1. Основные понятия – база данных, информационная система, СУБД. Принципы организации информационных систем.

*База данных* – совокупность структурированных, взаимосвязанных, динамически обновляемых данных некоторой предметной области.

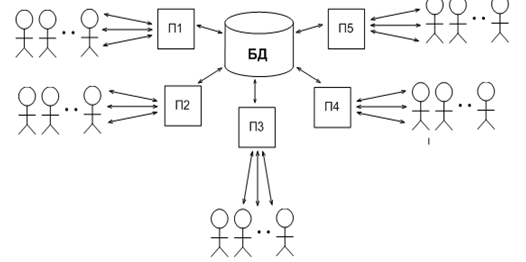
База данных (БД) вместе с поддерживающим ее программным обеспечением (ПО) образует *информационную систему* (ИС)*.* Коротко это можно записать в виде простой формулы БД + ПО= ИС.

Некоторые авторы понимают информационную систему более широко, включая в это понятие аппаратную составляющую и обслуживающий персонал, но для такой трактовки есть более подходящий термин – банк данных. Иногда встречается и более узкая трактовка информационной системы как совокупности данных и программных средств для решения конкретной прикладной задачи, например, задачи бухгалтерского или складского учета.

*СУБД* – сложный комплекс программных и языковых средств, предназначенных для создания и поддержки баз данных.

1. *Принцип интегрированности*

Принцип состоит в том, что единая интегрированная БД для всей предметной области совместно используется персоналом.



В интегрированной системе может быть достигнута *минимальная избыточность* (отсутствие дублирования) данных. Этот принцип обычно формулируется так: «Каждый факт - в одном месте».

В интегрированной системе легче добиться непротиворечивости (целостности) данных, т.к. ввиду отсутствия дублирования данных нет и их нестыковок. Имеется возможность контролировать целостность данных встроенными средствами СУБД, более подробная информация об этом содержится в разделе 2.1.

В интегрированной системе удобнее выполнять поиск данных, можно выполнять любые виды обработки и анализа данных.

Для интегрированной ИС проще решается проблема резервного копирования данных и восстановления поврежденных данных, так как эту задачу можно возложить на одного человека (АБД), который будет нести персональную ответственность за сохранность всех корпоративных данных.

2. *Принцип независимости прикладного программного обеспечения от способа организации данных.*

Между данными и прикладным программным обеспечением ИС находится, как минимум, два слоя базового программного обеспечения – операционная система и СУБД, которые берут на себя все низкоуровневые функции управления данными. Поэтому можно говорить об относительной независимости БД и ПрПО.

Различают следующие уровни независимости:а) логическая независимость – можно вносить некоторые изменения в структуру уже заполненной базы данных без коренной переделки прикладного программного обеспечения, например, можно добавить новые столбцы в уже заполненную таблицу базы данных, при этом все приложения не потеряют работоспособности, однако, при удалении таблиц или их столбцов некоторые приложения работать не смогут;

б) физическая независимость – можно изменить физический формат хранения данных, например, перейти на новую СУБД, без коренной переделки прикладного программного обеспечения (ПрПО о физическом формате хранения данных вообще ничего «не знает», поскольку работает с данными на логическом уровне).

3. *Принципы масштабируемости и переносимости*

Данные принципы вытекают из принципа независимости. Принцип масштабируемостиследует рассматривать в трех аспектах:

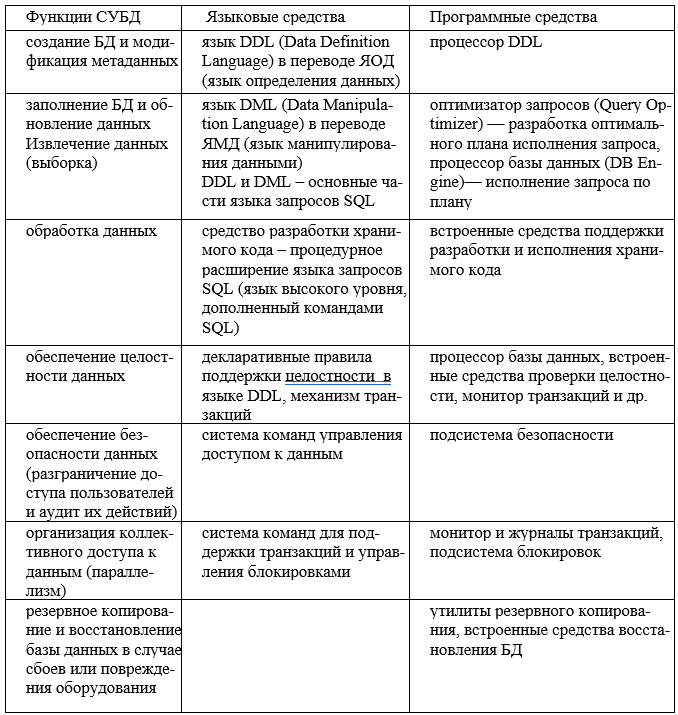
а) возможность неограниченного наращивания размеров БД;

б) неограниченное увеличение количества пользователей;

в) неограниченное увеличение количества приложений.

В определенный момент времени существующее базовое ПО перестанет удовлетворять требованиям эффективного управления данными, поэтому возникнет необходимость перенести данные на новую платформу без потери информации и коренной переделки ПрПО. Это свойство ИС называется *переносимостью.*

1. Функции СУБД. Языковые и программные средства для реализации этих функций.

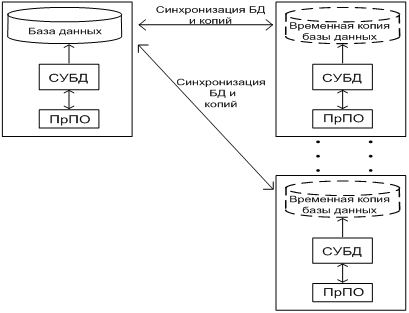


1. Архитектуры информационных систем.

*Архитектура ИС* – это совокупность существенных решений о её организации. Обычно в это понятие входят решения о компонентном составе системы и принципах взаимодействия компонентов.

**Архитектура «файл-сервер»**

В архитектуре файл-сервер *вся обработка данных выполняется на клиентских компьютерах*, сервер служит только в качестве хранилища файлов с данными (файловый сервер). Копии базы данных передаются для обработки на клиентские компьютеры, при этом постоянно выполняется синхронизация основной базы данных, размещённой на файловом сервере, с ее копиями в случае их обновления пользователями на клиентских компьютерах.



*Рисунок 1.4 - Архитектура файл-сервер*

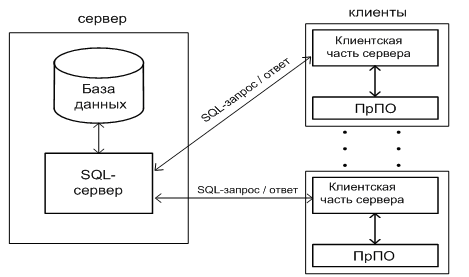
Недостаток архитектуры файл-сервер – большая нагрузка на сеть и клиентские компьютеры, поскольку на всех клиентских компьютерах должна быть установлена копия СУБД, которая выполняет все необходимые функции по обработке данных, при этом все изменения в копиях обязательно передаются по сети в основную базу данных, существенно повышая сетевой трафик.

Преимущество состоит в том, что не требуется мощный сервер. Такую архитектуру можно реализовать даже в одноранговой сети без выделенного сервера, необходимо только выделить один из компьютеров в качестве хранилища общей базы данных.

**Архитектура «клиент-сервер»**

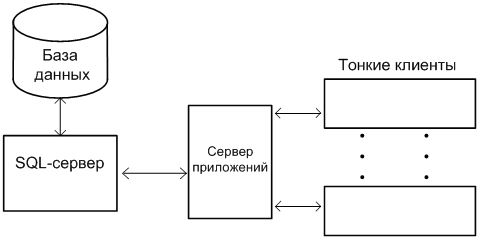
Применительно к информационным системам архитектура «клиент-сервер» интересна и актуальна главным образом потому, что обеспечивает простое и относительно дешевое решение проблемы коллективного (многопользовательского) доступа к базам данных в локальной или глобальной сети.

Информационная система архитектуры «клиент-сервер» разбивается на две части, которые могут выполняться в разных узлах сети, - клиентскую и серверную части. На серверную часть возлагаются функции хранения и значительной части обработки данных, на клиентскую – функции взаимодействия с пользователем и, частично, обработки данных, полученных с сервера (рисунок 1.5).



**Многозвенные архитектуры**

Многозвенные (многоуровневые) архитектуры позволяют обеспечить более оптимальную загрузку технических средств, чем классическая двухзвенная архитектура, и обеспечивают возможность плавного масштабирования информационной системы. Наиболее распространенным случаем является трехзвенная архитектура, в которой в качестве промежуточного слоя программного обеспечения между сервером и клиентом используется *сервер приложений* (рисунок 1.7). Сервер приложений берет на себя существенную часть обработки данных, позволяя разгрузить и серверную, и клиентскую части.

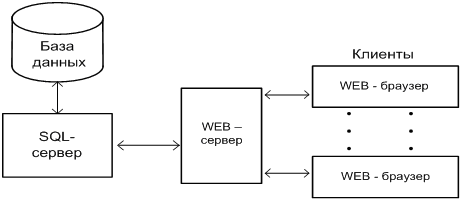


*Рисунок 1.6 – Трехзвенная архитектура*

В трехзвенной архитектуре часто используется так называемый *тонкий клиент (thin client)*, который вообще не выполняет никаких функций обработки данных, а только обеспечивает представление данных и взаимодействие с пользователем. В отличие от этого, клиента двухзвенной системы обычно называют *толстым клиентом (fat client).*

**Информационные системы на основе web-архитектуры**

Достижения в области технологий Internet привели к широкому распространению еще одной разновидности клиент-серверной архитектуры, которая получила название web-архитектуры (рисунок 1.7)



*Рисунок 1.7 – Web-архитектура*

В web-архитектуре обязательным является наличие еще одного дополнительного компонента – web-сервера, на котором устанавливается прикладное программное обеспечение, обеспечивающее всю необходимую функциональность ИС. В качестве такого сервера может использоваться Internet Information Server (IIS) фирмы Microsoft или свободно распространяемый Web-сервер Apache.

На клиентской стороне требуется только браузер для отображения html-страниц, принимаемых со стороны web-сервера, и взаимодействия с пользователем.

В современных ИС между SQL-сервером и Web-сервером часто находится еще одно звено – сервер приложений, который берет на себя большую часть обработки данных и позволяет одновременно разгрузить и SQL-сервер, и Web-сервер. В качестве примера подобной архитектуры можно привести информационные системы на платформе J2EE.

Преимуществом web-архитектуры является удобство администрирования ИС (на клиенте вообще не требуется устанавливать никакого специального программного обеспечения), возможность доступа пользователей к данным как с любого компьютера локальной сети, так и удаленно через Internet.

1. Модели данных.

Модель данных в общем виде можно представить в виде трех составляющих:

* логическая структура данных (в первую очередь, способ организации связей между элементами данных):
* набор допустимых операций по манипулированию данными для принятой логической структуры;
* правила поддержки целостности (непротиворечивости) данных.
* Можно считать понятия «модель данных» и «абстрактный тип данных» близкими (но не полностью идентичными) по смыслу.

К основным моделям данных относят следующие:

* иерархическая (на основе деревьев),
* сетевая (на основе сетевых структур—графов),
* реляционная (на основе таблиц),
* объектно-ориентированная (на основе принципов объектно-ориентированного программирования).
* расширения реляционной модели
  + постреляционная (таблицы с возможностью вложения одних таблиц в другие, т.е. поддерживается тип данных «таблица»),
  + объектно-реляционная (поддерживается возможность создания любых типов данных для столбцов таблиц)
* новые нереляционные модели
  + пары «ключ-значение»
  + документо-ориентированная модель
  + поколоночная модель
  + графовые модели.

**Иерархическая модель данных.** В СУБД иерархического типа вся информация представлена в виде деревьев, узлами которых являются записи (например, записи о подразделениях предприятия, сотрудниках подразделений и их детях образуют дерево)*.* Корневая запись каждого экземпляра дерева (в нашем случае запись о подразделении) обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках данного экземпляра дерева. Каждый экземпляр дерева называется *групповым отношением*, корневая запись называется *владельцем*, а все дочерние записи – *членами* группового отношения.

**Сетевая модель данных.** Сетевая модель данных определяется в тех же терминах, что и иерархическая. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:М. Но отличие сетевой модели состоит в том, что запись может быть членом *нескольких* групповых отношений.

**Реляционная модель данных.** Основным логическим объектом для хранения данных в реляционной модели является таблица. Для организации связей между данными различных таблиц используются общие столбцы.

**Объектно-ориентированная модель данных.** Данная модель считается перспективной в связи с распространением объектно-ориентированного подхода к разработке программных продуктов. В ней альтернативой реляционным таблицам являются *классы,* содержащие поля (элементы данных) и методы (программный код обработки данных)*.* Понятию строки таблицы в объектно-ориентированной модели соответствует экземпляр *класса – объект.*

**Расширения реляционной модели – постреляционная и объектно-реляционная.** *Постреляционная модель* в основе содержит реляционную модель, дополненную возможностью создания вложенных таблиц. Постреляционная модель уже поддерживается некоторыми СУБД, например, Oracle, DB2, Postgres и рядом других, которые содержат такие встроенные типы данных для столбцов таблицы как массивы и таблицы (т.е. получается таблица в таблице).

**Новые нереляционные модели.** В последние годы наблюдается стремительный рост объёмов хранимых данных, усложнение их структуры и усиление многообразия (для обозначения огромных хранилищ слабо структурированных данных часто используют термин Big Data – большие данные). Между тем, требования к доступности данных в таких хранилищах по-прежнему остаются высокими. СУБД, полностью поддерживающие реляционную модель, уже не могут обеспечить приемлемую производительность и масштабируемость при работе с Big Data.

**Модель «ключ-значение»** - суть этой модели заключается в хранении данных в виде одного большого ассоциативного массива. Доступ к данным осуществляется по ключу, при этом каждая запись (значение) может иметь свою собственную структуру и извлекается целиком. Основной способ реализации данной модели – хеш-таблица, которая обеспечивает самый быстрый поиск данных по ключу.

**Документо-ориентированная модель** предполагает хранение данных в виде коллекции документов, при этом каждый документ имеет иерархическую структуру. Иерархическая структура хранения позволяет вести поиск по большому количеству документов без загрузки всех документов в память целиком.

**Графовые модели** (их несколько) реализуют модель базы данных в виде графа и его обобщений. Основными элементами модели являются узлы и связи, поэтому данные модели позволяют хранить огромное количество произвольных связей между элементами данных. Графовые модели данных используются при моделировании социальных сетей, в биоинформатике, одна из графовых моделей была применена в проекте Semantic Web.

1. Обзор СУБД

**Классификация СУБД**

Имеется несколько существенных признаков для классификации СУБД. Выделим основные.

**По способу доступа к базе данных** (иначе – по архитектуре ИС, которую поддерживает СУБД). Различают встраиваемые, файл-серверные и клиент-серверные СУБД.

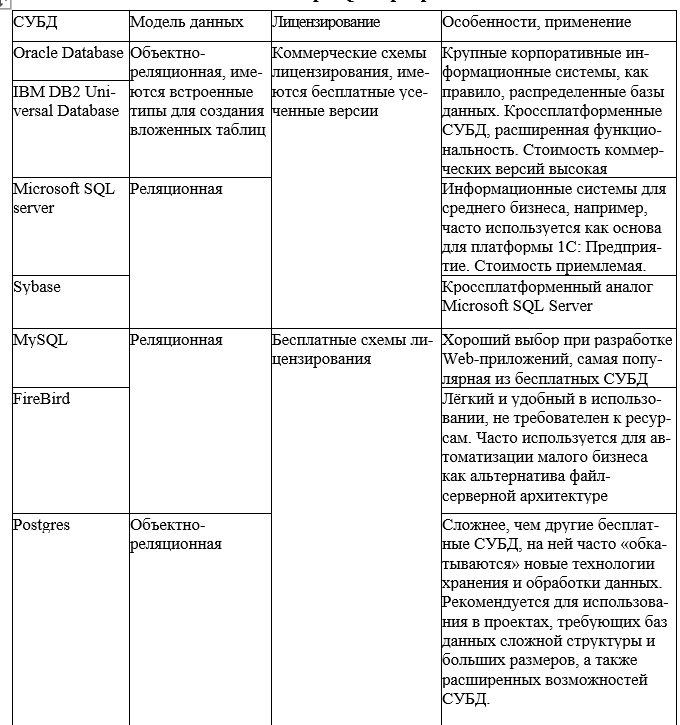
**По поддерживаемой модели данных.** Безусловный лидер по данному признаку – реляционная модель, основанная на стандартном языке запросов SQL, и её расширения – постреляционная и объектно-реляционная модели. Следует отметить также развивающийся быстрыми темпами сегмент СУБД, который условно называется NoSQL.

**По схеме лицензирования** различают свободно распространяемые (бесплатные) и коммерческие программные продукты. В обоих случаях имеется довольно много конкретных схем лицензирования, которых мы касаться не будем. В настоящее время большинство ведущих производителей СУБД, помимо коммерческих версий, предоставляют для свободного скачивания усечённые версии своих СУБД.

**Встраиваемые и файл-серверные СУБД**

Встраиваемые (Embedded) СУБД предназначены для локального хранения данных приложения и не рассчитаны на коллективное использование в сети.

**Клиент-серверные СУБД**



**Обзор NoSQL серверов**

|  |  |  |  |
| --- | --- | --- | --- |
| СУБД | Модель данных | Язык реализации | Особенности, применение |
| Redis | «Ключ-значение» | С | Максимальная производительность, поскольку вся БД размещена в оперативной памяти. Высокоскоростные сетевые хранилища данных |
| Riak | Erlang. C++, Java Script | Повышенная надежность хранения данных, поддерживаются хранилища больших размеров, поскольку выполняется выгрузка данных на диск |
| Couch DB | Документо-ориентированная модель | Erlang | Простота использования, применяется в известных проектах web-сайтов и web-сервисов различного назначения |
| MongoDB | C++ | Сочетание производительности и масштабируемости, используется в крупных распределенных хранилишах документов |
| BigTable | Поколоночные СУБД «Ключ-значение» |  | Разработана компанией Google и используется в проектах данной компании. Не распространяется, но возможно её использование через Google App Engine. |
| Apache Cassandra | Java | Свободно распространяемые реализации, основанные на концепции Google Big Table. Распределенность, отказоустойчивость, масштабируемость, возможность обработки огромных массивов данных |
| HyperTable | C++ |
| Neo4j | Графовая модель | Java | Используется для организации хранения данных, которые удобно моделировать с использованием сетевых структур, например, для хранения данных социальных сетей. |

1. Реляционная модель данных - определения. Свойства отношений. Потенциальные ключи. Первичный ключ, суррогатный ключ.

Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

*Отношение - это множество кортежей, соответствующих одной схеме отношения.*

Из определения отношения следуют его основные свойства:

в отношении не может быть двух одинаковых кортежей (согласно определению множества, все его элементы уникальны)

кортежи не упорядочены, атрибуты также не упорядочены (это свойство также является неотъемлемым свойством любого множества).

Добавим к этому, что имена всех атрибутов в пределах одного отношения должны быть уникальны.

*Потенциальным ключом отношения (Candidate Key - CK)* называют подмножество атрибутов отношения, которое удовлетворяет двум свойствам:

* уникальность (в отношении не существует двух одинаковых значений потенциального ключа);
* безизбыточность (никакое подмножество потенциального ключа не является потенциальным ключом).

Различают простые и составные потенциальные ключи (например, серия и номер паспорта сотрудника – составной потенциальный ключ, а его ИНН - простой).

В каждом отношении можно выделить один или несколько потенциальных ключей. Если таких ключей несколько, один из них выбирается в качестве *первичного ключа* (Primary Key - PK)*.*

*Первичные ключи используются в качестве общих столбцов для связывания таблиц.* Поэтому в качестве первичного ключа обычно выбирают самый короткий, чаще всего, числовой атрибут, для которого гарантируется не только уникальность, но и неизменность значений. На практике часто невозможно выделить такой атрибут, в этом случае используют дополнительный атрибут, называемый суррогатным ключом, который автоматически заполняется уникальными значениями и никогда не изменяется. Все потенциальные ключи отношения, которые не являются первичным ключом, называются *альтернативными* ключами.

Ни в одном из потенциальных ключей отношения NULL-значения недопустимы.

1. Внешние ключи. Правила ссылочной целостности.

Первичные ключи отношений используются в качестве общих атрибутов при связывании отношений. Для этой цели вводятся понятия *родительского (главного) отношения* и *дочернего (подчиненного) отношения.*

Первичный ключ родительского отношения, экспортированный в дочернее отношение в качестве связующего атрибута, называется *внешним ключом* дочернегоотношения*.* Посредством внешнего ключа кортежи дочернего отношения ссылаются на соответствующие им кортежи родительского отношения.

Определим данное понятие более формально. Назовем *внешним ключом* (Foreign Key - FK)такое подмножество атрибутов дочернего отношения, что для любого его непустого значения обязательно найдется равное значение первичного ключа главного отношения.

Обратное утверждение не справедливо, т.е. в родительском отношении могут найтись такие кортежи, на которые не ссылаются никакие кортежи дочернего отношения. Например, в отношении Сотрудники могут быть кортежи, для которых нет ни одного связанного кортежа в отношении Дети\_Сотрудников.

Значения внешних ключей не обязаны быть (и обычно не бывают) уникальными в своем отношении (продолжая пример с сотрудниками и детьми – у сотрудников может быть несколько детей, все они содержит одинаковые значения внешнего ключа).

Во внешнем ключе NULL-значения допустимы, в отличие от первичного ключа, однако на практике такая необходимость встречается крайне редко.

**Целостность ссылок (ссылочная целостность).** Сложные объекты реального мира представляются в реляционной базе данных в виде кортежей нескольких отношений, связанных между собой с помощью внешних ключей.

Требование целостности по ссылкам состоит в следующем:

все значения внешних ключей должны быть *согласованы.*

Правила поддержки ссылочной целостности зависят от выполняемой операции манипулирования данными.

При выполнении *операции вставки* ссылочная целостность контролируется только в случае наличия внешних ключей, ссылающихся на другие отношения. В этом случае проверяется существование соответствующих значений первичных ключей, в случае их отсутствия операция вставки отменяется.

Обеспечить ссылочную целостность при удалении можно несколькими способами.

* Запретить удаление кортежей в родительском отношении при наличии хотя бы одного ссылающегося кортежа (restrict - ограничить удаления).
* При удалении кортежа родительского отношения удалять все ссылающиеся на него кортежи дочернего отношения (cascade – каскадное удаление).
* При удалении кортежа родительского отношения установить во всех ссылающихся кортежах NULL-значения во внешних ключах (set null). Этот способ можно применять только в случае, если NULL-значения в соответствующем внешнем ключе разрешены.
* При удалении кортежа родительского отношения установить во всех ссылающихся кортежах значения по умолчанию во внешних ключах (set default). Значения по умолчанию задаются при создании базы данных.

Из предложенных способов наиболее безопасным является первый – запрет удаления, он применяется чаще всего. Способ каскадного удаления следует применять очень осторожно.

При выполнении операций обновления во внешних ключах ссылочная целостность обеспечивается так же, как и при добавлении нового кортежа.

1. Процедура проектирования базы данных. Диаграмма «сущности - связи». Использование CASE-средств.
2. Нормализация. Функциональные зависимости и декомпозиция отношений. Теорема Хеза.

В процессе разработки основ реляционной теории был предложен формальный математический аппарат, позволяющий проектировать реляционные базы данных с минимальной избыточностью, который получил название *нормализации* базы данных.

Нормализация основана на анализе *функциональных зависимостей (ФЗ)* между атрибутами отношений.

Если даны два атрибута X и Y некоторого отношения, то говорят, что Y функционально зависит от X, если в любой момент времени каждому значению X соответствует ровно одно значение Y.

Функциональная зависимость обозначается X → Y. Отметим, что X и Y могут представлять собой не только единичные атрибуты, но и группы (подмножества), составленные из нескольких атрибутов одного отношения.

*Подмножество атрибутов X называют детерминантом функциональной зависимости.*

Можно сказать, что функциональные зависимости представляют собой связи типа "один ко многим", существующие внутри отношения.

**Нежелательные ФЗ.** Некоторые функциональные зависимости, входящие в неприводимое множество ФЗ отношения, являются нежелательными, так как приводят к дублированию информации, что, в свою очередь, создает проблемы обновления данных и представляет угрозу целостности БД.

*Декомпозицией* – это разбиение имеющихся отношений. В процессе декомпозиции количество отношений в базе данных увеличивается, но общее количество хранимых данных, как правило, сокращается за счет устранения дублирования данных.

**Декомпозиция без потерь. Теорема Хеза.** Декомпозиция отношения есть не что иное, как взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали (возможно, с повторениями) *все* атрибуты исходного отношения. Иначе говоря, при декомпозиции *не должны теряться атрибуты* отношений. Но при декомпозиции также не должны потеряться и сами данные. Данные можно считать не потерянными в том случае, если по декомпозированным отношениям можно полностью восстановить исходное отношение *в прежнем виде,* используя операцию соединения отношений.

Проекции R1 и R2 отношения *R* называются *декомпозицией без потерь*, если отношение *R* *точно восстанавливается* из них при помощи естественного соединения *для любого состояния* отношения *R*.

**Теорема Хеза**

Пусть *R(A, B, C)* - отношение, *A, B, C* - атрибуты или множества атрибутов этого отношения. Если имеется функциональная зависимость*A → B*, то проекции R1(A, B) и R2 (A, C) образуют декомпозицию без потерь.

Иными словами, *любую ФЗ отношения можно вынести в отдельное отношение, оставив ее детерминант в исходном отношении.* При этом никакая информация не будет утеряна.

Обратимый пошаговый процесс декомпозиции отношений с устранением нежелательных функциональных зависимостей называется *нормализацией*.

1. Нормальные формы. Требования первой, второй и третьей нормальных форм.

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 в некотором смысле лучше предыдущей;

при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

*На практике обычно доводят базу данных до третьей нормальной формы.* Подробный анализ положительных и отрицательных сторон нормализации содержится в следующей лекции.

**Требования первой нормальной формы - 1NF.** Отношение находится в 1NF, если оно удовлетворяет двум условиям:

значения всех его атрибутов атомарны;

отсутствуют повторяющиеся группы атрибутов.

Первое из условий выполняется для любого отношения автоматически, поскольку оно следует из свойств отношений. Здесь требуется только уточнить понятие атомарности для текстовых атрибутов.

Второе условие требует отдельного пояснения. *Повторяющейся группой* называют подмножество атрибутов отношения, определенных на одном и том же домене и имеющих одинаковую семантику.

**Требования второй нормальной формы - 2NF.** Первичный ключ отношения иногда включает несколько атрибутов (в таком случае его называют составным). *Отношение находится в 2NF, если оно находится в 1NF и каждый неключевой атрибут функционально полно зависит от ключа.*

**Требования третьей нормальной формы - 3NF.**

Отношение находится в 3NF, если оно находится в 2NF и *каждый неключевой атрибут нетранзитивно зависит от первичного ключа.*

1. Плюсы и минусы нормализации. Сознательная денормализация.
2. Хранилища данных. Задачи аналитической обработки данных.
3. Общая характеристика языка SQL. Состав языка SQL.
4. Команда выборки select – общий вид, простые запросы.

Команда выборки SELECT. Результатом команды выборки всегда является новая таблица (возможно, пустая или содержащая одно значение, которое трактуется как таблица из одной строки и одного столбца).

Новая таблица является виртуальной, т.е. она существует (обычно в оперативной памяти) до тех пор, пока в ней есть необходимость. Она может участвовать в другой команде SQL (CREATE для различных объектов БД, INSERT), в другой команде SELECT (а также DELETE или UPDATE) как результат вложенного запроса или передается клиенту для последующей обработки.

**SELECT [DISTINCT] список\_выражений(\* - все столбцы таблицы)**

**FROM список\_таблиц, представлений, запросов**

**(возможно, с операцией соединения JOIN)**

**[WHERE условия\_отбора\_строк]**

**[GROUP BY список\_выражений\_для\_группировки]**

**[HAVING условия\_отбора\_групп\_с\_агрегатными\_функциями]**

**[ORDER BY список\_выражений\_для\_сортировки]**

**Выборка всех данных таблицы**

SELECT \* FROM имя\_таблицы (представления)

**Отбор столбцов (операция проекции)**

SELECT список\_имен\_столбцов FROM имя\_таблицы

Например:

выбрать столбцы с фамилией и датой рождения из таблицы студентов

SELECT name\_st, born FROM students

1. Способы соединения таблиц в запросе.

**Декартово произведение (CROSS JOIN)**

Это операция, при выполнении которой каждая строка одной таблицы соединяется (склеивается) с каждой строкой другой таблицы. Декартово произведение таблиц выполняется в том случае, если во фразе FROM присутствует список из нескольких таблиц, но не задается способ их соединения, или указывается операция соединения CROSS JOIN. Обычно таблицы, которые соединяются способом декартова произведения, не имеют общих столбцов.

Так, запрос вида:

select … from таблица1, таблица2

Или

select … from таблица1 CROSS JOIN таблица2

возвратит количество строк, равное произведению количества строк первой таблицы на количество строк второй таблицы, поскольку склеит каждую строку таблицы1 с каждой строкой таблицы2.

**Внутреннее соединение таблиц (INNER JOIN)**

Таким способом можно соединять только таблицы, имеющие общие столбцы (но не обязательно одноимённые!). При выполнении данной операции соединяются (склеиваются) только строки, имеющие одинаковые значения в столбце связи.

select students .name\_st, marks.mark

from students join marks

on students.cod\_st=marks.cod\_st

**Внешнее соединение таблиц (outer join)**

При внешнем соединении, в отличие от внутреннего, в результат выборки попадают не только все связанные строки обеих таблиц, но и строки одной из таблиц (или обеих), для которых нет связанных в другой таблице. Недостающим значениям столбцов другой таблицы при этом присваивается значение NULL.

Возможны три варианта внешнего соединения двух таблиц (выборка дополняется строками таблицы, стоящей слева от слова JOIN, или таблицы, стоящей справа, или обеих сразу), поэтому различают три вида внешних соединений:

* LEFT [OUTER] JOIN – левое внешнее соединение
* RIGHT [OUTER] JOIN – правое внешнее соединение
* FULL [OUTER] JOIN – полное внешнее соединение.

select st.cod\_st, st.name\_st, avg (m.mark) avg\_mark

from students st left join marks m on st.cod\_st=m.cod\_st  
group by st.cod\_st, st.name\_st

**Естественное соединение (NATURAL JOIN)**

Этот способ используется только для организации связи по одноимённым столбцам. Если в соединяемых таблицах есть столбцы с одинаковыми именами, операция NATURAL JOIN использует их, не требуя явного указания условия соединения в запросе.

select cod\_st, name\_st, m1ark

from students NATURAL join marks

будет выполнено соединение таблиц students и marks по общему одноимённому столбцу cod\_st, хотя в запросе это явно не указано.

**Самосоединения**

Это соединение таблицы со своей копией. Подобные соединения используются сравнительно редко, но при решении некоторых задач могут оказаться полезными.

Например, пусть требуется вывести всех однофамильцев, т.е. фамилии, которые встречаются в таблице студентов более одного раза:

SELECT DISTINCT s1.name\_st FROM students s1, students s2

WHERE s1.name\_st=s2.name\_st AND s1.cod\_st<>s2.cod\_st

1. Агрегатные функции. Запросы с группировкой. Отбор групп.
2. Вложенные и комбинированные запросы.
3. Команды DML для вставки, удаления, обновления.

Язык DDL (Data Definition Language) служит для создания, удаления и модификации всех объектов, входящих в состав базы данных.

Язык DDL содержит всего три основных команды **CREATE**, **ALTER** и **DROP**, которые могут быть применены к различным объектам базы данных.

В самом общем виде команду DDL можно определить так:

Создание (CREATE)

или

Изменение (ALTER) объект имя объекта [дополнительные параметры]

или

Удаление(DROP)

Примеры:

DROP TABLE t - удаление таблицы с именем t, здесь объектом является TABLE, его имя t.

Данная команда не требует никаких дополнительных параметров

CREATE TABLE tt (n NUMBER, x VARCHAR(50)) – создание таблицы с именем tt, в качестве дополнительных параметров задаются определения столбцов, в данном прмере их два: столбец n имеет цифровой тип, а столбец x - текстовый тип, причем длина текста не превышает 50 символов.

1. Команды языка DDL. Основные объекты реляционной базы данных.
2. Создание таблиц. Типы данных.

При создании таблицы задается ее уникальное имя и структура, т.е. первоначально создается пустая таблица, которая затем наполняется данными и при помощи команд DML.

Общий вид команды для создания таблицы:

**CREATE TABLE имя\_таблицы**

**(список\_определений\_столбцов [,список ограничений\_на\_таблицу]**

**)**

Элементы списка всегда разделяются запятыми (в дальнейшем это будет считаться само собой разумеющимся и особо оговариваться не будет).

Определение каждого столбца в списке имеет вид:

**имя\_столбца тип [DEFAULT значение\_по\_умолч.] [ограничения]**

Обязательными элементами описания столбца являются имя столбца и тип данных в столбце.

Имя столбца должно быть уникальным в пределах таблицы.

**Типы данных**

**Числовые типы.** Стандарт предусматривает следующие числовые типы.

* SMALLINT, INTEGER (INT) и BIGINT (последний появился только в SQL 2003) - целые двоичные числа со знаком, хотя стандарт не оговаривает размеры, в большинстве случаев это 2, 4 и 8 байт соответственно.
* NUMERIC(p,s) или NUMBER(p,s) или DECIMAL(p,s) - типы-синонимы для десятичных чисел с фиксированной точкой. Число p задает общее количество десятичных разрядов в числе, s - число разрядов после точки.
* FLOAT(p), REAL, DOUBLE PRECISION - двоичные числа с плавающей точкой. Разрядность типов REAL и DOUBLE PRECISION зависит от реализации.

**Символьные типы.** Их несколько.

* CHAR[ACTER] [(размер)] ) — строка фиксированной длины (по умолчанию размер типа – 1 байт), максимальный размер в Oracle 2000 байт, но во многих СУБД размер этого типа ограничивается 256 байтами.
* VARCHAR(размер) (в Oracle дополнительно поддерживается тип VARCHAR2, идентичный VARCHAR, но гарантированно неизменный вне зависимости от возможного изменения типа VARCHAR в стандарте) — строка переменной длины, в Oracle его размер не более 4000 байт.

**Типы даты и времени.** Стандарт поддерживает типы DATE, TIME и TIMESTAMP, в Oracle реализован тип DATE, которые служит для хранения даты и времени суток, а, начиная с версии 9i, еще и TIMESTAMP, который позволяет хранить момент времени с высокой точностью (до миллисекунд).

**Типы для больших объектов.** Современные базы данных позволяют хранить не только хорошо структурированную фактографическую информацию, но и такие данные, как рисунки (фотографии), звуковые или анимационные файлы, тексты формата Microsoft Word, Web-страницы в формате html и другую слабо структурированную информацию. Для хранения подобных данных используются типы больших объектов.

Стандарт поддерживает типы BLOB (Binary Large Object) и CLOB (Character Large Object) или NCLOB (National Character Large Object).

Oracle дополнительно поддерживает тип LONG для хранения больших текстов. Работа с типами LONG и CLOB выполняется значительно медленнее, чем с типом VARCHAR, поэтому они используются только для хранения больших текстовых документов.

1. Ограничения (Constraints) и их роль при создании таблиц.

Ограничения (constraints) позволяют при создании таблиц задавать дополнительные условия проверки вводимых данных, которые СУБД проверяет автоматически. Данные, которые не удовлетворяют условиям, заданным в ограничениях, отвергаются. Например, при вставке новой строки в таблицу она не будет добавлена, если хотя бы одно из значений не удовлетворяет ограничениям. Механизм ограничений позволяет поддерживать данные в непротиворечивом состоянии, соответствующем бизнес-правилам предметной области.

Любое ограничение может быть поименовано, в противном случае, имя для ограничения СУБД создает автоматически. Для явного задания имени, к определению ограничения следует добавить слева фразу CONSTRAINT имя\_ограничения (рекомендуется задавать явные имена для ограничений).

Можно задать ограничения для отдельного столбца или для таблицы в целом.

**Ограничения столбца:**

**NOT NULL/NULL** – допустимы ли пустые (неопределенные) значения в столбце, по умолчанию используется значение NULL (т.е. допустимы), что в большинстве случаев не соответствует бизнес-правилам предметной области, поскольку пропуски какой-либо нужной информации обычно не допустимы. Данное ограничение не именуется.

**PRIMARY KEY** – ограничение первичного ключа, при этом автоматически накладывается ограничение NOT NULL. При задании значений первичного ключа любое значение проверяется на уникальность и при обнаружении дубликата операция прерывается. В таблице может быть только один столбец с ограничением PRIMARY KEY.

**UNIQUE** – ограничение уникальности (альтернативный ключ), ограничение NOT NULL также накладывается автоматически. Фактически, UNIQUE ничем не отличается от PRIMARY KEY, однако количество столбцов с UNIQUE не ограничено.

**REFERENCES имя\_главной\_таблицы**— внешний ключ, который задается для столбца подчиненной (детальной) таблицы. Для значений внешнего ключа автоматически выполняется проверка на существование равного значения первичного ключа главной таблицы. При определении внешнего ключа могут быть дополнительно определены правила обеспечения ссылочной целостности.

**CHECK** – проверка дополнительных условий (логических выражений) при изменении данных в столбце, например, CHECK (имя\_столбца BETWEEN 1 AND 100).

**Ограничения таблицы**

Данные ограничения обычно используются в том случае, если они затрагивают сразу несколько столбцов:

**PRIMARY KEY** (список столбцов) — составной первичный ключ;

**UNIQUE** (список столбцов) — составной альтернативный ключ;

**FOREIGN KEY** имя\_внешнего\_ключа (список столбцов) REFERENCES имя\_главной\_таблицы [правила поддержки ссылочной целостности];

**CHECK** (логическое выражение, затрагивающее сразу несколько столбцов).

Например:

CREATE TABLE t1

( c1 NUMBER(3) NOT NULL,

c2 DATE NOT NULL,

c3 NUMBER(3) NOT NULL,

CONSTRAINT pk\_t1 PRIMARY KEY(c1,c2),

CONSTRAINT ck\_t1 CHECK(c1+c3<=200)

)

Создается таблица с именем t1, содержащая три столбца, обязательных для заполнения. Составной ключ таблицы включает столбцы c1 и с2 (обратим внимание, что каждый из столбцов таблицы не обладает свойствами уникальности, поэтому PRIMARY KEY не может быть ограничением одного столбца). Ограничение CHECK также затрагивает сразу два столбца, поэтому оформлено как ограничение таблицы.

1. Удаление таблицы. Изменение структуры таблицы

Команда удаления таблицы имеет вид:

**DROP имя\_таблицы**

Нельзя удалить таблицу, если существует внешние ключи, ссылающиеся на эту таблицу. Вместе с таблицей удаляются и созданные для нее индексы и триггеры.

Команда изменения структуры таблицы выглядит так:

**ALTER TABLE имя\_таблицы** указания\_по\_изменению\_структуры

Следует подчеркнуть, что команда ALTERTABLE служит для изменения в определении таблицы и может быть применена только к существующим таблицам. В качестве указаний по изменению структуры таблицы могут использоваться следующие фразы:

**ADD [COLUMN] определение\_столбца** – добавить столбец.

Например:

ALTER TABLE t ADD n NUMBER(4) NOT NULL

(t— имя существующей таблицы, n — имя нового столбца )

**ADD [CONSTRAINT] ограничение**  — добавить ограничение.

Например:

ALTER TABLE t ADD PRIMARY KEY(n)

или

ALTER TABLE t ADD CONSTRAINT pk\_t PRIMARY KEY(n)

В последнем примере ограничение первичного ключа получит имя pk\_t

**DROP COLUMN имя\_столбца**

**DROP [CONSTRAINT] ограничение**

Например:

ALTER TABLE t DROP PRIMARY KEY(n)

или

ALTER TABLE t DROP CONSTRAINT pk\_t

ALTER TABLE t DROP COLUMN n

**ALTER (MODIFY в Oracle) новое\_определение\_\_столбца**

При этом можно изменить тип, размер, ограничение NULL/NOT NULL, значение по умолчанию. Конечно, нельзя изменить имя столбца. Для этого следует удалить столбец со старым именем и добавить новый столбец с нужным именем;