# Базы данных

**Кореньков Владимир Васильевич**
профессор САУ,
зав. кафедры «Распределенные информационно-вычислительные системы»,
зам. директора Лаборатории информационных технологий ОИЯИ

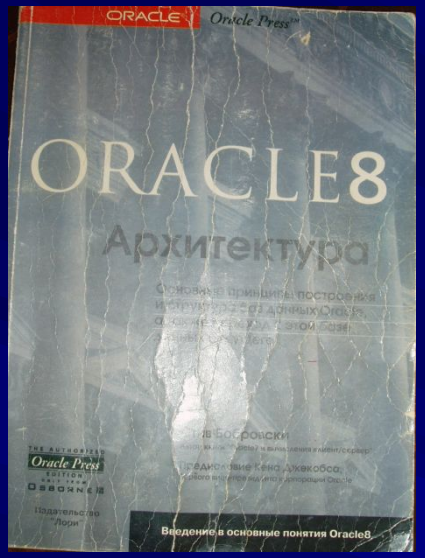
# Основные темы лекций по курсу СУБД

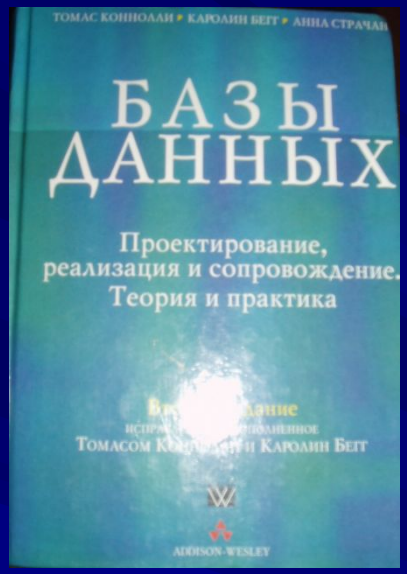
1. Основные понятия баз данных. Этапы развития СУБД. Требования к системам управления базами данных.
2. Архитектура баз данных. Логическая и физическая независимость данных. Схема прохождения запросов к БД. Режимы работы с базой данных. Схема прохождения запроса к БД. Классификация моделей данных. Архитектура и модели "клиент-сервер" в технологии БД.
3. Реляционная модель БД. Таблица, кортеж, атрибут, домен, первичный ключ, внешний ключ. Основные достоинства реляционной модели. Фундаментальные свойства отношений. Обеспечение целостности данных.
4. Операторы реляционной алгебры. Понятия полной и транзитивной функциональной зависимости. Нормализация, нормальные формы.
5. Проектирование баз данных. Семантические модели данных. ER - модель (Entity-Relationship, Сущность-Связи). Этапы проектирования баз данных.
6. Язык SQL, его структура, стандарты, история развития. Подмножество языка DML: операторы SELECT, INSERT, UPDATE, DELETE.
7. Подмножество языка DDL: операторы CREATE, ALTER, DROP. Поддержка ссылочной целостности данных. Представления, их значение. Обновляемые представления.
8. Объектные и системные привилегии. Операторы GRANT, REVOKE. Роли. Транзакции. Операторы управления транзакциями: COMMIT, ROLLBACK, SAVEPOINT. Журнал транзакций.

# Литература


- **Дейт К**. Введение в системы баз данных. – 8 изд., М., Вильямс, 2005
- **Т. Конноли, К. Бегг, А. Страхан** «Базы данных. Проектирование, реализация и сопровождение. Теория и практика». – М: «Вильямс», 2000
- **Кузнецов С.Д.** Базы данных: модели и языки: Учебник - М.: Бином-Пресс, 2008.
- **Андон Ф.,** Резниченко В. Язык запросов SQL - СПб.: BHV: Питер, 2006
- **Маркин А.В.** Построение запросов и программирование на SQL: Учебное пособие для вузов - М.: Диалог-МИФИ, 2008
- **Бобровски С**. Oracle 8: Архитектура – М.: Лори, 1998
- **Прайс Дж.** SQL для Oracle 10g. - М.: Лори, 2007
- **Санжей Мишра, Алан Бьюли.** Секреты Oracle SQL. : Символ-Плюс, 2006
- **А. Саймон** Стратегические технологии баз данных. – М., Финансы и статистика, 2000
- **М.Р. Когаловский** «Энциклопедия технологий баз данных». – М:, Финансы и статистика, 2002
- **Урман С.** Oracle 10g: Программирование на языке PL/SQL - М.: Лори, 2007
- **Абрамсон Я.,** Эбби Майкл С., Кори Майкл "Oracle 10g: Первое знакомство" - М.: Лори, 2007
- www.osp.ruwww.osp.ru, www.citforum.ru

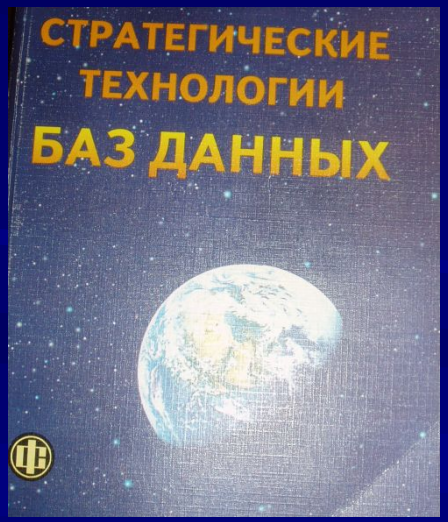
# Литература


ORACLE
Oracle Press
ORACLE8
Архитектура
Введение в основные понятия Oracle8


БАЗЫ ДАННЫХ
Проектирование, реализация и сопровождение. Теория и практика
ADDISON-WESLEY

СЕРИЯ «СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ»
ВВЕДЕНИЕ В СИСТЕМЫ БАЗ ДАННЫХ
ШЕСТОЕ ИЗДАНИЕ

М.Р.Когаловский
ЭНЦИКЛОПЕДИЯ ТЕХНОЛОГИЙ БАЗ ДАННЫХ
ЭВОЛЮЦИЯ ТЕХНОЛОГИЙ
ТЕХНОЛОГИИ И СТАНДАРТЫ
ИНФРАСТРУКТУРА
ТЕРМИНОЛОГИЯ


СТРАТЕГИЧЕСКИЕ ТЕХНОЛОГИИ БАЗ ДАННЫХ

Секреты Oracle SQL
O'REILLY

3-е издание
Oracle PL/SQL
ДЛЯ ПРОФЕССИОНАЛОВ
O'REILLY
ПИТЕР

ORACLE
Oracle Press
101
ORACLE PL/SQL
Как написать мощные и гибкие программы на PL/SQL
Кристофер Аллен
THE AUTHORIZED Oracle Press EDITION

# Основные понятия баз данных

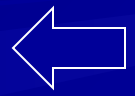
- Информация, хранящаяся в базах данных, является отражением объектов реального мира. В традиционной терминологии объекты реального мира, сведения о которых хранятся в базе данных, называются сущностями — entities, а их актуальные признаки — атрибутами (attributes).
- Объекты реального мира связаны друг с другом множеством сложных зависимостей, которые необходимо учитывать в информационной деятельности.
- Важнейшая задача компьютерных систем — хранение и обработка данных. Для ее решения были предприняты усилия, которые привели к появлению в конце 60-х годов специализированного программного обеспечения — систем управления базами данных (СУБД, database management systems).
- СУБД позволяют структурировать, систематизировать и организовать данные для их компьютерного хранения и обработки. Невозможно представить себе деятельность современного предприятия или учреждения без использования профессиональных СУБД, которые составляют фундамент информационной деятельности во всех сферах — начиная с производства и заканчивая финансами и телекоммуникациями.
- Сердцевиной, центральным компонентом любой СУБД является сервер базы данных. Его техническое качество в решающей степени определяет главные характеристики системы, такие как производительность, надежность, безопасность и т.д. В то же время богатство и разнообразие возможностей, заложенных в механизм его функционирования, сильно сказываются на эффективности разработки прикладных программ.
- Сервер БД является неотъемлемым компонентом модели взаимодействия "клиент-сервер", которая стала фактическим стандартом архитектуры современных СУБД и одним из этапов их развития от систем с централизованной архитектурой и систем с файловым сервером.

# Основные определения

***БД*** - это система специальным образом организованных данных (баз данных), программных, технических, языковых средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Система управления БД (СУБД) - это совокупность языковых и программных средств, обеспечивающих для выполнение всех операций, связанных с организацией хранения данных, их корректирования и доступа к ним.

БД - это поименованная совокупность взаимосвязанных данных находящихся под управлением СУБД.



# База данных

- База данных - это единое, большое хранилище данных, которое однократно определяется, а затем используется одновременно многими пользователями из разных подразделений. Вместо разрозненных файлов с избыточными данными, здесь все данные собраны вместе с минимальной долей избыточности. База данных является общим корпоративным ресурсом и хранит не только данные, но и их описания. По этой причине базу данных еще называют ***набором интегрированных записей с самоописанием***.
- Описание данных называется **системным каталогом** (system catalog), или **словарем данных** (data dictionary), а сами элементы описания принято называть **метаданными** (metadata), т.е. "данными о данных".
- Именно наличие самоописания данных в базе данных обеспечивает **независимость между программами и данными**.
- база данных — это совокупность описаний объектов реального мира и связей между ними, актуальных для конкретной прикладной области.

# СУБД

СУБД - это программное обеспечение, которое взаимодействует с прикладными программами пользователя и базой данных и обладает приведенными ниже возможностями.

Позволяет определять базу данных, что осуществляется с помощью **языка определения данных** (DDL - Data Definition Language). Язык DDL предоставляет пользователям средства указания типа данных и их структуры, а также средства задания ограничений для информации, хранимой в базе данных.

Позволяет вставлять, обновлять и извлекать информацию из базы данных, что осуществляется с помощью **языка управления данными** (DML - Data Manipulation Language). Наличие централизованного хранилища всех данных и их описаний позволяет использовать язык DML как общий инструмент организации запросов, который иногда называют **языком запросов**.

Предоставляет контролируемый доступ к базе данных с помощью перечисленных ниже средств:

- системы обеспечения безопасности, предотвращающей несанкционированный доступ к базе данных со стороны пользователей;
- системы поддержки целостности данных, обеспечивающей непротиворечивое состояние хранимых данных;
- системы управления параллельной работой приложений, контролирующей процессы их совместного доступа к базе данных;
- системы восстановления, позволяющей восстановить базу данных до предыдущего непротиворечивого состояния, нарушенного в результате сбоя аппаратного или программного обеспечения;

# Этапы развития СУБД

В истории развития и совершенствования систем управления базами данных, можно условно выделить три основных этапа.

1) Начальный этап был связан с созданием первого поколения СУБД, опиравшихся на иерархическую и сетевую модели данных (на основе спецификаций CODASYL). По времени он совпал с периодом, когда на рынке вычислительной техники доминировали большие ЭВМ (mainframe), которые в совокупности с СУБД первого поколения составили аппаратно-программную платформу больших информационных систем.

СУБД первого поколения были закрытыми системами: отсутствовал стандарт внешних интерфейсов, не обеспечивалась переносимость прикладных программ. Они не обладали средствами автоматизации программирования и имели массу других недостатков, в том числе и высокую стоимость.

2) С созданием реляционной модели данных был начат новый этап в эволюции СУБД. Простота и гибкость модели привлекли к ней внимание разработчиков и снискали ей множество сторонников. Реляционная модель данных стала доминирующей. Условно эту группу систем можно назвать "вторым поколением СУБД". Его характеризовали две основные особенности — реляционная модель данных и язык запросов SQL (Structured Query Language).

3) Представители второго поколения в настоящее время сохраняют определенную популярность среди производителей СУБД и развились в системы третьего поколения, к которому и относятся современные СУБД.

Для них характерны использование идей объектно-ориентированного подхода, управления распределенными базами данных, активного сервера БД, языков программирования четвертого поколения, фрагментации и параллельной обработки запросов, технологии тиражирования данных, многопотоковой архитектуры и других достижений в области обработки данных. СУБД третьего поколения — это сложные многофункциональные программные системы, работающие в открытой распределенной среде. Они предоставляют разработчикам мощные средства управления данными и богатый инструментарий для создания прикладных программ и систем.

# Требования к современным СУБД

- функциональность,
- производительность,
- защищенность,
- целостность
- масштабируемость
- надежность (катастрофоустойчивость),
- реактивность

# Архитектура Баз Данных

**Внешний уровень**

**Представление базы данных с точки зрения пользователей. Этот уровень описывает ту часть базы данных, которая относится к каждому пользователю**

**Концептуальный уровень**

**Обобщающее представление базы данных. Этот уровень описывает то, *какие* данные хранятся в базе данных, а также связи, существующие между ними**

- **На концептуальном уровне представлены следующие компоненты:**
- **все сущности, их атрибуты и связи;**
- **накладываемые на данные ограничения;**
- **семантическая информация о данных;**
- **информация о мерах обеспечения безопасности и поддержки целостности данных.**

**Внутренний уровень**

**Физическое представление базы данных в ЭВМ.**

**Этот уровень описывает, *как* информация хранится в базе данных.**

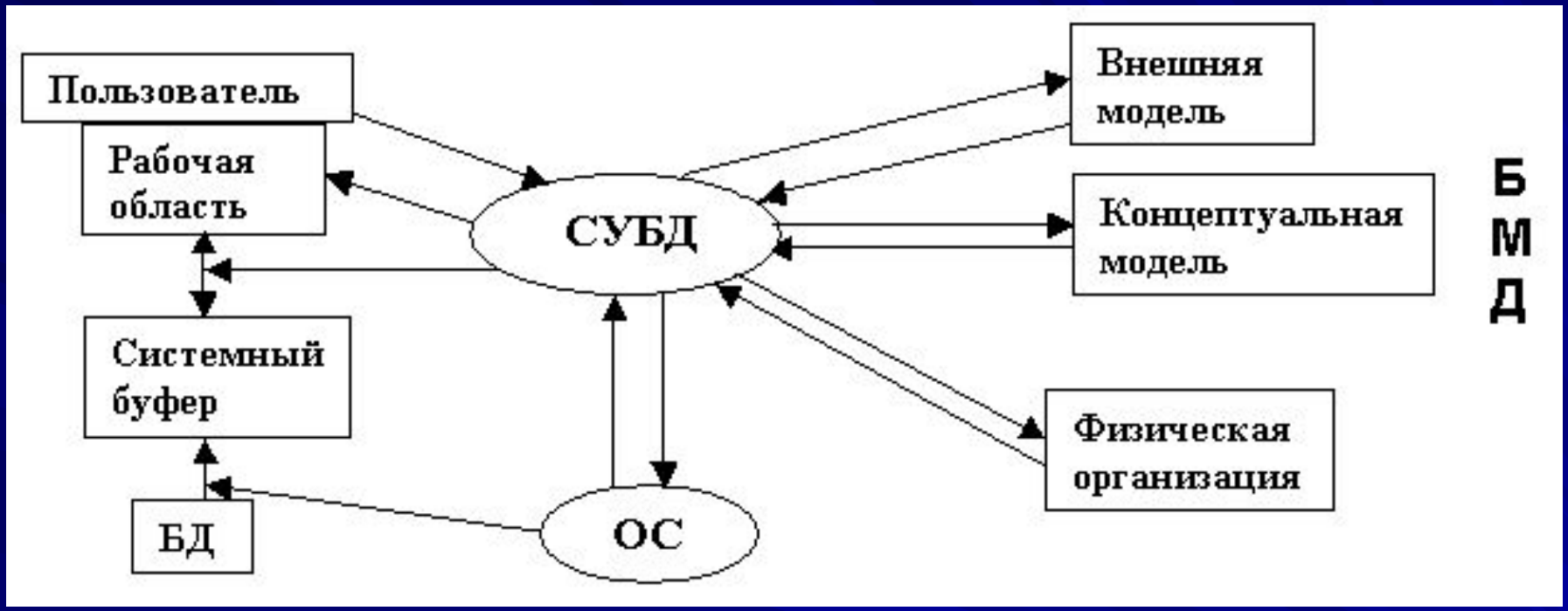
# Логическая и физическая независимость данных

- Основным назначением трехуровневой архитектуры является обеспечение **независимости от данных**, которая означает, что изменения на нижних уровнях никак не влияют на верхние уровни. Различают два типа независимости от данных: **логическую** и **физическую**.

- **Логическая независимость от данных -** означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему

- **Физическая независимость от данных -** означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему

# Схема прохождения запроса к базе данных




**БМД** – База метаданных, в которой хранится вся информация об используемых структурах данных, логической организации данных, правах доступа пользователей, физическое расположение данных. Для управления БМД существует специальное программное обеспечение администрирования баз данных, которое предназначено для корректного использования единого информационного пространства многими пользователями.

# Схема прохождения запроса к базе данных

1) Пользователь посылает запрос на получение данных из БД.
2) Анализ прав пользователя и внешней модели данных, соответствующей данному пользователю, подтверждает или запрещает доступ данного пользователя к запрошенным данным.
3) В случае запрета на доступ к данным СУБД сообщает пользователю об этом и прекращает дальнейший процесс обработки данных, иначе СУБД определяет часть концептуальной модели, которая затрагивается запросом пользователя.
4) СУБД получает информацию о запрошенной части концептуальной модели.
5) СУБД запрашивает информацию о местоположении данных на физическом уровне (файлы или физические адреса).
6) В СУБД возвращается информация о местоположении данных в терминах операционной системы (ОС).
7) СУБД обращается к средствам ОС для предоставления необходимых данных.
8) ОС пересылает данные из устройств хранения в системный буфер.
9) ОС сообщает СУБД об окончании пересылки.
10) СУБД выбирает из системного буфера нужную пользователю информацию и пересылает эти данные в рабочую область пользователя.

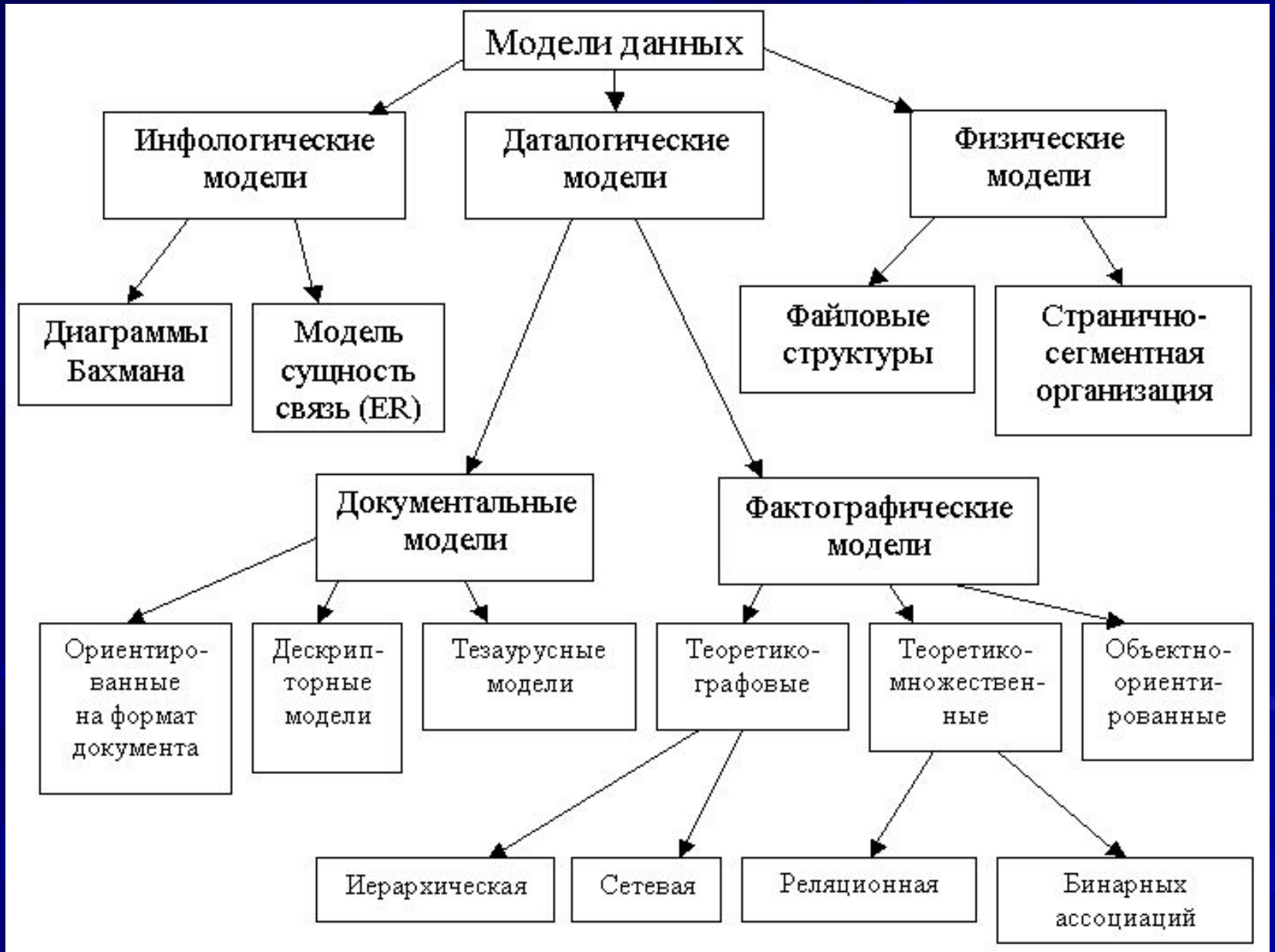
# Схема прохождения запроса к базе данных

Стоит отметить, что описанный выше процесс прохождения запроса не всегда выполняется полностью. Современные СУБД обладают средствами оптимизации выполнения запросов. В частности, если один и тот же пользователь повторно обращается к СУБД с новым запросом, то для него уже не будут проверяться внешняя модель и права доступа, а если дальнейший анализ запроса покажет, что данные могут находиться в системном буфере, то СУБД осуществит только последний шаг в цикле прохождения запроса.
Разумеется, механизм прохождения запроса в реальных СУБД гораздо сложнее, но и эта упрощенная схема показывает, насколько серьезными и сложными должны быть механизмы обработки запросов, поддерживаемые реальными СУБД.

# Данные и модели данных

Одними из основополагающих в концепции баз данных являются обобщенные категории «данные» и «модель данных».
Понятие «данные» в концепции баз данных – это набор конкретных значений, параметров, характеризующих объект, условие, ситуацию или любые другие факторы.
Примеры данных: Иванов Сергей Викторович, $100 и т.д.
Данные – не то же самое, что информация. Данные становятся информацией тогда, когда им задают определенную структуру, то есть придают им смысловое содержание.
Поэтому центральным понятием в области баз данных является понятие модели данных.
**Модель данных** – это некоторая абстракция, которая, будучи приложима к конкретным данным, позволяет трактовать их как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

# Классификация моделей данных



# Классификация моделей данных

Кроме трех рассмотренных уровней абстракции при проектировании БД существует еще один уровень, предшествующий им. Модель этого уровня должна выражать информацию о предметной области в виде, независимом от используемой СУБД.

Эти модели называются *инфологическими*, или *семантическими*, и отражают в естественной и удобной для разработчиков и других пользователей форме информационно-логический уровень абстрагирования, связанный с фиксацией и описанием объектов предметной области, их свойств и взаимосвязей.

Инфологические модели данных используются на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а *даталогические* модели уже поддерживаются уже конкретными СУБД.

*Документальные модели данных* соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке.

*Тезаурусные модели* основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязычных переводчиках. Принцип хранения информации в этих системах и подчиняется тезаурусным моделям.

# Классификация моделей данных

*Дескрипторные модели* – самые простые из документальных моделей, они широко использовались на ранних стадиях использования документальных баз данных. В этих моделях каждому документу соответствовал дескриптор – описатель. Этот дескриптор имел жесткую структуру и описывал документ в соответствии с теми характеристиками, которые требуются для работы с документами в разрабатываемой документальной БД. Обработка информации в таких базах данных велась исключительно по дескрипторам, то есть по тем параметрам, которые характеризовали документ, а не по самому тексту документа.

*Теоретико-графовые модели* данных отражают совокупность объектов реального мира в виде графа взаимосвязанных информационных объектов. В зависимости от типа графа выделяют *иерархическую* или *сетевую* модели. Исторически эти модели появились раньше, и в настоящий момент они используются реже, чем более современная *реляционная* модель данных. Однако до сих пор существуют системы, работающие на основе этих моделей, а одна из концепций развития объектно-ориентированных баз данных предполагает объединение принципов сетевой модели с концепцией реляционной.

# Режимы работы с базой данных

При размещении БД на персональном компьютере, который не находится в сети, БД всегда используется в монопольном режиме. Даже если БД используют несколько пользователей, они могут работать с ней только последовательно, и поэтому вопросов о поддержании корректной модификации БД в этом случае не возникает, они решаются организационными мерами.

Однако работа на изолированном компьютере с небольшой базой данных в настоящий момент становится уже нехарактерной для большинства приложений.

БД отражает информационную модель реальной предметной области, она растет по объему и резко увеличивается количество задач, решаемых с ее использованием, и в соответствии с этим увеличивается количество приложений, работающих с единой базой данных.

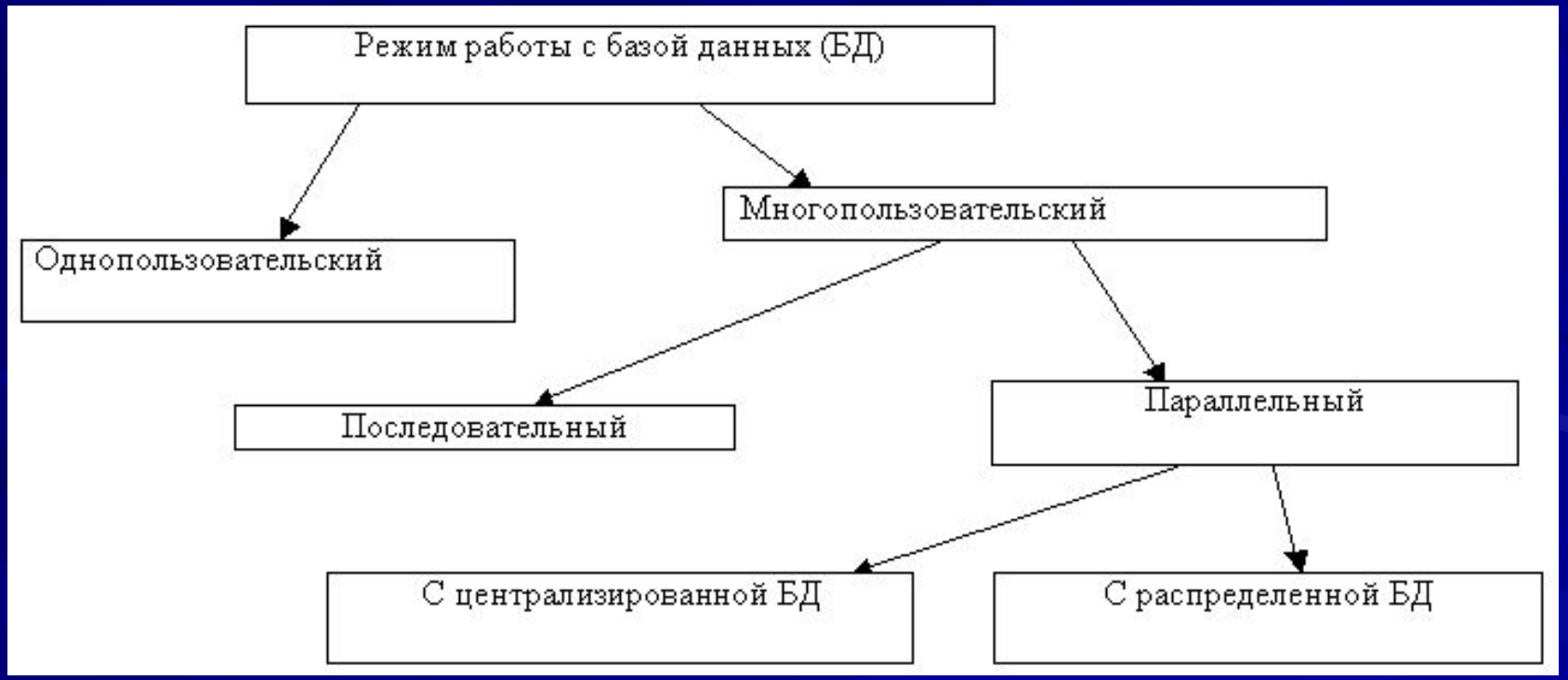
Компьютеры объединяются в локальные сети, и необходимость распределения приложений, работающих с единой базой данных по сети, является несомненной.

Действительно, даже когда вы строите БД для небольшой торговой фирмы, у вас появляется ряд специфических пользователей БД, которые имеют свои бизнес-функции и территориально могут находиться в разных помещениях, но все они должны работать с единой информационной моделью организации, то есть с единой базой данных.

# Режимы работы с базой данных

Параллельный доступ к одной БД нескольких пользователей, в том случае если БД расположена на одной машине, соответствует режиму распределенного доступа к централизованной БД. (Такие системы называют *системами распределенной обработки данных*.)

Если же БД распределена по нескольким компьютерам, расположенным в сети, и к ней возможен параллельный доступ нескольких пользователей, то мы имеем дело с параллельным доступом к распределенной БД. Подобные системы называются *системами распределенных баз данных*.



# Модель «клиент-сервер»

Вычислительная модель «клиент-сервер» исходно связана с парадигмой открытых систем, которая появилась в 90-х годах и быстро эволюционировала. Сам термин «клиент-сервер» исходно применялся к архитектуре программного обеспечения, которое описывало распределение процесса выполнения по принципу взаимодействия двух программных процессов, один из которых в этой модели называли «клиентом», а другой – «сервером». Клиентский процесс запрашивал некоторые услуги, а серверный процесс обеспечивал их выполнение. При этом предполагалось, что один серверный процесс может обслужить множество клиентских процессов.

Ранее приложение не разделалась на части, оно выполнялось некоторым монолитным блоком. Но возникла идея более рационального использования ресурсов сети. Действительно, при монолитном исполнении используются ресурсы только одного очень мощного компьютера, а остальные компьютеры в сети рассматриваются как терминалы и не обладают никакими вычислительными ресурсами. Но с появлением персональных компьютеров, в отличие от эпохи main-фреймов, все компьютеры в сети стали обладать собственными вычислительными ресурсами, и было бы разумным так распределить нагрузку на них, чтобы максимальным образом использовать их ресурсы.

# Разделение функций

Основной принцип технологии «клиент-сервер» применительно к технологии баз данных заключается в разделении функций стандартного интерактивного приложения на три части:

**Представление (Presentation Logic).**

**Обработка (Business Logic).**

**Хранение (Data manipulation Logic) и данные (Data).**

Презентационная логика (Presentation Logic) как часть приложения определяется тем, что пользователь видит на своем экране, когда работает с приложением, иными словами, это интерфейс приложения. Сюда относятся все интерфейсные экранные формы, которые пользователь видит или заполняет в ходе работы приложения (для веб-приложений – это HTML-страницы, загружаемые при помощи браузера на компьютер пользователя). К этой же части относится все то, что выводится пользователю на экран как результаты выполнения запрошенных действий. Презентационная логика всегда находится на компьютере пользователя, поскольку иначе пользователь не смог бы взаимодействовать с приложением.

Основными задачами презентационной логики являются:

формирование экранных изображений;

чтение и запись в экранные формы информации;

управление экраном, движением мыши, клавиатуры.

# Разделение функций

Бизнес-логика (Business Logic) – это исполняемая часть приложения, которая определяет алгоритмы решения конкретных задач приложения. Эта часть приложения может находиться как на клиентском компьютере, так и на сервере. В зависимости от того, в какой пропорции исполняемая часть распределена между клиентом и сервером, клиента и сервер могут называть «толстым» и «тонким». Чем больше функций приложения, реализующих алгоритм решения задачи, находится на клиенте, тем он «толще», чем меньше, тем он «тоньше»; то же относится и к серверу.

Логика обработки данных (Data manipulation Logic) и данные (Data) – это часть функций приложения, которая чаще всего возложена на сервер. Это собственно данные, составляющие базу данных, и функции по управлению хранением данных на сервере.

В зависимости от распределения описанных выше трех функций между клиентом и сервером различают четыре модели «клиент-сервер» в технологии баз данных.

# Модель файлового сервера

В модели файлового сервера (File Server, FS) презентационная логика и бизнес-логика располагаются на клиенте. На сервере располагаются файлы с данными, и поддерживается доступ к файлам. Клиент обращается к серверу с файловыми командами, а механизм управления всеми информационными ресурсами, собственно база метаданных, находится на клиенте.

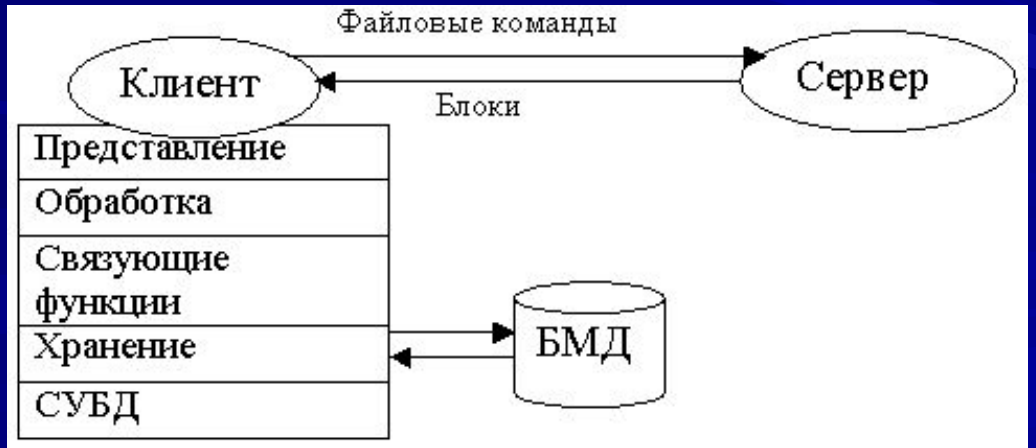
Единственным достоинством этой модели можно считать факт разделения функций между клиентом и сервером и возможность доступа к файлам, хранящимся на сервере одновременно многим пользователям.

К недостаткам модели можно отнести:

высокий сетевой трафик (пересылаются файлы целиком, даже полезной в нем является всего одна строка);

узкий спектр операций манипулирования с данными, который определяется файловыми командами;

отсутствие средств безопасности доступа к данным (на уровне файловой системы).



# Модель удаленного доступа к данным

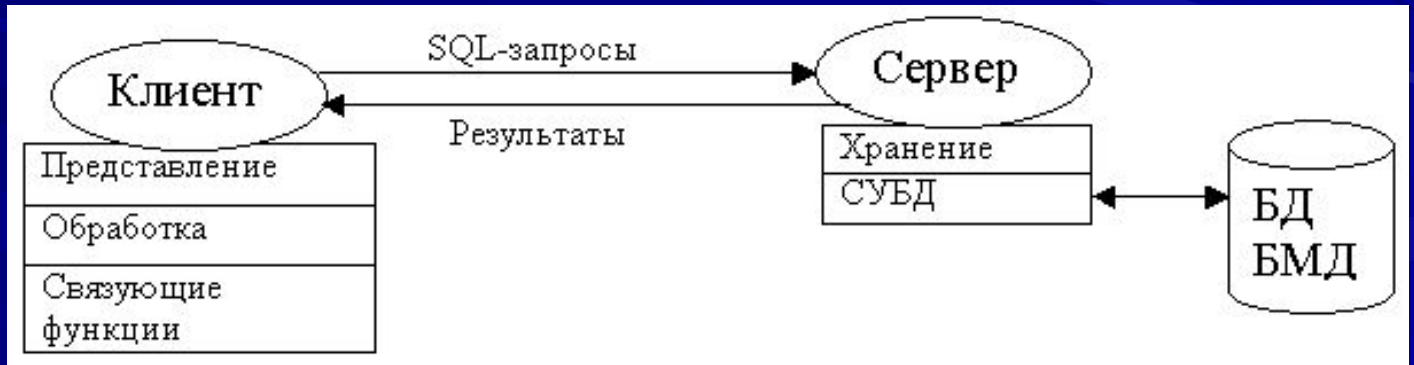
Отличием модели удаленного доступа к данным (Remote Data Access, RDA) от модели файлового сервера является то, что ядро СУБД расположено на сервере. Презентационная логика и бизнес-логика расположены на стороне клиента. Достоинством данной модели можно считать значительное сокращение сетевого трафика, так как по сети передаются не запросы на ввод-вывод в файловой терминологии, а запросы на языке SQL. Иными словами, вместо файлов по сети передается только полезная информация, определенная пользователем при помощи языка структурированных запросов (Structured Query Language, SQL), которая выделяется из файлов на уже стороне сервера самой СУБД.

Недостатки:

высокий сетевой трафик (несмотря на значительно сокращение сетевого трафика, по сравнению с модель файлового сервера, все-таки запросы на языке SQL при интенсивной работе клиентских приложений могут существенно загрузить сеть);

дублирование кода приложений (запросы на получение одних и тех же данных присутствуют в виде копий в различных приложениях);

пассивный сервер.



# Модель сервера баз данных

Модель сервера баз данных (Database Server, DBS) поддерживается многими современными СУБД: Informix, Ingres, Sybase, Oracle, MS SQL Server. Основу данной модели составляет механизм хранимых процедур как средство программирования SQL-сервера, механизм триггеров как механизм отслеживания текущего состояния информационного хранилища и механизм ограничений на пользовательские типы данных, который иногда называется механизмом поддержки доменной структуры. В этой модели бизнес-логика разделена между клиентом и сервером. На сервере бизнес-логика реализована в виде хранимых процедур – специальных программных модулей, которые хранятся в БД и управляются непосредственно СУБД. Клиентское приложение обращается к серверу с командой запуска хранимой процедуры, а сервер выполняет эту процедуру и регистрирует все изменения в БД, которые в ней предусмотрены. Сервер возвращает клиенту данные либо для вывода на экран, либо для выполнения части бизнес-логики, которая расположена на клиенте.

# Модель сервера баз данных

В данной модели сервер является активным, потому что не только клиент, но и сам сервер, используя механизм триггеров, может быть инициатором обработки данных в БД.

И хранимые процедуры, и триггеры хранятся в словаре БД, они могут быть использованы несколькими клиентами, что существенно уменьшает дублирование алгоритмов обработки данных в разных клиентских приложениях. Трафик обмена информацией между клиентом и сервером резко уменьшается. Недостатком данной модели является очень большая загрузка сервера, поскольку на него перекладывается большая часть бизнес-логики, предназначенной для обработки данных.

Однако это же одновременно является и плюсом, поскольку теперь требования к клиентам резко уменьшаются. Иногда такую модель называют моделью с «тонким» клиентом.

При построении приложений в модели сервера баз данных значительно упрощается организация обновления версий клиентских приложений, поскольку при обновлении приложения на стороне сервера, обновление части приложения на стороне клиента иногда даже не требуется. Кроме того, при использовании модели сервера баз данных увеличивается надежность и защищенность создаваемых приложений.

# Модель сервера приложений

Модель сервера приложений (Application Server, AS) является расширением двухуровневой модели и в ней вводится дополнительный промежуточный уровень между клиентом и сервером. Этот промежуточный уровень содержит один или несколько серверов приложений.

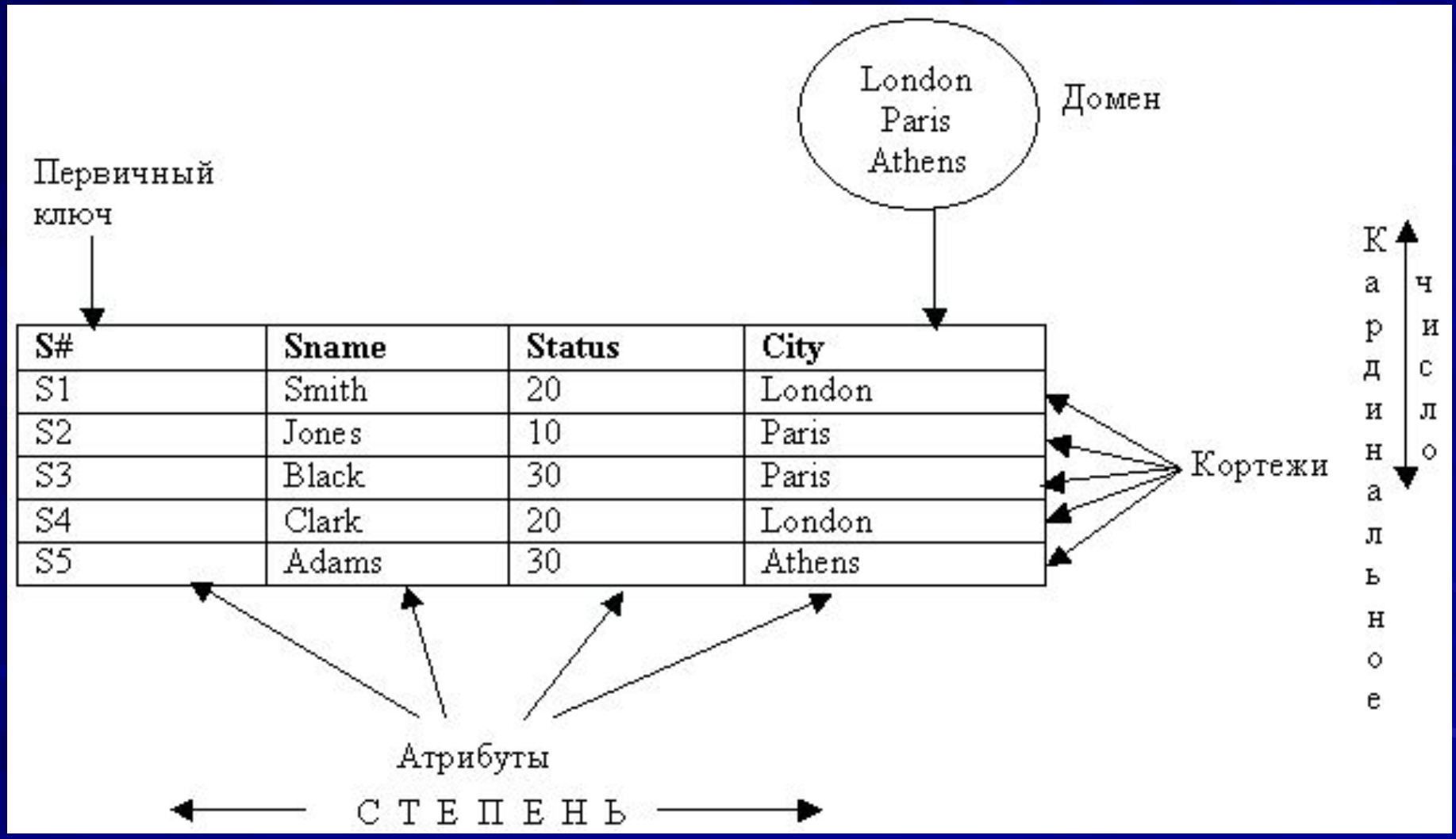
Здесь в наибольшей степени выражен принцип разделения функций, поскольку каждая из трех функций расположена теперь на отдельных компьютерах: презентационная логика находится на клиенте, базнес-логика реализована на серверах приложений, а данные хранятся на сервере баз данных.

Эта модель обладает большей гибкостью, чем двухуровневые модели. Наиболее заметны преимущества модели сервера приложений в тех случаях, когда клиенты выполняются сложные аналитические расчеты над базой данных, которые относятся в области OLAP-приложений (On-line analytical processing). В этой модели большая часть бизнес-логики клиента изолирована от возможностей расширенного SQL, реализованного в конкретной СУБД, и может быть выполнена на стандартных языках программирования, таких как C, C++, Object Pascal, Java. Это повышает переносимость системы, ее масштабируемость.

# Реляционная модель БД

- Реляционная модель данных была разработана Коддом  в 1970 году на основе математической теории отношений и опирается на систему понятий, важнейшими из которых являются таблица, отношение, строка, столбец, первичный ключ, внешний ключ.
- Реляционной считается такая база данных, в которой все данные представлены для пользователя в виде прямоугольных таблиц значений данных, и все операции над базой данных сводятся к манипуляциям с таблицами. Таблица состоит из строк и столбцов и имеет имя, уникальное внутри базы данных. Таблица отражает тип объекта реального мира (сущность), а каждая ее строка — конкретный объект. Каждый столбец таблицы — это совокупность значений конкретного атрибута объекта. Эти значения выбираются из множества всех возможных значений атрибута объекта, которое называется доменом (domain).
- Каждый столбец имеет имя, которое обычно записывается в верхней части таблицы. Оно должно быть уникальным в таблице, однако различные таблицы могут иметь столбцы с одинаковыми именами. Любая таблица должна иметь по крайней мере один столбец; столбцы расположены в таблице в соответствии с порядком следования их имен при ее создании. В отличие от столбцов, строки не имеют имен; порядок их следования в таблице не определен, а количество логически не ограничено.

# Реляционная модель БД



# Соответствие формальных реляционных терминов и их неформальных эквивалентов

| **Формальный реляционный термин** | **Неформальный эквивалент** |
|---|---|
| Отношение | Таблица |
| Кортеж | Строка |
| Кардинальное число | Количество строк |
| Атрибут | Столбец |
| Степень | Количество столбцов |
| Первичный ключ | Уникальный идентификатор |
| Домен | Совокупность допустимых значений |

- Вся информация в реляционных базах данных представляется значениями в *таблицах (table)*. В реляционных системах таблицы состоят из горизонтальных *строк (row)* и вертикальных *столбцов (column)*. Иногда можно встретить такие понятия, как *отношение (relation), кортеж (tuple)* и *(attribute)*. Это соответственно синонимы понятий таблица, строка и столбец.
  *Доменом* называется набор значений элементов данных одного типа, отвечающий поставленным условиям. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных, который забраковывает недопустимые значения.

# Основные достоинства реляционной модели

1) Наличие небольшого набора абстракций, которые позволяют моделировать предметную область и допускают точные формальные определения.
2) Наличие простого и достаточно мощного математического аппарата, опирающего на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных.
3) Возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

- **Фундаментальные свойства отношений**
  - нет одинаковых кортежей
  - кортежи не упорядочены
  - атрибуты не упорядочены
  - все значения атрибутов атомарные

# Обеспечение целостности данных

- Для пользователей информационной системы недостаточно, чтобы база данных просто отражала объекты реального мира. Важно, чтобы такое отражение было однозначным и непротиворечивым. В этом случае говорят, что база данных удовлетворяет условию целостности (integrity).
- Для того, чтобы гарантировать корректность и взаимную непротиворечивость данных, на базу данных накладываются некоторые ограничения, которые называют ограничениями целостности (data integrity constraints).
- Существует несколько типов ограничений целостности. Требуется, например, чтобы значения в столбце таблицы выбирались только из соответствующего домена. На практике учитывают и более сложные ограничения целостности, например, целостность по ссылкам (referential integrity). Ее суть заключается в том, что внешний ключ не может быть указателем на несуществующую строку в таблице

# Операторы реляционной алгебры

- ***Традиционные операции над множествами***
- Объединением (Union) двух отношений называется отношение, содержащее множество кортежей, принадлежащих либо первому, либо второму отношению.

$$R_1 = \{ r_1 \}, \quad R_2 = \{ r_2 \} \quad R_1 \cup R_2 = \{ r | r \in R_1 \vee r \in R_2 \}$$

- Пересечением (Intersect) отношений называется отношение, которое содержит множество кортежей, принадлежащих одновременно и первому и второму отношению.

$$R_1 \cap R_2 = \{ r | r \in R_1 \wedge r \in R_2 \}$$

- Разностью (Minus) отношений называется отношение, содержащее множество кортежей, принадлежащих $R_1$ и не принадлежащих $R_2$ :

$$R_1 \setminus R_2 = \{ r | r \in R_1 \wedge r \notin R_2 \}$$

- Декартовым произведением (Times) отношения степени n со схемой и отношения степени m со схемой , содержащее кортежи, полученные сцеплением каждого кортежа r отношения с каждым кортежем q отношения

$$R_1 = \{ r \}, \quad R_2 = \{ q \} \qquad R_1 \otimes R_2 = \{ (r, q) | r \in R_1 \wedge q \in R_2 \}$$

# Специальные операции реляционной алгебры

- Операция выбора (Select), заданная на отношении R в виде булевского выражения, определенного на атрибутах отношения R, называется отношение , включающее те кортежи из исходного отношения, для которых истинно условие выбора:

$$R[\alpha(r)] \quad \{r | r \in R \wedge \alpha(r) = "Истина" \}$$

- Операция проектирования (Project) или вертикального выбора называется отношение со схемой, соответствующей набору атрибутов B, содержащему кортежи, полученные из кортежей исходного отношения R путем удаления из них значений, не принадлежащих атрибутам из набора B.
- Операция соединения (Join) возвращает отношение, кортежи которого – это сочетание двух кортежей, имеющих общее значение для одного или нескольких общих атрибутов этих двух отношений.
- Операция деления (Divide) возвращает отношение, содержащее все значения одного атрибута отношения, которые соответствуют (в другом атрибуте) всем значениям во втором отношении.

# Понятия полной и транзитивной функциональной зависимости

- Функциональная зависимость ( functional dependence - FD) - в отношении R атрибут Y функционально зависит от атрибута X в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R. X - > R.Y
- Полная функциональная зависимость - функциональная зависимость R.X -> R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X (точным подмножеством множества X называется любое его подмножество, не совпадающее с X).
- Транзитивная функциональная зависимость - функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости

  R.X -> R.Z и R.Z -> R.Y.

# Нормализация

Вообще говоря, руководство по нормализации – это набор стандартов проектирования данных, называемых нормальными формами (normal form). Общепринятыми считаются пять нормальных форм, хотя их было предложено значительно больше. Создание таблиц в соответствии с этими стандартами называется нормализацией.

Нормальные формы изменяются в порядке от первой до пятой. Каждая последующая форма удовлетворяет требования предыдущей. Если вы следуете первому правилу нормализации, ваши данные будут представлены в первой нормальной форме. Если ваши данные удовлетворяют третьему правилу нормализации, они будут находиться в третьей нормальной форме (а также в первой и второй формах). Выполнение правил нормализации обычно приводит к разделению таблиц на две или больше таблиц с меньшим числом столбцов, выделению отношений первичный ключ - внешний ключ в меньшие таблицы, которые снова могут быть соединены с помощью операции объединения.

Одним из основных результатов разделения таблиц в соответствии с правилами нормализации является уменьшение избыточности данных в таблицах. При этом вас не должно смущать наличие в базе одинаковых столбцов первичных и внешних ключей. Такое преднамеренное дублирование – это не то же самое, что избыточность. Правила нормализации, подобно принципам объектного моделирования, развивались в рамках теории баз данных. Большинство разработчиков баз данных признают, что представление данных в третьей и четвертой нормальных формах полностью удовлетворяет все их потребности.

# Нормализация, нормальные формы

- Нормализация – это набор стандартов проектирования данных, называемых нормальными формами (normal form). Общепринятыми считаются пять нормальных форм. Создание таблиц в соответствии с этими стандартами называется нормализацией.
  Нормальные формы изменяются в порядке от первой до пятой.
  Каждая последующая форма удовлетворяет требования предыдущей.
- Выполнение правил нормализации обычно приводит к разделению таблиц на две или больше таблиц с меньшим числом столбцов, выделению отношений первичный ключ - внешний ключ в меньшие таблицы, которые снова могут быть соединены с помощью операции объединения.
  Одним из основных результатов разделения таблиц в соответствии с правилами нормализации является уменьшение избыточности данных в таблицах. Правила нормализации, подобно принципам объектного моделирования, развивались в рамках теории баз данных.
  Большинство разработчиков баз данных признают, что представление данных в третьей нормальной форме полностью удовлетворяет все их потребности.

- Первая нормальная форма (1NF) требует, чтобы на любом пересечении строки и столбца находилось единственное значение, которое должно быть атомарным. Кроме того, в таблице, удовлетворяющей первой нормальной форме, не должно быть повторяющихся групп.
- Вторая нормальная форма (2NF) - отношение R находится во второй нормальной форме в том и только в том случае, когда находится в первой нормальной форме (1NF) и каждый неключевой атрибут полностью зависит от первичного ключа.
- Второе правило нормализации требует, чтобы любой неключевой атрибут зависел от всего первичного ключа. Следовательно, таблица не должна содержать неключевых атрибутов, зависящих только от части составного первичного ключа.
- Третья нормальная форма (3NF)— отношение R находится в третьей нормальной форме в том и только в том случае, если находится во второй нормальной форме (2NF) и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.
- Третья нормальная форма повышает требования второй нормальной формы: она не ограничивается составными первичными ключами. Третья нормальная форма требует, чтобы ни один неключевой атрибут не зависел от другого неключевого атрибута. Любой неключевой атрибут должен зависеть только от первичного ключа.

# Четвертая и пятая нормальные формы

Четвертая нормальная форма запрещает независимые отношения типа один-ко-многим между ключевыми и неключевыми столбцами.

Пятая нормальная форма доводит весь процесс нормализации до логического конца, разбивая таблицы на минимально возможные части для устранения в них всей избыточности данных. Преимуществом преобразования базы данных в пятую нормальную форму является возможность управления целостностью. Изменение значения единственного ключа уже является очень серьезной проблемой. Вы должны найти все вхождения этого значения в вашей базе данных и внести соответствующие изменения.

- В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

• первая нормальная форма (1NF);
• вторая нормальная форма (2NF);
• третья нормальная форма (3NF);
• нормальная форма Бойса — Кодда (BCNF);
• четвертая нормальная форма (4NF);
• пятая нормальная форма, или нормальная форма проекции — соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

• каждая следующая нормальная форма улучшает свойства предыдущей;
• при переходе к следующей нормальной форме свойства предыдущих сохраняется.

# Проектирование баз данных

При проектировании базы данных решаются две основные проблемы:

Отображение объектов предметной области в абстрактные объекты модели данных таким образом, чтобы это отображение не противоречило семантике предметной области и было по возможности лучшим (эффективным, удобным и т.д.). Часто эту проблему называют проблемой логического проектирования баз данных.

Обеспечение эффективного выполнения запросов к базе данных, т.е. рациональное расположение данных во внешней памяти, создание полезных дополнительных структур (например, индексов) с учетом особенностей конкретной СУБД. Эту проблему называют проблемой физического проектирования баз данных.

# Семантические модели данных

Потребности проектировщиков баз данных в удобных и мощных средствах моделирования предметной области породили направление семантических моделей данных.

Наиболее часто на практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной  схеме.

Менее часто реализуется автоматизированная компиляция концептуальной схемы в реляционную.

Наконец, третья возможность - это непосредственная работа с базой данных в семантической модели, т.е. СУБД, основанные на семантических моделях данных.

# ER - модель (Entity-Relationship, Сущность-Связи)

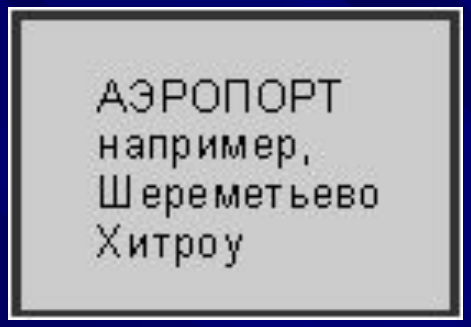
На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных. Модель была предложена Ченом (Chen) в 1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных.

Основные понятия и определения

Основными понятиями ER-модели являются сущность, связь и атрибут.

**Сущность** - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа.

Ниже изображена сущность АЭРОПОРТ с объектами Шереметьево и Хитроу:




- **Связь** - это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).
- Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой. При этом в месте "стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много (many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.

# Связи делятся на три типа по множественности:

**Один-к-одному (1:1)** экземпляр одной сущности связан с одним экземпляром другой сущности.

**Один-ко-многим (1:М**) экземпляр одной сущности связан с несколькими экземплярами другой сущности.

**Многие-ко-многим (М:М)** экземпляр одной сущности связан с несколькими экземплярами другой сущности и наоборот, любой экземпляр второй сущности связан с несколькими экземплярами первой сущности.

Как и сущность, связь - это типовое понятие, все экземпляры обеих пар связываемых сущностей подчиняются правилам связывания.

В изображенном ниже примере связь между сущностями БИЛЕТ и ПАССАЖИР связывает билеты и пассажиров. При том конец сущности с именем "для" позволяет связывать с одним пассажиром более одного билета, причем каждый билет должен быть связан с каким-либо пассажиром. Конец сущности с именем "имеет" означает, что каждый билет может принадлежать только одному пассажиру, причем пассажир не обязан иметь хотя бы один билет.

Лаконичной устной трактовкой изображенной диаграммы является следующая:
Каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;
Каждый ПАССАЖИР может иметь один или более БИЛЕТОВ.
На следующем примере изображена рекурсивная связь, связывающая сущность ЧЕЛОВЕК с ней же самой. Конец связи с именем "сын" определяет тот факт, что у одного отца может быть более чем один сын. Конец связи с именем "отец" означает, что не у каждого человека могут быть сыновья.
Лаконичной устной трактовкой изображенной диаграммы является следующая:

Каждый ЧЕЛОВЕК является сыном одного и только одного ЧЕЛОВЕКА;
Каждый ЧЕЛОВЕК может являться отцом для одного или более ЛЮДЕЙ.

# Язык SQL, его структура, стандарты, история развития.

Доступ к данным осуществляется в виде запросов, которые формулируются на стандартном языке запросов. Сегодня для большинства СУБД таким языком является SQL.

Появление и развития этого языка как средства описания доступа к базе данных связано с созданием теории реляционных баз данных. Прообраз языка SQL возник в 1970 году в рамках научно-исследовательского проекта System/R (IBM). Ныне SQL — это стандарт интерфейса с реляционными СУБД.

SQL не является языком программирования в традиционном представлении.

На нем пишутся не программы, а запросы к базе данных. Поэтому SQL — декларативный или непроцедурный язык. Это означает, что с его помощью можно сформулировать, что необходимо получить, но нельзя указать, как это следует сделать.

Первый международный стандарт языка SQL был принят в 1989 г. (SQL/89 или SQL1), в 1992 г. был принят стандарт языка SQL (SQL/92 или SQL2). В 1999 г. появился стандарт SQL3. В SQL3 введены новые типы данных, при этом предоставляется возможность задания сложных структурированных типов данных, которые в большей степени соответствуют объектной ориентации. Появились стандарты на события и триггеры, которые раньше не затрагивались в стандартах.

# Язык SQL делится на подмножества.

1) **Язык определения данных** (DDL - Data Definition Language) предоставляет пользователям средства указания типа данных и их структуры, а также средства задания ограничений для информации, хранимой в базе данных.
   Операторы – CREATE, ALTER, DROP.
2) **Язык манипулирования данными** (DML - Data Manipulation Language) позволяет вставлять, обновлять и извлекать информацию из базы данных.
   Операторы – SELECT, INSERT, DELETE, UPDATE.
3) **Язык управления данными** (DCL - Data Control Language) состоит из управляющих операторов.
   Операторы – GRANT, REVOKE.
4) **Язык управления транзакциями**.
   Операторы – COMMIT, ROLLBACK, SAVEPOINT.

Запрос на языке SQL состоит из одного или нескольких операторов, следующих один за другим и разделенных точкой с запятой.

Каждый столбец в любой таблице хранит данные определенных типов.

Различают базовые типы данных:

строки символов фиксированной длины - CHAR(n),

целые и вещественные числа - NUMERIC(n,m), DEC(n,m), INTEGER, SMALLINT, FLOAT(n), REAL, DOUBLE PRECISION ,

и дополнительные типы данных — строки символов переменной длины - VARCHAR(n), BIT VARYING(n),

денежные единицы, дату и время – DATE, TIMESTAMP, INTERVAL,

логические данные – BOOLEAN (два значения — "ИСТИНА" и "ЛОЖЬ").

# Пример базы данных для иллюстрации операторов языка SQL

*Таблица 1* **Salespeople (Продавцы)**

| SNUM | SNAME | CITY | COMM |
|---|---|---|---|
| 1001 | Peel | London | .12 |
| 1002 | Serres | San Jose | .13 |
| 1004 | Motika | London | .11 |
| 1007 | Rifkin | Barcelona | .15 |
| 1003 | Axelrod | New York | .10 |

snum уникальный номер продавца
sname имя продавца.
city расположение продавца ( город )
comm комиссионные, которые получает продавец от реализации заказа

***Таблица 2.*** **Customers (Покупатели)**

| CNUM | CNAME | CITY | RATING | SNUM |
|---|---|---|---|---|
| 2001 | Hoffman | London | 100 | 1001 |
| 2002 | Giovanni | Rome | 200 | 1003 |
| 2003 | Liu | San Jose | 200 | 1002 |
| 2004 | Grass | Berlin | 300 | 1002 |
| 2006 | Clemens | London | 100 | 1001 |
| 2008 | Cisneros | San Jose | 300 | 1007 |
| 2007 | Pereira | Rome | 100 | 1004 |

cnum уникальный номер покупателя
cname имя покупателя
city расположение покупателя ( город ).

rating код, указывающий уровень предпочтения данного покупателя перед другими ( рейтинг ).

***Таблица 3.*** **Orders (Заказы)**

| ONUM | AMT | ODATE | CNUM | SNUM |
|---|---|---|---|---|
| 3001 | 18.69 | 10/03/1990 | 2008 | 1007 |
| 3003 | 767.19 | 10/03/1990 | 2001 | 1001 |
| 3002 | 1900.10 | 10/03/1990 | 2007 | 1004 |
| 3005 | 5160.45 | 10/03/1990 | 2003 | 1002 |
| 3006 | 1098.16 | 10/03/1990 | 2008 | 1007 |
| 3009 | 1713.23 | 10/04/1990 | 2002 | 1003 |
| 3007 | 75.75 | 10/04/1990 | 2004 | 1002 |
| 3008 | 4723.00 | 10/05/1990 | 2006 | 1001 |
| 3010 | 1309.95 | 10/06/1990 | 2004 | 1002 |
| 3011 | 9891.88 | 10/06/1990 | 2006 | 1001 |

onum уникальный номер заказа
amt сумма заказа
odate дата выполнения заказа
cnum номер покупателя (из таблицы Заказчиков).
snum номер продавца (из таблицы Продавцов).

```
CREATE TABLE salespeople
        (snum        integer  NOT NULL PRIMARY KEY,
         sname       char(10)  NOT NULL,
         city        char(10),
         comm.       Decimal);
CREATE TABLE customers
        (cnum        integer  NOT NULL PRIMARY KEY,
         cname       char(10)  NOT NULL,
         city        char(10),
         rating      integer,
         snum        integer,
         FOREIGN KEY (snum) REFERENCES salespeople,
         UNIQUE (cnum,snum) );
CREATE TABLE orders
        (onum        integer  NOT NULL PRIMARY KEY,
         amt         decimal,
         odate       date      NOT NULL,
         cnum        integer NOT NULL,
         snum        integer NOT NULL,
         FOREIGN KEY (cnum,snum) REFERENCES
         customers (cnum,snum) );
```

# Последовательности

В реляционных или объектно-реляционных базах данных часто возникает необходимость в генерации целочисленных столбцов, где каждая строка имеет свое уникальное значение, как правило, используемое в качестве первичного ключа. Генераторы последовательностей позволяют избежать ненужного дискового ввода-вывода за счет помещения генерируемых чисел в специальный кэш-буфер, находящийся в оперативной памяти сервера. В прошлом, до появления генераторов последовательностей, разработчикам приложений приходилось создавать для хранения все возрастающих значений первичных ключей специальную таблицу, состоящую из единственной колонки. Подобное решение приводило к постоянной конкуренции пользователей за доступ к этой таблице, а также требовало значительных затрат ресурсов сервера для обеспечения блокировки доступа к ней в момент, пока одна из пользовательских сессий читала очередное значение ключа. Отметим, что в отличие от таблиц или представлений данных, приложения не могут осуществлять поиск определенных чисел непосредственно в сгенерированной последовательности.

Последовательность создается с помощью команды DDL CREATE SEQUENCE. После того, как последовательность создана, к ней можно обратиться:
*последовательность.*CURRVAL
или
*последовательность.*NEXTVAL,
где *последовательность* - это имя последовательности. CURRVAL (текущее значение) и NEXTVAL (следующее значение) – это псевдостолбцы, применяющиеся в последовательностях. CURRVAL возвращает текущее значение последовательности, а NEXTVAL увеличивает ее и возвращает новое значение. Как CURRVAL, так и NEXTVAL возвращают значения, имеющие тип NUMBER. Значения последовательностей могут быть использованы в списках выбора запросов, в списках значений (VALUES) операторов INSERT и в командах SET операторов UPDATE.

# Подмножество языка DML: операторы SELECT, INSERT, UPDATE, DELETE

Язык обработки данных DML позволяет добавлять (insert), изменять (update), удалять (delete) и выбирать (select) информацию в базе данных, т.е.  предназначен для работы с информационным содержанием базы данных. Операторы языка DML представляют собой SQL-команды, позволяющие изменять и дополнять хранящуюся в базе данных информацию.

# Оператор выбора SELECT

Синтаксис оператора имеет следующий вид:

SELECT [ ALL ! DISTINCT ] <список полей> ! *
[ FROM <список таблиц>
[ WHERE <условие выборки>
[ GROUP BY <список полей для группы>
[ HAVING <условие выборки для группы>
[ ORDER BY < список полей, по которым упорядочить вывод>

Используются:

- реляционные операторы =, < , > , <= , > = , < > ;
- булевые операторы AND, OR, NOT;
- IN (‘a1’, ‘a2’, …) – множество объектов;
- BETWEEN ‘a1’ AND ‘a2’ – задает границы параметров;
- LIKE – только для символьных полей задает шаблон ( символы % и _ )
  LIKE ‘G%’ LIKE ‘L% n % d_n’
- IS NULL, IS NOT NULL
- Агрегатные функции: COUNT, SUM, AVG, MAX, MIN
- строки и выражения:
  SELECT snum, sname, city, ‘%’, comm. * 100 FROM salespeople;

# Примеры:

1) Вывести все данные из таблицы Salespeople

```
SELECT snum, sname, city, comm
    FROM  Salespeople;
```

или

```
SELECT   *
    FROM  Salespeople;
```

2) Вывести имена всех продавцов из Лондона

```
SELECT  sname, city
    FROM  Salespeople
        WHERE city='London';
```

3) Вывести  все данные о покупателях в San Jose, которые имеют рейтинг ниже 200

```
SELECT  *
FROM Customers
    WHERE city = ' San Jose' AND rating < 200;
```

4) Вывести сумму и среднюю величину всех заказов

```
SELECT   SUM(amt), AVG(amt)
FROM orders;
```

5) Вывести максимальный заказ для каждого продавца

```
SELECT snum, MAX(amt)
FROM orders
GROUP BY  snum;
```

6) Вывести имена покупателей и обслуживающих их продавцов

```
SELECT customers.cname, salespeople.sname
FROM  customers, salespeople
WHERE  salespeople.snum=customers.snum;
```

7) Соединение двух копий одной таблицы (используем временные имена)

```
SELECT first.cname, second.cname, first.rating
FROM  customers first, customers second
WHERE  first.rating = second.rating;
```

8) Вложенные запросы

```
SELECT  *
FROM orders
WHERE snum=( SELECT snum
                FROM  salespeople
WHERE sname='Motika')
```

# Оператор ввода новых строк INSERT

Синтаксис оператора имеет следующий вид:

INSERT INTO <имя таблицы>
VALUES (<значение>,<значение>…)

1) Ввести строку в таблицу salespeople

```
INSERT INTO salespeople
    VALUES (1001, 'Peel', 'London', .12);
```

2) Извлечение строк из одной таблицы и вставка их в другую

```
INSERT INTO londonstaff
    SELECT *
        FROM salespeople
            WHERE city='London';
```

# Оператор удаления строк DELETE

1) Удаление всех строк в таблице

```
DELETE  FROM salespeople;
```

2) Удаление определенной строки в таблице

```
DELETE  FROM salespeople
        WHERE snum=1003;
```

3) Удаление группы  строк в таблице

```
DELETE  FROM salespeople
        WHERE city='London';
```

# Оператор изменения значений полей UPDATE

1) Изменение значения поля в определенной строке таблицы

```
UPDATE customers
    SET  rating = 200
        WHERE   snum = 1001;
```

2) Изменение значения поля во всех строках таблицы

```
UPDATE customers
    SET  rating = 200;
```

3) Удвоение размера комиссионных у продавцов из Лондона

```
UPDATE salespeople
    SET  comm.= comm.*2
        WHERE city=’London’;
```

# Подмножество языка DDL: операторы CREATE, ALTER, DROP

Язык определения данных DDL представляет собой подмножество команд языка SQL, предназначенное для создания и определения объектов базы данных. Определения объектов, созданные с помощью команд DDL, сохраняются в словаре базы данных.

Язык определения данных DDL обеспечивает:

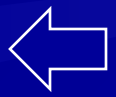
· Создание объектов базы данных (CREATE)
· Удаление объектов базы данных (DROP)
· Изменение свойств объектов базы данных (ALTER)

Современные базы данных содержат различные типы объектов:

таблицы, индексы, представления, роли, синонимы, последовательности, кластеры, триггеры, процедуры, функции, пакеты, пользователи, профили и т.д.

Все эти объекты создаются, изменяются и удаляются с помощью операторов DDL.

| Команда SQL | Назначение |
|---|---|
| alter procedure | Повторная компиляция хранимой процедуры |
| alter table | Добавление в таблицу нового столбца, переопределение существующего столбца, модификация отводимой области хранения на диске |
| analyze | Сбор статистики по производительности операций, выполняемых с конкретными объектами базы данных; собранные данные затем обрабатываются оптимизатором издержек |
| alter table add constraint | Добавление в таблицу условия целостности данных |
| create table | Создание таблицы |
| create index | Создание индекса |
| drop index | Удаление индекса |
| drop table | Удаление таблицы из базы данных |



# Оператор CREATE TABLE

- CREATE TABLE <имя таблицы>
- (<имя столбца> <тип> [<размер> [<ограничения> ]],
- <имя столбца> <тип> [<размер> [<ограничения> ]],…);
-
- Ограничения в таблицах:
- NOT NULL – значение столбца должно быть определено;
- UNIQUE – значения столбца являются уникальными;
- PRIMARY KEY – столбец является первичным ключом;
- CHECK – определяет условие, которому должны удовлетворять значения столбца;
- DEFAULT- присвоение значений «по умолчанию»

# Поддержка ссылочной целостности данных

FOREIGN KEY <список столбцов> REFERENCES < имя таблицы> [<список столбцов>]
FOREIGN KEY (snum) REFERENCES salespeople (snum)

Для изменения или исключения значений родительского ключа, на который имеется ссылка, есть следующие возможности:

- можно запретить изменение родительского ключа (NO ACTION);
- можно изменить родительский ключ и выполнить каскадные изменения (CASCADE);
- можно изменить родительский ключ и установить значения внешнего ключа в NULL (SET NULL).
- можно установить значение по умолчанию (SET DEFAULT)

[ON UPDATE CASCADE | SET NULL | SET DEFAULT | NO ACTION];
[ON DELETE CASCADE | SET NULL | SET DEFAULT | NO ACTION];

Параметры *ON UPDATE* и *ON DELETE* указываются при необходимости осуществлять каскадные действия при, соответственно, изменении или удалении значений атрибутов первичного ключа главной таблицы. При попытке изменения значения атрибута, входящего в состав первичного ключа, или удаления строки, на которую ссылаются строки из подчиненных таблиц, СУБД принимает решение либо о выполнении одного из каскадных действий, либо о запрещении SQL-оператора, изменяющего или удаляющего строку в главной таблице.

Если параметры ON UPDATE или ON DELETE равны NO ACTION или они не указаны совсем, СУБД запретит выполнение SQL-оператора, нарушающего ссылочную целостность.

Если после параметров ON UPDATE или ON DELETE стоит значение CASCADE, то произойдет каскадное изменение или удаление строк в подчиненных таблицах. Иными словами, при удалении строки из главной таблицы, будут удалены все строки в подчиненных таблицах, которые ссылались на удаляемую строку. При изменении значения первичного ключа главной таблицы и наличии параметра ON UPDATE со значением CASCADE все значения внешних ключей, ссылающихся на изменяемый атрибут, также изменят свое значение.

- Использовать значение CASCADE параметров ON UPDATE и ON DELETE следует с особой осторожностью, поскольку одна ошибочная команда, например, удаления строки в одной из таблиц может повлечь за собой необратимые последствия, которые могут привести к полной неработоспособности приложений базы данных. Если все-таки необходимо выполнять каскадное изменение или удаление данных в подчиненных таблицах, рекомендуется, по возможности, реализовывать это посредством хранимых процедур.
- Указание значения SET NULL для параметров ON UPDATE или ON DELETE приведет к тому, что значения атрибутов внешних ключей, ссылающихся на изменяемое значение первичного ключа, установится в NULL.
- Указание значения SET DEFAULT для параметров ON UPDATE или ON DELETE приведет к тому, что значения атрибутов внешних ключей, ссылающихся на изменяемое значение первичного ключа, примут значение по умолчанию, заданное при создании таблицы.

```
CREATE   TABLE   customers
    (cnum        integer  NOT NULL PRIMARY KEY,
     cname       char(10)  NOT NULL,
     city        char(10),
     rating      integer,
     snum        integer  REFERENCES salespeople,
     ON UPDATE of salespeople CASCADE,
     ON DELETE of salespeople NO ACTION);
CREATE   TABLE   orders
    (onum        integer  NOT NULL PRIMARY KEY,
     amt         decimal,
     odate       date     NOT NULL,
     cnum        integer NOT NULL REFERENCES customers,
     snum        integer  REFERENCES salespeople,
     ON UPDATE of customers CASCADE,
     ON DELETE of customers CASCADE,
     ON UPDATE of salespeople CASCADE,
     ON DELETE of salespeople SET NULL);
```

# Представления, их значение

Представление – объект базы данных, позволяющий получить определенную пользователем выборку данных из одной или нескольких таблиц. В отличие от таблицы, представление не содержит никаких данных, а лишь запрос на языке SQL, дающий возможность прочитать из базы данных необходимую информацию и представить ее в табличной форме.

Пользователь может не знать, работает он с представлением или с настоящей таблицей. Подобно таблицам, к представлениям можно применять операторы insert, update, delete и select.

Важно понимать принципы использования представлений. Их применение может оказаться необходимым в силу следующих причин:

1) Представления обеспечивают дополнительный уровень безопасности базы данных. Например, можно создать общую таблицу со сведениями обо всех сотрудниках компании, но разрешить менеджерам компании получать информацию только об их подчиненных.
2) Представления позволяют скрыть от пользователей сложность структуры хранимых данных.
3) Представления позволяют использовать разумные названия отдельных столбцов.
4) Представления обеспечивают гибкость при изменении формата одной или нескольких входящих в них таблиц.

# Примеры представлений

```
1)      CREATE VIEW londonstaff
            AS  SELECT  *
                FROM  salespeople
                    WHERE city = 'London';
2)      CREATE  VIEW  salesown
            AS  SELECT  snum, sname, city
                FROM  salespeople;
3)      CREATE   VIEW   Totalforday
            AS SELECT odate, COUNT(DISTINCT cnum),
                COUNT(DISTINCT snum), COUNT(onum),
AVG(amt), SUM(amt)
                FROM orders
                    GROUP BY odate;
```

# Обновляемые представления

Если к представлению можно применить операторы обновления, то представление является обновляемым (updateble), иначе оно является читаемым (read-only).

Приведем критерии того, является ли представление обновляемым в SQL:

- оно базируется на одной таблице;
- оно должно включать первичный ключ таблицы;
- оно не должно включать полей, полученных в результате применения функций агрегирования;
- оно не может содержать спецификации DISTINCT;
- оно не должно использовать GROUP BY  или HAVING;
- оно не должно использовать подзапросы;
- оно может быть определено на другом представлении, но это представление должно быть обновляемым;
- оно не может содержать константы, строки или выражения в списке выбираемых выходных полей;
- для INSERT  оно должно включать поля из таблицы, которые имеют ограничения NOT NULL.

# Пример обновляемого представления:

```
CREATE VIEW Highratings
        AS  SELECT  cnum,rating
              FROM  customers
                  WHERE rating = 300;
```

В это представление вводим новую строку:

```
            INSERT INTO Highratings
                VALUE  (2018, 200);
```

Это правильная команда для данного представления, строка через представление будет вставлена в таблицу customers, но не попадет в представление Highratings.

Эту проблему можно решить, указав WITH CHECK OPTION в определении представления.

# Объектные и системные привилегии

До сих пор предполагалось, что каждый пользователь базы данных может обращаться к объектам всех других пользователей базы данных и к информации, содержащейся в этих объектах. В действительности дело обстоит далеко не так. Сервер баз данных предоставляет администратору полный и систематический контроль над тем, что именно каждый отдельный пользователь может читать, модифицировать, стирать или изменять. В сочетании с использованием представлений данных можно полностью контролировать доступ к информации всех пользователей.

Права доступа используются для того, чтобы позволить одному пользователю работать с данными другого пользователя. После получения необходимых полномочий обладатель прав доступа может работать с объектами, принадлежащими другому пользователю.

Существуют привилегии двух различных видов:
объектные и системные.

*Объектная привилегия* (object privilege) разрешает выполнение определенной операции над конкретным объектом (например, над таблицей - SELECT, DELETE, INSERT, UPDATE, REFERENCES).

*Системная привилегия* (system privilege) разрешает выполнение операций над целым классом объектов.

# Операторы GRANT, REVOKE

Предоставление пользователям необходимых полномочий и лишение полномочий осуществляется с помощью операторов DCL: GRANT и REVOKE.
Оператор GRANT используется для открытия другой схеме доступа к привилегии, а оператор REVOKE - для запрещения доступа, разрешенного оператором GRANT. Оба оператора могут использоваться как для объектных, так и для системных привилегий.

Для объектных привилегий синтаксис оператора GRANT таков:
GRANT *привилегия* ON *объект* TO *обладатель_привилегий* [WITH GRANT OPTION];
где *привилегия* - это нужная привилегия,
*объект* - это объект, к которому разрешается доступ, а
*обладатель_привилегий* - пользователь, получающий привилегию.

Для системных привилегий синтаксис оператора GRANT таков:
GRANT *привилегия* TO *обладатель_привилегий* [WITH ADMIN OPTION];
где *привилегия* - это предоставляемая системная привилегия,
*обладатель_привилегий* - пользователь, получающий привилегию.

Для объектных привилегий синтаксис оператора REVOKE таков:
REVOKE *привилегия* ON *объект* FROM *обладатель_привилегий* [CASCADE CONSTRAINTS];
где *привилегия* - это отменяемая привилегия,
*объект* - это объект, на который предоставлена привилегия,
*обладатель_привилегий* - пользователь, получающий эту привилегию.

Для системных привилегий синтаксис оператора REVOKE таков:
REVOKE *привилегия* FROM *обладатель_привилегий;*
где *привилегия* - это отменяемая системная привилегия,
*обладатель_привилегий* - пользователь, который более не будет ее иметь.

Примеры операторов GRANT, REVOKE

1) GRANT select ON customers TO Adrian;
2) GRANT select, insert ON orders TO Adrian, Diana;
3) GRANT ALL ON customers TO Adrian;
4) GRANT select ON orders TO PUBLIC;
5) REVOKE insert ON orders FROM Adrian;

# Роли

Для организаций, в которых работает множество пользователей, управление привилегиями является достаточно сложной задачей. Для ее упрощения можно использовать средство, называемое ролями. Роль (role) является совокупностью привилегии, как объектных, так и системных.

Примеры ролей:

1) CREATE ROLE spaceadmin IDENTIFIED BY password
GRANT create session, alter session, restricted session, alter database, create rollback segment, alter rollback segment, drop rollback segment, create tablespace, alter tablespace, drop tablespace TO spaceadmin;

2) CREATE ROLE backupadmin IDENTIFIED BY password
GRANT create session, alter session, restricted session, manage tablespace, backup any table TO backupadmin;

# Транзакции

Транзакция представляет собой последовательность операторов языка SQL, которая рассматривается как некоторое неделимое действие над базой данных. В то же время, это логическая единица работы системы. Транзакция реализует некоторую прикладную функцию, например, перевод денег с одного счета на другой в банковской системе.

Существуют различные модели транзакций, которые могут быть классифицированы на основании различных свойств, включающих структуру транзакции, параллельность внутри транзакции, продолжительность и т.д. Чаще всего имеют в виду традиционные транзакции, характеризуемые четырьмя классическими свойствами: атомарности, согласованности, изолированности, долговечности (прочности) — ACID (Atomicity, Consistency, Isolation, Durability). Иногда традиционные транзакции называют ACID-транзакциями. Упомянутые выше свойства означают следующее.

1. **Атомарность**: Свойство атомарности выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе.
2. **Согласованность**: Свойство согласованности гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.
3. **Изолированность**: Свойство изолированности означает, что конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.
4. **Долговечность**: Свойство долговечности трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Таким образом, возможны два варианта завершения транзакции. Если все операторы выполнены успешно, и в процессе выполнения транзакции не произошло никаких сбоев программного или аппаратного обеспечения, транзакция фиксируется.

Фиксация транзакции — это действие, обеспечивающее запись на диск изменений в базе данных, которые были сделаны в процессе выполнения транзакции. До тех пор, пока транзакция не зафиксирована, возможно аннулирование этих изменений, восстановление базы данных в то состояние, в котором она была на момент начала транзакции. Фиксация означает, что все результаты выполнения транзакции становятся постоянными. Они станут видимыми другим транзакциям только после того, как текущая транзакция будет зафиксирована. До этого момента все данные, затрагиваемые транзакцией, будут "видны" пользователю в состоянии на начало текущей транзакции.

Если в процессе выполнения транзакции случилось нечто такое, что делает невозможным ее нормальное завершение, база данных должна быть возвращена в исходное состояние. Откат транзакции — это действие, обеспечивающее аннулирование всех изменений данных, которые были сделаны операторами SQL в теле текущей незавершенной транзакции.

Каждый оператор в транзакции выполняет свою часть работы, но для успешного завершения всей работы в целом требуется безусловное завершение их всех. Группирование операторов в транзакции сообщает СУБД, что вся эта группа должна быть выполнена как единое целое, причем такое выполнение должно поддерживаться автоматически.

# Операторы управления транзакциями: COMMIT, ROLLBACK, SAVEPOINT

COMMIT - зафиксировать транзакцию
ROLLBACK - отменить изменения в текущей транзакции
SAVEPOINT - создать контрольную точку внутри транзакции
Транзакции могут использоваться только с таблицами, принадлежащими БД.

**Журналы транзакций**

Журналы транзакций - это специальные файлы операционной системы, в которые СУБД записывает все изменения или транзакции, произведенные в базе данных. Поскольку все транзакции полностью сохраняются в журналах повтора, при необходимости с помощью этих журналов сервер базы данных всегда способен восстановить свое состояние на заданный момент времени. Каждая база данных обязательно должна иметь как минимум два оперативных журнала транзакций.

Журналы транзакций работают по циклическому принципу. Пусть в некоторой базе данных есть два журнала: logA и logB. По мере того, как транзакции создают, удаляют и модифицируют информацию в базе данных, все изменения заносятся в logA. Когда logA оказывается целиком заполненным, происходит переключение журналов, и все вновь произведенные транзакции начинают записываться в logB. По заполнении logB происходит новое переключение журналов, и транзакции опять сохраняются в logA.

# Режим ARCHIVELOG: возможность полного восстановления

При работе базы данных в режиме ARCHIVELOG все журналы повтора транзакций сохраняются. Таким образом, перед началом перезаписи оперативного журнала повтора в результате очередного циклического переключения его прежнее содержимое копируется в другой файл, В случае, если база данных не успела завершить подобное копирование, сервер Oracle приостанавливает все операции до его завершения. Следовательно, в режиме ARCHIVELOG сервер Oracle не позволит переписать старый журнал транзакций, пока не будет сделана его архивная копия. Таким образом, база данных хранит информацию обо всех произведенных транзакциях, что позволяет выполнить ее восстановление после любых типов сбоев, включая ошибочные действия пользователей или выход из строя жесткого диска. Это самый безопасный режим функционирования базы данных.

При работе базы данных в режиме ARCHIVELOG содержимое всех журналов транзакций сохраняется перед началом их перезаписи. Это обеспечивает широкие возможности по восстановлению базы данных, в том числе и восстановление ее состояния на заданный момент времени.

# Режим NOARCHIVELOG

При работе базы данных в режиме NOARCHIVELOG (именно этот режим устанавливается по умолчанию) сохранение старых журналов транзакций не производится. Поскольку в данном случае имеется информация лишь о последних по времени транзакциях, восстановление базы данных возможно только после некоторых типов сбоев, например после внезапного отключения питания. Необходимо еще раз подчеркнуть, что после заполнения оперативного журнала транзакций происходит переключение на второй журнал, в результате чего записанная в нем информация о старых транзакциях утрачивается.
Когда база данных работает в режиме NOARCHIVELOG, сохранение архивных копий журналов транзакций перед их перезаписью НЕ ПРОИЗВОДИТСЯ. Таким образом, в режиме NOARCHIVELOG имеются лишь ограниченные возможности по восстановлению после сбоев, и этот режим предназначен в первую очередь для защиты от разовых сбоев.

# SQL* Plus: резюме

SQL*Plus представляет собой вариант языка SQL, разработанный корпорацией Oracle, где слово "Plus" обозначает предложенные Oracle расширения SQL.

Любые операции с реляционными базами данных осуществляются исключительно с помощью SQL-подобных языков программирования.

SQL* Plus обладает дружественным пользователю интерфейсом.

При программировании на SQL*Plus пользователь имеет дело с наборами данных как с единым целым (другими словами, вам не придется контролировать обработку одной записи вслед за другой).

Реализованные в SQL*Plus расширения SQL значительно облегчают генерацию насыщенных информацией отчетов.

# Этапы проектирования баз данных

**Концептуальное проектирование базы данных**

Этап 1. Создание локальной концептуальной модели данных исходя из представлений о предметной области каждого из типов пользователей

Этап 1.1. Определение типов сущностей

Этап 1.2. Определение типов связей

Этап 1.3. Определение атрибутов и связывание их с типами сущностей и связей

Этап 1.4. Определение доменов атрибутов

Этап 1.5. Определение атрибутов, являющихся первичными ключами.

Этап 1.6. Создание диаграммы "сущность-связь".

Этап 1.7. Обсуждение локальных концептуальных моделей данных с конечными пользователями.

# Логическое проектирование базы данных (для реляционной модели)

Этап 2. Построение и проверка локальной логической модели данных на основе представления о предметной области каждого из типов пользователей.

Этап 2.1. Преобразование локальной логической модели данных на основе представления о предметной области каждого из типов пользователей

Этап 2.2. Определение набора отношений исходя из структуры локальной логической модели данных.

Этап 2.3. Проверка модели с помощью правил нормализации.

Этап 2.4. Проверка модели в отношении транзакций пользователей.

Этап 2.5. Создание диаграмм "сущность-связь".

Этап 2.6. Определение требований поддержки целостности данных.

Этап 2.7. Обсуждение разработанных локальных логических моделей данных с конечными пользователями.

Этап 3. **Создание и проверка глобальной логической модели данных.**
Этап 3.1. Слияние локальных логических моделей данных в единую глобальную модель данных.
Этап 3.2. Проверка глобальной логической модели данных.
Этап 3.3. Проверка возможностей расширения модели в будущем.
Этап 3.4. Создание окончательного варианта диаграммы "сущность-связь".
Этап 3.5. Обсуждение глобальной логической модели данных с пользователями.
**Физическое проектирование базы данных (с использованием реляционной СУБД)**
Этап 4. **Перенос глобальной логической модели данных в среду СУБД.**
Этап 4.1. Проектирование основных таблиц в среде СУБД.
Этап 4.2. Реализация бизнес-правил предприятия в среде СУБД.
Этап 5. **Проектирование физического представления базы данных.**
Этап 5.1. Анализ транзакций
Этап 5.2. Выбор файловой структуры.
Этап 5.3. Определение вторичных индексов.
Этап 5.4. Анализ необходимости введения контролируемой избыточности данных.
Этап 5.5. Определение требований к дисковой памяти.
Этап 6. **Разработка механизмов защиты**.
Этап 6.1. Разработка пользовательских представлений (видов).
Этап 6.2. Определение прав доступа.

Этап 7. **Организация мониторинга и настройка функционирования системы**