1. **Концепция и технология баз данных.Понятие банка данных,базы данных,СУБД.**

Принято считать, что использование **концепции** баз данных позволяет:

1. повысить надежность, целостность и сохранность данных;
2. сохранить затраты интеллектуального труда;
3. обеспечить простоту и легкость использования данных;
4. обеспечить независимость прикладных программ от данных (изменений их описаний и способов хранения);
5. обеспечить достоверность данных;
6. обеспечить требуемую скорость доступа к данным;
7. стандартизовать данные в пределах одной предметной области;
8. автоматизировать реорганизацию данных;
9. обеспечить защиту от искажения и уничтожения данных;
10. сократить дублирование информации за счет структурирования данных;
11. обеспечить обработку незапланированных запросов к хранимой информации;
12. создать предпосылки для создания распределенной обработки дaнныx.

**База данных**

База данных – есть совокупность взаимосвязанных именованных наборов данных с общими правилами организации, описания, хранения и обработки.

Эти наборы должны быть

интегрированными (с минимальным дублированием информации);

взаимосвязанными (взаимно дополняющими – до требуемой полноты сведений);

целенаправленными (содержать только необходимые сведения);

независимыми от процессов обработки.

База данных - это набор структурированной информации, предназначенный для совместного использования несколькими пользователями одновременно. Отдельные элементы данных в базе данных связаны между собой логическими связями, взаимозависимы. Именно понятие “структурированная” и “для совместного использования” отличают базу данных от простых файлов с данными, которые тоже являются наборами данных.

Структурированность означает, что данные имеют некоторую логическую структуру, некоторую схему, модель, которая связывает между собой разные данные.

**Банк данных**

Автоматизированный банк данных – это система информационных, математических, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоаспектного использования данных в некоторой предметной области. Банк данных включает в себя одну или несколько баз данных логически связанных между собой, систему управления ими (СУБД) и комплекс прикладных программ.

Банк данных должен обеспечить

* хранение и модификацию больших объемов многоаспектной информации;
* заданный уровень достоверности и непротиворечивость хранимой информации, ее восстановление после сбоев и отказов;
* поиск информации по произвольной совокупности признаков;
* одновременное обслуживание большого числа пользователей;
* оперативность в обработке запросов;
* простоту обращения;
* доступ к данным лишь тех пользователей, которые имеют необходимые полномочия.

**СУБД**

База данных предполагает наличие некоторого программного обеспечения, позволяющего пользователям работать с базой данной. Это программное обеспечение разрабатывается с помощью инструментальных программных средств, называемых системой управления базами данных (СУБД). С помощью СУБД можно создавать базы данных, модифицировать данные в базе данных, вносить новые данные, разрабатывать пользовательские приложения. СУБД должна выполнять некоторые задачи по администрированию и поддержанию непротиворечивости данных. СУБД - это инструмент, с помощью которого создается та или иная конкретная база данных.

В примере про систему резервирования билетов к системе управления базами данных относятся средства, с помощью которых разработаны программа, с которой работают операторы, сидящие в авиа-кассах, и программа, с помощью которой администратор меняет список рейсов. Утилита, с помощью которой по ночам создается резервная копия всей базы, также относится к СУБД.

Еще раз отметим разницу между базой данных и системой управления базой данных. Если какая-то фирма пишет в объявлении, что она продает базу данных, то это означает, что она продает информацию. Если же в рекламе написано о СУБД, то следует ожидать, что Вам предложат программные средства, с помощью которых Вы соберете свою собственную базу данных. Хотя, в реальной жизни, понятия базы данных и системы управления базой данных часто смешивают.

##### Информационно-поисковая система - система, выполняющая функции:  - хранения больших объемов информации;  - быстрого поиска требуемой информации;  - добавления, удаления и изменения хранимой информации;  - вывода информации в удобном для человека виде.

##### Различают:  - автоматизированные (coputerised);  - библиографические (reference);  - диалоговые (online);  - документальные и фактографические информационно-поисковые системы.

1. **Функции СУБД. Архитектура СУБД. Компоненты архитектуры и их характеристика.**

## Архитектура СУБД

**Основная функция СУБД** – организация обмена информацией между пользователями и базами данных с соответствующими процедурами контроля полномочий и процедур проверки. Среди пользователей СУБД выделяется лицо (или группа лиц), на которого обычно возлагаются следующие функции:

* определение информационного содержания базы данных (идентификация объектов и связей, представляющих интерес для данного предприятия, создание на этой основе концептуальной схемы (с помощью специального языка);
* определение структуры хранения и стратегии доступа;
* взаимодействие с пользователем (подготовка и написание внешних схем);
* определение стратегии дублирования и восстановления;
* управление эффективностью ответа на запросы пользователей;
* создание словаря данных.

Такое лицо получает статус администратора базы данных (АБД).  
Для обычного пользователя перечисленные выше функции либо недоступны, либо они ему не нужны. В этом проявляется еще один аспект независимости данных, а именно:  
СУБД должна предоставлять доступ к данным, не требуя от пользователя знаний о таких тонкостях как:

* физическое размещение в памяти данных и их описаний;
* управление данными во внешней памяти и управление буферами оперативной памяти;
* механизмы поиска запрашиваемых данных;
* проблемы, возникающие при одновременном запросе одних и тех же данных многими пользователями (как для прикладных, так и системных программам);
* способы обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа и множестве других функций СУБД, таких, например, как управление транзакциями или журнализации и восстановления баз данных после всевозможных сбоев.

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, которые практически не имеют и (или) не хотят иметь представления о:

1. физическом размещении в памяти данных и их описаний;
2. механизмах поиска запрашиваемых данных;
3. проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами);
4. способах обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа;
5. поддержании баз данных в актуальном состоянии и множестве других функций СУБД.

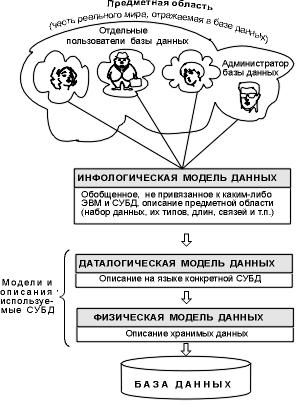
Естественно, что проект базы данных надо начинать с анализа предметной области и выявления требований к ней отдельных пользователей (сотрудников организации, для которых создается база данных). Проектирование обычно поручается человеку (группе лиц) – *администратору базы данных* (АБД). Им может быть как специально выделенный сотрудник организации, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться в будущих приложениях, АБД сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических формул, таблиц, графиков и других средств, понятных всем людям, работающих над проектированием базы данных, называют *инфологической моделью данных* (рис. 1.1.).

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. В конце концов этой средой может быть память человека, а не ЭВМ. Поэтому инфологическая модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторого определения, чтобы эта модель продолжала отражать предметную область.

Остальные модели, показанные на рис. 1.1., являются компьютерно-ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных. Нужные данные отыскиваются СУБД на внешних запоминающих устройствах по *физической модели данных*.

Так как указанный доступ осуществляется с помощью конкретной СУБД, то модели должны быть описаны на *языке описания данных* этой СУБД. Такое описание, создаваемое АБД по инфологической модели данных, называют *даталогической моделью данных*.



*Рис. 1. 1. Уровни моделей данных*

Трехуровневая архитектура (инфологический,

даталогический и физический уровни) позволяет

обеспечить *независимость хранимых данных* от

использующих их программ. АБД может при

необходимости переписать хранимые данные на

другие носители информации и (или) реорганизовать

их физическую структуру, изменив лишь

физическую модель данных. АБД может

подключить к системе любое число новых

пользователей (новых приложений), дополнив,

если надо, даталогическую модель. Указанные

изменения физической и даталогической моделей

не будут замечены существующими пользователями

системы (окажутся "прозрачными" для них), так

же как не будут замечены и новые пользователи.

Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.

1. **Основные свойства баз данных.**

1. Совместный доступ к данным, т.е. данные введенные единожды используются многократно.  
2. Минимизация избыточности данных. Ни одна БД не существует без избыточности. Избыточность существует двух видов. Управляемая - та, о которой мы знаем. Неуправляемая - хаотичная.  
3. Целостность и непротиворечивость данных. Целостность - семантические правила предметной области для БД, обеспечивающиеся чаще всего за счет расширения метаданных.  
4. Защита БД.  
5. Поддержка транзакций - последовательность операций над БД неделимых для СУБД.  
6. Увеличение гибкости обслуживания запросов данных - оптимизация запросов в СУБД.  
7. Обеспечение логической независимости программы от данных, т.е. возможность изменять логическую структуру БД без изменения программы.  
8. Физическая независимость, т.е. возможность изменения физического строения БД без изменения приложения.  
9. Наличие развитых средств поддержки БД.  
10. Сокращение времени разработки приложения за счет стандартных средств.  
11. Стандартизация.

1. **Этапы проектирования баз данных и их характеристика.**

1. Концептуальное проектирование — сбор, анализ и редактирование требований к данным. Для этого осуществляются следующие мероприятия:

* обследование предметной области, изучение ее информационной структуры
* выявление всех фрагментов, каждый из которых характеризуется пользовательским представлением, информационными объектами и связями между ними, процессами над информационными объектами
* моделирование и интеграция всех представлений

По окончании данного этапа получаем концептуальную модель, инвариантную к структуре базы данных. Часто она представляется в виде модели «сущность-связь».

2. Логическое проектирование — преобразование требований к данным в структуры данных. На выходе получаем СУБД-ориентированную структуру базы данных и спецификации прикладных программ. На этом этапе часто моделируют базы данных применительно к различным СУБД и проводят сравнительный анализ моделей.

3. Физическое проектирование — определение особенностей хранения данных, методов доступа и т. д.

Различие уровней представления данных на каждом этапе проектирования реляционной базы данных:

КОНЦЕПТУАЛЬНЫЙ УРОВЕНЬ — Представление аналитика (используется инфологическая модель «сущность-связь»)

\* сущности

\* атрибуты

\* связи

ЛОГИЧЕСКИЙ УРОВЕНЬ — Представление программиста

\* записи

\* элементы данных

\* связи между записями

ФИЗИЧЕСКИЙ УРОВЕНЬ — Представление администратора

\* группирование данных

\* индексы

\* методы доступа

1. **Case-средства для проектирования БД. Общая характеристика. Примеры.**

История систем автоматизации проектирования баз данных (CASE-средств) началась с автоматизации процесса рисования диаграмм, проверки их формальной корректности, обеспечения средств долговременного хранения диаграмм и другой проектной документации.

Эта идея, естественно, показалась разумной производителям CASE-средств проектирования БД. Подавляющее большинство подобных систем, представленных на рынке, **обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных, в реляционные схемы, специфицированные чаще всего на языке SQL**. У читателя может возникнуть вопрос, почему в предыдущем предложении говорится про «автоматизированное», а не про «автоматическое» преобразование? Все дело в том, что в типичной схеме SQL-ориентированной БД могут содержаться определения многих объектов (ограничений целостности общего вида, триггеров и хранимых процедур и т. д.), которые невозможно сгенерировать автоматически на основе концептуальной схемы. Поэтому на завершающем этапе проектирования реляционной схемы снова требуется ручная работа проектировщика.

Как правило, CASE-средства, автоматизирующие преобразование концептуальной схемы БД в реляционную, производят реляционную схему базы данных в третьей нормальной форме. Нормализация более высокого уровня усложняет программную реализацию и редко требуется на практике.

Наконец**, третья возможность**, которую следует упомянуть, хотя она еще не вышла (или только выходит, а может быть, никогда и не выйдет) за пределы исследовательских и экспериментальных проектов, – это работа с базой данных в семантической модели, т. е. СУБД, основанные на семантических моделях данных. При этом снова рассматриваются два варианта:

* обеспечение пользовательского интерфейса на основе семантической модели данных с автоматическим отображением конструкций этого интерфейса в реляционную модель данных (это задача примерно того же уровня сложности, что и автоматическая компиляция концептуальной схемы базы данных в реляционную схему)
* прямая реализация СУБД, основанная на какой-либо семантической модели данных.

Многие авторитетные специалисты полагают, что ближе всего ко второму подходу объектно-ориентированные СУБД, чьи модели данных по многим параметрам близки к семантическим моделям (хотя в некоторых аспектах они более мощны, а в некоторых – более слабы).

1. **Модели данных в БД. Основные понятия и определения.**

Центральным понятием в области баз данных является понятие модели.

**Модель данных** - это некоторая абстракция, которая, прикладываемая к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

На рис. 2.1. представлена классификация моделей данных.

**Модели данных**

**Инфологические модели**

**Даталогические модели**

Физические модели

Диаграммы

Бахмана

Модель сущность связь (ER)

Документальные модели

Фактографические модели

Основанные на файловых структурах

Основанные на странично-сегментной организации

Ориенированные на формат документа

Дескрипторные модели

Тезаурусные модели

Теоретико-графовые

Теоретико-множественные

Объектно-ориентированные

Иерархическая

Сетевая

Реляционная

Бинарных ассоциаций

Рис.2.1. Классификация моделей данных

Как отмечалось выше, *инфологическая модель* отображает реальный мир в некоторые понятные человеку концепции, полностью независимые от параметров среды хранения данных. Существует множество подходов к построению таких моделей: графовые модели, семантические сети, модель "сущность-связь" и т.д. Инфологическая модель должна быть отображена в компьютеро-ориентированную даталогическую модель, "понятную" СУБД. В процессе развития теории и практического использования баз данных, а также средств вычислительной техники создавались СУБД, поддерживающие различные даталогические модели.

Сначала стали использовать иерархические даталогические модели. Простота организации, наличие заранее заданных связей между сущностями, сходство с физическими моделями данных позволяли добиваться приемлемой производительности иерархических СУБД на медленных ЭВМ с весьма ограниченными объемами памяти. Но, если данные не имели древовидной структуры, то возникала масса сложностей при построении иерархической модели и желании добиться нужной производительности.

Сетевые модели также создавались для мало ресурсных ЭВМ. Это достаточно сложные структуры, состоящие из "наборов" – поименованных двухуровневых деревьев. "Наборы" соединяются с помощью "записей-связок", образуя цепочки и т.д. Прикладной программист должен знать массу терминов, изучить несколько внутренних языков СУБД, детально представлять логическую структуру базы данных для осуществления навигации среди различных экземпляров, наборов, записей и т.п. Один из разработчиков операционной системы UNIX сказал "Сетевая база – это самый верный способ потерять данные".

1. **Характеристика компонент моделей данных (реляционной, иерархической, сетевой). Абстракции в моделях данных. Примеры.**

## Иерархическая модель данных

**Иерархические,** или **древовидные**, *структуры* данных разработаны и используются достаточно давно. Например, большинство *методов индексирования* базируются именно на древовидных структурах данных. Иерархическая *модель* данных близка по своей идее к иерархической *структуре* данных. Но *модель* описывает не конкретные методы работы и манипулирования ссылками, а **способ логического представления данных**, то, какими терминами оперирует проектировщик структуры базы данных, когда отражает реальные зависимости с помощью имеющихся в СУБД механизмов.

Иерархическая модель позволяет строить иерархию элементов. То есть у каждого элемента может быть несколько “наследников” и существует один “родитель”. Для каждого уровня связи вводится интерпретация, зависящая от предметной области и описывающая взаимоотношение между “родителями” и “наследниками”. Каждый **элемент** представляется с помощью **записи**. Структура данных, обычно используемая для представления этой записи об элементе, обычно содержит некоторые атрибуты, характеристики каждого элемента.

Попробуем представить себе базу данных для описания тематических сборников по некоторой теме. Прежде всего, выделим уровни иерархии.

Первый уровень - это **издательства**. Каждое издательство характеризуется своим *названием, юридическим адресом, номером счета в банке*.

Каждое издательство выпускает несколько **сборников**. То есть издательство является “родителем” для сборника и связано со сборником соотношением “издает” (“публикует”, “печатает” и т.д.). Для каждого сборника появляются такие атрибуты, как *размер, периодичность, цена, ответственный редактор, корректор* и т.д.

В каждом сборнике есть несколько **статей** (хотя бы, одна). То есть сборник и статья связаны соотношением “включает”. Далее, у каждой статьи есть *название, авторы*.

**Авторы** представляются отдельным элементом и образуют следующий уровень иерархии. Каждый автор характеризуется *фамилией, именем, отчеством, гонораром* и т.д. Статьи связаны с автором соотношением “написаны”.

Графическое представление этого примера приведено на *Рис.2.2.* Элементы нарисованы прямоугольниками, их названия даны обычным шрифтом. Связи нарисованы стрелками и их названия даны курсивом, атрибуты для каждого элемента на этой схеме не показаны - они являются частью элемента данных.



Рис.2.2. Графическое представление иерархической модели данных (справа пример какой-то конкретной базы данных)

**Достоинства иерархических СУБД:** возможность реализовать быстрый поиск нужных значений, когда условия запроса соответствуют иерархии в схеме базе данных.

**Недостатки** **иерархических СУБД:** например, если запрос не соответствует имеющейся иерархии, то и его программирование, и его исполнение, потребуют значительных усилий. Например, попытки реализовать запрос типа *“в скольких сборниках статей опубликовал свои статьи какой-либо автор”* может оказаться весьма трудной задачей (мы можем искать в направлении от статьи к автору, но не наоборот). Другим недостатком иерархической модели является сложность внесения в нее изменений. Если, по каким-то причинам изменились условия задачи, и модель предметной области перестала быть иерархической (например, в нашем примере, мы хотим иметь не только зависимость авторов от статьи, но и статей от автора), то приведение схемы базы данных в соответствие предметной области становится нетривиальной задачей.

Недостатки иерархической модели проистекают оттого, что данная модель слишком жесткая. Иерархическая модель очень хорошо подходит для устоявшихся предметных областей с четкими зависимостями “родитель-потомок”, то есть к моделям, где есть четкая субординация между понятиями.

## Сетевая модель данных

**Сетевая модель данных** является развитием иерархической модели. В сетевой модели, так же как и в иерархической модели, есть понятие элемента данных и связи, которая может быть именована. Главное отличие сетевой модели от иерархической заключается в том, что к каждому элементу может идти связь не от одного элемента (“родителя”), а от нескольких.

Например, генеалогическое дерево, построенное только по мужской линии (или, только по материнской), является древовидной, иерархической структурой - у каждого человека (элемента), есть только один родитель. Если же включать в генеалогическое дерево всех родителей, то такое дерево с точки зрения структур данных будет уже не деревом, а сетью:



Рис.2.3. Представление фрагмента генеалогического дерева на основе сетевой модели данных

На данном рисунке представлены элементы только одно класса - описание людей, и на этом множестве для некоторых конкретных пар людей существуют связи, именуемые “муж”, “жена”, “отец”, “мать”, “ребенок”. Поэтому с точки зрения графического представления схемы этой базы данных (а не конкретных данных о семье Петровых), можно использовать следующий рисунок:



Рис.2.4. Представление схемы базы данных генеалогического дерева на основе сетевой модели данных

**Сетевая модель данных** основывается на понятии элемента данных и связей, задающих логику взаимоотношениями между данными. Связи от каждого элемента могут быть направлены на произвольное количество других элементов. На каждый элемент могут быть направлены связи от произвольного числа других элементов. Каждый элемент данных описывает некоторое понятие из предметной области и характеризуется некоторыми атрибутами. Для каждого элемента данных (элемент - это часть схемы) в реальной базе данных может существовать несколько экземпляров этого элемента. С каждым конкретным экземпляром по конкретной связи может быть связано разное число экземпляров другого элемента (например, у каждого человека разное число детей), но число видов связи одинаково для всех экземпляров одного элемента.

Если мы вернемся к нашему примеру про издательства тематических сборников (этот пример рассматривался в разделе про иерархические СУБД) и попытаемся расширить его, для того чтобы он более полно соответствовал реальным взаимоотношениям, то схема базы данных будет выглядеть следующим образом:



Рис.2.5. Представление расширенной схемы базы данных для описания издательств на основе сетевой модели

**К достоинствам сетевой модели** относится очень высокая скорость поиска и возможность адекватно представлять многие задачи в самых разных предметных областях. Высокая скорость поиска основывается на классическом способе физической реализации сетевой модели - на основе списков.

**Главным недостатком сетевой модели**, как, впрочем, и иерархической, является ее жесткость. Поиск данных, доступ к ним, возможен только по тем связям, которые реально существуют в данной конкретной модели. При поиске данных сетевая СУБД требует перемещаться только по существующим, заранее предусмотренным связям.

## Реляционная структура данных

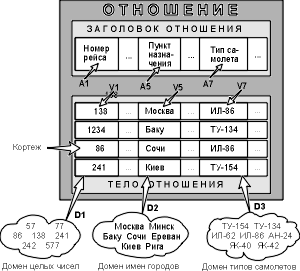
В конце 60-х годов появились работы, в которых обсуждались возможности применения различных табличных даталогических моделей данных, т.е. возможности использования привычных и естественных способов представления данных. Наиболее значительной из них была статья сотрудника фирмы IBM д-ра Э. Кодда (Codd E.F., A Relational Model of Data for Large Shared Data Banks. CACM 13: 6, June 1970), где, вероятно, впервые был применен термин "реляционная модель данных".

Будучи математиком по образованию, Э. Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как *отношение* – relation (англ.).

Наименьшая единица данных реляционной модели – это отдельное *атомарное* (неразложимое) для данной модели значение данных. Так, в одной предметной области фамилия, имя и отчество могут рассматриваться как единое значение, а в другой – как три различных значения.

*Доменом* называется множество атомарных значений одного и того же типа. Отношение на доменах D1, D2, ..., Dn (не обязательно, чтобы все они были различны) состоит из заголовка и тела. На рис. 2.6 приведен пример отношения для расписания движения самолетов.

*Заголовок* (интерпретация) состоит из такого фиксированного множества атрибутов A1, A2, ..., An, что существует взаимно однозначное соответствие между этими атрибутами Ai и определяющими их доменами Di (i=1,2,...,n).

*Рис. 2.6. Отношение с математической точки зрения (Ai - атрибуты, Vi - значения атрибутов)*

*Тело* состоит из меняющегося во времени множества *кортежей*, где каждый кортеж состоит в свою очередь из множества пар атрибут-значение (Ai:Vi), (i=1,2,...,n), по одной такой паре для каждого атрибута Ai в заголовке. Для любой заданной пары атрибут-значение (Ai:Vi) Vi является значением из единственного домена Di, который связан с атрибутом Ai.

*Степень отношения* – это число его атрибутов. Отношение степени один называют унарным, степени два – бинарным, степени три – тернарным, ..., а степени n – n-арным.

*Кардинальное число* или *мощность отношения* – это число его кортежей. Кардинальное число отношения изменяется во времени в отличие от его степени.

Поскольку отношение – это множество, а множества по определению не содержат совпадающих элементов, то никакие два кортежа отношения не могут быть дубликатами друг друга в любой произвольно-заданный момент времени.

Пусть R – отношение с атрибутами A1, A2, ..., An. Говорят, что множество атрибутов K=(Ai, Aj, ..., Ak) отношения R является возможным ключом R тогда и только тогда, когда удовлетворяются два независимых от времени условия:

1. Уникальность: в произвольный заданный момент времени никакие два различных кортежа R не имеют одного и того же значения для Ai, Aj, ..., Ak.
2. Минимальность: ни один из атрибутов Ai, Aj, ..., Ak не может быть исключен из K без нарушения уникальности.

Каждое отношение обладает хотя бы одним возможным ключом, поскольку по меньшей мере, комбинация всех его атрибутов удовлетворяет условию уникальности. Один из возможных ключей (выбранный произвольным образом) принимается за его первичный ключ. Остальные возможные ключи, если они есть, называются альтернативными ключами.

Вышеупомянутые и некоторые другие математические понятия явились теоретической базой для создания реляционных СУБД, разработки соответствующих языковых средств и программных систем, обеспечивающих их высокую производительность, и создания основ теории проектирования баз данных.

1. **Реляционная модель данных (РМД).Основные определения. Интерпретация отношения виде таблицы. Свойства табличного представления. Примеры.**

## Реляционная модель данных

История реляционных СУБД ведет свое начало с конца 60-х, когда одновременно несколькими авторами были выдвинуты предложения об использовании теоретико-множественных операторов для организации доступа к данным. Наиболее известными являются работы Кодда. Затем была экспериментальная система управления базами данных System R и использованный в ней язык SEQUEL, который можно считать непосредственным предшественником языка SQL. В настоящее время именно язык SQL является и де-юре, и де-факто стандартом для работы с реляционными СУБД. Например, семейство серверов реляционных баз данных Informix OnLine Dynamic Server поддерживают все эти стандарты и, кроме того, обеспечивают дополнительные возможности.

Интересно отметить, что еще в 1980 году Джефри Ульман (J.D. Ullman) писал в своей монографии "Основы систем баз данных" ("Principles of Databse Systems"), что почти все существующие коммерческие системы баз данных базируются либо на сетевой, либо на иерархической модели данных, но не реляционной модели и "эта ситуация будет меняться медленно". Тем не менее, уже в 1985 году ситуация резко поменялась - реляционные СУБД и язык SQL стали очень популярными. А в начале 90-х годов реляционные СУБД и язык SQL практически вытеснили все остальное с рынка СУБД. Причиной для такого кардинального изменения ситуации стала разработка эффективных, быстрых и надежных методов хранения и доступа к реляционным данным.

## Понятие отношения и таблицы

Давайте определим, что же такое реляционная СУБД и как в ней представляются данные. Для реляционной СУБД выбрано представление на основе математического понятия "отношение". Оно очень близко к знакомым всем нам понятию "таблица". По-английски отношение называется "relation", отсюда и название "реляционные СУБД (если называть такие базы "относительными", то это может исказить смысл).

Разница между таблицей в привычном для нас смысле и понятием отношения заключается в том, что в отношении нет порядка - это, вообще говоря, неупорядоченное множество записей. Это, конечно, не значит, что там совсем нет никакого порядка, просто он не подвластен ни программисту, ни администратору. Порядок определяется не отношением, а конкретной выборкой из отношения. Из одного и того же отношения я могу выбрать данные в порядке возрастания зарплаты, в алфавитном порядке фамилий и т.д.

В дальнейшем мы будем использовать все же термин "таблица", а не "отношение", так как этот термин понятен, привычен и самый распространенный язык доступа к реляционным базам данных - язык SQL - использует именно этот термин, хотя и понимает под ним "неупорядоченные" таблицы.

## Представление базы данных

Реляционная база данных - это набор информации, сгруппированной в одну или несколько таблиц. Таблицу можно представить как двумерный массив, или как набор записей одинаковой (для данной таблицы) структуры. Записи еще называют рядами. Другими словами, таблица состоит из рядов и столбцов. Число столбцов и записей для каждой таблицы теоретически неограниченно, хотя на практике ограничения, естественно, существуют. Превысить ограничения хорошего сервера базы данных практически невозможно - например, сервер баз данных Infomix OnLine DSA позволяет иметь до 32 767 столбцов и до 8 миллиардов записей в каждой таблице.

+-----+----+-----+-------+

¦ ¦ ¦ ¦ ¦ <-------- запись (ряд)

¦-----+----+-----+-------¦

¦ ¦XXXX¦ ¦ ¦ <-------- запись (ряд)

¦-----+-+--+-----+-------¦

¦ ¦ ¦

¦ L----------------+----> значение данного атрибута

¦ ................... ¦ для данной записи

¦-----+----+-----+-------¦

¦ ¦ ¦ ¦ ¦

+-----+----+-----+-------+

^ ^

¦ ¦

¦ L---------- столбец (атрибут, поле)

L-------------- столбец (атрибут, поле)

Рис.6. Структура таблицы (отношения).

Каждый столбец имеет определенный тип, неизменный для каждой записи внутри таблицы. Это может быть целое, дата, текст и т.д. Множество возможных значений конкретного столбца еще называют **доменом**. Важным для реляционной модели является требование того, чтобы значение каждого атрибута было атомарным, неделимым. Например, нельзя в качестве значения использовать массив целых. Если это правило не выполнено, то данную СУБД уже нельзя называть реляционной.

Каждый ряд в таблице описывает некий отдельный объект, поля содержат характеристики, значения неких признаков этих объектов. В таблице, которая, как мы уже отмечали, является набором записей, содержит записи, объединенные по какому-то признаку.

## Связь между таблицами

Итак, база данных - это набор таблиц. Как же обеспечить связь между таблицами? Связь между таблицами существует на мысленном, логическом уровне и определяется предметной областью. Практически связь между таблицами устанавливается за счет логически связанных данных в разных таблицах. В нашем примере с издательствами надо завести таблицу с описанием издательств, таблицу с описанием сборников, и т.д.

Таблица "Издательства":

|  |  |
| --- | --- |
| Название | Адрес |
| Бухгалтерия  Спорт  И т.д. | Москва,Ул.Правды,645  Москва, Ул.Советская, 897 |

Таблица "Сборники":

|  |  |
| --- | --- |
| Сборник | Издательство |
| Финансы и статистика  Все о налогах  Конный спорт  И т.д. | Бухгалтерия  Бухгалтерия  Спорт |

Рис.7. Структура некоторых таблиц из базы данных для издательств (отношения).

Теперь по названию сборника всегда можно определить название и адрес издательства (или нескольких издательств), его выпустившего. Для этого надо в таблице "Сборники" найти запись, у которой поле "Сборник" есть требуемое название, из найденной запись взять название издательства (атрибут “Издательство”), а потом в таблице "Издательства" найти нужный адрес.

Таким образом, для того, чтобы связать две таблицы мы использовали не жесткую, физически реализованную связь типа ссылки, а логическую связь через сравнение значений атрибутов, которую строили динамически, в процессе поиска нужной информации.

Основным достоинством реляционных СУБД, обеспечившим таким СУБД высокую популярность, является нефункциональность языков запроса, в частности, языка SQL. Это означает, что Вы формулируете не то, КАК Вам надо найти данные, а то, ЧТО Вам надо найти

1. **Определение понятия отношения и его элементов. Ключ отношения, его свойства. Представление объектов и связей инфологической модели в РМД. Примеры.**

*Отношение* - это ассоциация или "связь" между двумя сущностями. Отношение представляется в модели линией, соединяющей две сущности и глагольной конструкцией, которая описывает, как две сущности зависят друг от друга. Глагольная конструкция - механизм описания бизнес-правил, определяющих отношение. Хорошая глагольная конструкция описывает отношение в терминах бизнеса, а не на языке технических спецификаций.

Количество элементов отношения задает максимальное число экземпляров одной сущности, которые могут быть связаны с экземплярами другой сущности. Количество элементов определяется для обеих сторон отношения - для исходной и завершающей сущностей. Количество элементов определяет максимальное количество экземпляров сущностей, участвующих в отношении, в то время как обязательность определяет минимальное число экземпляров. Количество элементов часто выражается как *один* или ***много.*** Один и много могут появляться в трех различных комбинациях:

* **Один-к-одному** (1:1) - один и только один экземпляр сущности связан с одним и только одним экземпляром другой сущности.
* **Один-ко-многим** (1:N) - один и только один экземпляр родительской сущности связан со многими экземплярами подчиненной сущности.
* **Многие-ко-многим** (M:N) - много экземпляров одной сущности связаны с многими экземплярами другой сущности (также называется неспецифическим отношением).

**Ключи**

Что такое ключ? Набор столбцов. Он может состоять из одного столбца, или охватывать все столбцы таблицы. Для чего нам нужны ключи? Для идентификации строк таблицы. В чистой реляционной теории баз данных это единственный способ сослаться на строку. Ключи бывают разные - потенциальные, первичные, альтернативные, внешние, индексные, хеш-ключи, ключи сортировки, вторичные ключи, ключи шифрование и расшифровки и т.д. Но мы договаривались, что будем рассматривать только то, что нам понадобится в работе, вот и рассмотрим.

**Потенциальные (вероятностные) ключи.**Потенциальным ключом будем называть такую комбинацию столбцов, которая обладает следующими свойствами:

* *Уникальностью*В таблице нет двух разных строк с одиноковыми значениями в нашем потенциальном ключе.
* *Неизбыточностью.*Нельзя убрать один из столбцом из ключа, так, чтобы он не потерял уникльности.

Рассмотрим, например, такую таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ паспорта** | **Фамилия** | **Имя** | **Отчество** | **Должность** |
| 123456 | Иванов | Иван | Иванович | Директор |
| 234567 | Петров | Петр | Иванович | Его зам |
| 345678 | Сидорова | Мария | Ивановна | Секретарша |

Табличка у нас простая и небольшая. Но нам хватит.  
В данной таблице в качестве потенциального ключа можно рассматривать любой столбец. Но она у нас будет расширяться, так что будем смотреть в будущее.

Понятно, что отчество не может быть потенциальным ключом - есть совпадения. Фамилия - может, если только мы не планируем появления новых строк в таблице. Можно взять комбинацию фамилии и должности, вряд ли у нас будет два директора-однофамильца. Номер паспорта также подходит на роль потенциального ключа. Я думаю, вы поняли мою мысль - к каждой конкретной таблице потенциальнх ключей может быть много. Выбор потенциального ключа - дело программиста. Тот же номер паспорта может не подойти, если мы ожидаем кого-нибудь с поддельным паспортом. Выбор делается каждый раз заново для каждой ситуации.

**Первичные ключи**. Итак, с потенциальными ключами определились. Первичный ключ - это один из потенциальных ключей. Тот, который нам больше понравится. Вам какой больше нравиться? В реальной ситуации, новичок выберет номер паспорта. А что выберет профессионал? Профессионал добавит еще одно поле-счетчик, которое будет содержать уникальное для каждой записи значение. В Delphi такой тип поля называется AutoIncrement, в SQL Server есть целых 2 варианта - TimeStamp и свойство Identity поля.

**Альтернативные ключи**. Первичный ключ может быть только один на всю таблицу! После выбора первичного ключа из набора потенциальных ключей, оставшиеся ключи называются альтенативными. Это так, для знания терминологии. Пока нам о них больше ничего знать не надо.

**Внешние ключи**. Когда мы создаем какую-нибудь базу данных, например для начисления зарплаты, нам не удобно всех работников упоминать в одной таблице. Если, например, какой-нибудь из них упоминается там не один раз (зарплата, премия, надбавки, снятия, налоги и пр.), то при изменении его/ее фамилии надо будет пробежаться по всем строкам, и поменять все вхождения. Это неудобно. Так вот, имеем две таблицы:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | **Код работника** | **Вид движения** | **Сумма** | | 1 | Оклад | 100 | | 1 | Премия | 30 | | 1 | Налоги | -25 | | 2 | Оклад | 90 | | ... | ... | ... | | |  |  |  |  | | --- | --- | --- | --- | | **Код работника** | **Фимилия** | **Имя** | **Отчество** | | 1 | Иванов | Иван | Иванович | | 2 | Петров | Петр | Иванович | | 3 | Сидорова | Мария | Ивановна | |

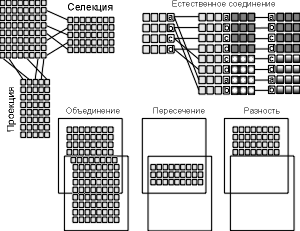
В первой таблице - с деньгами - столбец "Код работника" называется **внешним ключом**. Ясно, что он не может существовать без соответствующей строки из второй таблицы, в которой столбец "Код работника" - уже знакомый нам обычный первичный ключ. Вторая таблица - с фамилиями - является как бы "справочником фамилий" для первой.  
Хотя чистая реляционная теория требует, чтобы внешние ключи всегда ссылались на первичные ключи, мы это требование низведем до простой рекомендации: бывают ситуации, когда одна и та же таблица может служить справочником разным другим, причем в разном качестве. А первичный ключ, как мы знаем, может быть только один.

1. **Средства манипулирования данными (ЯМД), основанные на реляционной алгебре. Теоретико-множественные операции. Примеры.**

## Манипулирование реляционными данными

Предложив реляционную модель данных, Э.Ф. Кодд создал и инструмент для удобной работы с отношениями – реляционную алгебру. *Алгеброй* называется множество объектов с заданной на нем совокупностью операций, замкнутых относительно этого множества, называемого *основным множеством*. Основным множеством в реляционной алгебре является множество отношений. Всего Э.Ф. Коддом было предложено 8 операций. Каждая операция этой алгебры использует одну или несколько таблиц (отношений) в качестве ее операндов и продуцирует в результате новую таблицу, т.е. позволяет "разрезать" или "склеивать" таблицы (рис. 4.1). Все множество операций можно разделить на две группы: теоретико-множественные операции (в них входят 4 операции) и специальные операции. Три первые теоретико-множественные операции являются бинарными, т.е. в них участвуют два отношения и они требуют эквивалентных схем исходных отношений.

*Рис. 4.1. Некоторые операции реляционной алгебры*

Созданы языки манипулирования данными, позволяющие реализовать все операции реляционной алгебры и практически любые их сочетания. Среди них наиболее распространены SQL (Structured Query Language – *структуризованный язык запросов*) и QBE (Quere-By-Example – *запросы по образцу*). Оба относятся к языкам очень высокого уровня, с помощью которых пользователь указывает, какие данные необходимо получить, не уточняя процедуру их получения.

С помощью единственного запроса на любом из этих языков можно соединить несколько таблиц во временную таблицу и вырезать из нее требуемые строки и столбцы (селекция и проекция).

## Основные операции над таблицами и их интерпретация

### Теоретико-множественные операции реляционной алгебры

**Объединение** - операция, заключающаяся в построении нового отношения на основе двух других (все отношения должны иметь одинаковое число атрибутов). В объединение попадают все записи, как первого отношения, так и второго, и никакие другие записи.

Пусть заданы два отношения R1 = {r1}, R2 = {r2}, где r1 иr2 - соответственно кортежи отношений R1 и R2, то объединение R1 ∪ R2 = {r | r ∈ R1 ∨ r ∈ R2}. Здесь r - кортеж нового отношения, ∨ - операция логического сложения “ИЛИ”.

Например, отношения R1, R2 перечни деталей, изготавливаемых соответственно на первом и втором участках цеха. Отношение R3 содержит общий перечень деталей, изготавливаемых в цеху (общая номенклатура цеха).

Отношение R1

|  |  |
| --- | --- |
| Шифр детали | Название детали |
| 00011073 | Гайка М1 |
| 00011075 | Гайка М2 |
| 00011076 | Гайка М3 |
| 00011003 | Болт М1 |
| 00011006 | Болт М3 |
| 00013063 | Шайба М1 |
| 00013066 | Шайба М3 |

Отношение R2

|  |  |
| --- | --- |
| Шифр детали | Название детали |
| 00011073 | Гайка М1 |
| 00011076 | Гайка М3 |
| 00011077 | Гайка М4 |
| 00011004 | Болт М2 |
| 00011006 | Болт М3 |

Отношение R3

|  |  |
| --- | --- |
| Шифр детали | Название детали |
| 00011073 | Гайка М1 |
| 00011075 | Гайка М2 |
| 00011076 | Гайка М3 |
| 00011003 | Болт М1 |
| 00011006 | Болт М3 |
| 00013063 | Шайба М1 |
| 00013066 | Шайба М3 |
| 00011077 | Гайка М4 |
| 00011004 | Болт М2 |

Рис. 4.2. Объединение отношений R1 и R2

Совсем не обязательно, чтобы объединяемые таблицы, как и результирующая таблица, имели одинаковые названия атрибутов (в случае, если мы используем именованные атрибуты). Операция объединения позволяет собрать информацию из разных источников.

Например, если мы имеем отдельную таблицу со списком фирм-производителей товаров, и отдельную таблицу со списком фирм, которым мы поставляем некоторую продукцию, то, объединив эти две таблицы, мы можем получить список фирм-партнеров.

Таблица “Фирмы-производители”

|  |  |
| --- | --- |
| Производитель | Адрес |
| Вологда  Домик в деревне | Ул.1  Ул.2 465 |

Таблица “Фирмы-потребители”

|  |  |
| --- | --- |
| Потребитель | Адрес |
| АОО «Европа»  ОО «Мега» | Ул.4  Ул.3 |

Результирующая таблица “Фирмы-партнеры”

|  |  |
| --- | --- |
| Партнер | Адрес |
| Вологда  Домик в деревне  АОО «Европа»  ОО «Мега» | Ул.1  Ул.2  Ул.4  Ул.3 |

Рис. 4.3. Объединение двух таблиц

**Пересечением** отношений называется отношение, которое содержит множество кортежей, принадлежащих одновременно и первому и второму отношениям. R1 и R2:

R3 = R1 ∩ R2 = {r | r ∈ R1 ∧ r ∈ R2}, здесь ∧ - операция логического умножения (логическое “И”). В отношении R4 содержаться перечень деталей, которые выпускаются одновременно на двух участках цеха.

Отношение R4

|  |  |
| --- | --- |
| Шифр детали | Название детали |
| 00011073 | Гайка М1 |
| 00011076 | Гайка М3 |
| 00011006 | Болт М3 |

Рис. 4.4. Результат пересечения двух отношений

**Разностью** отношений R1 и R2 называется отношение, содержащее множество кортежей, принадлежащих R1 и не принадлежащих R2: R5 = R1 \ R2 = {r | r ∈R1 ∧ r ∉R2}.

Отношение R5 содержит перечень деталей, изготавливаемых только на участке 1, отношение R6 содержит перечень деталей, изготавливаемых только на участке 2. R6 = R2 \ R1 = {r | r ∈R2 ∧ r ∉R1}.

Отношение R5

|  |  |
| --- | --- |
| Шифр детали | Название детали |
| 00011075 | Гайка М2 |
| 00011003 | Болт М1 |
| 00013063 | Шайба М1 |
| 00013066 | Шайба М3 |

Отношение R6

|  |  |
| --- | --- |
| Шифр детали | Название детали |
| 00011077 | Гайка М4 |
| 00011004 | Болт М2 |

Рис. 4.5. Пример разности двух отношений

Операции объединение и пересечение являются коммутативными операциями, т.е. результат операции не зависит от порядка аргументов в операции. Операция разности является принципиально несимметричной операцией, т. е. результат операции будет различным для разного порядка аргументов, что и видно из сравнения отношений R5 и R6.

**Декартово произведение** - операция, заключающаяся в построении нового отношения на основе двух других путем по парной комбинацией всех возможных записей из первого отношения и второго отношения. Расширенным декартовым произведением отношения R1 степени n (атрибутов) со схемой, SR1 = (A1, A2,…, An) и отношения R2 степени m со схемой SR2 = (B1, B2,…, Bm) называется отношение R3 степени n+m со схемой SR3 = (A1, A2,…, An, B1, B2,…, Bm), содержащее кортежи, полученные сцеплением каждого кортежа r отношения R1 с каждым кортежем q отношения R2. Т.е., если R1 = {r}, R2 = {q}

R1⊕ R2 = {(r, q) | r ∈ R1 ∧ q ∈ R2}. В результирующей таблице будет r\*q записей и n + m атрибутов, т.е. получим отношение, которое характеризует все возможные комбинации между элементами различных множеств.

Декартово произведение позволяет связать разные объекты, объединить информацию, описывающую разные типы сущностей или объектов предметной области. Например, если мы имеем отдельную таблицу со списком фирм-производителей, и отдельную таблицу со списком всевозможных товаров, то декартово произведение этих двух таблиц позволит построить всевозможные пары “товар-производитель”:

Таблица “Фирмы-производители”

|  |  |
| --- | --- |
| Производитель | Адрес |
| Вологда  Домик в деревне | Ул.1  Ул.2 |

Таблица “Товары”

|  |
| --- |
| Товар |
| Кефир  Молоко |

Результирующая таблица “Товар-производитель”

|  |  |  |
| --- | --- | --- |
| Товар | Производитель | Адрес |
| Кефир  Молоко  Кефир  Молоко | Вологда  Вологда  Домик в деревне  Домик в деревне | Ул.1  Ул.1  Ул.2  Ул.2 |

Рис. 4.6. Декартово произведение двух таблиц

1. **ЯМД, основанный на реляционной алгебре. Специальные операции реляционной алгебры. Полная система операций реляционной алгебры. Примеры.**

Реляционная алгебра – это теоретический язык операций, позволяющих создавать на основе одного или нескольких отношений другое отношение без изменения исходных отношений. Оба операнда и результат являются отношениями, потому результаты одной операции могут применяться в другой операции – свойство замкнутости. Оно означает, что применение любого количества операций реляционной алгебры не приводит к созданию иных объектов, кроме отношений.

Реляционное исчисление эквивалентно реляционной алгебре. Реляционное исчисление в отличие от реляционной алгебры непроцедурный язык, и в основном используется для оценки избирательной мощности реляционных языков. Язык называется реляционно-полным, если он позволяет получить любое отношение, которое можно получить с помощью реляционного исчисления.

Первоначально Кодд предложил восемь операций, но впоследствии к ним были добавлены и некоторые другие. Пять основных операций реляционной алгебры, а именно выборка (selection), проекция (projection), декартово произведение (Cartesian product), объединение (union) и разность множеств (set difference), выполняют большинство действий по извлечению данных. На основании пяти основных операций можно также вывести дополнительные операции, такие как операции соединения (join), пересечения (intersection) и деления (division).

### Специальные операции реляционной алгебры

**Проекция** - операция, заключающаяся в удалении некоторых столбцов в отношении. Среди оставшихся записей с меньшим числом атрибутов может проводиться удаление дубликатов (одинаковых записей). Смысл операции проекции заключается в выделении из отношения той информации, которая нам нужна. Например, проекция отношения “Товары” по атрибутам “Производитель” и “Адрес” даст нам список всех поставщиков с их адресами:

|  |  |
| --- | --- |
| Производитель | Адрес |
| Вологда  Домик в деревне | Ул.1  Ул.2 |

Рис.4.7. Проекция таблицы “Товары” для получения списка фирм-производителей

**Селекция** - операция, заключающаяся в удалении некоторых записей в отношении на основе некоторого условия. Условие определяться как логическое выражение над значениями атрибутов. В этом логическом выражении можно использовать операции “больше”, “меньше”, “равно”, “принадлежит множеству” и т.д. Операция селекции, как и операция проекции, позволяет выделить из отношения ту информацию, которая нам нужна. Но, в отличие от операции проекции, выбираются подходящие записи, а не интересующие нас атрибуты.

Например, селекция отношения “Товары” по условию “*атрибут “Производитель” есть “Вологда””* позволит нам получить список всех товаров, поставляемых данной фирмой:

|  |  |  |
| --- | --- | --- |
| Товар | Производитель | Адрес |
| Кефир  Молоко  Сметана | Вологда  Вологда  Вологда | Ул.1  Ул.1  Ул.1 |

Рис.4.8. Селекция таблицы “Товары” по условию

Про тета-соединение и деление – см. лекции.

1. **Нормализация отношений, назначение и общая характеристика шагов нормализации. Понятие канонической схемы. Примеры.**

Процесс преобразования базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация позволяет обезопасить базу данных от логических и структурных проблем, называемых аномалиями данных. К примеру, когда существует несколько одинаковых записей в таблице, существует риск нарушения [целостности данных](http://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) при обновлении таблицы. Таблица, прошедшая нормализацию, менее подвержена таким проблемам, т.к. ее структура предполагает определение связей между данными, что исключает необходимость в существовании записей с повторяющейся информацией.

1. **1-ая нормальная форма (1НФ) отношения.Определение. Метод приведения отношения к 1НФ.**

### Первая нормальная форма (1NF)

**Таблица находится в первой нормальной форме, если каждый её атрибут атомарен.** Под выражением «атрибут атомарен» понимается, что атрибут может содержать только одно значение. Таким образом, не соответствуют 1NF таблицы, в полях которых могут храниться списки значений. Для приведения таблицы к 1NF обычно требуется разбить таблицу на несколько отдельных таблиц.

Замечание: в реляционной модели отношение всегда находится в 1 (или более высокой) нормальной форме в том смысле, что иные отношения не рассматриваются в реляционной модели. То есть само определение понятия отношение заведомо подразумевает наличие 1NF.

Пример приведения таблицы к первой нормальной форме

|  |  |
| --- | --- |
| Сотрудник | Номер телефона |
| Иванов И. И. | 283-56-82 |
| Иванов И. И. | 390-57-34 |
| Петров П. Ю. | 708-62-34 |

Исходная, ненормализованная, таблица:

|  |  |
| --- | --- |
| Сотрудник | Номер телефона |
| Иванов И. И. | 283-56-82 390-57-34 |
| Петров П. Ю. | 708-62-34 |

Таблица, приведённая к 1NF:

1. **Понятие функциональной зависимости (ФЗ) в отношениях. Свойства и аксиомы ФЗ. Примеры.**

### Функциональные зависимости.

Реляционная база данных содержит как структурную, так и семантическую информацию. Структура базы данных определяется числом и видом включенных в нее отношений, и связями типа "один ко многим", существующими между кортежами этих отношений. Семантическая часть описывает множество функциональных зависимостей, существующих между атрибутами этих отношений. Дадим определение функциональной зависимости.

**Определение:** 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, составленные из нескольких атрибутов одного отношения. Можно сказать, что функциональные зависимости представляют собой связи типа "один ко многим", существующие внутри отношения.

1. **2-аянормальная форма (2НФ) отношения. Определение полной функциональной зависимости и 2НФ. Характеристика отношения во 2НФ. Алгоритм приведения ко 2НФ. Теорема Хита. Примеры.**

**Понятие полной функциональной зависимости.**

Определение: неключевой атрибут **функционально полно зависит** от составного ключа если он функционально зависит от всего ключа в целом, но не находится в функциональной зависимости от какого-либо из входящих в него атрибутов.

Определение: **избыточная функциональная зависимость** - зависимость, заключающая в себе такую информацию, которая может быть получена на основе других зависимостей, имеющихся в базе данных.

### 2NF - вторая нормальная форма.

Определение второй нормальной формы: отношение находится во **2НФ**, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

Корректной считается такая схема базы данных, в которой отсутствуют избыточные функциональные зависимости. В противном случае приходится прибегать к процедуре декомпозиции (разложения) имеющегося множества отношений. При этом порождаемое множество содержит большее число отношений, которые являются проекциями отношений исходного множества. (Операция проекции описана в разделе, посвященном реляционной алгебре). Обратимый пошаговый процесс замены данной совокупности отношений другой схемой с устранением избыточных функциональных зависимостей называется нормализацией.

Условие обратимости требует, чтобы декомпозиция сохраняла эквивалентность схем при замене одной схемы на другую, т.е. в результирующих отношениях:

1)не должны появляться ранее отсутствовавшие кортежи;

2)на отношениях новой схемы должно выполняться исходное множество функциональных зависимостей.

#### Теорема Хита

Пусть дано отношение r\left( A,B,C \right).

Если *r* удовлетворяет функциональной зависимости A\to B, то оно равно соединению его проекций r\left[ A,B 
\right]и r\left[ A,C \right]\left( A\to B \right)\Rightarrow \left( 
r\left( A,B,C \right)=r\left[ A,B \right]\ \text{JOIN}\ r\left[ A,C 
\right] \right)

1. **3-я нормальная форма (3НФ) отношения. Определение транзитивной зависимости и 3НФ.Алгоритм приведения к 3НФ.Нормальная форма Бойса-Кодда (НФБК).Определение и алгоритм приведения к НФБК. Характеристика отношения в 3НФ и в НФБК. Примеры.**

### [Третья нормальная форма (3NF)](http://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D1%82%D1%8C%D1%8F_%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0)

Таблица находится в третьей нормальной форме, если она находится во второй нормальной форме, и при этом любой её неключевой атрибут функционально зависит только от первичного ключа. Или что тоже самое - "нет зависимостей неключевых атрибутов от других неключевых атрибутов + 2НФ".

При решении практических задач в большинстве случаев третья нормальная форма является достаточной. Процесс проектирования реляционной базы данных, как правило, заканчивается приведением к 3NF.

Пример приведения таблицы к третьей нормальной форме

|  |  |  |
| --- | --- | --- |
| Фамилия | Отдел | Телефон |
| Гришин | 1 | 11-22-33 |
| Васильев | 1 | 11-22-33 |
| Петров | 2 | 44-55-66 |

В результате приведения к 3NF получим две таблицы:

|  |  |
| --- | --- |
| Фамилия | Отдел |
| Гришин | 1 |
| Васильев | 1 |
| Петров | 2 |

|  |  |
| --- | --- |
| Отдел | Телефон |
| 1 | 11-22-33 |
| 2 | 44-55-66 |

### BCNF - нормальная форма Бойса-Кодда.

Эта нормальная форма вводит дополнительное ограничение по сравнению с 3НФ.   
Определение нормальной формы Бойса-Кодда:

Отношение находится в BCNF, если оно находится во 3НФ и в ней отсутствуют зависимости атрибутов первичного ключа от неключевых атрибутов.

Ситуация, когда отношение будет находится в 3NF, но не в BCNF, возникает при условии, что отношение имеет два (или более) возможных ключа, которые являются составными и имеют общий атрибут. Заметим, что на практике такая ситуация встречается достаточно редко, для всех прочих отношений 3NF и BCNF эквивалентны.

1. **Многозначные зависимости (МЗ). Определение. Свойства и аксиомы МЗ. Четвертая нормальная форма (4НФ) отношения. Характеристика отношения в 4НФ.**

### Многозначные зависимости и четвертая нормальная форма (4NF).

Четвертая нормальная форма касается отношений, в которых имеются повторяющиеся наборы данных. Декомпозиция, основанная на функциональных зависимостях, не приводит к исключению такой избыточности. В этом случае используют декомпозицию, основанную на многозначных зависимостях.

**Многозначная зависимость** является обобщением функциональной зависимости и рассматривает соответствия между множествами значений атрибутов.

В качестве примера рассмотрим отношение ПРЕПОДАВАТЕЛЬ (ИМЯ, КУРС, УЧЕБНОЕ\_ПОСОБИЕ), хранящее сведения о курсах, читаемых преодавателем, и написанных им учебниках. Пусть профессор N читает курсы "Теория упругости" и "Теория колебаний" и имеет соответствующие учебные пособия, а профессор K читает курс "Теория удара" и является автором учебников "Теория удара" и "Теоретическая механика". Тогда наше отношение будет иметь вид:

----------------------------------------------------

| ИМЯ | КУРС | УЧЕБНОЕ\_ПОСОБИЕ |

----------------------------------------------------

| N | Теория упругости | Теория упругости |

| N | Теория колебаний | Теория упругости |

| N | Теория упругости | Теория колебаний |

| N | Теория колебаний | Теория колебаний |

| K | Теория удара | Теория удара |

| K | Теория удара | Теоретическая механика |

----------------------------------------------------

добавляем:

----------------------------------------------------

| K | Теория упругости | Теория удара |

| K | Теория упругости | Теоретическая механика |

----------------------------------------------------

Это отношение имеет значительную избыточность и его использование приводит к возникновению аномалии обновления. Например, добавление информации о том, что профессор K будет также читать лекции по курсу "Теория упругости" приводит к необходимости добавить два кортежа (по одному для каждого написанного им учебника) вместо одного. Тем не менее, отношение ПРЕПОДАВАТЕЛЬ находится в NFBC (ключевой атрибут - ИМЯ).

Заметим, что указанные аномалии исчезают при замене отношения ПРЕПОДАВАТЕЛЬ его проекциями:

--------------------------- -------------------------------

| ИМЯ | КУРС | | ИМЯ | УЧЕБНОЕ\_ПОСОБИЕ |

--------------------------- -------------------------------

| N | Теория упругости | | N |Теория упругости |

| N | Теория колебаний | | N |Теория колебаний |

| K | Теория удара | | K |Теоретическая механика |

| K | Теория упругости | | K |Теория удара |

--------------------------- -------------------------------

Аномалия обновления возникает в данном случае потому, что в отношении ПРЕПОДАВАТЕЛЬ имеются:

1. зависимость множества значений атрибута КУРС от множества значений атрибута ИМЯ
2. зависимость множества значений атрибута УЧЕБНОЕ\_ПОСОБИЕ от множества значений атрибута ИМЯ.

Такие зависимости и называются многозначными и обозначаются как

ИМЯ ->> КУРС ИМЯ ->> УЧЕБНОЕ\_ПОСОБИЕ

Нетрудно показать, что многозначные зависимости всегда образуют связанные пары, поэтому их часто обозначают

ИМЯ ->> КУРС | УЧЕБНОЕ\_ПОСОБИЕ

Очевидно, что каждая функциональная зависимость является многозначной, но не каждая многозначная зависимость является функциональной.

Определение четвертой нормальной формы:

Отношение находится в 4NF если оно находится в BCNF и в нем отстутсвуют многозначные зависимости, не являющиеся функциональными зависимостями.

1. **Общая характеристика языка SQL. Стандарты SQL, способы его реализации. Структура языка SQL.**

Язык SQL, предназначенный для взаимодействия с базами данных, появился в середине 70-х гг. и был разработан в компании IBM в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structured English Query Language) только частично отражало суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционным БД. Но, в действительности, он почти с самого начала являлся **полным языком БД**, обеспечивающим помимо средств формулирования запросов и манипулирования БД следующие возможности:

* средства определения и манипулирования схемой БД;
* средства определения ограничений целостности и триггеров;
* средства определения представлений БД;
* средства определения структур физического уровня, поддерживающих эффективное выполнение запросов;
* средства авторизации доступа к отношениям и их полям;
* средства определения точек сохранения транзакции, и выполнения фиксации и откатов транзакций.

В языке отсутствовали средства явной синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

#### Структура языка SQL

Язык SQL, соответствующий последним стандартам SQL:2003, SQL:1999 (и даже SQL/92), это очень богатый и сложный язык, все возможности которого трудно сразу осознать и тем более понять. Поэтому приходится разбивать язык на уровни, или слои, такие, что каждый уровень языка включает все конструкции, входящие в более низкие уровни. В стандарте определяется несколько способов разбиения языка на уровни. В одной из классификаций язык разбивается на «базовый» (entry), «промежуточный» (intermediate) и «полный» (full) уровни.

Эта классификация ориентирована, прежде всего, на производителей СУБД, в которых поддерживается SQL. Реализация базового уровня языка является обязательным условием хотя бы какого-то соответствия стандарту. Реализация промежуточного уровня желательна, и обычно именно такой уровень языка поддерживается ведущими компаниями-производителями SQL-ориентированных СУБД. Наконец, полный уровень языка является целью, к достижению которой следует стремиться. В данной классификации критерием отнесения той или иной возможности языка к некоторому уровню является оцениваемая создателями стандарта SQL (большая часть которых является сотрудниками ведущих компаний, производящих SQL-ориентированные СУБД) техническая сложность реализации этой возможности. Конечно, такая классификация важна и для программистов приложений баз данных, но только для того, чтобы оценить реальные возможности конкретной СУБД. Для понимания языка SQL это разбиение на уровни несущественно.

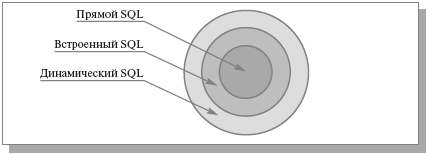


Рис. Один из способов разделения языка SQL на уровни

Другая классификация показана на рисунке. Среди всех конструкций языка SQL можно выделить такие конструкции, которые можно было использовать при «прямом» (direct) взаимодействии конечного пользователя с СУБД (например, в интерактивном режиме). В некотором смысле этот уровень также является базовым, поскольку соответствующие средства языка в наибольшей степени отражают его ориентированность на работу с множествами. На следующем уровне, уровне «встраиваемого» (embedded) SQL, язык расширяется конструкциями, позволяющими использовать возможности прямого SQL в программах, написанных на традиционных языках программирования. Наконец, на уровне «динамического» (dynamic) SQL во встраиваемый SQL добавляются конструкции, позволяющие приложениям обращаться к СУБД с конструкциями прямого SQL, которые динамически образуются во время выполнения программы.

1. **Операторы ЯМД в Т-SQL: состав и назначение. Примеры.**

**INSERT — осуществляет вставку строк в таблицу.**

**DELETE — осуществляет удаление строк из таблицы.**

**UPDATE — осуществляет модификацию данных в таблице.**

**SELECT — осуществляет выборку данных из таблиц по запросу.**

Все допустимые в SQL типы данных, которые можно использовать при определении столбцов, разбиваются на следующие категории:

* точные числовые типы (exact numerics);
* приближенные числовые типы (approximate numerics);
* типы символьных строк (character strings);
* типы битовых строк (bit strings);
* типы даты и времени (datetimes);
* типы временных интервалов (intervals);
* булевский тип (Booleans);
* типы коллекций (collection types);
* анонимные строчные типы (anonymous row types);
* типы, определяемые пользователем (user-defined types);
* ссылочные типы (reference types).

В столбцах таблиц, определенных на любых типах данных, наряду со значениями этих типов, допускается сохранение неопределенного значения, которое обозначается ключевым словом NULL. В языке определено, что результатом выражений вида x a\_op NULL, NULL a\_op x, NULL a\_op NULL является NULL для всех арифметических операций a\_op (+, - и т. д.), допустимых для типа данных выражения x (выражение NULL a\_op NULL является допустимым для любой арифметической операции a\_op). Также по определению полагается, что значением выражений x comp\_op NULL, NULL comp\_op x, NULL comp\_op NULL для всех операций сравнения (=, ne, >, < и т. д.), определенных для типа выражения x, является третье логическое значение unknown[4)](http://www.intuit.ru/department/database/dbbase/11/footnote.2.4.htm) (выражение NULL comp\_op NULL является допустимым для любой операции сравнения comp\_op).

1. **Способы определения правил целостности БД в Т-SQL. Задание правил целостности на уровне домена и таблицы.**

За контролем целостности данных следит сервер, и при нарушении правил целостности данных сервер известит клиента об ошибке.

К структурам контроля целостности данных относятся ограничители (constraint), которые привязаны к столбцам и триггеры (trigger), которые могут быть привязаны как к столбцам, так и к строкам в таблице.

**Ограничители** это элементарные проверки или условия, которые выполняются для операций вставки и модификации значения столбца. Если данная проверка не проходит или условие не выполняется, то вставка или модификация отменяется, а в программу клиента передается ошибка. SQL-серверы, как правило, поддерживают следующие ограничители.

* **NOT NULL** - проверка на непустое значение. NULL - специальное понятие в СУБД, которое означает "пусто". "Пусто" и "0(ноль)" не равны друг другу!
* **UNIQUE** - проверка на уникальность. Вставляемое значение должно быть уникально для данного столбца по всей таблице. Может содержать пустые значения.
* **PRIMARY KEY** - первичный ключ. Значение в столбце считается первичным ключом, если оно непустое и уникально в пределах столбца данной таблицы. Первичный ключ может быть составным и представлять собой комбинацию столбцов. Тогда чтобы считаться первичным ключом, каждое из группы значений не должно быть пустыми и формируемые строки значений первичного ключа должны быть уникальны в пределах таблицы. Первичный ключ - основа для построения индексов по таблице.

SQL-технология позволяет на уровне столбца задавать домены значений, т.е. строго определенные наборы или диапазоны значений, для помещаемых в столбец данных. В частности можно реализовывать ограничения ссылочной целостности (referential integrity constraint) и проверки фиксированного условия. Ограничение ссылочной целостности не позволяет значениям из столбца одной таблицы принимать значения кроме как из присутствующих в столбце другой таблицы. Это делается при помощи ограничителей FOREIGN KEY (внешний ключ) и REFERENCES (указатель ссылки). Таблица, содержащая FOREIGN KEY, считается родительской таблицей. Таблица, содержащая REFERENCES, считается дочерней таблицей. Внешний ключ и указатель ссылки могут находиться в одной таблице, т.е. родительская таблица одновременно является дочерней.

* **FOREIGN KEY** - внешний ключ. Назначает столбец или комбинацию столбцов в текущей (родительской) таблице в качестве внешнего ключа для ссылки из других таблиц.
* **REFERENCES** - указатель ссылки (или родительский ключ). Указывает на столбец (комбинацию столбцов) в родительской таблице, ограничивающую значения в текущей (дочерней) таблице.

Для использования ограничений ссылочной целостности должны выполняться некоторые условия. В частности, родительская и дочерняя таблицы должны находиться в пределах одного аппаратного сервера базы данных, они не могут находиться на различных узлах распределенной базы данных. Столбцы, участвующие в отношении ограничения ссылочной целостности обязаны иметь один и тот же тип данных. Ограничения ссылочной целостности используются при каскадном удалении, т.е. при удалении записи в родительской таблице удаляются все записи с указанным ключом из дочерних таблиц, и наоборот, при запрете удаления/модификации, т.е. при наличии зависимых записей в дочерних таблицах, значение ключа записи в родительской таблице нельзя удалить или модифицировать.

* **CHECK** - проверка фиксированного условия. В данном ограничителе явно указывается условие, которое должно выполняться для вставляемого или модифицируемого значения в столбце. Например: check (user in 'ALEX','JUSTAS') - в столбце user могут содержаться только значения 'ALEX' и 'JUSTAS', попытка вставки значения 'SHTIRLITZ' будет интерпретирована как ошибочная , check (user\_salary between 1000 and 5000) - столбец user\_salary может принимать целочисленные значения в диапазоне от 1000 до 5000 и т.д. При формировании условий с некоторыми ограничениями могут использоваться функции, например check (user = upper(user)), в данном случае имя пользователя должно вводиться только в верхнем регистре. Есть и ограничения, например, CHECK не может содержать подзапросы (SELECT).

Обычно ограничители задаются при создании таблиц. Но в дальнейшем их можно изменять, удалять или временно запрещать при помощи соответствующих команд СУБД.

1. **Т-SQL. Хранимые процедуры и их назначение. Типы хранимых процедур. Операторы создания, запуска, изменения и удаления хранимых процедур. Пример хранимой процедуры.**

Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде. Выполнение в базе данных хранимых процедур вместо отдельных операторов SQL дает пользователю следующие **преимущества:**

* необходимые операторы уже содержатся в базе данных;
* все они прошли этап синтаксического анализа и находятся в исполняемом формате; перед выполнением хранимой процедуры SQL Server генерирует для нее план исполнения, выполняет ее оптимизацию и компиляцию;
* хранимые процедуры поддерживают модульное программирование, так как позволяют разбивать большие задачи на самостоятельные, более мелкие и удобные в управлении части;
* хранимые процедуры могут вызывать другие хранимые процедуры и функции;
* хранимые процедуры могут быть вызваны из прикладных программ других типов;
* как правило, хранимые процедуры выполняются быстрее, чем последовательность отдельных операторов;
* хранимые процедуры проще использовать: они могут состоять из десятков и сотен команд, но для их запуска достаточно указать всего лишь имя нужной хранимой процедуры. Это позволяет уменьшить размер запроса, посылаемого от клиента на сервер, а значит, и нагрузку на сеть.

Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Они вызываются клиентской программой, другой хранимой процедурой или триггером. Разработчик может управлять правами доступа к хранимой процедуре, разрешая или запрещая ее выполнение. Изменять код хранимой процедуры разрешается только ее владельцу или члену фиксированной роли базы данных. При необходимости можно передать права владения ею от одного пользователя к другому.

#### Типы хранимых процедур

В SQL Server имеется несколько типов хранимых процедур.

* Системные хранимые процедуры предназначены для выполнения различных административных действий. Практически все действия по администрированию сервера выполняются с их помощью. Можно сказать, что системные хранимые процедуры являются интерфейсом, обеспечивающим работу с системными таблицами, которая, в конечном счете, сводится к изменению, добавлению, удалению и выборке данных из системных таблиц как пользовательских, так и системных баз данных. Системные хранимые процедуры имеют префикс sp\_, хранятся в системной базе данных и могут быть вызваны в контексте любой другой базы данных.
* Пользовательские хранимые процедуры реализуют те или иные действия. Хранимые процедуры – полноценный объект базы данных. Вследствие этого каждая хранимая процедура располагается в конкретной базе данных, где и выполняется.
* Временные хранимые процедуры существуют лишь некоторое время, после чего автоматически уничтожаются сервером. Они делятся на локальные и глобальные. Локальные временные хранимые процедуры могут быть вызваны только из того соединения, в котором созданы. При создании такой процедуры ей необходимо дать имя, начинающееся с одного символа #. Как и все временные объекты, хранимые процедуры этого типа автоматически удаляются при отключении пользователя, перезапуске или остановке сервера. Глобальные временные хранимые процедуры доступны для любых соединений сервера, на котором имеется такая же процедура. Для ее определения достаточно дать ей имя, начинающееся с символов ##. Удаляются эти процедуры при перезапуске или остановке сервера, а также при закрытии соединения, в контексте которого они были созданы.

#### Создание, изменение и удаление хранимых процедур

Создание хранимой процедуры предполагает решение следующих задач:

* определение типа создаваемой хранимой процедуры: временная или пользовательская. Кроме этого, можно создать свою собственную системную хранимую процедуру, назначив ей имя с префиксом sp\_ и поместив ее в системную базу данных. Такая процедура будет доступна в контексте любой базы данных локального сервера;
* планирование прав доступа. При создании хранимой процедуры следует учитывать, что она будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь;
* определение параметров хранимой процедуры. Подобно процедурам, входящим в состав большинства языков программирования, хранимые процедуры могут обладать входными и выходными параметрами;
* разработка кода хранимой процедуры. Код процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

{CREATE | ALTER } PROC[EDURE] имя\_процедуры [;номер]

[{@имя\_параметра тип\_данных } [VARYING ] [=default][OUTPUT] ][,...n]

[WITH { RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION }]

[FOR REPLICATION]

AS

sql\_оператор [...n]

**Удаление** хранимой процедуры осуществляется командой:

DROP PROCEDURE {имя\_процедуры} [,...n]

Для ***выполнения*** *хранимой процедуры* используется команда:

[[ EXEC [ UTE] имя\_процедуры [;номер]

[[@имя\_параметра=]{значение | @имя\_переменной}

[OUTPUT ]|[DEFAULT ]][,...n]

Для ***обращения*** *к процедуре* можно использовать команды:

EXEC my\_proc1 или my\_proc1

**Пример** *Процедура без параметров*. Создать процедуру для уменьшения цены товара первого сорта на 10%.

CREATE PROC my\_proc2

AS

UPDATE Товар SET Цена=Цена\*0.9

WHERE Сорт=’первый’

1. **Т-SQL. Курсоры: назначение, описание, применение. Пример**

**Курсор** — ссылка на контекстную область памяти. В некоторых реализациях [информационно-логического языка](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) [SQL](http://ru.wikipedia.org/wiki/SQL) ([Oracle](http://ru.wikipedia.org/wiki/Oracle_%28%D0%A1%D0%A3%D0%91%D0%94%29), [Microsoft SQL Server](http://ru.wikipedia.org/wiki/Microsoft_SQL_Server)) — получаемый при выполнении [запроса](http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BF%D1%80%D0%BE%D1%81) результирующий набор и связанный с ним указатель текущей записи.

В [PL/SQL](http://ru.wikipedia.org/wiki/PL/SQL) поддерживаются два типа курсоров: явные и неявные. Явный курсор объявляется разработчиком, а неявный курсор не требует объявления.

Курсор может возвращать одну строку, несколько строк или ни одной строки. Для запросов, возвращающих более одной строки, можно использовать только явный курсор. Для повторного создания результирующего набора для других значений параметров курсор следует закрыть, а затем повторно открыть.

Курсор может быть объявлен в секциях объявлений любого блока [PL/SQL](http://ru.wikipedia.org/wiki/PL/SQL), подпрограммы или пакета.

### Операторы управления явным курсором

* Оператор CURSOR выполняет объявление явного курсора.
* Оператор OPEN открывает курсор, создавая новый результирующий набор на базе указанного запроса.
* Оператор FETCH выполняет последовательное извлечение строк из результирующего набора от начала до конца.
* Оператор CLOSE закрывает курсор и освобождает занимаемые им ресурсы

### Атрибуты курсора

* %ISOPEN — возвращает значение TRUE, если курсор открыт.
* %FOUND — определяет, найдена ли строка, удовлетворяющая условию.
* %NOTFOUND — возвращает TRUE, если строка не найдена.
* %ROWCOUNT — возвращает номер текущей строки.

1. **Т-SQL. Триггеры и их назначение. Типы триггеров. Операторы создания, изменения, включения/отключения, удаления триггеров. Ограничения использования триггеров. Примеры.**

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных. И все же их использование часто связано с дополнительными затратами ресурсов на операции ввода/вывода. В том случае, когда тех же результатов (с гораздо меньшими непроизводительными затратами ресурсов) можно добиться с помощью хранимых процедур или прикладных программ, применение триггеров нецелесообразно.

Каждый триггер привязывается к конкретной таблице. Все производимые им модификации данных рассматриваются как одна транзакция. В случае обнаружения ошибки или нарушения целостности данных происходит откат этой транзакции. Тем самым внесение изменений запрещается. Отменяются также все изменения, уже сделанные триггером.

Создает триггер только владелец базы данных. Это ограничение позволяет избежать случайного изменения структуры таблиц, способов связи с ними других объектов и т.п.

Триггер представляет собой весьма полезное и в то же время опасное средство. Так, при неправильной логике его работы можно легко уничтожить целую базу данных, поэтому триггеры необходимо очень тщательно отлаживать.

В отличие от обычной подпрограммы, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же он не имеет аргументов. Приведение его в действие иногда называют запуском триггера. С помощью триггеров достигаются следующие цели:

* проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
* выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
* накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;
* поддержка репликации.

Основной формат команды CREATE TRIGGER показан ниже:

CREATE TRIGGER имя\_триггера

INSTEAD OF | AFTER <триггерное\_событие>

ON <имя\_таблицы>

[REFERENCING

<список\_старых\_или\_новых\_псевдонимов>]

[FOR EACH { ROW | STATEMENT}]

[WHEN(условие\_триггера)]

<тело\_триггера>

Триггерные события состоят из вставки, удаления и обновления строк в таблице. В последнем случае для триггерного события можно указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов BEFORE (триггер запускается до выполнения связанных с ним событий) или AFTER (после их выполнения).

#### Типы триггеров

В SQL Server существует два параметра, определяющих поведение триггеров:

* **AFTER**. Триггер выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной транзакции: если произойдет откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp\_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER-триггерами.
* **INSTEAD OF**. Триггер вызывается вместо выполнения команд. В отличие от AFTER-триггера INSTEAD OF-триггер может быть определен как для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Триггеры различают по типу команд, на которые они реагируют.

Существует три типа триггеров:

* **INSERT TRIGGER** – запускаются при попытке вставки данных с помощью команды INSERT.
* **UPDATE TRIGGER** – запускаются при попытке изменения данных с помощью команды UPDATE.
* **DELETE TRIGGER** – запускаются при попытке удаления данных с помощью команды DELETE.

Конструкции [ DELETE] [,] [ INSERT] [,] [ UPDATE] и FOR | AFTER | INSTEAD OF } { [INSERT] [,] [UPDATE] определяют, на какую команду будет реагировать триггер. При его создании должна быть указана хотя бы одна команда. Допускается создание триггера, реагирующего на две или на все три команды.

Аргумент WITH APPEND позволяет создавать несколько триггеров каждого типа.

При создании триггера с аргументом NOT FOR REPLICATION запрещается его запуск во время выполнения модификации таблиц механизмами репликации.

Конструкция AS sql\_оператор[...n] определяет набор SQL- операторов и команд, которые будут выполнены при запуске триггера.

Отметим, что внутри триггера не допускается выполнение ряда операций, таких, например, как:

* создание, изменение и удаление базы данных;
* восстановление резервной копии базы данных или журнала транзакций.

Выполнение этих команд не разрешено, так как они не могут быть отменены в случае отката транзакции, в которой выполняется триггер. Это запрещение вряд ли может каким-то образом сказаться на функциональности создаваемых триггеров. Трудно найти такую ситуацию, когда, например, после изменения строки таблицы потребуется выполнить восстановление резервной копии журнала транзакций.

**программирование триггера**

При выполнении команд добавления, изменения и удаления записей сервер создает две специальные таблицы: *inserted* и *deleted*. В них содержатся списки строк, которые будут вставлены или удалены по завершении транзакции. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется *триггер*. Для каждого *триггера* создается свой комплект таблиц inserted и deleted, поэтому никакой другой *триггер* не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение *триггера*, содержимое таблиц inserted и deleted может быть разным:

* команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения *триггера* все строки из таблицы inserted переместятся в исходную таблицу;
* команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; *триггер* может проверить каждую строку и определить, разрешено ли ее удаление; в таблице inserted не окажется ни одной строки;
* команда UPDATE – при ее выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении *триггера*. Новые значения строк содержатся в таблице inserted. Эти строки добавятся в исходную таблицу после успешного выполнения *триггера*.

Для *удаления триггера* используется команда

DROP TRIGGER {имя\_триггера} [,...n]

1. **Т-SQL. Ссылочная целостность. Правила ссылочной целостности и поддержка их с помощью триггеров. Примеры.**

При создании баз данных большое внимание должно быть уделено средствам поддержания данных в целостном состоянии. Рассмотрим предусмотренные стандартом языка SQL функции, которые предназначены для поддержания целостности данных. Эта поддержка включает средства задания ограничений, они вводятся с целью защиты базы данных от нарушения согласованности сохраняемых в ней данных. К таким типам поддержки целостности данных относятся:

* обязательные данные;
* ограничения для доменов полей;
* целостность сущностей;
* ссылочная целостность;
* требования конкретного предприятия.

#### Ссылочная целостность

Внешние ключи представляют собой столбцы или наборы столбцов, предназначенные для связывания каждой из строк дочерней таблицы, содержащей этот внешний ключ, со строкой родительской таблицы, содержащей соответствующее значение потенциального ключа. Стандарт SQL предусматривает механизм определения внешних ключей с помощью предложения FOREIGN KEY, а фраза REFERENCES определяет имя родительской таблицы, т.е. таблицы, где находится соответствующий потенциальный ключ. При использовании этого предложения система отклонит выполнение любых операторов INSERT или UPDATE, с помощью которых будет предпринята попытка создать в дочерней таблице значение внешнего ключа, не соответствующее одному из уже существующих значений потенциального ключа родительской таблицы. Когда действия системы выполняются при поступлении операторов UPDATE и DELETE, содержащих попытку обновить или удалить значение потенциального ключа в родительской таблице, которому соответствует одна или более строк дочерней таблицы, то они зависят от правил поддержки ссылочной целостности, указанных во фразах ON UPDATE и ON DELETE предложения FOREIGN KEY. Если пользователь предпринимает попытку удалить из родительской таблицы строку, на которую ссылается одна или более строк дочерней таблицы, язык SQL предоставляет следующие возможности:

* CASCADE - выполняется удаление строки из родительской таблицы, сопровождающееся автоматическим удалением всех ссылающихся на нее строк дочерней таблицы;
* SET NULL - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы записывается значение NULL;
* SET DEFAULT - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносится значение, принимаемое по умолчанию;
* NO ACTION - операция удаления строки из родительской таблицы отменяется. Именно это значение используется по умолчанию в тех случаях, когда в описании внешнего ключа фраза ON DELETE опущена.

Те же самые правила применяются в языке SQL и тогда, когда значение потенциального ключа родительской таблицы обновляется.

Определитель MATCH позволяет уточнить способ обработки значения NULL во внешнем ключе.

При определении таблицы предложение FOREIGN KEY может указываться произвольное количество раз.

В операторе CREATE TABLE используется необязательная фраза DEFAULT, которая предназначена для задания принимаемого по умолчанию значения, когда в операторе INSERT значение в данном столбце будет отсутствовать.

Фраза CONSTRAINT позволяет задать имя ограничению, что позволит впоследствии отменить то или иное ограничение с помощью оператора ALTER TABLE.

1. **Т-SQL. Персональные, списковые и количественные запросы. Агрегатные функции. Особенности использования фразы group by. Реализация количественного запроса по одному или нескольким столбцам с использованием Т-SQL. Примеры.**

**Агрегатные функции**

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(\*), фактический (т.е. требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций - null.

Вычисление функции COUNT(\*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением null во всех строках.

Стандартом предусмотрены следующие агрегатные функции:

|  |  |
| --- | --- |
| Функция | Описание |
| COUNT(\*) | Возвращает количество строк источника записей. |
| COUNT(<имя поля>) | Возвращает количество значений в указанном столбце. |
| SUM(<имя поля>) | Возвращает сумму значений в указанном столбце. |
| AVG(<имя поля>) | Возвращает среднее значение в указанном столбце. |
| MIN(<имя поля>) | Возвращает минимальное значение в указанном столбце. |
| MAX(<имя поля>) | Возвращает максимальное значение в указанном столбце. |

Все эти функции возвращают единственное значение. При этом функции COUNT, MIN и MAX применимы к любым типам данных, в то время как SUM и AVG используются только для числовых полей. Разница между функцией COUNT(\*) и COUNT(<имя поля>) состоит в том, что вторая при подсчете не учитывает NULL-значения.

**Предложение GROUP BY**

Предложение GROUP BY позволяет вам определять подмножество значений в особом поле в терминах другого пол, и применять функцию агрегата к подмножеству. Это дает вам возможность объединять пол и агрегатные функции в едином предложении SELECT. Например, предположим что вы хотите найти наибольшую сумму приобретений полученную каждым продавцом. Вы можете сделать раздельный запрос для каждого из них, выбрав MAX (amt) из таблицы Порядков для каждого значения пол snum. GROUP BY, однако, позволит Вам поместить их все в одну команду:

=============== SQL Execution Log ==============

| |

| SELECT snum, MAX (amt) |

| FROM Orders |

| GROUP BY snum; |

| =============================================== |

| snum |

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

| 1001 767.19 |

| 1002 1713.23 |

| 1003 75.75 |

| 1014 1309.95 |

================================================

GROUP BY применяет агрегатные функции независимо от серий групп которые определяются с помощью значения поля в целом. В этом случае, каждая группа состоит из всех строк с тем же самым значением пол snum, и MAX функция применяется отдельно для каждой такой группы. Это значение пол, к которому применяется GROUP BY, имеет, по определению, только одно значение на группу вывода, также как это делает агрегатная функция. Результатом является совместимость которая позволяет агрегатам и полям объединяться таким образом. Вы можете также использовать GROUP BY с многочисленными полями. Совершенству вышеупомянутый пример далее, предположим что вы хотите увидеть наибольшую сумму приобретений получаемую каждым продавцом каждый день. Чтобы сделать это, вы должны сгруппировать таблицу Порядков по датам продавцов, и применить функцию MAX к каждой такой группе, подобно этому:

=============== SQL Execution Log ==============

| |

| SELECT snum, odate, MAX (amt) |

| FROM Orders |

| GROUP BY snum, odate; |

| =============================================== |

| snum odate |

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

| 1001 10/03/1990 767.19 |

| 1001 10/05/1990 4723.00 |

| 1001 10/06/1990 9891.88 |

| 1002 10/03/1990 5160.45 |

| 1002 10/04/1990 75.75 |

| 1002 10/06/1990 1309.95 |

| 1003 10/04/1990 1713.23 |

| 1014 10/03/1990 1900.10 |

| 1007 10/03/1990 1098.16 |

=================================================

На месте имен полей могут быть использованы их порядковые номера в списке полей результирующей таблицы.

1. **Транзакция, ее определение иназначение. Свойства транзакций.**

Транзакцией называется последовательность действий, которая или полностью фиксируется в базе данных, или полностью отменяется. Иногда под транзакцией также подразумевают не группу SQL-операторов, а интервал времени, выполняемые в течение которого SQL-операторы можно или все зафиксировать или все отменить.

Так, операция перевода денег с одного счета на другой должна составлять единую транзакцию. Иначе может возникнуть ситуация, когда первый SQL-оператор переведет деньги на другой счет, а второй, выполняющий снятие их со счета, не доведет дело до конца из-за непредвиденного сбоя.

Принцип использования транзакций

Новая транзакция начинается с начала каждого сеанса работы с базой данных. Далее все выполняемые SQL-операторы будут входить в одну транзакцию до тех пор, пока не будет выполнен оператор COMMIT WORK или ROLLBACK WORK.

Оператор COMMIT WORK завершает текущую транзакцию, выполняя фиксацию сделанных изменений в базе данных. Иногда говорят, что оператор COMMIT WORK фиксирует транзакцию.

Оператор ROLLBACK WORK выполняет откат транзакции, отменяя действие всех SQL-операторов, выполненных в текущей транзакции.

Логически транзакция должна объединять только выполнение взаимосвязанных операций. Так, если делать транзакции "очень большими", состоящими из последовательности не связанных между собой операторов, то любой сбой, автоматически выполняющий откат транзакции, повлияет на отмену действий, которые могли бы быть успешно завершены при более "коротких" транзакциях.

# Транзакции

Многопользовательские системы широко используют понятие транзакций. Транзакция - это логическая единица работы, которая состоит из одного или нескольких SQL-выражений. Группа выражений отмеченных как транзакция рассматривается как единое и неделимое целое. В случае если в одном из выражений обработки данных происходит ошибка, то транзакция отменяется целиком. Таким образом, система возвращается в состояние предшествующее началу транзакции, обеспечивая при этом физическую и логическую непротиворечивость данных.

Например, мы пытаемся модифицировать таблицу при помощи оператора UPDATE. В одном из столбцов этим оператором устанавливается недопустимое значение с точки зрения правил целостности для этой таблицы. Срабатывание ограничителя приведет к тому, что сервер СУБД не позволит выполнить такую модификацию и известит нас ошибкой, а механизм контроля транзакций вызывает отмену всего выполняемого выражения и производит откат к предыдущему состоянию таблицы, сохраняя, таким образом, целостность и непротиворечивость данных. В данном примере транзакция работает с одним SQL-выражением. В случае если выражений несколько, то откатывается результат работы всех выражений составляющих единую транзакцию.

Чтобы транзакциями можно было пользоваться, в системе должен быть включен режим регистрации транзакций. После этого система сохраняет информацию о предыдущих состояниях объектов в системе в так называемых журналах транзакций. Журналы транзакций - это специальные файлы, управляемые сервером СУБД, в которых записываются все изменения произошедшие с момента начала транзакций для всех транзакций в системе.

Если происходит явное сохранение изменений в системе (по команде COMMIT) или неявное сохранение изменений (по завершению группы SQL-выражений, формирующих транзакцию или по завершению сеанса пользователя), то все изменения произошедшие с момента начала транзакции вносятся в систему, и информация о данной транзакции удаляется из журнала.

Для облегчения управления системой в режиме регистрации транзакций существует возможность задания так называемых промежуточных точек сохранения. Промежуточная точка сохранения по команде SAVEPOINT <имя\_точки\_останова> явно помечает состояние системы и предоставляет возможность восстановления состояния БД на момент ее сохранения по команде ROLLBACK. В данном случае ROLLBACK <имя\_точки\_останова> откатывает систему к указанной точке. Обычно промежуточных точек сохранения для одного пользователя может быть несколько.

В случае если транзакция по каким-то причинам не может быть завершена, то происходит неявный откат. Его причиной, могут быть, например, ошибка при выполнении одного из SQL-выражений, составляющих транзакцию, или обрыв связи с инициатором транзакции. При этом по информации из журнала восстанавливается предыдущее состояние объектов, которые пыталась модифицировать текущая транзакция, после чего информация о транзакции удаляется из журнала. Откат может быть и явным - по команде ROLLBACK.

Данная схема справедлива для Oracle, где транзакция начинается с выполнением первого оператора, прочие сервера могут работать по-другому. Например в Informix DS, транзакция начинается явно, при помощи команды BEGIN WORK.

В SQL-бочке меда есть своя ложка дегтя. Для всех SQL-серверов использующих журнальный режим регистрации транзакций существует проблема, так называемых "длинных" транзакций. Это транзакции, которые затрагивают очень большой объем данных (сопоставимый с количеством свободного места на дисках) и в этом случае журналы регистрации транзакций могут переполниться. Если их рост ничем неограничен, то они могут израсходовать у ОС всю доступную дисковую память, что не есть хорошо, т.к. операционная система и сервер СУБД в этом случае остаются в непредсказуемом состоянии. Если их рост ограничен, то при переполнении журналов СУБД выдает соответствующую ошибку и операция откатывается. Чтобы избежать таких ситуаций программист должен разделить длинную транзакцию на короткие транзакции.

**Требования ACID**

Atomicity — Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся в исходное состояние.

Consistency — Согласованность, одно из самых сложных и неоднозначных свойств из четвёрки ACID. В соответствии с этим требованием, система находится в согласованном состоянии до начала транзакции и должна остаться в согласованном состоянии после завершения транзакции. Не нужно путать требование согласованности с требованиями целостности (integrity). Последние правила являются более узкими и, во многом, специфичны для реляционных СУБД: есть требования целостности типов (domain integrity), целостности ссылок (referential integrity), целостности сущностей (entity integrity), которые не могут быть нарушены физически в силу особенностей реализации системы. Согласованность является более широким понятием. Например, в банковской системе может существовать требование равенства суммы, списываемой с одного счёта, сумме, зачисляемой на другой. Это бизнес-правило и оно не может быть гарантировано только проверками целостности, его должны соблюсти программисты при написании кода транзакций. Если какая-либо транзакция произведёт списание, но не произведёт зачисление, то система останется в некорректном состоянии и свойство согласованности будет нарушено. Наконец, ещё одно замечание касается того, что в ходе выполнения транзакции согласованность не требуется. В нашем примере, списание и зачисление будут, скорее всего, двумя разными подоперациями и между их выполнением внутри транзакции будет видно несогласованное состояние системы. Однако не нужно забывать, что при выполнении требования изоляции, никаким другим транзакциям эта несогласованность не будет видна. А атомарность гарантирует, что транзакция либо будет полностью завершена, либо будет полностью откачена, тем самым эта промежуточная несогласованность является скрытой.

Isolation — Изоляция. Во время выполнения транзакции другие процессы не должны видеть данные в промежуточном состоянии. Например, если транзакция изменяет сразу несколько полей в базе данных, то другой запрос, выполненный во время выполнения транзакции, не должен вернуть одни из этих полей с новыми значениями, а другие с исходными.

Durability — Долговечность. Независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

1. **Блокировки при реализации транзакций. Свойство устойчивости транзакций. Т-SQL.**

*Блокировками* (locks) называются механизмы, применяемые для управления параллельными изменениями данных.

Существует два типа *блокировок*:

* *оптимистические блокировки* (optimistic locks) - предотвращают возникновение конфликтных ситуаций, выполняя при необходимости откат *транзакции* (такие *блокировки* в стандарте SQL-92 не поддерживаются);
* *пессимистические блокировки* (pessimistic locks) - предотвращают возникновение конфликтных ситуаций, блокируя одновременный доступ к данным для одновременных *транзакций*.

*Блокировки* используются для приостановки выполнения одних SQL-операторов, пока выполняются другие.

Если при *пессимистической блокировке* выполнен SQL-оператор, который может вызвать конфликтную ситуацию для другого SQL-оператора, то выполнение второго SQL-оператора будет приостановлено. При *оптимистической блокировке* могут выполняться любые SQL-операторы, но в случае возникновения конфликтной ситуации все изменения будут потеряны.

*Блокировки*, используемые уровнями изоляции, подразделяются на:

* *разделяемые блокировки (S-locks)*, которые могут одновременно устанавливаться несколькими пользователями;
* *исключительные блокировки* (X-locks), которые устанавливаются только одним пользователем, получающим эксклюзивный доступ к данным.

Существуют следующие логические и физические уровни*блокировок*:

* *блокировка на уровне таблицы* (table-level locking);
* *блокировка на уровне строк* (row-level locking);
* *блокировка на уровне элемента таблицы* (item-level locking);
* *блокировка на уровне БД* (dbspace-level locking);
* *блокировка на уровне табличного пространства* (tablespace-level locking);

*блокировка на уровне страницы или блока* (page-level locking).

1. **База данных и ее объекты. Структура языка SQL: операторы определения объектов БД.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица 5.1. Операторы определения данных DDL** | | | | |
| **Оператор** | | **Смысл** | | **Действие** |
| CREATE TABLE | | Создать таблицу | | Создает новую таблицу в БД |
| DROP TABLE | | Удалить таблицу | | Удаляет таблицу из БД |
| ALTER TABLE | | Изменить таблицу | | Изменяет структуру существующей таблицы или ограничения целостности, задаваемые для данной таблицы |
| CREATE VIEW | | Создать представление | | Создает виртуальную таблицу, соответствующую некоторому SQL-запросу |
| ALTER VIEW | | Изменить представление | | Изменяет ранее созданное представление |
| DROP VIEW | | Удалить представление | | Удаляет ранее созданное представление |
| CREATE INDEX | | Создать индекс | | Создает индекс для некоторой таблицы для обеспечения быстрого доступа по атрибутам, входящим в индекс |
| DROP INDEX | | Удалить индекс | | Удаляет ранее созданный индекс |
| **Таблица 5.2. Операторы манипулирования данными Data Manipulation Lanquaqe (DMP)** | | | | |
| **Оператор** | **Смысл** | | **Действие** | |
| DELETE | Удалить строки | | Удаляет одну или несколько строк, соответствующих условиям фильтрации, из базовой таблицы. Применение оператора согласуется с принципами поддержки целостности, поэтому этот оператор не всегда может быть выполнен корректно, даже если синтаксически он записан правильно | |
| INSERT | Вставить строку | | Вставляет одну строку в базовую таблицу. Допустимы модификации оператора, при которых сразу несколько строк могут быть перенесены из одной таблицы или запроса в базовую таблицу | |
| UPDATE | Обновить строку | | Обновляет значения одного или нескольких столбцов в одной или нескольких строках, соответствующих условиям фильтрации | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Таблица 5.3. Язык запросов Data Query Lanquaqe (DQL)** | | | |
| **Оператор** | **Смысл** | **Действие** | |
| SELECT | Выбрать строки | Оператор, заменяющий все операторы реляционной алгебры и позволяющий сформировать результирующее отношение, соответствующее запросу | |
| **Таблица 5.4. Средства управления транзакциями** | | | |
| **Оператор** | **Смысл** | | **Действие** |
| COMMIT | Завершить транзакцию | | Завершить комплексную взаимосвязанную обработку информации, объединенную в транзакцию |
| ROLLBACK | Откатить транзакцию | | Отменить изменения, проведенные в ходе выполнения транзакции |
| SAVEPOINT | Сохранить промежуточную точку выполнения транзакции | | Сохранить промежуточное состояние БД, пометить его для того, чтобы можно было в дальнейшем к нему вернуться |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Таблица 5.5. Средства администрирования данных** | | | | |
| **Оператор** | | **Смысл** | **Действие** | |
| ALTER DATABASE | | Изменить БД | Изменить набор основных объектов в базе данных, ограничений, касающихся всей базы данных | |
| ALTER DBAREA | | Изменить область хранения БД | Изменить ранее созданную область хранения | |
| ALTER PASSWORD | | Изменить пароль | Изменить пароль для всей базы данных | |
| CREATE DATABASE | | Создать БД | Создать новую базу данных, определив основные параметры для нее | |
| CREATE DBAREA | | Создать область хранения | Создать новую область хранения и сделать ее доступной для размещения данных | |
| DROP DATABASE | | Удалить БД | Удалить существующую базу данных (только в том случае, когда вы имеете право выполнить это действие) | |
| DROP DBAREA | | Удалить область хранения БД | Удалить существующую область хранения (если в ней на настоящий момент не располагаются активные данные) | |
| GRANT | | Предоставить права | Предоставить права доступа на ряд действий над некоторым объектом БД | |
| REVOKE | | Лишить прав | Лишить прав доступа к некоторому объекту или некоторым действиям над объектом | |
| **Таблица 5.6. Программный SQL** | | | | |
| **Оператор** | **Смысл** | | | **Действие** |
| DECLARE | Определяет курсор для запроса | | | Задает некоторое имя и определяет связанный с ним запрос к БД, который соответствует виртуальному набору данных |
| OPEN | Открыть курсор | | | Формирует виртуальный набор данных, соответствующий описанию указанного курсора и текущему состоянию БД |
| FETCH | Считать строку из множества строк, определенных курсором | | | Считывает очередную строку, заданную параметром команды из виртуального набора данных, соответствующего открытому курсору |
| CLOSE | Закрыть курсор | | | Прекращает доступ к виртуальному набору данных, соответствующему указанному курсору |
| PREPARE | Подготовить оператор SQL к динамическому выполнению | | | Сгенерировать план выполнения запроса, соответствующего заданному оператору SQL |
| EXECUTE | Выполнить оператор SQL, ранее подготовленный к динамическому выполнению | | | Выполняет ранее подготовленный план запроса |

INSERT INTO table\_name

{ [ (column\_commalist) ] query\_expression

| DEFAULT VALUES

UPDATE table\_name SET update\_assignment\_commalist

WHERE conditional\_expression

update\_assignment ::= column\_name =

{ value\_expression | DEFAULT | NULL }

DELETE FROM table\_name

WHERE conditional\_expression

1. **Т-SQL. Поиск данных с помощью оператора Select. Структура команды Select. Функции between, in, like и null. Агрегатные функции. Опции group by, having, order by. Примеры.**

## SELECT

Оператор (предложение) *выборки* **SELECT** выбирает из одной или нескольких таблиц множество значений, которое и является результатом его работы.

SELECT [предикат] {\*|[таблица.]поле\_1 [AS псевдоним\_2] [,...]}

FROM выражение [, ...]

[WHERE... ]

[GROUP BY... ]

[HAVING... ];

или

SELECT <поле\_1> [,...] FROM выражение [WHERE <условие>];

Оператор **SELECT** может осуществлять поиск сразу по нескольким таблицам. В этом случае после слова FROM в операторе SELECT надо указать таблицы, по которым производится поиск. Если в нескольких полях имеются одноименные поля (например, поле *название* присутствует и в таблице «Фирма-поставщик», и в таблице «Товар»), то для устранения неясностей надо перед именем поля указать имя таблицы и символ точка.

Выдать список товаров и названий фирм, их производящих:

SELECT товар.название, фирма\_поставщик.название FROM товар,фирма\_поставщик

WHERE товар.id\_фирмы = фирма\_поставщик.id\_фирмы;

Например, чтобы получить список всех людей, информация о которых есть в базе, вместе с названием фирмы, в которой они работают, то надо выполнить запрос:

SELECT фирма.название, фирма.адрес, сотрудники.фамилия, сотрудники.имя FROM фирма, сотрудники WHERE фирма.id\_фирмы = сотрудники.id\_фирмы;

**SELECT** будет работать следующим образом. Он последовательно будет перебирать все записи из таблицы «Фирма». Для каждой записи из «Фирма» будет просмотрена таблица «Сотрудники». Как только будет найдена ***запись*** из «Сотрудники», удовлетворяющая условию WHERE, в результат добавится новый ряд, сформированный из полей *название* и *адрес* записи из «Фирма» и полей *фамилия* и *имя* записи из «Сотрудники». Если же в «Сотрудники» не будет найдено ни одной ***записи***, то информация о текущей записи из «Фирма» в результат не попадет.

Таблица «Фирма»

|  |  |  |
| --- | --- | --- |
| ID\_фирмы | название | адрес |
| 102  103  104 | «АО Восток»  «ОСО Запад»  «АО Мираж» | Пр. Андропова 345  Ул.Толстого 45  Ул.Советская 90 |

Таблица «Сотрудники»

|  |  |  |
| --- | --- | --- |
| ID\_фирмы | фамилия | имя |
| 102  103  102  105 | Иванов  Петров  Сидоров  Антонов | Михаил  Иван  Николай  Юрий |

Рис.8.1. Пример таблиц «Фирма» и «Сотрудники»

### INNER JOIN, LEFT JOIN, RIGHT JOIN

SELECT DISTINCTROW фирма.название, фирма.адрес, сотрудники.фамилия, сотрудники.имя FROM фирма INNER JOIN сотрудники ON фирма.id\_фирмы = сотрудники.id\_фирмы;

**INNER JOIN** – самый обычный тип связывания, объединяет записи двух таблиц, если связующие поля обеих таблиц содержат одинаковые значения.

|  |  |  |  |
| --- | --- | --- | --- |
| название | адрес | Фамилия | имя |
| «АО Восток»  «АО Восток»  «ОСО Запад» | Пр. Андропова 345  Пр. Андропова 345  Ул.Толстого 45 | Иванов  Сидоров  Петров | Михаил  Николай  Иван |

Рис.8.2. Результат связывания двух таблиц после выполнения запроса

При выборке из двух таблиц мы получаем все возможные комбинации значений, при этом из всех таблиц выбираются только те записи, которые удовлетворяют заданному условию. Например, в приведенном выше примере на получение информации о фирмах и людях, в них работающих, мы получим список только тех компаний, для которых в таблице «Сотрудники» есть хотя бы один человек. А если мы хотим в этом запросе получить информацию обо всех фирмах, в том числе и тех, в которых нет ни одного сотрудника?

SELECT фирма.название, фирма.адрес, сотрудники.фамилия, сотрудники.имя FROM фирма LEFT JOIN сотрудники ON фирма.id\_фирмы = сотрудники.id\_фирмы;

**LEFT JOIN** - используется для создания левого внешнего объединения, при котором все записи из первой (левой) таблицы включаются в динамический набор, даже если во второй (правой) таблице нет соответствующих им записей.

|  |  |  |  |
| --- | --- | --- | --- |
| название | адрес | Фамилия | имя |
| «АО Восток»  «АО Восток»  «ОСО Запад»  «АО Мираж» | Пр. Анропова 345  Пр. Андропова 345  Ул.Толстого 45  Ул.Советская 90 | Иванов  Сидоров  Петров  NULL | Михаил  Николай  Иван  NULL |

Рис.8.3. Результат оператора SELECT с использованием указателя LEFT JOIN

Для отбора всех сотрудников, в том числе и тех, которые не приписаны ни к одной из фирм, надо выполнить запрос:

SELECT фирма.название, фирма.адрес, сотрудники.фамилия, сотрудники.имя FROM фирма RIGHT JOIN сотрудники ON фирма.id\_фирмы = сотрудники.id\_фирмы;

**RIGHT JOIN** - используется для создания правого внешнего объединения, при котором все записи из второй (правой) таблицы включаются в динамический набор, даже если в первой (левой) таблице нет соответствующих им записей.

|  |  |  |  |
| --- | --- | --- | --- |
| название | адрес | Фамилия | имя |
| «АО Восток»  «АО Восток»  «ОСО Запад»  NULL | Пр. Андропова 345  Пр. Андропова 345  Ул.Толстого 45  NULL | Иванов  Сидоров  Петров  Антонов | Михаил  Николай  Иван  Юрий |

Рис.8.4. Результат оператора SELECT с использованием указателя RIGHT JOIN

### Использование BETWEEN

С помощью BETWEEN ... AND ... (находится в интервале от ... до ...) можно отобрать строки, в которых значение какого-либо столбца находятся в заданном диапазоне.

Например, выдать перечень продуктов, в которых значение содержания белка находится в диапазоне от 10 до 50:

SELECT Продукт, Белки FROM Продукты WHERE Белки BETWEEN 10 AND 50;

Можно задать и NOT BETWEEN (не принадлежит диапазону между), например

SELECT Продукт, Белки, Жиры FROM Продукты

WHERE Белки NOT BETWEEN 10 AND 50 AND Жиры > 100;

BETWEEN особенно удобен при работе с данными, задаваемыми интервалами, начало и конец которых расположен в разных столбцах.

Если, например, потребовалось узнать, какие изменения минимальных окладов производились в 1993/94 учебном году, то можно выдать запрос

SELECT Начало, Оклад FROM Оклад

WHERE Начало BETWEEN #01-09-93# AND #31-08-94#;

Отметим, что при формировании запросов значения дат следует заключать в #.

Для выявления всех значений минимальных окладов, которые существовали в 1993/94 учебном году, можно сформировать запрос

SELECT \* FROM Оклад WHERE Начало BETWEEN #01-09-93# AND #31-08-94#

OR Конец BETWEEN #01-09-93# AND #31-08-94#;

### Проверка на вхождение во множество IN

Логическая операция, которую можно использовать в условии для операторов SELECT, UPDATE и DELETE ­это операция **IN**. Эта логическая операция возвращает истину, когда значение слева от слова IN входит во множество значений, указанного справа от слова IN. Множество возможных значений может быть указано явно - через запятую в скобках, а может формироваться другим оператором SELECT:

<выражение> IN (<значение 1>, <значение 2>, ....)

<выражение> IN (<оператор SELECT>)

Выдать сведения о блюдах на основе яиц, крупы и овощей

SELECT\* FROM Блюда WHERE Основа IN (Яйца Крупа Овощи);

Форма IN является в действительности просто краткой записью последовательности отдельных сравнений, соединенных операторами OR. Предыдущее предложение эквивалентно такому:

SELECT \* FROM Блюда WHERE Основа = Яйца OR Основа = Крупа OR Основа = Овощи;

Например, получить адреса фирм «АО Сделай Сам» и «ОСО Довгань»:

SELECT название,адрес FROM фирма WHERE название IN ("АО Сделай Сам", "ООО Довгань");

Это запрос можно сформулировать и с помощью операций "=" и "OR", но во многих случаях использование операции IN более наглядно и компактно.

Например, найти названия и адреса всех фирм, поставляющих молоко. Этот запрос выполняется одним SQL-оператором:

SELECT название, адрес FROM фирма WHERE id\_фирмы IN (SELECT товар.id\_фирмы FROM товар WHERE название = "молоко");

Найти все фирмы, которые еще не поставляют молоко:

SELECT название, адрес FROM фирма WHERE id\_фирмы NOT IN(SELECT товар.id\_фирмы FROM товар WHERE название = "молоко");

Если известно, что оператор SELECT возвращает одну запись, то можно вместо слова IN использовать проверку на равенство.

Например, получить список самых дорогих товаров, а именно их названия и цену (цена будет для всех одна - максимальная), то надо выполнить запрос:

SELECT название, цена FROM товар WHERE цена = (SELECT MAX(цена) FROM товар);

### Использование LIKE

Используется для сравнения строкового выражения с образцом в выражении SQL.

Выдать перечень сыров, имеющихся на складе.

SELECT название, цена FROM товар WHERE название Like "сыр\*";

Следовательно, в приведенном примере SELECT будет осуществлять выборку записей из таблицы товар, для которых значение в столбце название начинается сочетанием «сыр» и содержит любую последовательность из нуля или более символов, следующих за сочетанием «сыр». Если бы среди продуктов были «Голанский сыр», «Тильзицкий сыр» и т.п., то они не были бы найдены. Для их отыскания надо изменить фразу WHERE:

WHERE название LIKE "\*сыр\*";

Это позволит отыскать все сыры.

**Предложение GROUP BY**

Предложение GROUP BY позволяет вам определять подмножество значений в особом поле в терминах другого пол, и применять функцию агрегата к подмножеству. Это дает вам возможность объединять пол и агрегатные функции в едином предложении SELECT. Например, предположим что вы хотите найти наибольшую сумму приобретений полученную каждым продавцом. Вы можете сделать раздельный запрос для каждого из них, выбрав MAX (amt) из таблицы Порядков для каждого значения пол snum. GROUP BY, однако, позволит Вам поместить их все в одну команду:

=============== SQL Execution Log ==============

| |

| SELECT snum, MAX (amt) |

| FROM Orders |

| GROUP BY snum; |

| =============================================== |

| snum |

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

| 1001 767.19 |

| 1002 1713.23 |

| 1003 75.75 |

| 1014 1309.95 |

================================================

GROUP BY применяет агрегатные функции независимо от серий групп которые определяются с помощью значения поля в целом. В этом случае, каждая группа состоит из всех строк с тем же самым значением пол snum, и MAX функция применяется отдельно для каждой такой группы. Это значение пол, к которому применяется GROUP BY, имеет, по определению, только одно значение на группу вывода, также как это делает агрегатная функция. Результатом является совместимость которая позволяет агрегатам и полям объединяться таким образом. Вы можете также использовать GROUP BY с многочисленными полями. Совершенству вышеупомянутый пример далее, предположим что вы хотите увидеть наибольшую сумму приобретений получаемую каждым продавцом каждый день. Чтобы сделать это, вы должны сгруппировать таблицу Порядков по датам продавцов, и применить функцию MAX к каждой такой группе, подобно этому:

=============== SQL Execution Log ==============

| |

| SELECT snum, odate, MAX (amt) |

| FROM Orders |

| GROUP BY snum, odate; |

| =============================================== |

| snum odate |

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

| 1001 10/03/1990 767.19 |

| 1001 10/05/1990 4723.00 |

| 1001 10/06/1990 9891.88 |

| 1002 10/03/1990 5160.45 |

| 1002 10/04/1990 75.75 |

| 1002 10/06/1990 1309.95 |

| 1003 10/04/1990 1713.23 |

| 1014 10/03/1990 1900.10 |

| 1007 10/03/1990 1098.16 |

=================================================

На месте имен полей могут быть использованы их порядковые номера в списке полей результирующей таблицы.

### ПРЕДЛОЖЕНИЕ HAVING

Предположим, что в предыдущем примере, вы хотели бы увидеть только максимальную сумму приобретений, значение которой выше $3000.00. Вы не сможете использовать агрегатную функцию в предложении WHERE (если вы не используете подзапрос, описанный позже ), потому что предикаты оцениваются в терминах одиночной строки, а агрегатные функции оцениваются в терминах групп строк. Это означает, что вы не сможете сделать что-нибудь подобно следующему:

SELECT snum, odate, MAX (amt)

FROM Oreders

WHERE MAX ((amt)) > 3000.00

GROUP BY snum, odate;

Это будет отклонением от строгой интерпретации ANSI. Чтобы увидеть максимальную стоимость приобретений свыше $3000.00, вы можете использовать предложение HAVING. Предложение HAVING определяет критерии используемые чтобы удалять определенные группы из вывода, точно также как предложение WHERE делает это для индивидуальных строк. Правильной командой будет следующая:

=============== SQL Execution Log ==============

| |

| SELECT snum, odate, MAX (amt) |

| FROM Orders |

| GROUP BY snum, odate |

| HAVING MAX (amt) > 3000.00; |

| =============================================== |

| snum odate |

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

| 1001 10/05/1990 4723.00 |

| 1001 10/06/1990 9891.88 |

| 1002 10/03/1990 5160.45 |

| |

================================================

**ORDER BY** сортирует строки результирующей таблицы данных. Если ORDER BY используется внутри GROUP BY, то строки сортируются внутри каждой группы результирующих строк. Вместо имен полей могут быть использованы их порядковые номера в списке полей результирующей таблицы. ASC сортирует данные в восходящем порядке, DESC - в обратном.

Пример:

SELECT tkunden.knum, tverkauf.vnum, tverkauf.prov

FROM tkunden, tverkauf

WHERE tkunden.vnum=tverkauf.vnum order by tkunden.knum asc;

Пример сортировки по нескольким полям:

SELECT departments.dep\_name,employees.name,employees.salary

FROM departments,employees

WHERE departments.dep\_id=employees.dep\_id\_ref

ORDER BY departments.dep\_id asc, employees.salary desc;

1. **Т-SQL. Операторы создания и удаления таблиц БД, индексов.**

***Таблица*** – основной объект для хранения информации в реляционной *базе данных*. Она состоит из содержащих данные *строк* и *столбцов*, занимает в *базе данных* физическое пространство и может быть постоянной или временной.

Поле, также называемое в реляционной *базе данных* *столбцом*, является частью *таблицы*, за которой закреплен определенный тип данных. Каждая *таблица* *базы данных* должна содержать хотя бы один *столбец*. *Строка* данных – это запись в *таблице* *базы данных*, она включает поля, содержащие данные из одной записи *таблицы*.

Приступая к *созданию таблицы*, необходимо иметь ответы на ряд вопросов:

* Как будет называться *таблица*?
* Как будут называться *столбцы* (поля) *таблицы*?
* Какие типы данных будут закреплены за каждым *столбцом*?
* Какой размер памяти должен быть выделен для хранения каждого *столбца*?
* Какие *столбцы* *таблицы* требуют обязательного ввода?
* Из каких *столбцов* будет состоять первичный ключ?

Базовый синтаксис оператора *создания таблицы* имеет следующий вид:

<определение\_таблицы> ::=

CREATE TABLE имя\_таблицы

(имя\_столбца тип\_данных

[NULL | NOT NULL ] [,...n])

С течением времени структура базы данных меняется: создаются новые таблицы, а прежние становятся ненужными и удаляются из базы данных с помощью оператора:

DROP TABLE имя\_таблицы [RESTRICT | CASCADE]

Следует отметить, что эта команда удалит не только указанную таблицу, но и все входящие в нее строки данных. Если требуется удалить из таблицы лишь данные, сохранив структуру таблицы, следует воспользоваться командой DELETE.

Оператор DROP TABLE дополнительно позволяет указывать, следует ли операцию удаления выполнять каскадно. Если в операторе указано ключевое слово RESTRICT, то при наличии в базе данных хотя бы одного объекта, существование которого зависит от удаляемой таблицы, выполнение оператора DROP TABLE будет отменено. Если указано ключевое слово CASCADE, автоматически удаляются и все прочие объекты базы данных, чье существование зависит от удаляемой таблицы, а также другие объекты, зависящие от удаляемых объектов. Общий эффект от выполнения оператора DROP TABLE с ключевым словом CASCADE может оказаться весьма ощутимым, поэтому подобные операторы следует использовать с максимальной осторожностью.

Чаще всего оператор DROP TABLE используется для исправления ошибок, допущенных при создании таблицы. Если таблица была создана с некорректной структурой, можно воспользоваться оператором DROP TABLE для ее удаления, после чего создать таблицу заново.

***Индексы*** представляют собой структуру, позволяющую выполнять ускоренный доступ к *строкам* *таблицы* на основе значений одного или более ее *столбцов*. Наличие *индекса* может существенно повысить скорость выполнения некоторых запросов и сократить время поиска необходимых данных за счет физического или логического их упорядочивания. ***Индекс*** – это набор ссылок, упорядоченных по определенному *столбцу* *таблицы*, который в данном случае будет называться индексированным *столбцом*. Хотя *индекс* и связан с конкретным *столбцом* (или *столбцами*) *таблицы*, все же он является самостоятельным объектом *базы данных*.

Физически *индекс* – всего лишь упорядоченный набор значений из индексированного *столбца* с указателями на места физического размещения исходных *строк* в структуре *базы данных*. Когда пользователь выполняет обращающийся к индексированному *столбцу* запрос, СУБД автоматически анализирует *индекс* для поиска требуемых значений.

Однако, поскольку *индексы* должны обновляться системой при каждом внесении *изменений* в их базовую *таблицу*, они создают дополнительную нагрузку на систему.

*Индексы* обычно создаются с целью удовлетворения определенных критериев поиска после того, как *таблица* уже находилась некоторое время в работе и увеличилась в размерах. *Создание индексов* не предусмотрено стандартом SQL, однако большинство диалектов поддерживают как минимум следующий оператор:

*CREATE [ UNIQUE ] INDEX имя\_индекса*

ON имя\_таблицы(имя\_столбца[ASC|DESC][,...n])

Указанные в операторе *столбцы* составляют *ключ индекса*. *Индексы* могут создаваться только для базовых *таблиц*, но не для представлений. Если в операторе указано ключевое слово UNIQUE, уникальность значений *ключа индекса* будет автоматически поддерживаться системой. Требование уникальности значений обязательно для первичных ключей, а также возможно и для других *столбцов* *таблицы* (например, для альтернативных ключей). Хотя *создание индекса* допускается в любой момент, при его построении для уже заполненной данными *таблицы* могут возникнуть проблемы, связанные с дублированием данных в различных *строках*. Следовательно, *уникальные индексы* (по крайней мере, для первичного ключа) имеет смысл создавать непосредственно при формировании *таблицы*. В результате система сразу возьмет на себя контроль за уникальностью значений данных в соответствующих *столбцах*.

Если созданный *индекс* впоследствии окажется ненужным, его можно удалить с помощью оператора

DROP INDEX имя\_индекса

#### Создание индекса

Если выборка данных из таблицы требует значительного времени, это означает, что для нее необходимо создать индекс. Индексы могут существенно повысить производительность выполнения операций поиска и выборки данных. При выборе столбца для индекса следует проанализировать, какие типы запросов чаще всего выполняются пользователями и какие столбцы являются ключевыми, т.е. задающими критерии выборки данных, например, порядок сортировки.

В среде SQL Server реализовано несколько типов индексов:

* кластерные индексы;
* некластерные индексы;
* уникальные индексы.

#### Некластерный индекс

Некластерные индексы – наиболее типичные представители семейства индексов. В отличие от кластерных, они не перестраивают физическую структуру таблицы, а лишь организуют ссылки на соответствующие строки.

Для идентификации нужной строки в таблице некластерный индекс организует специальные указатели, включающие в себя:

* информацию об идентификационном номере файла, в котором хранится строка;
* идентификационный номер страницы соответствующих данных;
* номер искомой строки на соответствующей странице;
* содержимое столбца.

В большинстве случаев следует ограничиваться 4-5 индексами.

#### Кластерный индекс

Принципиальным отличием кластерного индекса от индексов других типов является то, что при его определении в таблице физическое расположение данных перестраивается в соответствии со структурой индекса. Логическая структура таблицы в этом случае представляет собой скорее словарь, чем индекс. Данные в словаре физически упорядочены, например по алфавиту.

Кластерные индексы могут дать существенное увеличение производительности поиска данных даже по сравнению с обычными индексами. Увеличение производительности особенно заметно при работе с последовательными данными. Если в таблице определен некластерный индекс, то сервер должен сначала обратиться к индексу, а затем найти нужную строку в таблице. При использовании кластерных индексов следующая порция данных располагается сразу после найденных ранее данных. Благодаря этому отпадают лишние операции, связанные с обращением к индексу и новым поиском нужной строки в таблице.

Естественно, в таблице может быть определен только один кластерный индекс. В качестве такового следует выбирать наиболее часто используемые столбцы. При этом стоит следовать общим рекомендациям создания индексов и не индексировать слишком длинные столбцы.

Кластерный индекс может включать несколько столбцов. Однако количество таких столбцов рекомендуется по возможности свести к минимуму.

Необходимо избегать создания кластерного индекса для часто изменяемых столбцов, поскольку сервер должен будет выполнять физическое перемещение всех данных в таблице, чтобы они находились в упорядоченном состоянии, как того требует кластерный индекс. Для интенсивно изменяемых столбцов лучше подходит некластерный индекс.

При создании в таблице первичного ключа (PRIMARY KEY) сервер автоматически создает для него кластерный индекс, если его не существовало ранее или если при определении ключа не был явно указан другой тип индекса.

Когда же в таблице определен еще и некластерный индекс, то его указатель ссылается не на физическое положение строки в базе данных, а на соответствующий элемент кластерного индекса, описывающего эту строку, что позволяет не перестраивать структуру некластерных индексов всякий раз, когда кластерный индекс меняет физический порядок строк в таблице.

#### Уникальный индекс

Уникальность значений в индексируемом столбце гарантируют уникальные индексы. При их наличии сервер не разрешит вставить новое или изменить существующее значение таким образом, чтобы в результате этой операции в столбце появились два одинаковых значения.

Уникальный индекс является своеобразной надстройкой и может быть реализован как для кластерного, так и для некластерного индекса. В одной таблице может существовать один уникальный кластерный и множество уникальных некластерных индексов.

Уникальные индексы следует определять только тогда, когда это действительно необходимо. Для обеспечения целостности данных в столбце можно определить ограничение целостности UNIQUE или PRIMARY KEY, а не прибегать к уникальным индексам. Их использование только для обеспечения целостности данных является неоправданной тратой пространства в базе данных. Кроме того, на их поддержание тратится и процессорное время.

Средства языка SQL предлагают несколько способов определения индекса:

* автоматическое создание индекса при создании первичного ключа;
* автоматическое создание индекса при определении ограничения целостности UNIQUE;
* создание индекса с помощью команды CREATE INDEX.

Последняя команда имеет следующий формат:

CREATE [ UNIQUE ]

[ CLUSTERED | NONCLUSTERED ]

INDEX имя\_индекса ON имя\_таблицы(имя\_столбца

[ASC|DESC][,...n])

[WITH [PAD\_INDEX]

[[,] FILLFACTOR=фактор\_заполнения]

[[,] IGNORE\_DUP\_KEY]

[[,] DROP\_EXISTING]

[[,] STATISTICS\_NORECOMPUTE] ]

[ON имя\_группы\_файлов ]

Рассмотрим некоторые параметры приведенной команды.

Имя индекса должно быть уникальным в пределах таблицы, а сам индекс создается исключительно для таблицы текущей базы данных.

Параметр UNIQUE используется при необходимости ввода в определенное поле только уникальных значений. При указании этого ключевого слова будет создан уникальный индекс. В индексируемом столбце желательно запретить хранение значений NULL, чтобы избежать проблем, связанных с уникальностью значений. После того как для столбца появится уникальный индекс, сервер не разрешит выполнение команд INSERT и UPDATE, которые приведут к появлению дублирующих значений.

Параметр CLUSTERED использует возможность физического индексирования данных и позволяет произвести так называемое кластерное индексирование, в результате чего будут отсортированы данные в самой таблице согласно порядку этого индекса, а вся добавляемая информация станет приводить к изменению физического порядка данных. Кластерным может быть только один индекс в таблице.

Параметр NONCLUSTERED позволяет создавать некластерные индексы.

Параметр FILLFACTOR осуществляет настройку разбиения индекса на страницы и заметно оптимизирует работу SQL-сервера. Коэффициент FILLFACTOR определяет в процентном соотношении размер создаваемых индексных страниц. При этом имеется обратно пропорциональная зависимость частоты работы с таблицей и коэффициента FILLFACTOR.

Параметр PAD\_INDEX определяет заполнение внутреннего пространства индекса и применяется совместно с FILLFACTOR.

Параметр DROP\_EXISTING при использовании кластерного индекса определяет его повторное создание, что позволяет предотвратить нежелательное обновление кластерных индексов.

Параметр STATISTICS\_NORECOMPUTE определяет функции автоматического обновления статистики для таблицы.

Параметр имя\_группы\_файлов позволяет осуществить выбор файловой группы, в которой будет находиться создаваемый индекс. Использование индекса из другой файловой группы повышает производительность некластерных индексов в связи с параллельностью выполнения процессов ввода/вывода и работы с самим индексом.

#### Удаление индекса

Удаление индекса выполняется командой

DROP INDEX 'имя\_индекса'[,...n]

**Пример 3.5.** Создать уникальный кластерный индекс для таблицы Клиент по столбцу Фамилия в первичной группе файлов.

CREATE UNIQUE CLUSTERED INDEX index\_klient1

ON Клиент (Фамилия)

WITH DROP\_EXISTING

ON PRIMARY

**Пример 3.6.** Создать уникальный некластерный индекс для таблицы Клиент по столбцам Фамилия и Имя в первичной группе файлов. Кроме того, элементы индекса будут упорядочены по убыванию. Также запретим автоматическое обновление статистики при изменении данных в таблице и установим фактор заполнения индексных страниц на уровне 30%.

CREATE UNIQUE NONCLUSTERED INDEX index\_klient2

ON Клиент (Фамилия DESC,Имя DESC)

WITH FILLFACTOR=30,

STATISTICS\_NORECOMPUTE

ON PRIMARY

1. **Т-SQL. Операторы загрузки таблиц, удаления и обновления данных таблицы. Типы данных.**

#### Изменение таблицы

Структура существующей таблицы может быть модифицирована с помощью команды ALTER TABLE, упрощенный синтаксис которой представлен ниже:

ALTER TABLE имя\_таблицы

{[ADD [COLUMN] имя\_столбца тип\_данных [

NULL | NOT NULL ]]

| [DROP [COLUMN] имя\_столбца]}

В среде MS SQL Server упрощенный синтаксис команды модификации таблицы имеет вид:

ALTER TABLE имя\_таблицы

{[ALTER COLUMN имя\_столбца

{новый\_тип\_данных [(точность[,масштаб])]

[ NULL | NOT NULL ]}]

| ADD { [имя\_столбца тип\_данных]

| имя\_столбца AS выражение } [,...n]

| DROP {COLUMN имя\_столбца}[,...n] }

Команда позволяет добавлять и удалять столбцы, изменять их определения.

Одно из основных правил при добавлении столбцов в существующую таблицу гласит: когда в таблице уже содержатся данные, добавляемый столбец не может быть определен с атрибутом NOT NULL. Этот атрибут означает, что для каждой строки данных соответствующий столбец должен содержать некоторое значение, поэтому добавление столбца с атрибутом NOT NULL приводит к появлению противоречия – уже существующие строки данных таблицы не будут иметь в новом столбце ненулевых значений.

Тем не менее, существует способ добавления обязательных полей в существующую таблицу. Для этого необходимо:

* добавить в таблицу новый столбец, определив его с атрибутом NULL (т.е. столбец не обязан содержать каких-либо значений);
* ввести в новый столбец какие-либо значения для каждой строки данных таблицы;
* убедившись, что новый столбец содержит ненулевые значения для каждой строки данных, изменить структуру таблицы, заменив атрибут этого столбца на NOT NULL.

При изменении определений столбцов следует принимать во внимание некоторые общепринятые правила:

* размер столбца может быть увеличен до максимального значения, допускаемого соответствующим типом данных;
* размер столбца может быть уменьшен только в том случае, если содержащееся в нем наибольшее значение не будет превосходить его нового размера;
* количество разрядов числового типа данных всегда может быть увеличено;
* количество разрядов числового типа данных может быть уменьшено только в том случае, если количество разрядов наибольшего значения в соответствующем столбце не будет превосходить нового числа разрядов, определенного для этого столбца;
* количество десятичных знаков числового типа данных может быть уменьшено или увеличено;
* тип данных столбца, как правило, может быть изменен.

Некоторые реализации фактически могут ограничить разработчика в использовании некоторых опций команды ALTER TABLE. Например, может оказаться недопустимым удаление столбцов из существующей таблицы. Чтобы добиться этого, сначала потребуется удалить саму таблицу и только потом заново ее построить с нужными столбцами. Причем уже внесенные в таблицу данные будут потеряны.

Возможны трудности, связанные с удалением из таблицы столбца, который зависит от некоторого столбца другой таблицы. В таком случае сначала придется удалить ограничение столбца, а затем сам столбец.

**Пример 3.4.** Добавить в таблицу Клиент поле для номера расчетного счета.

ALTER TABLE Клиент ADD Рас\_счет CHAR(20)

1. **Т-SQL . Задание ограничений целостности в команде create table. Примеры.**
2. **Понятие об администрировании баз данных Средства администрирования БД в SQLServer 2005.**

### Основные функции группы администратора БД

1. **Анализ предметной области**: описание предметной области, выявление ограничений целостности, определение статуса (доступности, секретности) информации, определение потребностей пользователей, определение соответствия "данные—пользователь", определение объемно-временных характеристик обработки данных.
2. **Проектирование структуры БД**: определение состава и структуры файлов БД и связей между ними, выбор методов упорядочения данных и методов доступа к информации, описание БД на языке описания данных (ЯОД).
3. **Задание ограничений целостности при описании структуры БД и процедур обработки БД**:
   * задание декларативных ограничений целостности, присущих предметной области;
   * определение динамических ограничений целостности, присущих предметной области в процессе изменения информации, хранящейся в БД;
   * определение ограничений целостности, вызванных структурой БД;
   * разработка процедур обеспечения целостности БД при вводе и корректировке данных;
   * определение ограничений целостности при параллельной работе пользователей в многопользовательском режиме.
4. **Первоначальная загрузка и ведение БД**:
   * разработка технологии первоначальной загрузки БД, которая будет отличаться от процедуры модификации и дополнения данными при штатном использовании базы данных;
   * разработка технологии проверки соответствия введенных данных реальному состоянию предметной области. База данных моделирует реальные объекты некоторой предметной области и взаимосвязи между ними, и на момент начала штатной эксплуатации эта модель должна полностью соответствовать состоянию объектов предметной области на данный момент времени;
   * в соответствии с разработанной технологией первоначальной загрузки может понадобиться проектирование системы первоначального ввода данных.
5. **Защита данных:**
   * определение системы паролей, принципов регистрации пользователей, создание групп пользователей, обладающих одинаковыми правами доступа к данным;
   * разработка принципов защиты конкретных данных и объектов проектирования; разработка специализированных методов кодирования информации при ее циркуляции в локальной и глобальной информационных сетях;
   * разработка средств фиксации доступа к данным и попыток нарушения системы защиты;
   * тестирование системы защиты;
   * исследование случаев нарушения системы защиты и развитие динамических методов защиты информации в БД.
6. **Обеспечение восстановления БД**:
   * разработка организационных средств архивирования и принципов восстановления БД;
   * разработка дополнительных программных средств и технологических процессов восстановления БД после сбоев.
7. **Анализ обращений пользователей БД**: сбор статистики по характеру запросов, по времени их выполнения, по требуемым выходным документам
8. **Анализ эффективности функционирования БД:**
   * анализ показателей функционирования БД;
   * планирование реструктуризации (изменение структуры) БД и реорганизации БнД.
9. **Работа с конечными пользователями**:
   * сбор информации об изменении предметной области;
   * сбор информации об оценке работы БД;
   * обучение пользователей, консультирование пользователей;
   * разработка необходимой методической и учебной документации по работе конечных пользователей.
10. **Подготовка и поддержание системных средств:**
    * анализ существующих на рынке программных средств и анализ возможности и необходимости их использования в рамках БД;
    * разработка требуемых организационных и программно-технических мероприятий по развитию БД;
    * проверка работоспособности закупаемых программных средств перед подключением их к БД;
    * курирование подключения новых программных средств к БД.
11. **Организационно-методическая работа по проектированию БД**:
    * выбор или создание методики проектирования БД;
    * определение целей и направления развития системы в целом;
    * планирование этапов развития БД;
    * разработка общих словарей-справочников проекта БД и концептуальной модели;
    * стыковка внешних моделей разрабатываемых приложений;
    * курирование подключения нового приложения к действующей БД;
    * обеспечение возможности комплексной отладки множества приложений, взаимодействующих с одной БД.

**Средства администрирования БД в SQL Server 2005**

1. Редактор кода в среде SQL Server Management Studio обеспечивают следующие возможности.

* Шаблоны, которые могут быть использованы для быстрой подготовки сценариев для SQL Server, служб SQL Server 2005 Analysis Services (SSAS) и SQL Server 2005 Compact Edition. Шаблоны — это файлы, содержащие базовый набор инструкций, необходимых для создания объектов в базе данных.
* Выделение цветом синтаксических конструкций, облегчающее читаемость сложных инструкций.
* Создание запросов в графическом конструкторе запросов методом перетаскивания.
* Представление окон запросов в виде вкладок окна документа или в виде отдельных документов.
* Представление результатов выполнения запроса в виде табличной сетки или текстового окна с возможностью перенаправления в файл.
* Отображение табличной сетки результатов в виде отдельных окон с вкладками.
* Графическое отображение результатов инструкции Showplan, отражающих логические шаги построения плана выполнения инструкции Transact-SQL. Management Studio при подключении к экземплярам SQL Server 2005 получает план от SQL Server Database Engine в формате XML, а при подключении к экземплярам SQL Server 2000 — в текстовом виде.
* Среда изменения текста с развитыми возможностями, поддерживающая поиск и замену, комментирование блоков, пользовательские шрифты и цвета и нумерацию строк. Некоторые типы редакторов поддерживают дополнительные возможности, такие как структурирование и автозавершение.
* Режим SQLCMD для выполнения сценариев, содержащих команды операционной системы.

Редакторы запросов содержат следующие окна.

* Редактор запросов. Это окно используется для ввода и выполнения сценариев.
* Результаты. Это окно используется для просмотра результатов выполнения запроса. Результаты в нем могут отображаться в виде текста или табличной сетки.
* Сообщения. В этом окне отображаются сведения о том, как выполнен запрос. Например, в окне «Сообщения» могут выводиться сообщения об ошибках или число возвращенных строк.
* Статистика клиента. В этом окне отображаются сведения о выполнении запроса, сгруппированные по категориям. При выборе пункта **Включить статистику клиента** из меню **Запрос** в ходе выполнения запроса появляется окно **Статистика клиента**. Статистика успешно выполненных запросов приводится вместе со средними значениями. Чтобы сбросить средние значения, выберите пункт **Сбросить статистику клиента** в меню **Запрос**.

1. [Диспетчер конфигурации SQL Server](http://msdn.microsoft.com/ru-ru/library/ms174212(SQL.90).aspx) — это средство, предназначенное для управления службами, связанными с SQL Server; для настройки сетевых протоколов, которые используются SQL Server; а также для управления конфигурацией подключений с клиентских компьютеров SQL Server. Диспетчер конфигурации SQL Server представляет собой оснастку консоли управления, доступ к которой можно получить из меню «Пуск» и которую можно добавить в любой экран консоли управления. Консоль управления (mmc.exe) для открытия диспетчера конфигурации SQL Server использует файл SQLServerManager.msc в папке Windows System32. Диспетчер конфигурации SQL Server сочетает в себе функциональные возможности следующих средств SQL Server 2000: программа Server Network Utility, программа Client Network Utility и диспетчер служб.

Диспетчер конфигурации SQL Server и среда SQL Server Management Studio используют инструментарий WMI для просмотра и изменения некоторых параметров сервера. Инструментарий WMI обеспечивает единообразный интерфейс с API-вызовами, которые управляют операциями с реестром, запрашивающими средства SQL Server, а также улучшенный контроль и управление выбранными SQL-службами оснастки «Диспетчер конфигурации SQL Server».

1. Помощник по настройке ядра СУБД
2. **Т-SQL. Командные и объектные полномочия. Команды grant и revoke. Примеры.**

Оператор предоставления привилегий имеет следующий формат:

GRANT {<список действий> | ALL PRIVILEGES}

ON <имя\_объекта>

TO {<имя\_пользователя> | PUBLIC }

[WITH GRANT OPTION ]

Здесь список действий определяет набор действий из общедопустимого перечня действий над объектом данного типа.

Параметр ALL PRIVILEGES указывает, что разрешены все действия из допустимых для объектов данного типа.

<имя\_объекта> — задает имя конкретного объекта: таблицы, представления, хранимой процедуры, триггера.

<имя\_пользователя> или PUBLIC определяет, кому предоставляются данные привилегии.

Параметр WITH GRANT OPTION является необязательным и определяет режим, при котором передаются не только права на указанные действия, но и право передавать эти права другим пользователям. Передавать права в этом случае пользователь может только в рамках разрешенных ему действий.

Общий формат оператора назначения привилегий для объекта типа таблица будет иметь следующий синтаксис:

GRANT {[SELECT][,INSERT][,DELETE][,UPDATE (<список столбцов>)]}

ON <имя\_таблицы>

TO {<имя\_пользователя> | PUBLIC } [WITH GRANT OPTION ]

Тогда резонно будет выполнить следующие назначения:

GRANT INSERT

ON Tab1

TO user2

GRANT SELECT

ON Tab1

TO user3

Эти назначения означают, что пользователь user2 имеет право только вводить новые строки в отношение Tab1, а пользователь user3 имеет право просматривать все строки в таблице Tab1.

Для отмены ранее назначенных привилегий в стандарте SQL определен оператор REVOKE. Оператор отмены привилегий имеет следующий синтаксис:

REVOKE {<список операций> | ALL PRIVILEGES}

ON <имя\_объекта>

FROM {<список пользователей> | PUBLIC }

{CASCADE | RESTRICT }

Параметры CASCADE или RESTRICT определяют, каким образом должна производиться отмена привилегий. Параметр CASCADE отменяет привилегии не только пользователя, который непосредственно упоминался в операторе GRANT при предоставлении ему привилегий, но и всем пользователям, которым этот пользователь присвоил привилегии, воспользовавшись параметром WITH GRANT OPTION.

Например, при использовании операции:

REVOKE ALL PRIVILEGES

ON Tab1

TO user4 CASCADE

будут отменены привилегии и пользователя user5, которому пользователь user4 успел присвоить привилегии.

Параметр RESTRICKT ограничивает отмену привилегий только пользователю, непосредственно упомянутому в операторе REVOKE. Но при наличии делегированных привилегий этот оператор не будет выполнен. Так, например, операция:

REVOKE ALL PRIVILEGES

ON Tab1 TO user4 RESTRICT

не будет выполнена, потому что пользователь user4 передал часть cвоих полномочий пользователю user5.

Посредством оператора REVOKE можно отобрать все или только некоторые из ранее присвоенных привилегий по работе с конкретным объектом. При этом из описания синтаксиса оператора отмены привилегий видно, что можно отобрать привилегии одним оператором сразу у нескольких пользователей или у целой группы PUBLIC.

1. **Т-SQL. Добавление, удаление и обновление данных в представлении. Примеры.**

представление - это сохраняемое в каталоге базы данных выражение запросов, обладающее собственным именем и, возможно, собственными именами столбцов.

create\_view ::=

CREATE [ RECURSIVE ] VIEW table\_name [ column\_name\_comma\_list ]

AS query\_expression

[ WITH [ CASCADED | LOCAL ] CHECK OPTION ]

В операциях выборки к любому представлению можно адресоваться таким же образом, как и к любой базовой таблице. Естественно, возникает вопрос: а можно ли использовать имена представлений и в операциях обновления базы данных и если такая возможность допускается, то как это следует понимать?

Напомним, что в соответствии с семантикой языка SQL при выполнении запроса, в разделе FROM которого прямо или косвенно присутствует имя представления, прежде всего, производится материализация представления, т.е. вычисляется результат соответствующего выражения запросов, сохраняется во временной базовой таблице, и далее запрос выполняется по отношению к этой базовой таблице. Хотя в реализациях SQL обычно стремятся избегать материализации представлений, любая реализация обязана обеспечить такое выполнение запроса над представлением, которое было бы эквивалентно выполнению запроса с явной материализацией представления.

Если допустить выполнение над представлениями операций обновления (сразу заметим, что, вообще говоря, в языке SQL это всегда разрешалось), то в этом случае семантика материализации явно не подходит. На первое место выходит требование, чтобы операция обновления над представлением однозначно отображалась в одну или несколько операций обновления над теми постоянно хранимыми базовыми таблицами, над которыми прямо или косвенно определено данное представление.

1. **Тенденции развития СУБД. Понятие ООСУБД, принципы и проблемы реализации.**

**Объектно-ориентированная СУБД** — реализующая объектно-ориентированный подход. Эта система управления обрабатывает данные как абстрактные объекты, наделённые свойствами, в виде неструктурированных данных, и использующие методы взаимодействия с другими объектами окружающего мира.

Пример Объектно-ориентированной СУБД:

* IBM Lotus Notes/Domino
* Jasmine
* ObjectStore
* Caché
* СООБЗ Cerebrum

### Объектно-ориентированная парадигма.

Сразу же необходимо заметить, что общепринятого определения "объектно-ориентированной модели данных" не существует. Сейчас можно говорить лишь о неком "объектном" подходе к логическому представлению данных и о различных объектно-ориентированных способах его реализации.

Мы знаем, что любая модель данных должна включать три аспекта: структурный, целостный и манипуляционный. Посмотрим, как они реализуются на основе объектно-ориентированная парадигмы программирования:

**Структура:**

Структура объектной модели описываются с помощью трех ключевых понятий:

* **инкапсуляция** - каждый объект обладает некоторым внутренним состянием (хранит внутри себя запись данных), а также набором методов - процедур, с помощью которых (и *только* таким образом) можно получить доступ к данным, определяющим внутреннее состояние объекта, или изменить их. Таким образом, объекты можно рассматривать как самостоятельные сущности, отделенные от внешнего мира.
* **наследование** - подразумевает возможность создавать из классов объектов новые классы объекты, которые наследуют структуру и методы своих предков, добавляя к ним черты, отражающие их собственную индивидуальность. Наследование может быть простым (один предок) и множественным (несколько предков).
* **полиморфизм** - различные объекты могут по разному реагировать на одинаковые внешние события в зависимости от того, как реализованы их методы.

**Целостность данных:**

Для поддержания целостности объектно-ориентированный подход предлагает использовать следующие средства:

* автоматическое поддержание отношений наследования
* возможность объявить некоторые поля данных и методы объекта как "скрытые", не видимые для других объектов; такие поля и методы используются только методами самого объекта
* создание процедур контроля целостности внутри объекта

**Средства манипулирования данными:**

К сожалению, в объектно-ориентированном программировании отсутствуют общие средства манипулирования данными, такие как реляционная алгебра или реляционное счисление. Работа с данными ведется с помощью одного из объектно-ориентированных языков программирования общего назначения, обычно это SmallTalk, C++ или Java.

**Подведем теперь некоторые итоги:**

В объектно-ориентированных базах данных, в отличие от реляционных, хранятся не записи, а объекты. ОО-подход представляет более совершенные средства для отображения реального мира, чем реляционная модель:

* естественное представление данных. В реляционной модели все отношения принадлежат одному уровню, именно это осложняет преобразование иерархических связей модели "сущность-связь" в реляционную модель. ОО-модель можно рассматривать послойно, на разных уровнях абстракции.
* имеется возможность определения новых типов данных и операций с ними.

В то же время, ОО-модели присущ и ряд недостатков:

* осутствуют мощные непроцедурные средства извлечения объектов из базы. Все запросы приходится писать на процедурных языках, проблема их оптимизации возлагается на программиста.
* вместо чисто декларативных ограничений целостности (типа явного объявления первичных и внешних ключей реляционных таблиц с помощью ключевых слов PRIMARY KEY и REFERENCES) или полудекларативных триггеров для обеспечения внутренней целостности приходится писать процедурный код.

Очевидно, что оба эти недостатка связаны с отсутствием развитых средств манипулирования данными. Эта задача решается двумя способами - расширение ОО-языков в сторону управления данными (стандарт ODMG), либо добавление объектных свойств в реляционные СУБД (SQL-3, а также так называемые объектно-реляционных СУБД).

1. **Тенденции развития СУБД. ОРСУБД. Принципы и проблемы реализации . Пример.**

Чтобы определить Объектно-Реляционную Систему Управления Базами Данных (ОРСУБД) достаточно воспользоваться простым уравнением: ОРСУБД = ОСУБД + РСУБД = (O + Р) \* СУ \* БД. На логическом уровне ОРСУБД есть методы обработки СУ применяемые к структуре данных БД, которая характеризуется понятиями О объектная и Р реляционная. Блин, над этим гениальным уравнением я рыдал.

Все необходимое для объектного представления доступно в объектной СУБД (ОСУБД). Обычно ОСУБД приравнивают к OОСУБД, а именно к СУБД интегрированной с Объектно-Ориентированным (OO) языком программирования как C++ и Java. Характерные свойства OОСУБД - 1) **комплексные данные**, 2) **наследование типа**, и 3) **объектное поведение**. Комплексные данные могут быть реализованы через постоянные объекты (persistent objects) и XML. OO языки программирования с их определением класса формируют наследование и объектное поведение.

Реляционный концепт в контексте СУБД определен реляционной моделью доктора Е. F. Codd, которая базируется на отношениях в форме двумерных таблиц рядов и столбцов. Преобразование запросов к реляционной алгебре - основное подтверждение, относящее базу данных к реляционной модели. Это - предубеждение, думать, что язык SQL2 - единственный и необходимый критерий РСУБД, точно так же как думать, что Java - единственный язык ОО программирования. Примечательная особенность РСУБД - возможность обрабатывать быстро большую массу однотипных n-элементных кортежей (рядов или записей).

Три основных свидетельства, 1.комплексные данные, 2.наследование типа, и 3.объектное поведение, достаточны, чтобы классифицировать РСУБД или СУБД как ОРСУБД или ОСУБД соответственно. Внутренний язык СУБД как SQL или Zigzag - не критерий, а только материал для классификации на логическом уровне. Язык Zigzag соотносится с объектным SQL3, по крайней мере, в объектно-ориентированном представлении данных. Zigzag - более выразителен и помогает работать со структурно более гибкими данными. Однако SQL3 с его схематизацией типа позволяет устанавливать более строгий контроль однородных данных. Основное отличие проявляется в том, что Zigzag может видеть тип как объект данных и объект данных как тип для других объектов. Это позволяет создать иерархию данных, не только иерархию типа, семантически более точно.

Как может быть замечено, значения атрибутов не наследуются, так что "типография" и "нерегулярно" повторены в каждом объекте. Есть уверенность, следующие версии SQL решат эту проблему, тип будет содержать статический атрибут со значением подобно статическому полю в Java (статические методы уже существуют в SQL у Oracle и DB2). Такой статический атрибут должен быть полезен в запросе, выбирающем все объекты тех типов, которые имеют нужное значение.

 В будущем SQL должен владеть менее выразительными, чем Zigzag, но разумно простыми конструкциями, чтобы рассматривать типы в роли объектов. Гибкость, основанная на рассмотрении объектов в качестве типов, останется прерогативой таких языков, как Zigzag. Чтобы создать объекты типа  "учебник" и "беллетристика", SQL3 разработчик создаст новые типы и новые таблицы или, что лучше, преобразует существующую таблицу в новые. Zigzag разработчик использует объекты "учебник" и "беллетристика" как уже существующие типы. Например, чтобы добавить объект "физика" он может ограничить себя составлением только одной инструкции подобной *:учебник:физика(...)*.

**Объектно-реляционная СУБД (ОРСУБД)** — реляционная СУБД (РСУБД), поддерживающая некоторые технологии, реализующие объектно-ориентированный подход.

Разница между объектно-реляционными и объектными СУБД: первые являют собой надстройку над реляционной схемой, вторые же изначально объектно-ориентированы. Главная особенность и отличие объектно-реляционных, как и объектных, СУБД от реляционных заключается в том, что О(Р)СУБД интегрированы с Объектно-Ориентированным (OO) языком программирования, внутренним или внешним как C++, Java. Характерные свойства OРСУБД - 1) комплексные данные, 2) наследование типа, и 3) объектное поведение.

**Комплексные данные** могут быть реализованы через постоянно-хранимые объекты (persistent objects). Создание комплексных данных в большинстве существующих ОРСУБД основано на предварительном определении схемы через определяемый пользователем тип (UDT - user-defined type). Используются также встроенные конструкторы составных типов, например массив (ARRAY).

Иерархия структурных комплексных данных предлагает дополнительное свойство, **наследование типа**. То есть структурный тип может иметь подтипы, которые используют все его атрибуты и содержат дополнительные атрибуты, специфицированные в подтипе.

**Объектное поведение** закладывается через описание программных объектов. Такие объекты должны быть сохраняемыми и переносимыми для обработки в базе данных, поэтому они называются обычно как постоянные (или долговременные) объекты. Внутри базы данных все отношения с постоянным программным объектом есть отношения с его объектным идентификатором (OID).

Объектно-реляционными СУБД являются, к примеру, широко известные Oracle Database, Microsoft SQL Server 2005, PostgreSQL, а также Sav Zigzag, IBM Cloudscape, FirstSQL/J

1. **Понятие OLAPи OLTPсистемы. Принципы и реализации многомерных СУБД.**

**OLAP** (англ. *online analytical processing*, аналитическая обработка в реальном времени) — технология обработки информации, включающая составление и динамическую публикацию отчётов и документов. Используется аналитиками для быстрой обработки сложных запросов к базе данных. Служит для подготовки бизнес-отчетов по продажам, маркетингу, в целях управления, т. н. data mining — добыча данных (способ анализа информации в базе данных с целью отыскания аномалий и трендов без выяснения смыслового значения записей).

Причина использования OLAP для обработки запросов — это скорость. Реляционные БД хранят сущности в отдельных таблицах, которые обычно хорошо нормализованы. Эта структура удобна для операционных БД (системы OLTP), но сложные многотабличные запросы в ней выполняются относительно медленно. Более хорошей моделью для запросов, а не для изменения, является Пространственная БД.

OLAP делает мгновенный снимок реляционной БД и структурирует её в пространственную модель для запросов. Заявленное время обработки запросов в OLAP составляет около 0.1 % от аналогичных запросов в реляционную БД.

OLAP-структура, созданная из рабочих данных, называется OLAP-куб. Куб создаётся из соединения таблиц с применением схемы звезды. В центре «звезды» находится таблица фактов, которая содержит ключевые факты, по которым делаются запросы. Множественные таблицы с измерениями присоединены к таблице фактов. Эти таблицы показывают, как могут анализироваться агрегированные реляционные данные. Количество возможных агрегирований определяется количеством способов, которыми первоначальные данные могут быть иерархически отображены.

Например, все клиенты могут быть сгруппированы по городам или по регионам страны (Запад, Восток, Север и т. д.), таким образом, 50 городов, 8 регионов и 2 страны составят 3 уровня иерархии с 60 членами. Также клиенты могут быть объединены по отношению к продукции; если существуют 250 продуктов по 2 категориям, три группы продукции и 3 производственных подразделения, то количество агрегатов составит 16560. При добавлении измерений в схему, количество возможных вариантов быстро достигает десятков миллионов и более.

OLAP-куб содержит в себе базовые данные и информацию об измерениях (агрегатах). Куб потенциально содержит всю информацию, которая может потребоваться для ответов на любые запросы. Из-за громадного количества агрегатов, зачастую полный расчёт происходит только для некоторых измерений, для остальных же производится «по требованию».

Вместе с базовой концепцией существуют три типа OLAP — OLAP со многими измерениями (Multidimensional OLAP — MOLAP), реляционный OLAP (Relational OLAP — ROLAP) и гибридный OLAP (Hybrid OLAP — HOLAP). MOLAP — это классическая форма OLAP, так что её часто называют просто OLAP. Она использует суммирующую БД, специальный вариант процессора пространственных БД и создаёт требуемую пространственную схему данных с сохранением как базовых данных, так и агрегатов. ROLAP работает напрямую с реляционным хранилищем, факты и таблицы с измерениями хранятся в реляционных таблицах, и для хранения агрегатов создаются дополнительные реляционные таблицы. HOLAP использует реляционные таблицы для хранения базовых данных и многомерные таблицы для агрегатов. Особым случаем ROLAP является ROLAP реального времени (Real-time ROLAP — R-ROLAP). В отличие от ROLAP в R-ROLAP для хранения агрегатов не создаются дополнительные реляционные таблицы, а агрегаты рассчитываются в момент запроса. При этом многомерный запрос к OLAP-системе автоматически преобразуется в SQL-запрос к реляционным данным.

Каждый тип хранения имеет определённые преимущества, хотя есть разногласия в их оценке у разных производителей. MOLAP лучше всего подходит для небольших наборов данных, он быстро рассчитывает агрегаты и возвращает ответы, но при этом генерируются огромные объёмы данных. ROLAP оценивается как более масштабируемое решение, использующее к тому же наименьшее возможное пространство. При этом скорость обработки значительно снижается. HOLAP находится посреди этих двух подходов, он достаточно хорошо [масштабируется](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D1%82%D0%B0%D0%B1%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C) и быстро обрабатывается. Архитектура R-ROLAP позволяет производить многомерный анализ OLTP-данных в режиме реального времени.

Сложность в применении OLAP состоит в создании запросов, выборе базовых данных и разработке схемы, в результате чего большинство современных продуктов OLAP поставляются вместе с огромным количеством предварительно настроенных запросов. Другая проблема — в базовых данных. Они должны быть полными и непротиворечивыми.

## Реализации OLAP

Первым продуктом, выполняющим OLAP-запросы, был Express (компания IRI). Однако, сам термин OLAP был предложен [Эдгаром Коддом](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4%D0%B4), «отцом реляционных БД». А работа Кодда финансировалась Arbor, компанией, выпустившей свой собственный OLAP-продукт — Essbase (позже купленный Hyperion, которая в 2007 г. была поглощена компанией Oracle) — годом ранее. Как результат, «[12 законов аналитической обработки в реальном времени](http://ru.wikipedia.org/w/index.php?title=12_%D0%B7%D0%B0%D0%BA%D0%BE%D0%BD%D0%BE%D0%B2_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B9_%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%B2_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8&action=edit&redlink=1)» Кодда появились в их описании Essbase.

Другие хорошо известные OLAP-продукты включают [Microsoft Analysis Services](http://ru.wikipedia.org/w/index.php?title=Microsoft_Analysis_Services&action=edit&redlink=1) (ранее называвшиеся OLAP Services, часть [SQL Server](http://ru.wikipedia.org/wiki/SQL_Server)), [Oracle OLAP Option](http://ru.wikipedia.org/w/index.php?title=Oracle_OLAP_Option&action=edit&redlink=1), [DB2 OLAP Server](http://ru.wikipedia.org/w/index.php?title=DB2_OLAP_Server&action=edit&redlink=1) от [IBM](http://ru.wikipedia.org/wiki/IBM) (фактически, EssBase с дополнениями от IBM), [SAP BW](http://ru.wikipedia.org/w/index.php?title=SAP_BW&action=edit&redlink=1), продукты [Brio](http://ru.wikipedia.org/w/index.php?title=Brio&action=edit&redlink=1), [BusinessObjects](http://ru.wikipedia.org/w/index.php?title=BusinessObjects&action=edit&redlink=1), [Cognos](http://ru.wikipedia.org/wiki/Cognos), [MicroStrategy](http://ru.wikipedia.org/w/index.php?title=MicroStrategy&action=edit&redlink=1) и других производителей.

C технической точки зрения, представленные на рынке продукты делятся на «физический OLAP» и «виртуальный».

В первом случае наличествует программа, выполняющая предварительный расчет агрегатов, которые затем сохраняются в специальную многомерную БД, обеспечивающую быстрое извлечение. Примеры таких продуктов — Microsoft Analysis Services, Oracle OLAP Option, Oracle/Hyperion EssBase, Cognos PowerPlay.

Во втором случае данные хранятся в реляционных СУБД, а агрегаты могут не существовать вообще или создаваться по первому запросу в СУБД или кэше аналитического ПО. Примеры таких продуктов — SAP BW, BusinessObjects, Microstrategy.

Системы, имеющие в своей основе «физический OLAP» обеспечивают стабильно лучшее время отклика на запросы, чем системы «виртуальный OLAP». Поставщики систем «виртуальный OLAP» заявляют о большей масштабируемости их продуктов в плане поддержки очень больших объемов данных.

С точки зрения пользователя оба варианта выглядят похожими по возможностям.

Наибольшее применение OLAP находит в продуктах для бизнес-планирования и [хранилищах данных](http://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

**OLTP** (Online Transaction Processing) — онлайновая обработка транзакций. Способ организации [БД](http://ru.wikipedia.org/wiki/%D0%91%D0%94), при котором система работает с [транзакциями](http://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) небольшими по размерам, но идущими большим потоком, и при этом [клиенту](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%B5%D0%BD%D1%82) требуется от системы максимально быстрое время ответа.

Термин OLTP применяют также к системам (приложениям). OLTP-системы предназначены для ввода, структурированного хранения и обработки информации (операций, документов) **в режиме реального времени**.

## Использование

OLTP-приложениями охватывается широкий спектр задач во многих отраслях - банковские и биржевые операции, в промышленности -регистрация прохождения детали на [конвейере](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B2%D0%B5%D0%B9%D0%B5%D1%80), фиксация в статистике посещений очередного посетителя веб-сайта, автоматизация бухгалтерского, складского учета и учета документов и т. п. Приложения OLTP, как правило, автоматизируют структурированные, повторяющиеся задачи обработки данных, такие как ввод заказов и банковские транзакции. OLTP-системы проектируются, настраиваются и оптимизируются для выполнения максимального количества транзакций за короткие промежутки времени. Как правило, большой гибкости здесь не требуется, и чаще всего используется фиксированный набор надежных и безопасных методов ввода, модификации, удаления данных и выпуска оперативной отчетности. Показателем эффективности является количество транзакций, выполняемых за секунду. Обычно аналитические возможности OLTP-систем сильно ограничены (либо вообще отсутствуют).

## Требования

* Сильно [нормализованные](http://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0) модели данных.
* При возникновении ошибки, транзакция должна целиком откатиться и вернуть систему к состоянию, которое было до начала транзакции.
* Обработка данных в [реальном времени](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F).

## Преимущества, Недостатки

OLTP-системы оптимизированы для небольших дискретных транзакций. А вот запросы на некую комплексную информацию (к примеру поквартальная динамика объемов продаж по определённой модели товара в определённом филиале), характерные для аналитических приложений ([OLAP](http://ru.wikipedia.org/wiki/OLAP)), породят сложные соединения таблиц и просмотр таблиц целиком. На один такой запрос уйдет масса времени и компьютерных ресурсов, что затормозит обработку текущих транзакций.

1. **Распределенные СУБД. Основные принципы реализации.**

СУБД и централизация обработки информации позволили устранить такие недостатки традиционных файловых систем, как несвязанность, несогласованность и избыточность данных. По мере роста баз данных и особенно при их использовании в территориально разделенных организациях появляются другие проблемы. Так, для централизованной СУБД, находящейся в узле телекоммуникационной сети, с помощью которой различные подразделения организации получают доступ к данным, с ростом объема информации и количества транзакций возникают следующие трудности:

* большой поток обменов данными;
* низкая надежность;
* низкая общая производительность;
* большие затраты на разработку.

Хотя в централизованной базе данных легче обеспечить безопасность, целостность и непротиворечивость информации при обновлениях, перечисленные проблемы создают определенные трудности. В качестве возможного решения этих проблем предполагается децентрализация данных. При децентрализации достигается:

* более высокая степень одновременности обработки вследствие распределения нагрузки;
* улучшенное использование данных на местах при выполнении удаленных (дистанционных) запросов;
* меньшие затраты;
* простота управления.

Затраты на создание сети, в узлах которой находятся малые ЭВМ, гораздо ниже, чем затраты на создание аналогичной системы с использованием большой ЭВМ.

Дадим следующее определение:

**распределенная база данных** — это набор файлов (отношений), хранящихся в разных узлах информационной сети и логически связанных таким образом, чтобы составлять единую совокупность данных (связь может быть функциональной или через копии одного и того же файла).

Распределенная база данных предполагает хранение и выполнение функций управления данными в нескольких узлах и передачу данных между этими узлами в процессе выполнения запросов. Разбиение данных в распределенной базе данных может достигаться путем хранения различных таблиц на разных компьютерах или даже хранения разных частей и фрагментов одной таблицы на разных компьютерах. Для пользователя (или прикладной программы) не должно иметь значения, каким образом распределены данные между компьютерами. Работать с распределенной базой данных, если она действительно распределенная, следует так же, как и с централизованной, т. е. размещение базы данных должно быть прозрачно.

Несмотря на то, что распределенная база данных состоит из нескольких локальных баз данных, у пользователя должна сохраняться иллюзия работы с централизованной базой данных, что вызывает потребность в использовании некоторого общего представления о данных — глобальной концептуальной схемы. Определение данных в такой концептуальной схеме должно быть аналогичным определению в централизованной базе данных.

Отличия начинаются, когда требуется хранить данные в нескольких узлах. Чтобы произвести разбиение данных, нужно секционировать таблицы глобальной схемы на фрагменты. Существует два типа секционирования: горизонтальное и вертикальное. При секционировании таблицы по строкам выполняется горизонтальное секционирование, при разбиении по столбцам — вертикальное.

Таким образом, архитектура распределенной СУБД должна содержать информацию о секционировании исходных таблиц базы данных, что предполагает создание дополнительного уровня — фрагментного.

Самый высший уровень архитектуры распределенной СУБД — это интерфейс прикладной программы и интерфейс процессора запросов.

Взгляд на базу данных отдельных пользователей представлен в архитектуре отдельным 1-м уровнем, что аналогично внешнему уровню в классической архитектуре СУБД.

Для реализации и объяснения распределенной природы базы данных выделяются два уровня: фрагментный (см. выше) и уровень распределенного представления. Последний показывает географическое распределение данных по рабочим станциям, расположение экземпляра каждого фрагмента.

1—4 уровни архитектуры распределенной СУБД относятся к сетевой СУБД. Однако выделяют еще локальные СУБД, где определяют представление данных на каждой рабочей станции. Каждый уровень поддерживает различные представления базы данных; каждый уровень взаимодействует только со смежными уровнями представления.

Дейт установил **12 свойств** или качеств идеальной распределенной базы данных:

1. Локальная автономия (local autonomy) – управление данными на каждом из узлов распределенной системы выполняется локально. База данных, расположенная на одном из узлов, является неотъемлемым компонентом распределенной системы. Будучи фрагментом общего пространства данных, она в то же время функционирует как полноценная локальная база данных; управление ею выполняется локально и независимо от других узлов системы.
2. Независимость узлов (no reliance on central site) – в идеальной системе все узлы равноправны и независимы, а расположенные на них базы являются равноправными поставщиками данных в общее пространство данных. База данных на каждом из узлов самодостаточна: она включает полный собственный словарь данных и полностью защищена от несанкционированного доступа.
3. Непрерывность операции (continuous operation) – возможность непрерывного доступа к данным (известная 24 часа в течение суток, семь дней в неделю) в рамках DDB вне зависимости от их расположения и вне зависимости от операций, выполняемых на локальных узлах. Это качество можно выразить лозунгом: данные доступны всегда, а операции над ними выполняются непрерывно.
4. Прозрачность расположения (location independence) – пользователь, обращающийся к DDB, ничего не должен знать о реальном, физическом размещении данных в узлах информационной системы. Все операции над данными выполняются без учета их местонахождения. Транспортировка запросов к базам данных осуществляется встроенными системными средствами.
5. Прозрачная фрагментация (fragmentation independence) – возможность распределенного (то есть на различных узлах) размещения данных, логически представляющих собой единое целое. Существует фрагментация двух типов: горизонтальная и вертикальная. Первая означает хранение строк таблицы на различных узлах (фактически, хранение строк одной логической таблицы в нескольких идентичных физических таблицах на различных узлах). Вторая означает распределение столбцов логической таблицы по нескольким узлам.
6. Прозрачное тиражирование (асинхронный в общем случае процесс переноса изменений объектов исходной базы данных в базы, расположенные на других узлах распределенной системы) (replication independence) – тиражирование возможно и достигается внутрисистемными средствами.
7. Обработка распределенных запасов (distributed query processing) – возможность выполнения операций выборки над распределенной базой данных, сформулированных в рамках обычного запроса на языке SQL.
8. Обработка распределенных транзакций (distributed transaction processing) – возможность выполнения операций обновления распределенной базы данных (INSERT, UPDATE, DELETE), не разрушающегоцелостность и согласованность данных. Эта цель достигается применением двухфазового или двухфазного протокола фиксации транзакций (two-phase commit protocol), ставшего фактическим стандартом обработки распределенных транзакций. Его применение гарантирует согласованное изменение данных на нескольких узлах в рамках распределенной, или глобальной транзакции.
9. Независимость от оборудования (hardware independence) – в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей — от мэйнфреймов до персоналок.
10. Независимость от операционных систем (operationg system independence) – вытекает из предыдущего и означает многообразие операционных систем, управляющих узлами распределенной системы.
11. Прозрачность сети (network independence) – спектр поддерживаемых конкретной СУБД сетевых протоколов не должен быть ограничением системы с распределенными базами данных. Данное качество формулируется максимально широко: в распределенной системе возможны любые сетевые протоколы.
12. Независимость от баз данных (database independence) – в распределенной системе могут мирно сосуществовать СУБД различных производителей, и возможны операции поиска и обновления в базах данных различных моделей и форматов.

Исходя из определения Дэйта, СУБД в общем случае можно рассматривать как слабосвязанную сетевую структуру, узлы которой представляют собой локальные базы данных. Локальные базы данных автономны, независимы и самоопределены; доступ к ним обеспечивается от различных поставщиков. Связи между узлами — это потоки тиражируемых данных. Топология DDB варьируется в широком диапазоне: возможны варианты иерархии, структур типа звезда и т. д. В целом топология DDB определяется географией информационной системы и направленностью потоков тиражирования данных.

Рассмотрим теперь проблемы реальных распределенных баз данных. Проблемы централизованных СУБД существуют и здесь, однако децентрализация добавляет новые:

* Какова общая модель данных распределенной системы? Мы должны иметь единую концептуальную схему всей сети. Это обеспечит логическую прозрачность данных для пользователя, в результате чего он сможет формировать запрос ко всей базе, находясь за отдельным терминалом (т. е. как бы работая с централизованной базой данных).
* Необходима схема, определяющая местонахождение данных в сети. Это обеспечит прозрачность размещения данных, благодаря которой пользователь может не указывать, куда переслать запрос, чтобы получить требуемые данные.
* Распределенные базы данных могут быть однородными или неоднородными по аппаратным и программным средствам. Проблему неоднородности сравнительно легко решить, если распре деленная база является неоднородной по аппаратным средствам, но однородной по программным средствам (одинаковые СУБД в узлах). Если же в узлах распределенной системы используются разные СУБД, необходимы средства преобразования структур данных и языков. Это должно обеспечить прозрачность преобразования в узлах распределенной базы данных.
* Управление словарями. Для обеспечения всех видов прозрачности в распределенной базе данных нужны программы, управляющие многочисленными справочниками или словарями.
* Методы выполнения запросов в распределенной базе данных отличаются от аналогичных методов централизованных СУБД, так как отдельные части запроса нужно выполнять на месте рас положения соответствующих данных и передавать частичные результаты на другие узлы; при этом должна быть обеспечена координация всех процессов.
* В распределенной базе данных нужен сложный механизм управления одновременной обработкой, который, в частности, должен обеспечивать синхронизацию при обновлениях информации, то гарантирует непротиворечивость данных.

Развитая методология распределения и размещения данных, включая разбиение, является одним из основных требований к распределенной базе данных.

**С У Б Д Access**

1. **Общая характеристика и возможности системы.**

**Access** – самая популярная система управления базами данных (СУБД) общего назначения. Это комплекс программных средств, предназначенных для создания структуры новой базы данных, наполнения её содержимым, редактирования содержимого, отбора данных в соответствии с заданными критериями, их упорядочивания, оформления, печати.

**Access** работает под управлением *Windows* и поэтому может использовать все возможности DDE и OLE. DDE позволяет выполнять функции и производить обмен данными между **Access** и любыми другими приложениями *Windows*, поддерживающим DDE. Для осуществления динамического обмена данными с другими приложениями можно использовать макросы или процедуры на Visual Basic.

В **Access** база данных обозначает файл, содержащий набор информации. Каждая база данных в **Access** состоит из основных объектов: *таблиц, запросов, форм, отчётов, страниц, макросов* и *модулей.*

**Access** может работать одновременно только с одной базой данных. Но одна **БД Access** может включать сотни *таблиц*, *форм*, *запросов*, *отчётов*, *макросов* и *модулей*, которые хранятся в одном файле с расширением mdb.

***ТАБЛИЦА –*** это объект, определяемый и используемый для хранения данных. Каждая таблица включает информацию определённого типа. *Таблица* содержит поля (столбцы), в которых хранятся данные и записи (строки). В записи собрана вся информация о конкретном объекте. Для каждой *таблицы* можно определить первичный ключ и один или несколько индексов с целью увеличения скорости доступа к данным.

**Access** позволяет изменить структуру *таблицы*, просматривать, редактировать, удалять и добавлять записи, осуществлять поиск, замену, сортировку данных, изменять вид *таблицы*, создавать связь между таблицами и удалять их.

***ЗАПРОС –*** это объект, который позволяет пользователю получить данные из одной или несколько таблиц.

***ФОРМЫ –*** это объект, предназначенный для просмотра, ввода и редактирования записей базы данных.

***ОТЧЁТ –*** это объект, предназначенный для создания документа, который впоследствии может бать распечатан либо включён в документ другого приложения.

***СТРАНИЦЫ –*** это объект, представляющий собой специальный тип *Web-страниц*, предназначенный для просмотра и работы через Интернет или интрасеть с данными.

***МАКРОС –*** это объект, представляющий собой последовательность макрокоманд для автоматизации наиболее часто выполняемых действий при работе с базой.

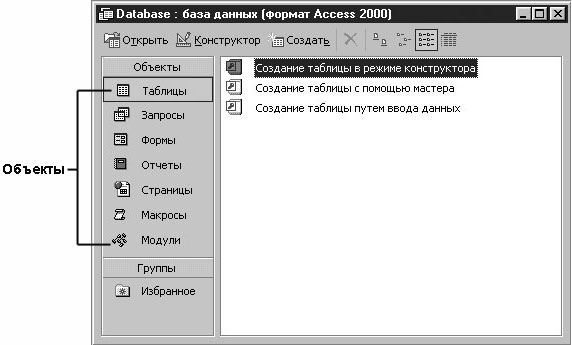
***МОДУЛЬ –*** это объект, автоматизирующий комплексные операции и предоставляющий программисту более полный контроль, чем *макрос*.

1. **Способы представления информации. Примеры.**

Объекты Access могут отображаться в различных режимах. Для таблиц обычно используются **режим таблицы**, предназначенный для ввода данных, и **режим конструктора**, позволяющий просмотреть и модифицировать структуру таблицы. Можно также щелкнуть на стрелке кнопки **Вид (View)** и выбрать нужный режим из списка. В табличном режиме содержимое таблицы отображается в виде столбцов (полей) и строк (записей).

1. **Структура объектов системы и их классификация. Примеры.**

Таблицы представляют собой один из типов объектов, входящих в базу данных Access. На следующем рисунке представлено окно базы данных, где перечислены все типы объектов.



Из всех типов объектов только таблицы предназначены для хранения информации. Остальные используются для просмотра, редактирования, обработки и анализа данных - иначе говоря, для обеспечения эффективного доступа к информации.

**Таблицы:** Таблицы играют ключевую роль в базах данных, поскольку именно в них хранится информация. База данных может содержать тысячи таблиц, размеры которых ограничиваются только доступным пространством на жестком диске компьютера. Для таблиц обычно используются режим таблицы, предназначенный для ввода данных, и режим конструктора, позволяющий просмотреть и модифицировать структуру таблицы.

**Запросы:** Запросы предназначены для поиска в базе данных информации, отвечающей определенным критериям. Найденные записи, называемые результатами запроса, можно просматривать, редактировать и анализировать различными способами. Кроме того, результаты запроса могут использоваться в качестве основы для создания других объектов Access. В сущности, запрос представляет собой вопрос, сформулированный в терминах базы данных. Существует различные типы запросов. Наиболее распространенными являются запросы на выборку, параметрические и перекрестные запросы. Для создания простых запросов используется мастер, в менее тривиальных случаях можно создать запрос вручную в режиме конструктора.

**Формы:** Информация хранится в таблицах в том виде, в котором была введена. Это не имеет особого значения, если никто, кроме вас, не работает с базой данных. Однако если база данных предназначена для пользователей, имеющих смутное представление об Access, работа с таблицами может показаться им чрезмерно сложной, не говоря уже о том, как это отразится на сохранности информации. В таких случаях лучше воспользоваться формами, которые позволяют упростить и сделать более эффективными ввод и обработку содержимого таблиц.В сущности, форма представляет собой окно, куда можно поместить элементы управления, предназначенные для ввода и отображения данных. Access включает панель, которая содержит многие стандартные элементы управления Windows, в том числе поля, надписи, флажки и кнопки выбора. Не требуется особых талантов, чтобы с помощью этих элементов создать формы, которые выглядят и функционируют примерно так же, как диалоговые окна в приложениях Microsoft Windows.Формы используются для ввода и редактирования записей в таблицах базы данных. Подобно таблицам и запросам, их можно отображать в трех режимах: в режим формы, предназначенном для ввода данных, в режиме таблицы, где данные представлены в табличном формате, и в режиме конструктора, позволяющем изменить внешний вид, содержание и функционирование формы. На следующем рисунке приведен пример формы в режиме конструктора.

**Отчеты:** Отчеты используются для отображения информации, содержащейся в таблицах, в отформатированном виде, который легко читается как на экране компьютера, так и на бумаге. Помимо данных, извлеченных из нескольких таблиц и запросов, отчеты могут включать элементы оформления, свойственные печатным документам, как, например, названия, заголовки и колонтитулы.Отчет можно отобразить в трех режимах: в режиме конструктора, позволяющем изменить внешний вид и макет отчета, в режиме просмотра образца, где можно просмотреть все элементы готового отчета, но в сокращенном виде, и в режиме предварительного просмотра, где отчет отображается в том виде, в каком будет напечатан.

**Страницы :** Чтобы предоставить доступ к информации, хранящейся в базе данных, пользователям Интернета или интранета, можно создать страницы, называемые страницами доступа к данным. Работа с данными на странице доступа в Web осуществляется примерно так же, как в Access - пользователи могут просматривать таблицы, выполнять запросы и заполнять поля форм.Хотя публикация информации из базы данных в Web на первый взгляд кажется сложной, Access включает мастер, которые берет на себя большую часть кропотливой работы по созданию страницы доступа. При желании созданную мастером страницу можно доработать в режиме конструктора.

**Макросы:** Макросы представляют собой небольшие программы, с помощью которых обеспечивается реакция Access на такие события, как открытие формы, щелчок кнопки или обновление записи. Это особенно удобно, если предполагается передать базу данных неквалифицированным пользователям. Например, можно написать макросы, содержащие последовательность команд, выполняющих рутинные задачи, или связать такие действия, как открытие формы или печать отчета, с кнопками кнопочной формы.

**Модули:** Модули представляют собой программы на Visual Basic for Applications (VBA), языке программирования высокого уровня, разработанного Microsoft для создания приложений Windows. Помимо стандартного набора команд VBA, каждая программа Microsoft Office имеет собственные команды. В отличие от макросов, позволяющих автоматизировать не более пяти, шести десятков операций, VBA включает сотни команд и может неограниченно расширяться за счет дополнений, вносимых другими компаниями и частными лицами. Программы VBA используются для решения задач, слишком сложных для макросов, как, например, извлечение определенной информации из рабочих листов Excel.

1. **Средства создания и коррекции структуры базы данных. Примеры.**

С помощью мастера **Создание баз данных (Database Wizard)** можно создать быстро базу данных, хотя мастер не всегда создает то, что нужно, он создает нечто, весьма близкое, и очень быстро.

1. Создание файла (New File)

2. На панели задачи в разделе Создание с помощью шаблона (New from template) щелкните на Общие шаблоны (General templates), а затем щелкните на вкладке Базы данных (Databases), чтобы отобразить возможные варианты (шаблоны).

3. Щелкните дважды на Контакты (Contact Management). Появится диалоговое окно Файл новой базы данных (File New Database), позволяющее ввести имя и расположение новой базы данных.

4. Появится первая страница мастера Создание баз данных (Database Wizard), где описывается тип информации, которая будет храниться в базе данных.

5. В левой части страницы указаны три таблицы, входящие в базу данных, а в правом окошке перечислены поля, которые могут быть включены в эти таблицы. Обязательные поля помечены флажками, необязательные отображаются курсивом.

Чтобы включить поле в таблицу, следует выделить соответствующий флажок.

6. Поочередно щелкните на каждой таблице и просмотрите списки доступных полей.

7. Просмотрите все стили, щелкнув на каждом из них. Завершив просмотр стилей.

8. Поочередно щелкните на каждом стиле оформления отчетов.

9. Оставьте заданные по умолчанию установки без изменений и щелкните на кнопке Готово (Finish).

После завершения работы мастера откроется созданная база данных Контакты с главной кнопочной формой на первом плане.

10. Создание файла (New File)

11. На панели задачи в разделе Создание (New) щелкните на кнопке Новая база данных (Blank Database).

Появится окно базы данных, которое не содержит ни таблиц, ни форм, ни запросов, ни каких-либо других объектов.

12. На панели инструментов окна базы данных щелкните на кнопке Создать (New), чтобы отобразить диалоговое окно Новая таблица (New Table).

13. Щелкните дважды на пункте Мастер таблиц (Table Wizard), чтобы отобразить первую страницу мастера.

1. **Организация обработки данных. Примеры.**

База данных, в которой отображается текущая деятельность компании, полезна только в том случае, если информация в ней постоянно обновляется, а поиск нужных сведений осуществляется быстро. Хотя ввод, корректировка и извлечение табличных данных не представляют особого труда, едва ли можно поручить эти функции пользователям, не

имеющим представления об Access. Дело не только в неэффективности подобного подхода, но и в количестве ошибок, возникающих при работе в режиме таблицы, особенно при вводе информации в связанные таблицы. Решением этой проблемы - и первым шагом на пути превращения базы данных в приложение – являются формы. Форма представляет собой объект базы данных, содержащий упорядоченный набор элементов управления, которые обеспечивают интерактивное взаимодействие с полями одной или нескольких таблиц. С помощью элементов управления можно вводить новые данные, редактировать и удалять существующие и осуществлять поиск информации.

Подобно печатным формам, формы Access включают поля, предназначенные для ввода данных, и надписи к ним. Но в отличие от печатных форм, они могут включать такие элементы, как кнопки выбора или командные кнопки, что превращает формы Access в объекты, подобные диалоговым окнам Windows или страницам мастеров. Как и любой другой объект Access, форма может быть создана вручную или с помощью мастера. Формы, предназначенные для перемещения по объектам и организации работы, рекомендуется создавать вручную в режиме конструктора. Формы же, основанные на таблицах, следует создавать с помощью мастера и при необходимости дорабатывать в режиме конструктора. И не потому, что создание формы вручную требует особых усилий - просто нет смысла делать то, с чем легко справится мастер. режде чем приступать к созданию формы, нужно решить, на какой таблице она базируется и

для каких целей предназначена. Решив эти вопросы, можно создать форму, воспользовавшись мастером Создание формы (Form Wizard). Если созданная мастером форма не совсем отвечает вашим нуждам, ее можно модифицировать в режиме

конструктора, как и большинство объектов Access.

1. На панели объектов щелкните на пункте Формы (Forms).

2. Щелкните дважды на команде Создание формы с помощью мастера (Create form by using wizard).

3. В списке Таблицы и запросы (Table/Query) щелкните на Таблица: Клиенты (Table: Customers), чтобы отобразить поля этой таблицы в списке Доступные поля (Available Fields).

4. Щелкните на кнопке >>, чтобы переместить все поля таблицы Клиенты в список Выбранные поля (Selected Fields), и щелкните на кнопке Далее (Next). Вторая страница мастера позволяет выбрать внешний вид формы. Если щелкнуть на одном из вариантов, представленных справа, в левой части страницы отобразится соответствующий макет.

5. Щелкните на варианте В один столбец (Columnar), а затем щелкните на кнопке Далее (Next). Появится страница, позволяющая выбрать стиль оформления формы. Каждый стиль можно просмотреть на образце, выделив его в списке.

6. Так как форма базируется на таблице Клиенты, Access предлагает имя таблицы в качестве заголовка формы. Примите предложение, оставьте выделенным флажок Открыть форму для просмотра и ввода данных (Open the form to view or enter

information) и щелкните на кнопке Готово (Finish). Откроется новая форма Клиенты, в которой отображается первая запись таблицы Клиенты.

7. Воспользуйтесь кнопками перехода внизу формы, чтобы просмотреть несколько записей. Как и с таблицами, с формами можно работать в нескольких режимах. Чаще всего используются режим формы, предназначенный для просмотра и корректировки данных, и режим конструктора, позволяющий добавлять элементы управления, изменять их свойства и положение.

В созданной мастером форме каждое поле, выбранное из таблицы, представлено двумя элементами управления: полем (куда вводятся данные) и надписью к нему. Такие формы связаны с конкретной таблицей, которая является источником записей для формы, а поле таблицы является источником данных для элемента управления. Каждый элемент

управления обладает набором свойств, таких как шрифт и его атрибуты, цвет фона или стиль обрамления. По умолчанию мастер присваивает всем свойствам значения, но их можно изменить, чтобы улучшить внешний вид формы и ее функционирование.

1. **Способы ускорения поиска данных: индексация и сортировка. Примеры.**

Индекс – средство, ускоряющее поиск и сортировку в таблицы за счет использования ключевых значений, которое позволяет обеспечить уникальность строк таблицы. Access автоматически индексирует поле первичного ключа. Для этого нужно выделить в режиме конструктора таблиц поле или поля, которые нужно проиндексировать, щелкнуть правой

кнопкой мыши, и выбрать в контекстном меню пункт Ключевое поле. Кроме того, в том же режиме конструктора можно на вкладке свойств поля выбрать свойство Индексированное поле для любого поля. Потом при желании иметь отсортированную по возрастанию (убыванию) информацию, можно, нажав кнопку Индексы на панели инструментов, выбрать в поле Порядок сортировки соответствующую строку и установить желаемый порядок. Сортировка базы данных в Access представляет собой очень простое действие. При открытии таблицы можно выбрать меню Записи/Сортировка или нажать на панели инструментов кнопки Сортировка по возрастанию (убыванию). Access, выполняя эту команду, отсортирует записи по данному полю так, что текст идет в алфавитном порядке, в порядке возрастания чисел и от более ранней даты к более поздней, – и наоборот. При индексировании создается отдельный индексный файл, при этом сама база данных остается неизменной. При сортировке же изменяется база данных, порядок следования записей.

1. **Способы организации связи между файлами. Примеры.**
2. **Средства создания приложений Примеры.**

Формы предназначены для вывода данных на экран в удобном виде, форма может использоваться для поиска данных. Если изъять формы из MS Access, то программа превратится в заурядную СУБД, каких множество. С одной стороны, формы позволяют пользователям вводить данные в таблицы базы данных без непосредственного доступа к самим таблицам. С другой стороны, они позволяют выводить результаты работы запросов не в виде скупых результирующих таблиц, а в виде красиво оформленных форм. В связи с таким разделением существует два вида формирования структуры форм: на основе таблицы и на основе запроса, хотя возможен и комбинированный подход, – это вопрос творчества [1].

Создадим форму «Список студентов по группам» в режиме мастера форм (одиночная на основании таблицы «Студент», в столбец). Перед тем, как создать эту форму, создадим запрос «Список студентов по группам»:

SELECT Группа.[Обозначение группы], Группа.[Количество студентов], Группа.[Средний балл в группе при поступлении], Студент.[Номер зачетной книжки], Студент.Фамилия, Студент.Имя, Студент.Отчество, Студент.[Год рождения], Студент.[Балл при поступлении]

FROM Группа INNER JOIN Студент ON Группа.[Код группы]=Студент.[Код группы]

ORDER BY Студент.[Номер зачетной книжки], Студент.Фамилия;

После создания, переименуйте эту форму как «Список студентов по группам» и

просмотрите данные через форму (рис. 26).

Форма «Список студентов по группам»

Формы предназначены и для заполнения базы данных пользователями. Создадим в режиме автоформы формы «Группа» и «Студент» и введем в формы данные (рис. 27). В соответствующих таблицах базы данных появились новые, введенные нами, данные для группы ДХГ-31. Аналогично в режиме автоформы следует создать формы «Кафедра»,

«Преподаватель», «Предмет», «План», «Успеваемость».

Заполнение таблицы «Группа» базы через автоформу «Группа» Для вывода на печать документов на основе данных из базы используются отчеты. Отчеты во многом похожи на формы, но имеют иное функциональное назначение – они

служат для форматированного вывода данных на печатающие устройства и, соответственно, при этом должны учитывать параметры принтера и параметры используемой бумаги. Создадим отчет «Список студентов по группам» в режиме мастера отчетов на основании запроса «Список студентов по группам», выполненного ранее. Отредактируем в

режиме конструктора отчет, если это требуется, чтобы привести отчет в пригодный для печати вид (рис. 28).

Аналогично, следует создать отчет «Список преподавателей по кафедрам» в режиме мастера отчетов на основании запроса «Список преподавателей по кафедрам», который сформирован на основе рис. 3 предметной области.



Отчет «Список студентов по группам» подготовленный для печати

1. **Средства задания ссылочной целостности.**

***Ссылочная целостность*** – это ограничение базы данных, гарантирующее, что ссылки между данными являются действительно правомерными и неповрежденными. Ссылочная целостность является фундаментальным принципом теории баз данных и проистекает из той идеи, что база данных должна не только сохранять данные, но и активно содействовать обеспечению их качества

В Microsoft Access обеспечивается возможность автоматической проверки целостности данных в связанных полях. *Целостность даных* означает систему правил, используемых для поддержания связей между записями в связанных таблицах, а также для обеспечения защиты от случайного удаления или изменения связанных данных. Установить проверку целостности данных можно, если выполнены следующие условия:

* связанное поле главной таблицы является ключевым полем или имеет уникальный индекс;
* связанные поля имеют один тип данных. Здесь существует два исключения. Поле счетчика может быть связано с числовым полем, если в последнем поле в свойстве **Размер поля** (FieldSize) указано значение **Длинное целое** (Long Integer), или в обоих полях свойство **Размер поля** (FieldSize) имеет значение **Код репликации** (Replication ID);
* обе таблицы принадлежат одной базе данных Microsoft Access.

Чтобы обеспечить целостность, работа с данными должна производиться с учетом нижеперечисленных правил.

* Невозможно ввести в связанное поле подчиненной таблицы значение, отсутствующее в связанном поле главной таблицы. Однако можно ввести пустое значение, показывающее, что для данной записи связь отсутствует.
* Не допускается удаление записи из главной таблицы, если существуют связанные с ней записи в подчиненной таблице.
* Невозможно изменить значение ключевого поля в главной таблице, если существуют записи, связанные с данной таблицей.

Чтобы эти правила контролировались для конкретной связи, при ее создании следует установить флажок **Обеспечение целостности данных** (Enforce Referential Integrity). Тогда любая попытка выполнить действие, нарушающее одно из перечисленных выше правил, приведет к выводу на экран предупреждения, а само действие выполнено не будет.

Чтобы преодолеть ограничения на удаление или изменение связанных записей, сохраняя при этом целостность данных, следует установить флажки **каскадное обновление связанных полей** (Cascade Update Related Fields) и **каскадное удаление связанных записей** (Cascade Delete Related Records). Если установлен флажок **каскадное обновление связанных полей** (Cascade Update Related Fields), то при изменении ключевого поля главной таблицы автоматически будут изменены и соответствующие значения поля связанных записей. Если установлен флажок **каскадное удаление связанных записей** (Cascade Delete Related Records), то при удалении записи в главной таблице удаляются и все связанные записи в подчиненной таблице.

**CASE-средство Erwin**

1. **CASE-средство ERwin. Назначение, состав и характеристика инструментальных средств Erwin. Основные этапы проектирования концептуальной модели предметной области и КМ базы данных с использованием CASE-средства ERwin. Примеры.**

ERwin - средство разработки структуры базы данных (БД). ERwin сочетает графический интерфейс Windows, инструменты для построения ER-диаграмм, редакторы для создания логического и физического описания модели данных и прозрачную поддержку ведущих реляционных СУБД и настольных баз данных. С помощью ERwin можно создавать или проводить обратное проектирование (реинжиниринг) баз данных.

Реализация моделирования в ERwin базируется на теории реляционных баз данных и на методологии IDEF1X.  
Методология IDEF1X была разработана для ВВС США и теперь используется, в частности, в правительственных, аэрокосмических и финансовых учреждениях, а также в большом числе частных компаний.

Возможны две точки зрения на информационную модель и, соответственно, два уровня модели. Первый - логический (точка зрения пользователя) - описывает данные, задействованные в бизнесе предприятия. Второй - физический - определяет представление информации в БД. ERwin объединяет их в единую диаграмму, имеющую несколько уровней представления.

1. **CASE-средство ERwin. Компоненты диаграммы Erwin и основные виды представления диаграммы. Инструменты для создания логической модели БД.**

Компоненты диаграммы ERwin и основные виды представлений диаграммы

Диаграмма ERwin строится из трех основных блоков - сущностей, атрибутов и связей. Если рассматривать диаграмму как графическое представление правил предметной области, то сущности являются существительными, а связи - глаголами.  
Выбор между логическим и физическим уровнем отображения осуществляется через линейку инструментов или меню. Внутри каждого из этих уровней есть следующие режимы отображения:

* Режим "сущности" - внутри прямоугольников отображается имя сущности (для логической модели) или имя таблицы (для физического представления модели); служит для удобства обзора большой диаграммы или размещения прямоугольников сущностей на диаграмме.
* Режим "определение сущности" служит для презентации диаграммы другим людям.
* Режим "атрибуты". При переходе от предметной области к модели требуется вводить информацию о том, что составляет сущность. Эта информация вводится путем задания атрибутов (на физическом уровне - колонок таблиц). В этом режиме прямоугольник-сущность делится линией на две части - в верхней части отображаются атрибуты (колонки), составляющие первичный ключ, а в нижней - остальные атрибуты. Этот режим является основным при проектировании на логическом и физическом уровнях.
* Режим "первичные ключи" - внутри прямоугольников - сущностей показываются только атрибуты/колонки, составляющие первичный ключ.
* Режим "пиктограммы". Для презентационных целей каждой таблице может быть поставлена в соответствие пиктограмма (bitmap).
* Режим "показ глагольной фразы". На дугах связей показываются глагольные фразы, связывающие сущности (для логического уровня) или имена внешних ключей (для физического уровня).

Диаграмма может занимать более чем один экран и более чем один лист при печати. Для обзора модели предусмотрены, кроме прокруток экрана, режимы уменьшения/увеличения изображения, отображение всей модели, отображение выделенной части модели.

Инструменты для создания модели в ERwin

Основные инструменты создания модели доступны как из меню, так и через окно инструментов. С их помощью создаются независимые и зависимые сущности, идентифицирующие и неидентифицирующие связи, полные и неполные категории, неспецифические связи и текстовые элементы.  
Нажатием мыши над сущностью производится вход в один из многочисленных редакторов ERwin:

* редакторы, связанные с сущностью в целом (определение сущности, дополнительная информация, триггеры, индексы, характеристики таблицы, хранимые процедуры, связанные с таблицей);

редакторы атрибутов (определение атрибутов, колонки таблицы в физическом представлении модели, репозитарий средства 4GL, например, расширенные атрибуты в PowerBuilder ).

1. **Сущности и связи в ERwin. Альтернативные ключи, инвертированные индексы, унификация атрибутов, связи категоризации.**

Анатомия сущности

*Сущность* служит для представления набора реальных или абстрактных предметов (людей, мест, событий и т.п.), которые обладают общими атрибутами или характеристиками. Сущность - “логический” объект, который в физической среде СУБД представлен таблицей. Сущность в ERwin обычно описывает три части информации: атрибуты, являющиеся первичными ключами, неключевые атрибуты и тип сущности.

Анатомия связи

Данные, относящиеся к связям, очень важны и часто являются критическими данными, которые мы используем в повседневном бизнесе. Например, важно знать о каком-то типе инструмента, но знание того, к кому относится конкретный инструмент (связь между человеком и инструментом) может иметь критическую важность. Связь - это соотношение либо между двумя сущностями, либо между сущностью и этой же сущностью. Связь - “логический” объект, представленный одним или несколькими атрибутами - внешними ключами. Связь в ERwin обычно содержит пять типов информации: тип связи, родительский конец связи, дочерний конец связи, знак “обязательности” связи и кардинальность связи.

Назначение альтернативных ключей

В целях улучшения работы базы данных в таблице обычно имеется несколько индексов. Преимущество множественных индексов в том, что у Вас есть несколько точек доступа к данным в таблице. В ER*win* атрибут(ы) первичного ключа автоматически индексируются. Кроме этого, индексируются альтернативные ключи и Inversion Entry. *Альтернативным ключом* называется атрибут или группа атрибутов, уникальным образом определяющие экземпляр сущности. Если у сущности есть несколько атрибутов, уникальным образом определяющих каждый экземпляр, то Вы можете назначить любой из этих атрибутов, за исключением атрибутов первичного ключа, альтернативным ключом, и ER*win* создаст дополнительные индексы.

ER*win* создает уникальный индекс для каждого альтернативного ключа. Информация об альтернативных ключах вводится в редакторе Entity-Attribute.

**Как назначить атрибут или группу атрибутов альтернативным ключом**

1. Войдите в редактор Entity-Attribute.

2. Щелкните по концу атрибута, который находится в неключевом окне и который Вы хотите назначить альтернативным ключом или частью ключа.

3. Введите “(АК1)”, т.е. альтернативный ключ 1, после имени каждого атрибута, входящего в альтернативный ключ 1.

Например, если у Вас три атрибута вместе составляют альтернативный ключ, то каждый из них должен заканчиваться на “(АК1)”, и все три будут по очереди проиндексированы вместе.

4. Нажмите “ОК” для сохранения изменений.

У сущности может быть несколько альтернативных ключей. Они нумеруются последовательно - АК1, АК2, АК3 и т.д. Если атрибут - часть нескольких альтернативных ключей, отделите метки разных ключей, стоящие в скобках, запятыми (АК1, АК2).

**Порядок создания альтернативных ключей**

При генерации схемы индексы создаются в определенном порядке. Сначала создается индекс первичного ключа, затем индексы альтернативных ключей: АК1, АК2, АК3, затем индексы Inversion Entry: IE1, IE2, IE3 и т.д.

**Связь подтипа**

Связью подтипа (другое название - категоризационная связь) называют связь между сущностью подтипа и ее групповым родителем. Связь подтипа всегда связывает один экземпляр группового родителя с 0 или одним экземпляром подтипа.

**Унификация**

Слияние двух или более атрибутов внешнего ключа в один атрибут внешнего ключа на основе утверждения, что значения исходных атрибутов внешнего ключа должны быть идентичны.

1. **Прямое и обратное проектирование. Синхронизация с базой данных. Интерфейсы к СУБД. Поддержка задания правил целостности и начальных значений.**

**Обратное проектирование (Reverse Engineering) объектов памяти**

Когда Вы производите обратное проектирование базы данных, ER*win* может импортировать имена и определения объектов физической памяти, определенных на сервере, таким же образом, каким он импортирует физические таблицы, индексы и другую информацию по физической схеме.

Когда Вы импортируете информацию об объектах физической памяти из сервера, ER*win* использует информациюоб адресе каждой таблицы базы данных, так что впоследствии Вы можете заново создать базу данных, используя то же самое распределение памяти. Вам *не* нужно заново вручную распределять память под таблицы.

После того как Вы импортировали объекты физической памяти в ER*win*, Вы можете просматривать или изменять определения объекта и связи таблиц в редакторах Physical Object и Table Property так же, как Вы работали с объектами физической памяти, созданными в ER*win*.

**Прямое проектирование объектов физической памяти**

Если Вы генерируете физическую схему в ORACLE, то Вы можете включить любую базу данных, табличное пространство или сегменты отката, которые Вы определили в ERwin, как часть схемы. ERwin автоматически транслирует определения физических объектов в команды CREATE TABLE, CREATE TABLESPACE, CREATE ROLLBACK SEGMENT и вставляет информацию о заданных параметрах с соблюдением синтаксиса SQL.

Когда Вы генерируете физическую схему, ERwin сначала создает заданные родительские объекты памяти, затем - дочерние объекты памяти и физические таблицы, расположенный по заданным адресам в памяти.

**Синхронизация физических объектов**

После того как Вы произвели обратное проектирование базы данных, если Вы добавляете или изменяете объекты в ERwin или создаете, удаляете и изменяете объекты непосредственно на сервере базы данных, возникнут %расхождения между моделью данных и физической схемой. Когда это происходит, ERwin позволяет Вам синхронизировать определения, так чтобы информация, хранимая в ERwin, соответствовала информации, хранящейся на сервере.

Во время процесса синхронизации ERwin сравнивает имена логической модели и имена физической схемы, хранящиеся на сервере, и показывает Вам, какие из объектов схемы не синхронизированы.

**Интерфейсы к СУБД**

ERwin поддерживает прямой интерфейс с основными СУБД: DB2 версии 2 и 3, Informix версий 5.1, 6.0, 7.1, Ingres, NetWare SQL, ORACLE версий 6 и 7, Progress, Rdb версий 4 и 6, SQL/400 версий 2 и 3, SQLBase версий 5 и 6, SQL Server версий 4 и 6, Sybase версии 4.2, Sybase System 10 и 11, Watcom SQL. Отметим, что поддерживаются как самые современные, так и предыдущие версии основных СУБД.

ERwin поддерживает также настольные (desktop) СУБД: Microsoft Access, FoxPro, Clipper, dBASE III, dBASE IV и Paradox.  
 Проектирование на физическом уровне выполняется в терминах той базы данных, которую предполагается использовать в системе. Важно, что ERwin "известны" соответствия между возможностями СУБД различных производителей, вследствие чего возможно преобразование физической схемы, спроектированной для одной СУБД, в другую.   
 Для создания физической структуры БД может быть запрошена генерация DDL-скрипта (data definition language). При этом используется диалект SQL для выбранного типа и версии сервера. Хотя сгенерированный код не нуждается в модификации, имеется возможность его сохранить в файл или распечатать.

**Правила и начальные значения**

В ERwin поддерживаются два типа правил (проверок допустимости значений) и начальных (по умолчанию) значений. Правило и умолчание может быть указано для проверки со стороны клиента (например, в PowerBuilder) и со стороны сервера.   
 При задании правила или умолчания для клиентской части эти атрибуты переносятся в репозитарий средства 4GL.  
 Показан диалог для задания значений по умолчанию, устанавливаемых в PowerBuilder. Заметьте, что в одном и том же диалоге задаются умолчания, подставляемые как на стороне клиента, так и на стороне сервера (в данном случае - Sybase).

1. **Генерация отчетов.**

По завершении работы над информационной моделью, как правило, распечатываются логический и физический уровни диаграммы, а также отчет по соответствиям сущность-таблица, атрибут-имя колонки, сущность-атрибуты. Диаграмма физической модели является необходимым, почти достаточным и очень удобным материалом для разработчиков программ. Дополнительная информация для группы разработчиков прикладных программ содержится в отчете "имена таблиц и колонок", который может быть легко построен с помощью ERwin. Выбор режима построения отчета показан на рис.12.

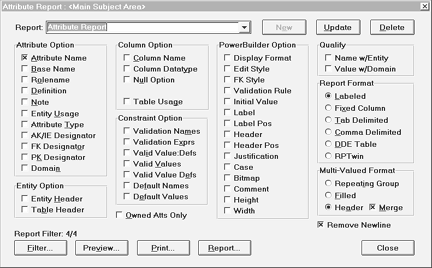


Рис. 12. Варианты выдачи отчета

Сгенерированный отчет может быть сохранен на диск (колонки разделяются запятыми, выравниваются или разделяются табуляцией), или передан в текстовый процессор (или электронную таблицу) через интерфейс DDE.  
Для подготовки развитых отчетов может быть использован специальный генератор отчетов фирмы Logic Works - RPTwin, который интегрирован с ERwin. **Декабрь2010.**