным.
3. При использовании таких простых понятий, как суррогатный первичный ключ, мы гарантированно получаем таблицы, которые находятся во второй нормальной форме.
4. Если в качестве единственного потенциального ключа рассматривать первичный суррогатный ключ, то мы получим таблицу в усиленной третьей нормальной форме

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.

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, Томас, Кун, Дарл .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. В.В. Кириллов, Г.Ю. Громов. 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. - M.: 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. Когаловский М. Р. Энциклопедия технологий баз данных. М.: Финансы и статистика, 2002. ISBN 5-279-02276-4
14. SQL. Библия пользователя, 2-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2010. — 752 с. : ил. ISBN 978-5-8459-1546-7 (рус.)
15. Codd, Edgar F. The relational model for database management: Version 2. — Addison-Wesley, 1990. — ISBN 9-780-201-14192-4.
16. Фейерштейн С., Прибыл Б., Oracle PL/SQL. Для профессионалов. 6-е изд. — СПб.: Питер, 2015. — 1024 с. ил., ISBN 978-5-496-01152-5.
17. Иван Шамаев Электронный ресурс: «Основы PL/SQL» -<https://ivan-shamaev.ru/pl_sql_functions_procedures_variables_cursors_cycles/>
18. Reinhard Pichler Электронный ресурс «Database TheoryVU. Codd’s Theorem» - <https://www.dbai.tuwien.ac.at/staff/pichler/dbt/slides/dbt03.pdf/>

Заключение

# Приложение A. Сокращения принятые в этой книге

**БД** — база данных

**ДЛМ** — даталогическая модель базы данных

**ИЛМ** — инфологическая модель предметной области

**ИМ** — иерархическая модель данных

**НФ** — нормальная форма отношения

**ОЦ** — ограничения целостности

**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 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.

***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.

Далее приведено соответствие терминов в различных областях знаний.

|  |  |  |
| --- | --- | --- |
| База данных | Реляционная алгебра | EXCEL |
| Таблица | Отношение | Таблица |
| Запись | Кортеж | Строка |
| Поле | Атрибут | Столбец |

***Ключ*** - поле или суперпозиция несколько полей и\или их фрагментов (называемых *ключевыми*), однозначно определяющая запись 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 быть первичным ключом, но не является оным, поскольку первичный ключ только один.

***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*** между родительской и дочерней таблицами (сущностями) это сопоставление записей, которое производится по равенству значений внешнего и первичного ключей в обеих таблицах.

***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:М)***. Одной записи родительской таблицы соответствует несколько записей дочерней таблицы. При этом внешний ключ участвующий в связи не уникален.

***Связь типа “Многие-ко-многим” (М:М)***. 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 (*таблица*), содержащий записи, каждая из которых содержит два значения: 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 содержащая всю служебную информацию СУБД.

***Запрос -*** это 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](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется [ограничением целостности](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5_%D1%86%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1). Такие ограничения должны быть формально объявлены в [СУБД](https://wiki2.org/ru/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), после чего СУБД должна контролировать их выполнение

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 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. Средства 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 строки в заданной таблице, или изменением UPDATE данных в определенном столбце 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. Применялось в прошлом веке при проектировании БД.

***Реляционная база данных*** представляет собой набор взаимосвязанных двухмерных таблиц. Эта модель предложена сотрудником фирмы “IBM” Эдгаром Коддом в 1970 году.

Таблица базы данных соответствует одной сущности предметной области и состоит из фиксированного числа полей, 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](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4), использующий структуру [задачи](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0) 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) — механизм [синхронизации](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) содержимого нескольких копий БД. Репликация — 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.

***Тиражирование данных*** предполагает создание копий (репликаций) 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 доступ*** дает возможность одновременной работы с БД многим пользователям. Для корректной работы при одновременной корректировке отдельных записей возможны блокировки записей.

***Администратор базы данных*** ‑ лицо или группа лиц, ответствен­ная за проектирование и эффективное использование БД.

*Основные функции администратора базы данных:*

1. Участие в разработке проекта по созданию БД.
2. Обеспечение согласованности БД (копирование, репликация, восстановление).
3. Обучение пользователей работе с БД (вход в БД, ввод паролей, работа в нормальной и экстремальной ситуациях и др.).
4. Отслеживание трафика загрузки БД, сбоев, характеристик (время доступа, общее время обработки запросов и т.д.).
5. Реорганизация БД с целью улучшения характеристик базы.
6. Реализация многопользовательского режима работы с БД (защита от несанкционированного доступа, шифрование данных, разделение доступа и др.).

***Администратор СУБД (DBA) -*** 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 СУБД*** – 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 он и выполняется. При необходимости сервер приложения формирует запрос к серверу БД.

***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: столбцы и строки таблиц и даже поля строк (значения). Таблицы базы данных и представления имеют владельца или создателя.

***Правила разграничения доступа*** (security policy) — совокупность правил, регламентирующих права субъектов доступа 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.

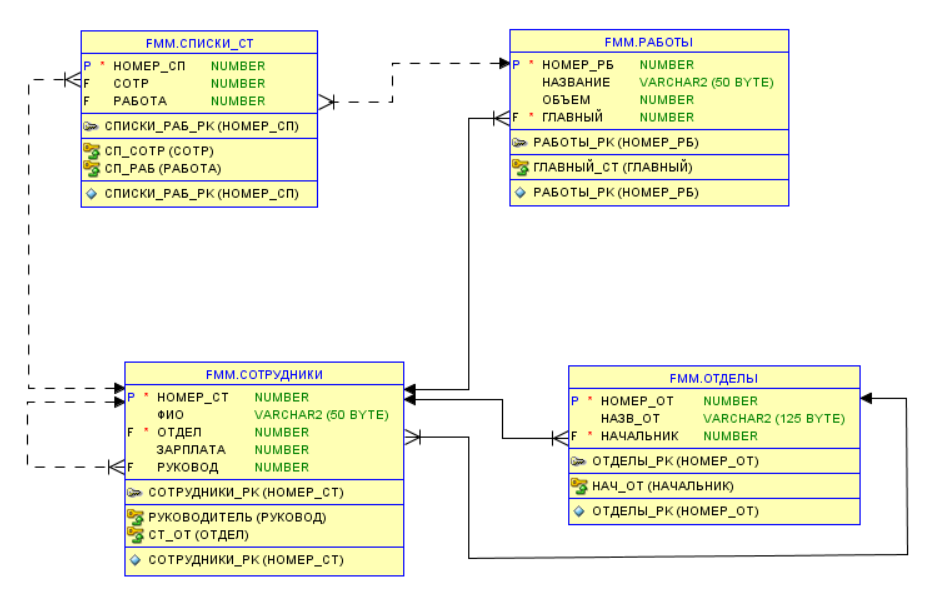
***Идентификация*** (identification) — присвоение объектам и субъектам доступа идентификатора и (или) сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов.

***Пароль*** (password) — идентификатор субъекта, который является его секретом.

***Аутентификация*** (authentification) — проверка принадлежности субъекту доступа предъявленного им идентификатора, подтверждение подлинности.

***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

 Рис. 16. Полная схема таблиц схемы для изучения SQL.

# 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 популярные так называемые «реляционные» СУБД не соответствуют многим критериям.

**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 столбца.

**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 строки.

**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 целостности, 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 строке реляционной таблицы, но и по отношению к любому множеству строк.

**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 целостности** (*Integrity Independence*):

Вся информация, необходимая для поддержания целостности, должна находиться в словаре данных. Язык для работы с данными должен выполнять проверку входных данных и автоматически поддерживать целостность 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 целостности, которые поддерживаются языком более высокого уровня.