

Вопросы по курсу «Управление данными»

1. Определения: управление данными, база данных, система управления базами данных.

Управление данными - деятельность, направленная на определение, создание, хранение, поддержку данных, а также на обеспечении доступа к данным и процессам манипулирования ими в одной или более информационной системе

База данных (БД) - совокупность специальным образом организационных данных, хранимых в памяти вычислительной системы, отображающих состояние объектов и их взаимосвязей в предметной области

Система управления базами данных (СУБД) - совокупность программных средств для создания, ведения и совместного использования баз данных многими пользователями

2. Понятие системы базы данных.

С точки зрения общего управления, система баз данных включает в себя 5 частей:

- Аппаратное обеспечение (компьютер, устройства хранения данных, сетевые устройства);
- Программное обеспечение (операционная система, СУБД, прикладные программы);
- Люди (все пользователи баз данных);
- Процедуры и стандарты - инструкции и правила, которые определяют использование системы баз данных, обеспечивают системы и стандарты по которым ведется бизнес в организации;
- Данные.

3. Основные функции СУБД.

Основные функции СУБД

- управление данными во внешней памяти (обеспечивает необходимые структуры во внешней памяти, как для хранения данных входящих в БД, так...);
- управление буферами оперативной памяти (буфер - область оперативной памяти, предназначенная для ускорения обмена между внешней и оперативной памятью);
- управление транзакциями (транзакция - последовательность операций над БД, рассматриваемая как единое целое: либо транзакция вся выполняется, тогда СУБД фиксирует все изменения, либо никакие);
- ведение журнала изменений (выполняется СУБД для восстановления последнего согласованного состояния базы данных после аппаратного или программного сбоя), журнал СУБД - это особая БД или часть основной БД, недоступная пользователю и используемая для записи информации обо всех измерениях БД;
- поддержка языков БД (язык баз данных (ГОСТ 33707) - язык с использованием формального синтаксиса, предназначенный для определения ю, создания, организации доступа и поддержки базы данных).

4. Типовая организация современной СУБД.

Компоненты СУБД

Ядро СУБД отвечает за:

- управление данными во внешней памяти (менеджер данных)
- управление буферами оперативной памяти (менеджер буферов)
- управление транзакциями (менеджер транзакций)

- журнализацию (менеджер журнала)

Компилятор (процессор) языка БД преобразует операторы языка БД в некоторую выполняемую программу, оптимизирует запросы на извлечение и изменение данных

Подсистема поддержки времени исполнения интерпретирует программы управления данными, создающие пользовательский интерфейс с СУБД

Утилиты предназначены для оказания помощи администраторам БД в эффективном администрировании БД

5. Классификация баз данных на основе хранения и доступа к данным.

На основе хранения и доступа к данным

- Настольные (индивидуального доступа, desktop)
- С единым хранением данных
- Распределенные

Настольные БД – это база данных, предназначенная для локального использования одним пользователем. Локальные БД могут создаваться каждым пользователем самостоятельно, а могут извлекаться из общей БД.

СУБД: ACCESS, Paradox, dBase, FoxPro. **Настольные СУБД** — это набор различных средств и методов, которые предназначены для формирования, сопровождения и использования информационных баз данных.

БД с единым хранением – обеспечивают единое хранение данных и работу с этими данными многих пользователей Два подхода к построению: файл-серверные (БД хранится на сервере, сервер не выполняет обработку данных) и клиент-серверные системы

Распределенная БД (РБД) – совокупность логически взаимосвязанных разделяемых данных, физически распределенных в компьютерной сети. Распределенные БД, кроме того, имеют характерные особенности, связанные с тем, что физически разные части БД могут быть расположены на разных ЭВМ, а логически, с точки зрения пользователя, они должны представлять собой единое целое.

6. Архитектура «Клиент-сервер» и «Файл-сервер», их сравнение.

Сетевое многопользовательское приложение строится по принципу **файл-серверной архитектуры**. Данные в виде одного или нескольких файлов размещаются на файловом сервере. Файловый сервер принимает запросы, поступающие по сети от компьютеров-клиентов, и передает им требуемые данные. Однако обработка этих данных выполняется на компьютерах-клиентах.

В архитектуре "клиент-сервер" сервер базы данных не только обеспечивает доступ к общим данным, но и берет на себя всю обработку этих данных. Клиент посылает на сервер запросы на чтение или изменение данных, которые формулируются на языке SQL. Сервер сам выполняет все необходимые изменения или выборки, контролируя при этом целостность и согласованность данных, и результаты в виде набора записей или кода возврата посылает на компьютер клиента.

Недостатки архитектуры с файловым сервером очевидны и вытекают главным образом из того, что данные хранятся в одном месте, а обрабатываются в другом. Это означает, что их нужно передавать по сети, что приводит к очень высоким нагрузкам на сеть и, вследствие этого, резкому снижению производительности приложения при увеличении числа одновременно работающих клиентов. Вторым важным недостатком такой архитектуры является

децентрализованное решение проблем целостности и согласованности данных и одновременного доступа к данным. Такое решение снижает надежность приложения.

Архитектура "клиент-сервер" позволяет устранить все указанные недостатки. Кроме того, она позволяет оптимальным образом распределить вычислительную нагрузку между клиентом и сервером, что также влияет на многие характеристики системы: стоимость, производительность, поддержку.

7. Особенности распределенных баз данных. Распределенные СУБД. Локальные и глобальные приложения.

Распределенная обработка данных (ГОСТ 33707) - обработка данных, при которой выполнение операций распределено по узлам вычислительной сети

Распределенная БД (РБД) - совокупность логически взаимосвязанных разделяемых данных, физически распределенных в компьютерной сети

Распределенная СУБД - программный комплекс, предназначенный для управления РБД и обеспечивающий прозрачный доступ пользователей к распределенной информации

Пользователи взаимодействуют с базой данных через локальные и глобальные приложения

Пользователи взаимодействуют с распределенной базой данных через приложения. **Локальные приложения** не требуют доступа к данным на других узлах, **глобальные приложения** требуют подобного доступа. В распределенной СУБД должно существовать хотя бы одно глобальное приложение, поэтому любая РСУБД должна иметь следующие особенности [2].

- ♦ Набор логически связанных разделяемых данных.
- ♦ Сохраняемые данные разбиты на некоторое количество фрагментов.
- ♦ Между фрагментами может быть организована репликация данных.
- ♦ Фрагменты и их реплики распределены по различным узлам.
- ♦ Узлы связаны между собой сетевыми соединениями.
- ♦ Работа с данными на каждом узле управляется СУБД.
- ♦ СУБД на каждом узле способны поддерживать автономную работу локальных приложений.

Из определения РСУБД следует, что для конечного пользователя распределенность системы должна быть совершенно **прозрачна** (невидима). Другими словами, для пользователя распределенная система должна выглядеть так же, как нераспределенная система. В некоторых случаях это требование называют **фундаментальным принципом** построения распределенных СУБД [1].

8. Распределенные базы данных. Подходы к разработке распределенных БД.

Распределенная обработка данных (ГОСТ 33707) - обработка данных, при которой выполнение операций распределено по узлам вычислительной сети

Распределенная БД (РБД) - совокупность логически взаимосвязанных разделяемых данных, физически распределенных в компьютерной сети

Распределенная СУБД - программный комплекс, предназначенный для управления РБД и обеспечивающий прозрачный доступ пользователей к распределенной информации

9. Распределенные базы данных. Проблемы создания распределенных БД.

Распределенная обработка данных (ГОСТ 33707) - обработка данных, при которой выполнение операций распределено по узлам вычислительной сети

Распределенная БД (РБД) - совокупность логически взаимосвязанных разделяемых данных, физически распределенных в компьютерной сети

Распределенная СУБД - программный комплекс, предназначенный для управления РБД и обеспечивающий прозрачный доступ пользователей к распределенной информации

Основные проблемы создания распределенной БД:

- фрагментация данных и распределение по компьютерам
- составление глобального каталога, содержащего информацию о каждом фрагменте БД и его местоположении в сети
- организация обработки запросов:
 - синхронизация нескольких запросов к одним и тем же данным
 - исключение аномалий удаления и обновления одних и тех же данных
 - оптимизация последовательности шагов при обработке запроса и т.д.

10. Классификация баз данных по характеру организации данных, по типу используемых моделей.

По (типу) характеру организации данных: (только символьные базы данных)

- **Структурированные** - предварительно описана (определена) модель структуры:
 - Теоретико-графовые:
 - Иерархические
 - Сетевые
 - Теоретико-множественные:
 - Реляционные
 - Пост реляционные
 - Многомерные
 - Объектно-ориентированные
- **Неструктурированные** – БД, в которых информация представлена в виде семантических сетей
- **Частично-структурированные** – содержат информацию в виде текста, возможно, гипертекста

Иерархическая (древовидная) модель БД (является одной из самых первых) - наиболее естественным образом отражает множественные связи между объектами реального мира, когда один объект выступает в качестве главного (родительского), с которым связано большое количество подчиненных объектов.

Иерархическая модель БД – упорядоченная совокупность экземпляров деревьев, каждое из которых содержит экземпляры записей

Сетевая база данных – это модель данных, где несколько записей или файлов могут быть связаны с несколькими владельцами файлов и наоборот.

Реляционной считается такая БД в которой все данные представлены в виде прямоугольных таблиц значений данных и все операции над БД сводятся к манипуляциям с таблицами

Постреляционная модель является расширением реляционной модели. Она снимает ограничение неделимости данных, допуская многозначные поля, значения которых состоят из подзначений, и набор значений воспринимается как самостоятельная таблица, встроенная в главную таблицу.

Многомерной называют базу данных, в которой данные организованы не в виде множества связанных двумерных таблиц, как в реляционных структурах, а в виде упорядоченных многомерных массивов

11. Иерархические структуры данных. Основные информационные единицы. Пример.

Ограничения иерархической модели.

Иерархические модели наиболее естественным образом отражают множественные связи между объектами реального мира, когда один объект выступает в роли главного с которым связано большое количество подчиненных объектов

IMS (Information Management System) - результат работы фирм IBM и Rockwell по созданию СУБД для поддержки лунного проекта Аполлон

Информационные единицы:

- поле (минимальное, неделимое, доступное пользователю)
- запись
- групповое отношение
- база данных

Тип записи - именованное совокупность полей с указанием типов

СТОЛ (ЦВЕТ символьный, ДЛИНА числовой, ШИРИНА числовой, ВЫСОТА числовой, МАТЕРИАЛ символьный)

Экземпляр записи - набор конкретных значений в последовательности и соответствующей определению типа этой записи

СТОЛ (Коричневый, 1.30, 0.9, 0.75, дерево)

Тип записи определяет все множество одинаковых объектов, а экземпляр указывает на конкретный объект из этого множества

Иерархическая модель БД - упорядоченная совокупность экземпляров деревьев, каждое из которых содержит экземпляры записей

Ограничения иерархической модели БД:

- существует только одна корневая запись, которая не связана ни с какой исходной записью
- каждая логически исходная запись может быть связана с произвольным числом логически подчиненных записей
- каждая логически подчиненная запись может быть связана только с одной логически исходной записью



12. Сетевые структуры данных. Основные понятия сетевой модели. Пример.

Сетевые модели позволяют строить структуры данных более общего вида, нежели иерархические

CODASYL (Conference on Data Systems Language), 1971 г.

Набор:

- один и тот же тип записи может участвовать в нескольких наборах
- для любых двух типов записей может быть задано любое количество наборов, которые их связывают

Между типами организация и сотрудник связь типа многих ко многим. Тип записи счет участвует в двух наборах, с типом записи банк и с типом записи сотрудник. Каждый экземпляр набора состоит из одного экземпляра записи владельца и упорядоченного набора записей членов набора

Сетевая база данных - совокупность взаимосвязанных наборов

Сетевая база данных – это модель данных, где несколько записей или файлов могут быть связаны с несколькими владельцами файлов и наоборот.

Каждый экземпляр набора – список записей-членов набора, поставленных в соответствие записи-владельцу набора

Записи-члены набора могут быть упорядочены:

- ✓ произвольным способом
- ✓ хронологическим в порядке их поступления
- ✓ обратнхронологическим
- ✓ упорядоченным по некоторому ключу

Сетевая модель БД – совокупность взаимосвязанных наборов

13. Реляционная модель данных. Основные элементы и формы их представления.

Реляционная модель данных (РМД) разработана на основе математической теории отношений и опирается на систему базовых понятий

Объекты реального мира - информация, которая хранится в базе данных (сущность)

Реляционный называется такая база данных, в которой все данные представлены в виде прямоугольной таблицы

Элемент РМД	Форма представления
Отношение	Таблица
Схема отношения	Заголовок таблицы
Кортеж	Строка таблицы
Атрибут	Заголовок столбца таблицы
Значение атрибута	Значение поля таблицы

Домен - множество всех возможных значений атрибута объекта

Определяется заданием базового типа и произвольного логического выражения применяемого к элементу типа данных

Домен НАЗВАНИЕ:

- базовый тип - строка символов
- логическое выражение: строки начинаются с букв, за исключением букв Ь, Ъ, Ы

A_i – атрибут, D_i – его домен
 $A_i = \text{Материал}$
 $D_i = \text{dom}(A_i) = \{\text{«Сталь»}, \text{«Олово»}, \text{«Цинк»}, \dots\}$
Схема отношения – именованное множество пар
{имя атрибута, имя домена}
ДЕТАЛЬ (Шифр, Название, Вес, Материал)
Схема БД – набор именованных схем отношений
Для схемы отношения $R(A_1, A_2, \dots, A_n)$ k -ый кортеж:
 $\langle a_{1k}, a_{2k}, \dots, a_{nk} \rangle$, a_{ik} – значение i -ого атрибута в k -ом кортеже

Схема отношения - именованное множество пар {имя атрибута, имя домена}

Схема БД - набор именованных схем отношений

Степень отношения - это число его атрибутов

Мощность отношения - это число его кортежей

Отношение r со схемой отношения $R(A_1, A_2, \dots, A_n)$ – это множество кортежей, соответствующих одной схеме отношения

14. Реляционная модель данных. Основные свойства отношений. Возможные ключи. Пример.

Реляционная модель данных (РМД) разработана на основе математической теории отношений и опирается на систему базовых понятий

Объекты реального мира - информация, которая хранится в базе данных (сущность)

Реляционный называется такая база данных, в которой все данные представлены в виде прямоугольной таблицы

Свойства отношений

1. Отсутствие кортежей-дубликатов (т.к. отношения это множество кортежей, а во множестве элементы не повторяются);
2. Отсутствие упорядоченных кортежей;
3. Отсутствие упорядоченности атрибутов (схема отношения есть множество пар состоящих из имени его атрибута и домена то в этом множестве атрибуты не упорядочиваются);

Первичный ключ - набор атрибутов, значения которых однозначно определяют каждый кортеж отношения

Отношение $R(A_1, A_2, \dots, A_n)$

$K = (A_1, A_2, \dots, A_m)$ – возможный ключ \Leftrightarrow

А) Уникальность

Б) Минимальность

r	A	B	C
	a1		c1
	a1		c2
	a1		c3
	a2		c1
	a2		c2

A – нет

B – нет

C – нет

AB – да

BC – нет

AC – да

ABC – да

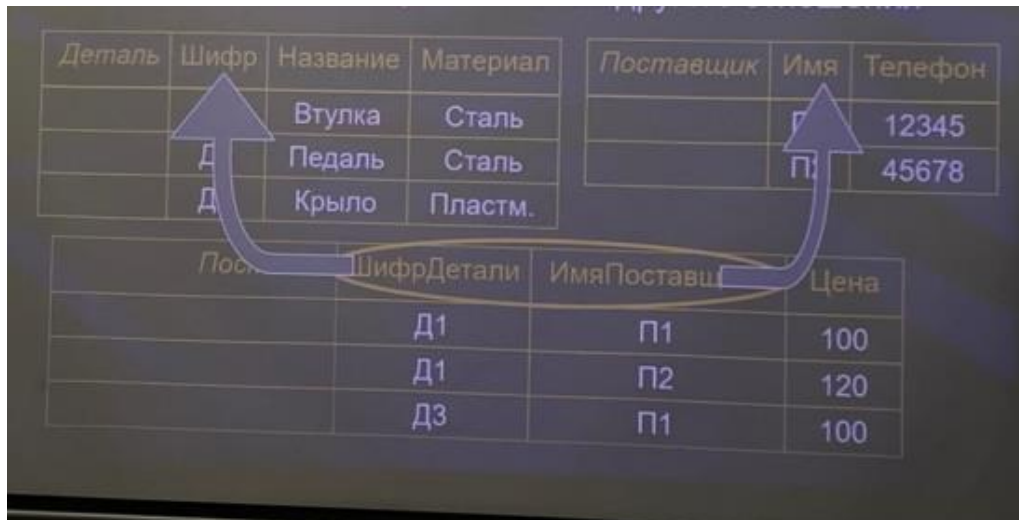
Деталь	Шифр	Название	Вес	Материал
	Д1	Втулка	0.8	Сталь
	Д2	Педадь	1.0	Сталь
	Д3	Крыло	0.8	Пластмасса
	Д4	Крыло	0.9	Пластмасса

15. Связывание таблиц в реляционной модели данных. Суть связывания. Внешние ключи. Основные виды связи. Пример.

В иерархической и сетевой моделях объекты представляются с помощью полей и записей, а связи между объектами с помощью групповых отношений (иерархическая модель) или наборами (сетевая модель). В реляционной модели и объекты, и связи представляются с помощью отношений.

Связь между отношениями устанавливается с помощью внешних ключей.

Внешний ключ - атрибут (или множество атрибутов) отношения, являющийся ключом другого отношения

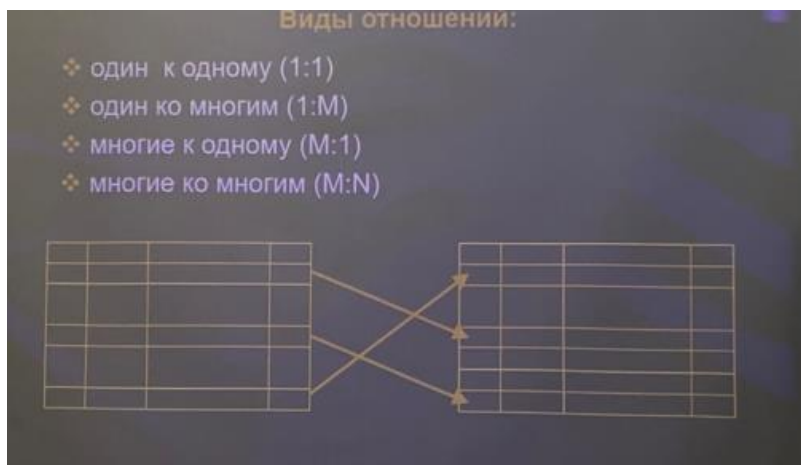


При связывании таблиц обычно выделяют основную таблицу и подчиненную

Виды отношений:

- один к одному (1:1) образуется в том случае, когда связующие поля обеих таблиц являются ключевыми
- один ко многим (1:M) одной связи основной таблицы соответствует несколько записей подчиненной таблицы
- многие к одному (M:1) нескольким связям основной таблицы соответствует одна запись подчиненной таблицы
- многие ко многим (M:N) нескольким записям в основной таблице соответствует несколько записей подчиненной таблицы

Возможны случаи, когда удобней иметь не одну, а две или более таблиц, связанных отношением 1:1: а) необходимость ускорить обработку информации; б) повысить удобство работы нескольких пользователей с этой информацией; в) обеспечить степень защиты информации



16. Понятие целостности в реляционной модели данных.

Целостность базы данных — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением целостности.

Требования целостности:

- требование целостности сущностей (т.к. любому объекту реального мира в РМД соответствует кортеж отношений, то любой кортеж отношения должен быть отличим от другого кортежа реального мира)
- требование целостности по ссылкам

Контроль целостности связей - анализ содержимого таблиц на соблюдение следующих правил:

- каждой записи основной таблицы соответствует нуль или более записей подчиненной таблицы
- в подчиненной таблице нет записей, которые не имеют родительских записей в основной таблице
- каждая запись подчиненной таблицы имеет только одну родительскую запись основной таблицы

При изменении записей в обеих таблицах не должны нарушаться ни одно из этих правил

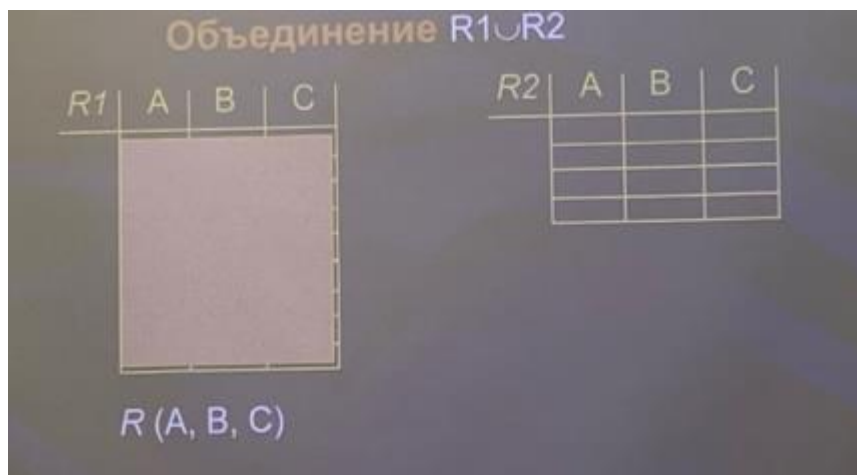
17. Теоретико-множественные операции реляционной алгебры. Примеры.

Основная идея реляционной алгебры, т.к. отношения являются множественными, то средства манипулирования ими могут базироваться на традиционных операциях над множествами, дополнив их специальными операциями, характерными для базы данных

Операция над одним отношением - унарная. Над двумя отношениями - бинарная. При выполнении бинарных операций иногда требуется чтобы оба отношения были совместимы по структуре

Теоретико-множественные операции:

- объединение - Объединением двух совместимых отношений одинаковой размерности, является отношение содержащее все кортежи исходных отношений, за исключением повторений



- пересечение - Пересечением двух совместимых отношений одинаковой размерности, является отношение, содержащее кортежи одновременно принадлежащие к обоим отношениям

Пересечение $R1 \cap R2$

R1	A	B	C

R2	A	B	C

Вычитание $R1 - R2$

- разность - Вычитанием двух совместимых отношений одинаковой размерности, является отношение содержащее кортежи принадлежащие R1, но не принадлежащие R2

Вычитание $R1 - R2$

R1	A	B	C

R2	A	B	C

- произведение - Произведение $R1 \times R2$ отношения R1 размерности $n1$ и отношения R2 размерности $n2$ - отношение R степени $(n1+n2)$, содержащее кортежи, в которых первые $n1$ элементов принадлежат множеству R1, а последние $n2$ элементов - множеству R2

элементов – множеству R2

R1	Имя
	П1
	П2

R2	Шифр
	Д1
	Д2
	Д3

$R1 \times R2$	Имя	Шифр
	П1	Д1
	П1	Д2
	П1	Д3
	П2	Д1
	П2	Д2
	П2	Д3

18. Специальные реляционные операции реляционной алгебры. Примеры.

Специальные реляционные операции:

- ограничение (селекция, выборка)

1. Выборка $\sigma_f(R)$ отношения R по формуле f

Деталь	Шифр	Название	Вес	Материал
	Д1	Втулка	0.8	Сталь
	Д2	Педадь	1.0	Сталь
	Д3	Крыло	0.8	Пластмасса
	Д4	Крыло	0.9	Пластмасса

$\sigma_{\text{Вес} > 0.8}(\text{Деталь})$	Шифр	Название	Вес	Материал
	Д2	Педадь	1.0	Сталь
	Д4	Крыло	0.9	Пластмасса

- проекция

2. Проекция $\Pi_{A_1, A_2, \dots, A_n}(R)$

Деталь	Шифр	Название	Вес	Материал
	Д1	Втулка	0.8	Сталь
	Д2	Педадь	1.0	Сталь
	Д3	Крыло	0.8	Пластмасса
	Д4	Крыло	0.9	Пластмасса

$\Pi_{\text{Название, Материал}}(\text{Деталь})$	Название	Материал
	Втулка	Сталь
	Педадь	Сталь
	Крыло	Пластмасса

- соединение - Соединением двух отношений R_1, R_2 по формуле f , является отношение содержащее кортежи декартового произведения R_1, R_2 , удовлетворяющие формуле f : а) тета-соединение (если формула f имеет произвольный вид); б) эквисоединение (если в формуле f только оператор равенства); в) естественное соединение (по одноименным атрибутам)

4. Соединение $R_1 \bowtie R_2$

- Θ -соединение (тета-соединение)
- эквисоединение
- естественное соединение $R_1 \bowtie R_2$

Деталь	Шифр	Цвет
	Д1	Красный
	Д2	Красный
	Д2	Синий

Поставщик	Имя	Шифр Детали
	П1	Д1
	П2	Д2

Поставщик \bowtie Деталь	Имя	Шифр	Цвет
	П1	Д1	Красный
	П2	Д2	Красный
	П2	Д2	Синий

- деление - Результат деления $R1 \div R2$ отношения $R1(A,B)$ на отношение $R2(B)$ - отношение с заголовком А и телом, состоящим из кортежей r таких, что в отношении $R1$ имеются кортежи (r, s) , причем множество значений s включает все значения атрибута В отношения $R2$

R1	Фамилия	Курс
	Иванов	Алгебра
	Иванов	Физика
	Петров	Физика
	Петров	Черчение
	Петров	Химия
	Сидоров	Физика
	Сидоров	Химия
	Алексеев	Физика

R2	Курс
	Физика
	Химия

$R1 \div R2$	Фамилия
	Петров
	Сидоров

19. Функциональные зависимости. Нормализация реляционных баз данных. Пример.

Одним из преимуществ РМД является наличие формального механизма оценки качества логической структуры базы данных. Из анализа видно, что в нем есть большое количество избыточных (повторяющихся) данных. Избыточность этих данных может вызвать нарушение целостности данных. Аномалии модификации:

- аномалия удаления (проявляется в том, что удаляю факты относящиеся к одному объекту, мы непроизвольно удаляем факты другого объекта)
- аномалия добавления (нельзя поместить информацию об одном объекте, не указав дополнительную информацию о другом объекте)
- аномалия обновления (меняя информацию об объекте в одном месте, необходимо изменить ее же множество раз)

Нормализация - это разбиение (декомпозиция) исходного отношения на два и более, обладающих лучшими свойствами при удалении, добавлении и обновлении

Цель нормализации сводится к получению такой базы данных, в которой каждый факт появляется только в одном месте

Процесс нормализации представляет собой переход между схемами отношений, при чем каждая следующая нормальная форма обладает свойствами лучшими нежели предыдущими, но при этом свойства предыдущих не меняются

Отношения находится в первой нормальной форме (1НФ), если все его атрибуты являются простыми

Отношение находится в первой нормальной форме (1НФ), если все его атрибуты являются простыми

№сотр	Фамилия	Должность	Проект
1111	Алексеев А.А.	Программист I	Альфа, Омега
2222	Иванов И.И.	Вед. программист	Альфа, Омега, Гамма, Сигма
3333	Сидоров С.С.	Программист II	Альфа

Функциональные зависимости

Атрибут В функционально зависит от атрибута А (А и В могут быть составными), если каждому значению А соответствует только одно значение В Функциональная зависимость называется полной, если атрибут В не зависит функционально от подмножества атрибута А

Два или более атрибутов взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от другого

$A \rightarrow B$

Фамилия	Должность	Зарплата	Проект	Задание
Алексеев А.А.	Программист I	100	Альфа	Задание1
Алексеев А.А.	Программист I	100	Омега	Задание2
Иванов И.И.	Вед. программист	150	Альфа	Задание3
Сидоров С.С.	Программист II	150	Альфа	Задание1

Фамилия \rightarrow Зарплата Фамилия, Проект \rightarrow Задание
Фамилия \rightarrow Должность Фамилия, Проект \rightarrow Должность
Должность \rightarrow Зарплата Фамилия, Проект \rightarrow Зарплата
Фамилия, Проект \rightarrow Задание, Должность, Зарплата

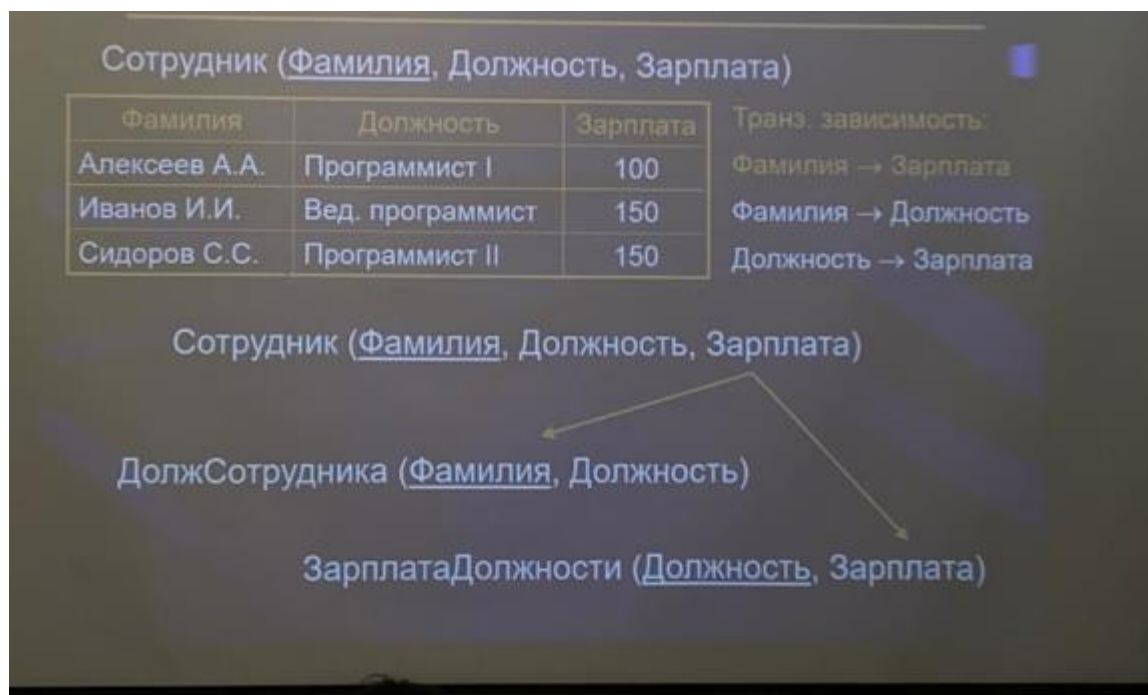
Отношение находится во второй нормальной форме (2НФ), если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа

Отношение находится во второй нормальной форме (2НФ), если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа

Фамилия	Должность	Зарплата	Проект	Задание
Алексеев А.А.	Программист I	100	Альфа	Задание1
Алексеев А.А.	Программист I	100	Омега	Задание2
Иванов И.И.	Вед. программист	150	Альфа	Задание3
Сидоров С.С.	Программист II	150	Альфа	Задание1

Если первичный ключ простой, то он автоматически будет во 2НФ

Отношение находится в третьей нормальной форме (3НФ), если оно находится в 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключа



20. Концептуальное проектирование баз данных. Анализ информационных задач и пользователей системы. Составление диаграммы вариантов использования. Пример. Классификация данных по частоте их изменения.

концептуальное проектирование - цель концептуального проектирования: определение содержания базы данных

Концептуальное проектирование заключается в формализованном описании предметной области, которое должно быть таким, чтобы с одной стороны можно было проанализировать корректность, а с другой стороны это описание не должно быть привязано к конкретной СУБД.

Любая БД создается для решения определенных прикладных задач, в простейшем случае для накопления и выдачи данных, ну а в более сложных случаях, для решения задач, которое используют данные из БД в качестве исходных данных.

При анализе задач требуется ответить на 2 вопроса: а) для решения каких задач создается БД? б) какие данные нужны для решения этих задач? Анализ хранимых объектов: а) данные о ком или о чем должны храниться в БД? б) какими атрибутами они характеризуются?

Анализ задач

UML (англ. Unified Modeling Language - унифицированный язык моделирования) Диаграмма вариантов использования, главное назначение ДВИ заключается в формализации функциональных требований к системе с помощью действующих лиц (потенциальный пользователь системы), второй графический элемент - вариант использования (задача, с которой действующее лицо обращается к разрабатываемому программному продукту)

Рекомендуемые ограничения: количество действующих лиц не более 20, количество вариантов использования не более 50.

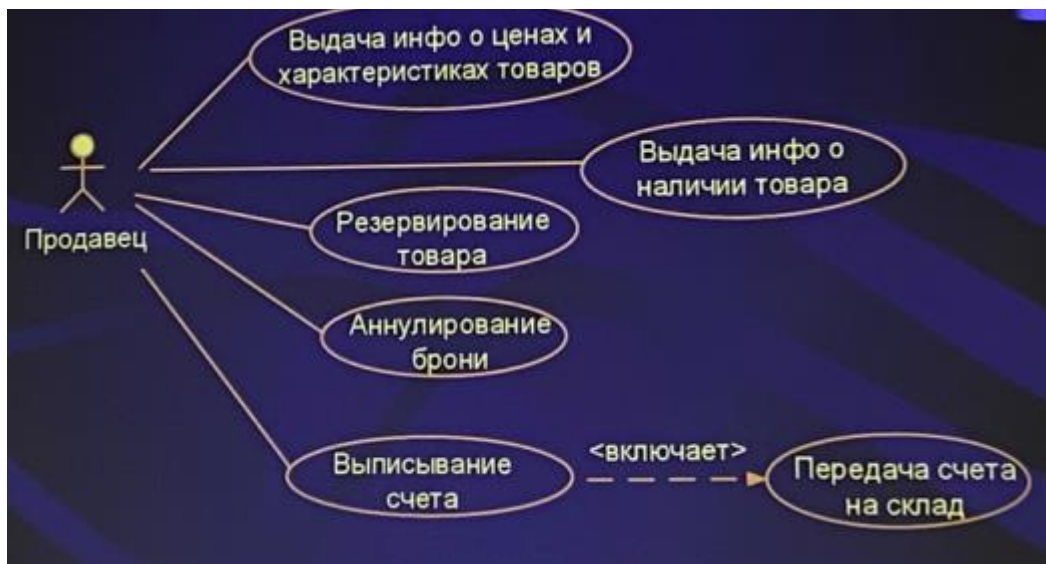
Создавая диаграмму вариантов использования, стоит сосредоточиться на том, какие задачи будут решать с помощью БД, а не на том как это будет сделано

Составление ДВИ позволяет:

- Экспертам предметной области описать взгляд на систему извне с такой степенью детализации, что разработчики сумеют сконструировать ее внутреннее представление
- Варианты использования позволяют разработчикам понять назначение разрабатываемой БД
- Варианты использования являются основой тестирования элементов БД на всем протяжении его жизненного цикла

Последовательность составления ДВИ:

- Выделить действующие лица
- Для каждого действующего лица выделить его варианты использования. После этого составить общий список вариантов использования
- Связать между собой действующие лица и варианты использования
- Проанализировать в первом приближении наличие общих подзадач у разных вариантов использования, при наличии выделить их и установить отношения



Классификация данных по частоте их изменений:

- условно-постоянные данные
- данные, которые оперативно обновляются при каждом решении задач

Для проведения этой классификации необходимо проанализировать выделенные задачи: какие исходные данные требуются, какие из них каждый раз надо вводить заново, а какие имеют условно постоянный характер и меняются редко. Аналогичным образом требуется проанализировать данные, которые являются результатом решения задач: могут носить разовый характер, либо они могут быть сохранены в базе данных

21. Концептуальное проектирование баз данных. Анализ хранимых объектов.

концептуальное проектирование - цель концептуального проектирования: определение содержания базы данных

Концептуальное проектирование заключается в формализованном описании предметной области, которое должно быть таким, чтобы с одной стороны можно было проанализировать корректность, а с другой стороны это описание не должно быть привязано к конкретной СУБД.

Анализ хранимых объектов

Цель: в первом приближении определить данные о каких объектах должны храниться в БД и каким перечнем атрибутов они характеризуются. Первоначальный перечень атрибутов можно получить в ходе анализа задач, выделенных на предидущем этапе, но кроме этого можно создать задел на будущее (включить в БД атрибуты, значения которых потребуются при расширении состава решаемых задач)

Рекомендации по определению состава атрибутов:

- Имя атрибута должно быть существительным
- Типы атрибутов пока концептуальны и не должны быть связаны со средой реализации
- Среди атрибутов не стоит иметь атрибуты, значения которых можно легко вычислить, зная значения других
- Наличие составных атрибутов нецелесообразно, если составные части могут использоваться независимо друг от друга
- При наличии атрибутов, принимающих значение из фиксированного множества значений, целесообразно предусмотреть кодификаторы

22. Семантическое моделирование данных, ER-диаграммы. Пример.

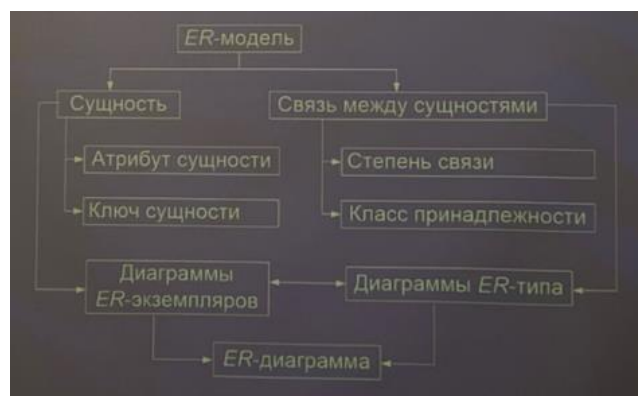
Семантическое моделирование данных направлено на отображение семантики (смысла) предметной области на модель БД

ER-модели (Entity - сущность, Relation - связь)

ER-модель представляет собой формальную конструкцию, которая не предписывает никаких графических средств для ее визуализации

Наиболее известным представителем класса семантических моделей предметной области является модель «сущность-связь» или ER-модель.

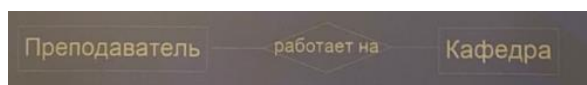
С её помощью можно выделить ключевые сущности и обозначить связи, которые могут устанавливаться между этими сущностями. Во время проектирования баз данных происходит преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных (реляционной, объектной, сетевой или др.)



Сущность - это объект, информация о котором должна сохраняться в БД (сущ.)

Связь 2 или более сущностей предполагает зависимости между атрибутами этих сущностей, название связи - глагол

- диаграммы ER-экземпляров
- диаграммы ER-типа (ER-диаграмма)



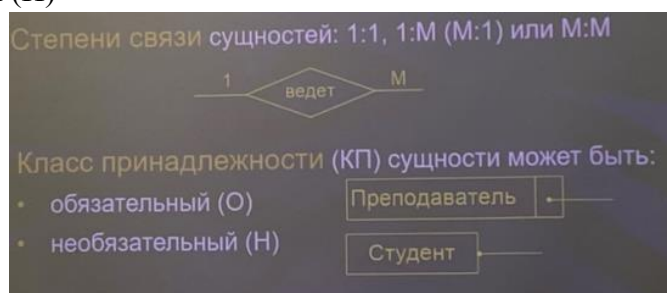
На начальном этапе проектирования выделяют только те атрибуты которые составляют ключ сущностей, поэтому под каждым прямоугольником указывается ключ сущности, выделяемый подчёркиванием, многоточие после ключа означает, что возможны и другие атрибуты, но ни один из них не будет частью ключа

Степень связи сущностей: 1:1, 1:M (M:1) или (M:M)

Между любой парой сущностей может быть задано произвольное количество связи, с разными смысловыми нагрузками

Класс принадлежности (КП) сущности может быть:

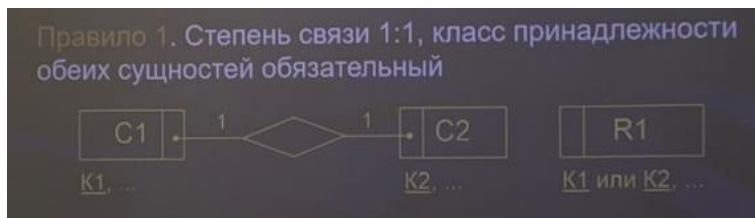
- обязательный (О), когда в связи должен участвовать каждый экземпляр сущности
- необязательный (Н)



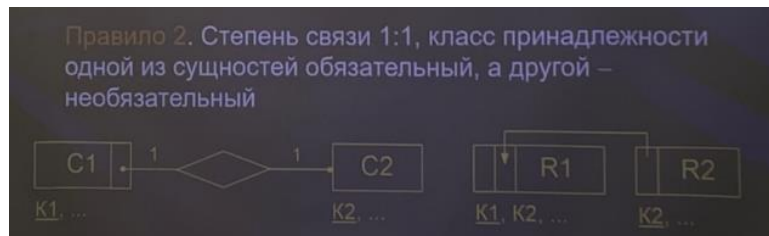
23. Правила формирования отношений на основе диаграмм ER-типа. Пример.

Правила формирования отношений

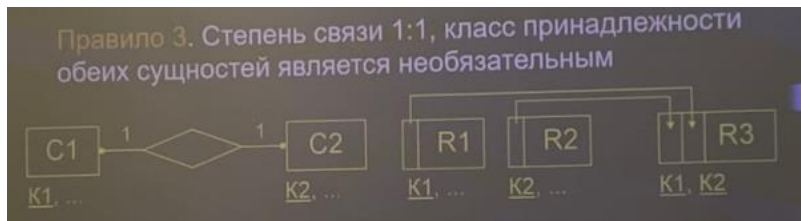
Правило 1. Если степень связи 1:1 и класс принадлежности обеих сущностей обязательный, то требуется всего одно отношение, объединяющее эти сущности, при этом ключом отношений может стать ключ любой сущности



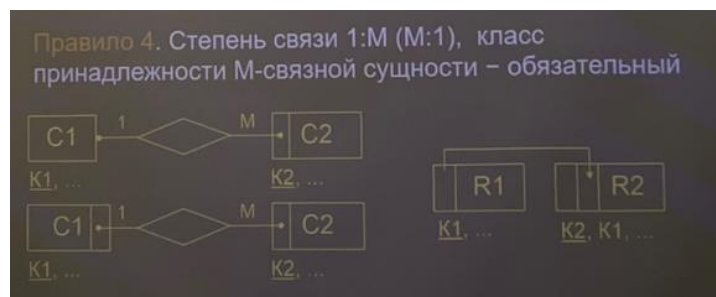
Правило 2. Если степень связи 1:1, класс принадлежности одной сущности обязательный, а другой необязательный, то для каждой из сущностей формируется по одному отношению, причём ключ сущности с необязательным классом принадлежности добавляется в качестве атрибута в другое отношение



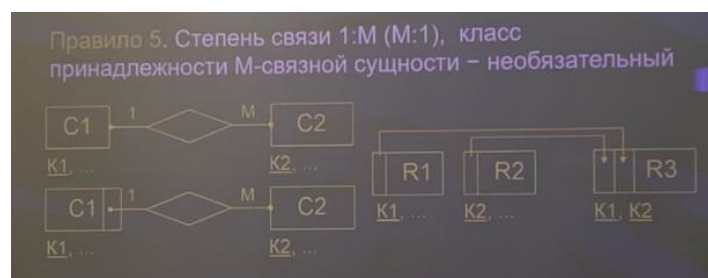
Правило 3. Если степень связи 1:1, класс принадлежности обеих сущностей является необязательным, то формируются 3 отношения, по одному для каждой сущности со своими ключами, и третье выражающее связь и содержащее ключи из обеих сущностей



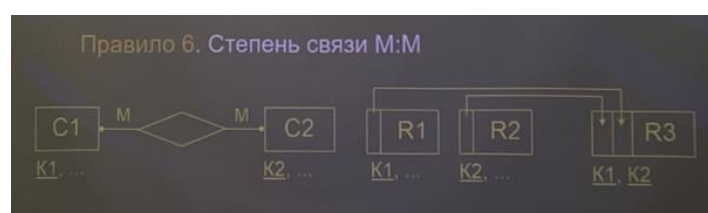
Правило 4. Если степень связи 1:M (M:1), класс принадлежности многосвязной сущности обязательный, то независимо от класс принадлежности другой сущности достаточно сформировать 2 отношения, по одному для каждой сущности со своими ключами, при этом ключ односвязной сущности добавляется в качестве атрибута в отношение для многосвязной сущности



Правило 5. Если степень связи 1:M (M:1), класс принадлежности многосвязной сущности необязательный, то необходимо формирование 3 отношений, по одному для каждой сущности со своими ключами, и третье выражающее связь и содержащее ключи из обеих сущностей



Правило 6. Если степень связи M:M, то независимо от класса принадлежности сущностей требуется формирование 3 отношений, по одному для каждой сущности со своими ключами, и третье выражающее связь и содержащее ключи из обеих сущностей



24. Основные этапы логического проектирования. Исключение особенностей, несовместимых с реляционной моделью. Примеры.

Логическое проектирование БД

Цель: определение состава и структуры таблиц БД на основе результатов концептуального проектирования и проверка полученной модели с помощью методов нормализации

Стадии логического проектирования:

Этап 1. Исключение особенностей, несовместимых с реляционной моделью

Этап 2. Формирование отношений на основе логической модели данных

Этап 3. Проверка отношений с использованием средств нормализации

Этап 4. Определение ограничений целостности

- Удаление рекурсивных связей - это особый тип связи в которой сущность соединена сама с собой (необязательно)



- Удаление сложных связей - в которой участвуют 3 или более сущностей
- Удаление многозначных атрибутов

№сотр	Фамилия	Должность	Проект
1111	Алексеев А.А.	Программист I	Альфа, Омега
2222	Иванов И.И.	Вед. программист	Альфа, Омега, Гамма, Сигма
3333	Сидоров С.С.	Программист II	Альфа

Сложные типы связи - необходимо выполнить декомпозицию сложной связи путем введения промежуточной сущности, сложная связь заменяется необходимым количеством двухсторонней связи

Для каждого многозначного атрибута создается отдельное отношение и в это новое отношение передается первичный ключ в качестве внешнего ключа, тогда первичный ключ нового отношения включает себя 2 атрибута - название многозначного атрибута, и название ключа

25. Основные этапы логического проектирования. Формирование отношений на основе логической модели данных.

Этап 1. Исключение особенностей, несовместимых с реляционной моделью

Этап 2. Формирование отношений на основе логической модели данных

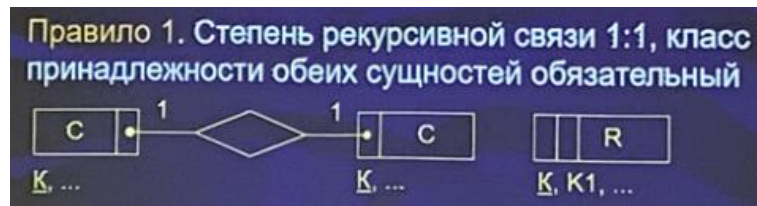
Этап 3. Проверка отношений с помощью правил нормализации

Этап 4. Определение требований поддержки целостности данных

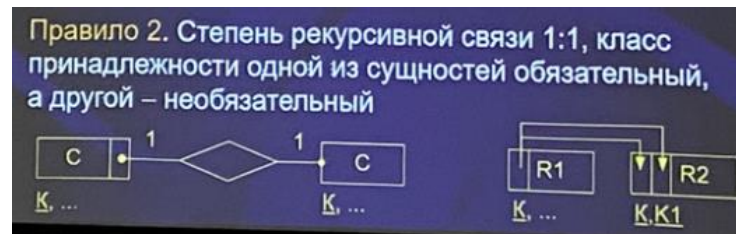
На данном этапе определяются наборы отношений необходимых для представления сущностей, связей и атрибутов. Связь между двумя сущностями реализуется с использованием механизма - первичный ключ и внешний ключ. Родительская сущность передает свой

первичный ключ в дочернюю сущность, в качестве внешнего ключа. Основные способы формирования отношений (6 правил) смотри лекцию по ER-моделям

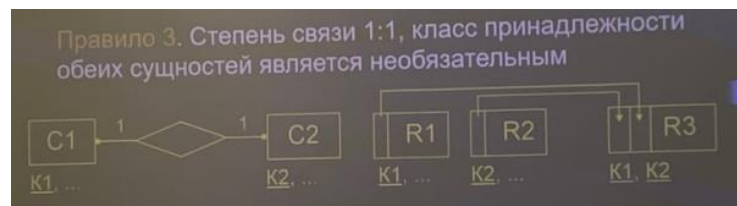
Правило 1. Степень рекурсивной связи 1:1, класс принадлежности обеих сущностей обязательный



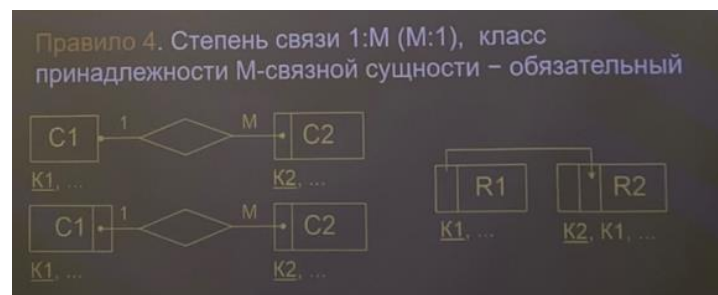
Правило 2. Степень рекурсивной связи 1:1, класс принадлежности одной из сущностей обязательный, а другой - необязательный



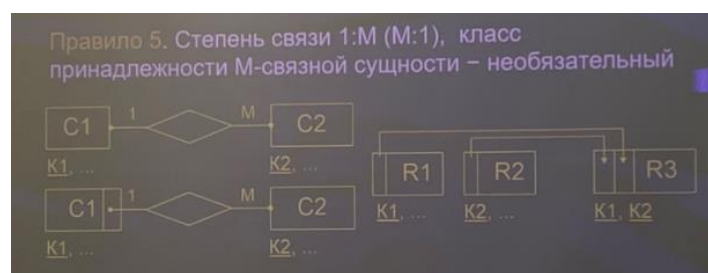
Правило 3. Степень рекурсивной связи 1:1, класс принадлежности обеих сущностей - необязательный



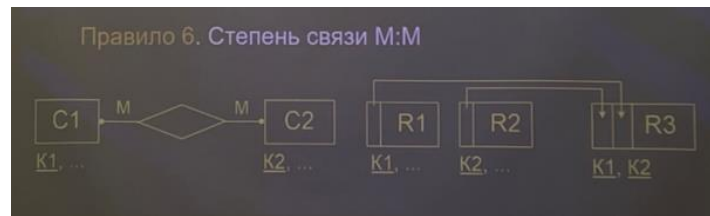
Правило 4. Степень рекурсивной связи 1:M (M:1), класс принадлежности M-связной сущности - необязательный



Правило 5. Если степень связи 1:M (M:1), класс принадлежности многосвязной сущности необязательный, то необходимо формирование 3 отношений, по одному для каждой сущности со своими ключами, и третье выражающее связь и содержащее ключи из обеих сущностей



Правило 6. . Степень рекурсивной связи M:M



26. Основные этапы логического проектирования. Определение требований поддержки целостности данных.

- Этап 1. Исключение особенностей, несовместимых с реляционной моделью
- Этап 2. Формирование отношений на основе логической модели данных
- Этап 3. Проверка отношений с помощью правил нормализации
- **Этап 4. Определение требований поддержки целостности данных**

Определите требований поддержки целостности данных. Вводится с целью предотвращения появления в БД противоречивых данных 5 типов ограничений целостности данных:

- обязательные данные (атрибуты не могут иметь другого значения)
- ограничения для доменов атрибутов
- целостность сущностей (не может содержать пустого значения)
- ссылочная целостность
- ограничения предметной области

Ссылочная целостность означает, что если внешний ключ содержит какое-то значение, то оно обязательно должно присутствовать в первичном ключе родительского значения Организация поддержки ссылочной целостности

Случай 1. Вставка новой строки в дочернее отношение

Случай 2. Удаление строки из дочернего отношения

Случай 3. Обновление внешнего ключа в строке дочернего отношения

Случай 4. Вставка строки в родительское отношение

Случай 5. Удаление строки из родительского отношения

- NO ACTION
- CASCADE
- SET NULL
- SET DEFAULT
- NO CHECK

Случай 6. Обновление первичного ключа в строке родительского отношения

Добавление или изменение сущностей может регламентироваться правилами принятыми в организации

Ограничения предметной области

Необходимо зафиксировать сведения о том всех установленных ограничениях в словесной форме

27. Этапы физического проектирования реляционной БД.

Исходной информацией для этапа физического проектирования БД является логическая модель данных, состоящая из ER-диаграммы или диаграммы отношений, а также из реляционной схемы

Обратная связь между физическим проектированием, логическим проектированием и разработкой приложений

Физические возможности разных СУБД сильно различаются друг от друга

Физическое проектирование тесно связано с особенностями конкретной СУБД

- физическое проектирование
 - цель физического проектирования: реализация спроектированной БД на конкретной СУБД и обеспечение выполнения всех требований к ней

Этап 1) Проектирование основных отношений

Определение каждого отношения включает:

- имя отношения список простых атрибутов
- определение первичного ключа и альтернативных (AK) и внешних (FK) ключей
- список производных атрибутов и описание способов их вычисления
- определение требований ссылочной целостности для любых внешних ключей

Для каждого атрибута следует указать:

- определение его домена, включающее тип данных, размерность внутреннего представления атрибута и любые требуемые ограничения на допустимые значения
- значение атрибута, принимаемое по умолчанию
- допустимость значения NULL

Этап 2) Разработка способов получения производных атрибутов

На этапе физического проектирования необходимо определить, должен ли производный атрибут храниться в БД, либо он будет вычисляться каждый раз когда в нем возникает необходимость

Два варианта затрат:

- на хранение производных данных и поддержание их согласованности с реальными данными
- на вычисление производных данных по мере необходимости

Этап 3) Реализация ограничений предметной области

Способы реализации ограничений:

- с использованием возможностей стандарта языка SQL
- с применением триггеров
- непосредственно в самом приложении

28. Показатели, использующиеся для оценки эффективности физического проекта базы данных.

- физическое проектирование
 - цель физического проектирования: реализация спроектированной БД на конкретной СУБД и обеспечение выполнения всех требований к ней

Показатели оцени эффективности хранения данных:

- производительность выполнения транзакций

Транзакция - последовательность операций над БД, рассматриваемая как единое целое

- время ответа

- дисковая память

29. Стандартный язык баз данных SQL. Типы данных SQL. Особенности типов данных в разных СУБД.

Язык баз данных SQL

SQL (Structured Query Language) - структурированный язык запросов

Язык sql предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными из таблиц (выборка, изменения и т.д.)

Язык sql - не процедурный язык, поскольку не содержит операции управления, ввода, вывода. Это встроенный язык программирования СУБД, у каждой СУБД свой встроенный язык программирования

Встроенные языки программирования СУБД:

- язык FoxPro для СУБД Visual FoxPro
- язык Visual Basic for Application для СУБД Access

При статическом использовании sql, в тексте программы имеются вызовы функции языка sql, которые жестко включаются в исполняемый модуль после компиляции. Изменения в вызываемых функциях могут быть на уровне отдельных параметров с использованием переменных языка программирования

При динамическом использовании sql, предполагает динамическое построение вызовов sql функций и уже интерпретация этих вызовов при выполнении программы. Используется если в приложении заранее не известно какая sql команда будет вызвана, определяется в ходе диалога с пользователем

Типы данных:

Тип данных «строка символов»

Символьные строки фиксированной длины

CHARACTER (длина) или CHAR (длины) 0<длина

Строки переменной длины (не стандартизированный тип)

VARCHAR (длина)

Тип данных VARCHAR

позволяет более экономно использовать память, выделяемую для хранения символьных значений. И оказывается более удобным при выполнении операций связанных со сравнением строк

Числовые типы данных

- INTEGER или INT - целые числа от -2^{31} до $+2^{31}$
- SMALLINT - целые числа от -2^{15} до 2^{15}
- DECIMAL, DEC или NUMERIC (точность, масштаб) - десятичное число с фиксированной точкой
- FLOAT ([точность]), REAL - числа с плавающей точкой

Дата и время

Тип DATE в СУБД Oracle: с 1 января 4712 года до н.э. по 31 декабря 4712 года

Тип DATETIME в СУБД Sybase: с 01.01.1753 по 31.12.9999

Тип SMALLDATETIME: с 01.01. 1900 по 06.06.2079

Специальная арифметика дат и времен $13.01.2015 + 10 = 23.01.2015$

Дополнительные типы данных: INTERVAL, MONEY, BINARY

30. Язык SQL. Операторы создания и изменения структуры таблиц. Пример.

Язык определения данных (DDL – Data Definition Language) и язык манипулирования данными (DML – Data Manipulation Language)		
Язык	Название оператора	Назначение оператора
DDL	CREATE TABLE	создание таблицы
	DROP TABLE	удаление таблицы
	ALTER TABLE	изменение структуры таблицы
	CREATE INDEX	создание индекса
	DROP INDEX	удаление индекса
	CREATE VIEW	создание представления
	DROP VIEW	удаление представления
	GRANT	назначение привилегий
	REVOKE	удаление привилегий

Создание таблиц

Оператор CREATE TABLE создает пустую таблицу

CREATE TABLE <имя таблицы>

(<имя столбца> <тип данных>,
<имя столбца > < тип данных >, ...)

Отношение Сотрудник (Номер, ФИО, Дата Рождения, Зарплата, Пол)

```
CREATE TABLE sotr
(sotr_id INTEGER,
fio VARCHAR(20),
birthday DATE,
salary DECIMAL(8,2),
gender CHAR(1))
```

Оператор ALTER TABLE модифицирует структуру и параметры существующей таблицы
Добавление столбцов в таблицу

ALTER TABLE <имя таблицы> ADD

(<имя столбца> <тип данных>,
<имя столбца > < тип данных >, ...)

Изменение описания столбцов (изменение размеров, добавление / удаление ограничений на значения)

ALTER TABLE <имя таблицы> MODIFY

(<имя столбца> <тип данных>,
<имя столбца > < тип данных >, ...)

Ограничение модификаций характеристик столбца:

- изменение типа данных возможно, только если столбец пуст
- для незаполненного столбца можно изменять размер/точность, для заполненного - размер/точность можно увеличить, но не понизить

Удаление столбцов из таблицы

ALTER TABLE DROP <имя таблицы> DROP

(<имя столбца>, <имя столбца >, ...)

Оператор DROP TABLE удаляет существующую таблицу

31. Язык SQL. Ограничения на множество допустимых значений данных. Примеры

При создании и изменении таблицы можно определить ограничения на вводимые значение, в этом случае sql будет отвергать любое из них при несоответствии халатным параметрам

Ограничения являются статическими, если заранее известен диапазон значений вставляемых в столбец

Динамические ограничения определяются связью со значениями находящимися в другом столбце этой же таблицы или другой таблицы

2 основных типа ограничений:

- ограничения на столбцы
- ограничения на таблицу

CREATE TABLE <имя таблицы>

(<имя столбца> <тип данных> <ограничения на столбец>, <имя столбца> <тип данных>
<ограничения на столбец>, ...)

Запрет использования пустых значений - NOT NULL

CREATE TABLE sotr

(sotr_id INTEGER NOT NULL,
fio VARCHAR(20) NOT NULL,
birthday DATE)

Ограничение UNIQUE - запрет на использование повторяющихся значений

CREATE TABLE sotr

(sotr_id INTEGER NOT NULL UNIQUE,
fio VARCHAR(20) NOT NULL,
birthday DATE)

NOT NULL и UNIQUE всегда используются в пар

Первичный ключ таблицы PRIMARY KEY - комбинирование ограничений UNIQUE и NOT NULL

```
CREATE TABLE sotr
(sotr_id INTEGER PRIMARY KEY,
fio VARCHAR(20) NOT NULL,
...
gender CHAR(1))
CREATE TABLE project
(sotr_id INTEGER NOT NULL,
n_proj INTEGER NOT NULL,
task VARCHAR(50),
PRIMARY KEY(sotr_id, n_proj))
```

Ограничение CHECK - определение условия на вводимое значение

Значение зарплаты в таблице Sotr должно превышать 200

```
CREATE TABLE sotr
(sotr_id INTEGER PRIMARY KEY,
salary DECIMAL(8,2)
...
CHECK (salary > 200, ...))
```

DEFAULT - определение значения по умолчанию

DEFAULT «значение по умолчанию»

Большинство сотрудников, информация о которых хранится в таблице Sotr, - мужчины

32. Язык SQL. Поддержка целостности данных. Примеры.

Внешний ключ - это атрибут (или множество атрибутов) отношения, являющийся ключом другого отношения

FOREIGN KEY ограничивает допустимое множество значений столбца значениями родительского ключа ссылка на который указывается после слова REFERENCES

Варианты действий для сохранения ссылочной ценности данных:

- ограничение NO ACTION или RESTRICT (изменение значений внешнего ключа запрещено, если у него есть дочерние записи)
- ограничение CASCADE (изменение значений в родительской таблице -> изменение значений в дочерней таблице)
- ограничение SET NULL (изменять значения родительского ключа можно, но при это везде в значении дочерне записи будет стоять NULL)
- ограничение SET DEFAULT (изменять значения родительского ключа можно, но при это везде в значении дочерне записи будет стоять DEFAULT)

33. Язык SQL. Использование индексации для быстрого доступа к данным. Преимущества и недостатки использования индексов. Пример.

Индексация данных, является основным способом повышения производительности операций языка SQL

Индекс содержит упорядоченный список содержимого столбца (или группы столбцов) индексируемой таблицы вместе с идентификаторами этих строк

Индексы классифицируются и делятся на уникальные и не уникальные, в уникальном индексе есть только одна ссылка для каждого значения, а в не уникальном может быть произвольное количество ссылок для каждого значения

Индексы являются невидимыми для пользователя, все операции с индексами являются невидимыми для пользователя, чаще всего индексы используются в запросах, при выполнении запроса, СУБД решает какие индексы надо применить и надо ли вообще

Индексы способны ускорить операции:

- определение минимального и максимального значений по индексированному столбцу
- сортировку и группировку столбцов таблицы
- поиск типа is null или is not
- извлечение данных, когда нужны только проиндексированные данные

Недостатки использования индексов:

- при всяком изменении или удалении содержимого индексированного столбца, а также при добавлении новой строки индекс необходимо обновлять
- индекс сам по себе занимает дополнительное место
- при извлечении из таблицы очень большого числа строк использование индекса приводит к потере времени

34. Язык SQL. Операторы манипулирования данными. Примеры.

Манипулирование данными

INSERT добавляет в таблицу новую строку

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

Значения вводятся в том порядке в котором были указаны столбцы при создании таблицы

INSERT INTO <имя таблицы> (<столбец>, <столбец>, ...)

VALUES (<значение>, <значение>, ...)

Столбцы, имена которых не указаны в списке будут заполнены либо значениями по умолчанию, либо значением null

DELETE удаляет строки из таблицы

DELETE FROM <имя таблицы> WHERE <условие>

UPDATE изменяет значение полей в строках таблицы

UPDATE <имя таблицы> SET <столбец>=<значение>, <столбец>=<значение>, ... WHERE <условие>

35. Поисковые операции на языке SQL. Запросы по одной таблице. Упорядочивание результата запроса. Примеры.

Запросы по одной таблице

Запрос - предписание на выдачу информации хранимой в БД

SELECT (DISTINCT) <список атрибутов> FROM <имя таблицы>

[WHERE <условие выборки>]

[ORDER BY <список атрибутов>]

[GROUP BY <список атрибутов>]

[HAVING <условие>]

36. Поисковые операции на языке SQL. Группировка и агрегирующие функции. Примеры.

Выборка данных

После слова SELECT перечисляется список выводимых атрибутов, константы, выражения

Ключевое слово AS изменяет имя столбика выводимого в запросе

Таблица, получаемая в результате выполнения SQL запроса может не отвечать требованиям реляционной алгебры, а именно, в ней могут оказаться повторяющиеся строки

Для исключения из результата SELECT запроса повторяющиеся записи используется SELECT DISTINCT

WHERE - условие выдачи данных

После WHERE записывается логическое условие

При написании логических условий можно писать:

- операции сравнения: <=, >=, =, <>
- логические операторы AND, OR, NOT
- специальные операторы написания условий: IN, BETWEEN, LIKE

Оператор IN используется для сравнения проверяемого значения поля с заданным списком (указывается в круглых скобках сразу после оператора IN)

Оператор BETWEEN используется для проверки условия принадлежности значения заданному интервалу

Оператор LIKE используется только для символьных значений, определяет входит ли заданная подстрока (образец поиска) в проверяемое поле

Спец. символы: _ (заменяет один любой символ), % (любое кол-во символов)

Упорядочивание результата запроса

ORDER BY позволяет упорядочить выводимые записи в соответствии со значениями одного или нескольких выбранных столбцов

ASC - по возрастанию

DESC - по убыванию

Агрегирующие функции

Агрегирующие функции позволяют получать из таблицы сводную информацию, выполняя операции над группой строк таблицы

COUNT - кол-во строк или значений поля (не являющиеся null значениями)

SUM - сумма значений заданного атрибута

AVG - среднее значение заданного атрибута

MAX - максимальное значение заданного атрибута

MIN - минимальное значение заданного атрибута

Если до применения агрегирующей функции необходимо исключить повторяющиеся значения в столбце, то в круглых скобках перед именем столбца нужно написать ключевое слово DISTINCT

Группировка

GROUP BY позволяет группировать записи в подмножества и применять агрегирующие функции не ко всей таблице, а отдельно к каждой сформированной группе. В одну группу включаются те записи, которые имеют одинаковые значения в полях по которым выполняется группировка

По стандарту SQL в запросе с группировкой после ключевого слова SELECT можно указывать только те поля, по которым идет группировка или агрегирующие функции

Определить дату рождения самого молодого мужчины и самой молодой женщины в каждом отделе Порядок полей GROUP BY не влияет на результат

При необходимости часть сформированных групп (GROUP BY) можно исключить ключевым словом HAVING

Условие применяется к агрегирующим функциям

Этот запрос определяет наибольшую заработную плату в отделах где больше 2 сотрудников

37. Язык SQL. Запросы по нескольким таблицам. Примеры.

Запросы по нескольким таблицам

В результате выполнения запросов в котором указаны 2 таблицы происходит операция декартового произведения

Соединение таблиц имеет смысл в том случае, если соединяются не все строки исходных таблиц, а только их часть. Это ограничение записывается в разделе WHERE

Необходимо вывести список сотрудников с указанием проектов в которых они участвовали и заданиями, которые они выполняют

```
SELECT * FROM Sotr, Project
Результат (Sotr_id, Fio, Birthday, Dep, Salary, Gender,
Sotr_id, Proj_n, Task)
SELECT Sotr.Sotr_id, Fio, Proj_n, Task FROM Sotr, Project
WHERE Sotr.Sotr_id = Project.Sotr_id
SELECT S.Sotr_id, Fio, Proj_n, Task FROM Sotr S,
Project P WHERE S.Sotr_id = P.Sotr_id
```

38. Язык SQL. Команды с подзапросами. Типы подзапросов. Простые и коррелированные подзапросы. Примеры.

Часто невозможно решить задачу с использованием только одного запроса, это особенно актуально, когда при использовании условия отбора (в разделе WHERE или HAVING) значение с которым нужно сравнивать заранее не известно и должно быть вычислено в момент выполнения команды SELECT, в этом случае используются законченные команды SELECT, внедренные в тело другой команды

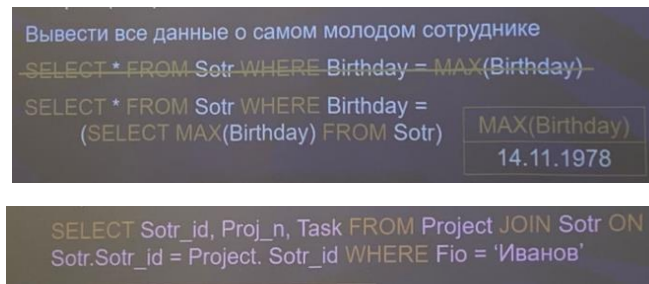
Подзапрос или вложенный запрос - это SELECT заключенный в круглые скобки и встроенное в тело другого SELECT или других команд

Типы подзапросов:

- скалярные (возвращает единственное значение)
- квантифицированный предикатные (возвращается список значений)
 - IN
 - ALL/ANY
 - EXISTS

Скалярные подзапросы

Для использования данного вида подзапроса пользователь должен быть уверен, что подзапрос вернёт единственное значение



Запрос работает корректно, если подзапрос вернёт только одно значение, в противном случае подзапрос будет считаться ошибочным

Для получения единственного значения:

- DISTINCT
- агрегирующие функции

Подзапросы, возвращающие множество значений

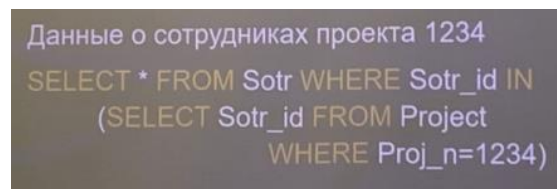
В большинстве случаев вложенные подзапросы генерируют неименованные промежуточные отношения (временную таблицу)

Применяемые к подзапросу операции множества:

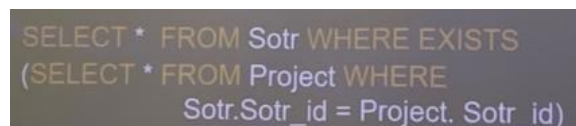
IN, NOT IN, ALL, ANY, EXISTS, NOT EXISTS

IN используется в основном запросе для отбора только тех записей, которые содержат значения, совпадающие с одним из отображенных вложенных запросов

ALL требует выполнения условия для всех значений найденных подзапросом, ANY требует выполнения хотя бы для одного такого значения



Предикат EXISTS используется для определения того, должен ли подчиненный запрос возвращать какие либо записи



Виды вложенных подзапросов:

- простой вложенный подзапрос (может вычисляться как независимый запрос, иначе говоря, результаты подзапроса подставляются во внешний запрос, первый обрабатывается вложенный запрос самого нижнего уровня, значения полученные в результате его выполнения, используются в запросе более высокого уровня и т.д.)
- коррелированный подзапрос (не может выполняться как независимый, поскольку он содержит условие отбора, зависящее от значения полей в основном запросе, запросы с EXISTS являются коррелированными и обрабатываются системой в обратном порядке,

сначала выбирается первая строка таблицы основного запроса, из нее отбираются значения тех столбцов, которые используются во вложенном подзапросе, если эти значения удовлетворяют условиям вложенного подзапроса, то первая строка включается в результат и т.д., пока не будут рассмотрены все строки основной таблицы)

Подзапросы в командах манипулирования данными

Insert into использует данные возвращаемые подзапросом для помещения их в другую таблицу

Обновляем и удаляем данные только из одной таблицы

```
Заполнить таблицу Sotr_1 записями обо всех сотрудниках,
получающих зарплату свыше 40000 рублей
INSERT INTO Sotr_1
  SELECT * FROM Sotr WHERE Salary > 40000

Увеличить зарплату сотрудникам, выполнявшим проект
3456, на 1500 руб.
UPDATE Sotr SET salary = salary+1500 WHERE Sotr_id IN
  (SELECT Sotr_id FROM Project WHERE Proj_n=3456)

Удалить из таблицы сведения о самом старом сотруднике
DELETE FROM Sotr WHERE Birthday =
  (SELECT MIN(Birthday) FROM Sotr)
```

39. Язык SQL. Оператор объединения. Пример.

Оператор объединения

Union используется для объединения результатов двух или более select запросов в единое множество строк и столбцов с удалением повторяющихся результатов

Использование оператора union возможны только при объединении тех запросов, столбцы которых совместимы

```
UNION
< SELECT -запрос>
UNION
< SELECT-запрос>
[UNION ... < SELECT-запрос>]

Получить в одной таблице фамилии сотрудников и
руководителей всех отделов
SELECT Dep AS 'Отдел', 'Руководитель' AS
'Должность', Fio AS 'Фамилия' FROM Boss
UNION
SELECT Dep AS 'Отдел', 'Сотрудник' AS 'Должность',
Fio AS 'Фамилия' FROM Sotr
```

Сортировка указывается один раз, для объединения

Если необходимо оставить дубликаты, используется union all

40. Язык SQL. Работа с представлениями. Примеры.

Представления

Это виртуальная таблица, в которой собраны данные из реально существующих таблиц

Таблица представлений не содержит собственных данных, фактически, представления - это именованная таблица, содержимое которой является результатом запроса заданного при описании представления

При чем этот запрос выполняется каждый раз, когда происходит обращение к таблице представлений

Представления позволяют:

- ограничивать число столбцов
- ограничивать число строк
- выводить дополнительные столбцы, преобразованные из других столбцов базовой таблицы

```
CREATE VIEW First_Dep AS
SELECT * FROM Sotr WHERE Dep = 1
SELECT * FROM First_Dep
```

Sotr_id	Fio	Birthday	Dep	Salary	Gender
001	Иванов	12.01.1976	1	37500	м
066	Филин	31.10.1976	1	38000	м
087	Белова	10.06.1966	1	47300	ж

Представление используется для объединения нескольких таблиц в одну виртуальную

41. Объекты БД. Схемы в БД. Синтаксис языка Transact SQL. Имена объектов. Локальные переменные.

- язык определения данных (DDL - Data Definition Language)
- язык манипулирования данными (DML - Data Manipulation Language)
- язык управления данными (Data Control Language), содержащий операторы для разграничения доступа пользователей к объектам БД
- алгоритмическую составляющую языка, включающую в себя процедуры, переменные, циклы, условные конструкции и т.п.

Схема в БД - логическое образование, которое объединяет несколько объектов БД

В БД могут существовать объекты одного вида с одинаковыми именами, но при этом они должны принадлежать к разным схемам

Схемы удобно использовать для предоставления разрешений

Преимущества применения схем:

- каждый объект БД относится к какой-либо схеме
- для каждого пользователя определяется своя схема по умолчанию
- нескольким пользователям можно назначить одну и ту же схему по умолчанию
- один пользователь может являться владельцем сразу нескольких схем
- при создании объекта можно явно указать схему, в которую его нужно поместить

Имя сервера - сервер на котором располагается объект

Имена объектов должны состоять из символов латинского алфавита, цифр, спец. символов

- Локальные переменные используются для:
- проведения различных вычислений передачи данных в другие процедуры и функции и возвращения данных из процедур и функций
- хранения промежуточных данных и т.п.

DECLARE @a INT, @i AS REAL

DECLARE @s CHAR(15), @name CHAR(15)

При объявлении переменной ей присваивается значение NULL

SET и SELECT - присвоить значение

SET @name = 'Ирина'

SELECT @name = 'Ирина'

Задание. Присвоить переменной @name значение имени пользователя с номером 1

- Объявление переменных

DECLARE @id INT, @name VARCHAR(40)

- Задание значения переменной @id

SET @id = 1

- Задание значения переменной @name

SELECT @name = UserName FROM Users WHERE UserID = @id

/* Вывод переменной @name в результат запроса (в форме таблицы) */

SELECT @name AS [Имя пользователя]

- или печать на экран

PRINT Имя пользователя - '+' @name

Программный модуль на языке TSQL состоит из отдельных предложений, это могут быть операторы, команды SQL и других элементов языка

42. Язык Transact SQL. Операторы языка. Условные и циклические конструкции.

Арифметические операторы: +, -, *, /, %

Операторы сравнения: =, >, <, <=, >=

Логические операторы, возвращающие TRUE/FALSE: NOT, ALL, OR, AND, ANY, IN, BETWEEN, EXISTS, LIKE

Группировка двух или более команд в единый блок BEGIN END

Конструкция ветвления (условные конструкции)

IF <лог.выражение>

Оператор

[ELSE

Оператор]

В зависимости от того если ли сотрудник с фамилией 'Агеев' выполняются те или иные действия

IF EXISTS (SELECT * FROM Sotr WHERE Fio LIKE 'Агеев%')

PRINT 'Такой сотрудник имеется'

ELSE

BEGIN

INSERT INTO Sotr (Sotr_id, Fio) VALUES (114, 'Aree')

```
PRINT 'Сотрудник был добавлен'
```

```
END
```

CASE встраиваемая конструкция, которая проверяет значения нескольких выражений и в зависимости от результата проверки возвращает тот или иной результат

- С входным выражением (только равенство)

```
CASE <входное выражение>
```

```
WHEN <выражение 1> THEN <результат 1>
```

```
[WHEN <выражение 2> THEN <результат 2> ...]
```

```
[ELSE <результат> ]
```

```
END
```

Вывести список сотрудников с указанием их пола

```
SELECT Fio, CASE Gender
```

```
    WHEN 'м' THEN 'Мужчина'
```

```
    WHEN 'ж' THEN 'Женщина'
```

```
    ELSE 'Не указан'
```

```
END
```

```
FROM Sotr
```

- Без входного выражения

```
CASE
```

```
    WHEN <лог. выражение 1> THEN <результат 1>
```

```
    [WHEN <лог. выражение 2> THEN <результат 2> ...]
```

```
    [ELSE <результат> ]
```

```
END
```

```
SELECT Fio, CASE
```

```
    WHEN Salary < 8000 THEN 'Низкая'
```

```
    WHEN Salary BETWEEN 8000 AND 15000 THEN 'Средняя'
```

```
    WHEN Salary > 15000 THEN 'Высокая'
```

```
END
```

```
FROM Sotr
```

CASE используется также в операторе UPDATE, INSERT и DELETE

```
IIF (<логическое выражение>, <результат 1>, <результат 2>)
```

Вывести список сотрудников с указанием их пола

```
SELECT Fio, IIF (Gender = 'м', 'Мужчина', 'Женщина')
```

```
FROM Sotr
```

Изменить зарплату сотрудникам в зависимости от пола

```
UPDATE Sot SET Salary =
```

```
    IIF (Gender = 'М', Salary* 1.25, Salary*1.2)
```

Команда BREAK принудительно останавливает цикл

Команда CONTINUE останавливает задачу и начинает цикл заново

43. Язык Transact SQL. Табличные переменные.

Переменная типа таблица, по структуре соответствует обычной таблице БД, в нее можно загрузить данные, можно выполнять все те же самые операции, что и над обычной таблицей. Единственное ограничение - область действия

Можно задать:

- описание столбцов таблицы (имя, тип данных, значение по умолчанию, ограничения на столбец, возможность NULL значений, первичный ключ, уникальность и т.п.)
- описание ограничений на уровне таблицы

В таблице может быть только один счётчик

Опция IDENTITY (<начальное значение>, <шаг>)

Для табличных переменных не предусмотрено использование FOREIGN KEY, они не связаны с другими таблицами

44. Язык Transact SQL. Курсоры. Создание курсоров и определение их характеристик.

Курсор - это особый временный объект SQL, который создается на основе таблиц или представлений БД, и имеет средства построчного передвижения по результирующему набору строк запроса

• 1. Создание курсора

```
DECLARE <имя_курсора> CURSOR
[<видимость>]
[<прокрутка>]
[<тип>]
[<блокировка>]
FOR <SELECT_запрос>
[FOR UPDATE [OF <имена_столбцов>]]
```

Ограничения на команду SELECT:

- не может содержать INTO для создания новой таблицы
- не может содержать COM или COMPUTE BY, но может содержать функции агрегирования

Основные характеристики курсора:

- способность отражать изменения в исходных данных
- способность осуществлять прокрутку во множестве строк
- способность модифицировать множество строк

Видимость курсора определяется с помощью слов LOCAL или GLOBAL, глобальный существует до закрытия соединения с сервером, а локальный в рамках одной программы

Прокрутка: FORWARD_ONLY - только вперед, от начала до конца. SCROLL - в любом направлении

Тип курсора:

- STATIC (статический курсор), формируется один раз и не обновляется, не чувствует изменений в структуре или в значении исходных данных, открывается в режиме «только для чтения»
- KEYSET (ключевой курсор), формирует курсор только из столбцов первичного ключа, на них нет ограничений
- DYNAMIC (динамический курсор), всегда отображение последнее изменение БД
- FAST_FORWARD (курсор быстрого доступа), оптимизированы для быстрого доступа к данным, допускает только чтение

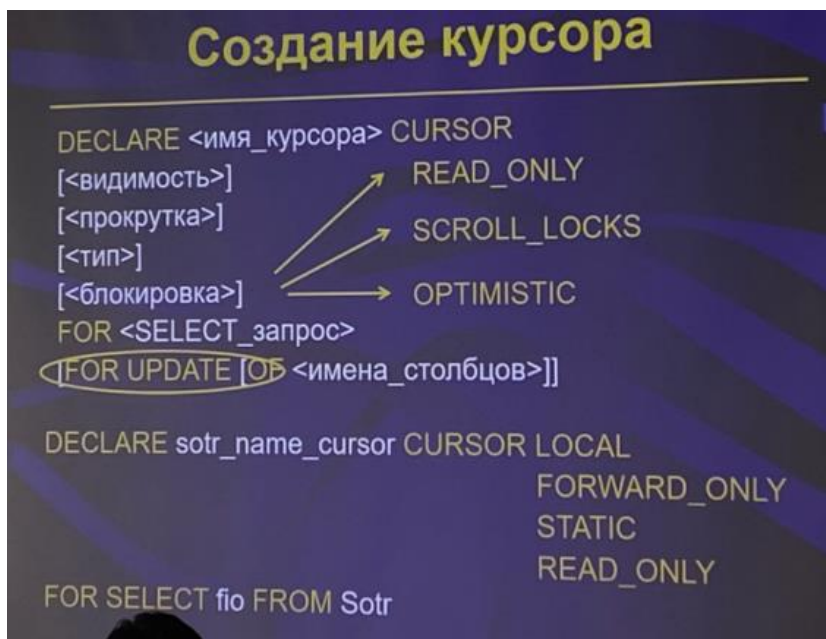
Блокировка: READ_ONLY - курсор не может вносить изменения в исходные данные.

SCROLL_LOCKS - обновление в исходных данных может выполнить только курсор.

OPTIMISTIC - позволяет обновлять строки исходных данных как курсором, так и вне его.

FOR UPDATE - ограничения на курсор, ограничение на столбцы, только их можно будет менять.

Создаем локальный, последовательный, статический курсор, пример:



45. Язык Transact SQL. Открытие курсора и работа с ним. Пример.

2. Открытие курсора

DECLARE создает курсор, но не заполняет его данными

OPEN [GLOBAL] <имя_курсора>

3. Чтение курсора

FETCH <имя_курсора>

Извлекает строку из множества строк курсора

FETCH INTO <имя_курсора> INTO <список_переменных>

FETCH <режим> FROM <имя_курсора> INTO <список_переменных>

Режимы:

- NEXT
- PRIOR
- FIRST
- LAST
- ABSOLUTE n
- RELATIVE n

4. Закрытие курсора

CLOSE [GLOBAL] <имя_курсора>

Используется в том случае, когда необходимо открыть его повторно

5. Освобождение курсора

DEALLOCATE [GLOBAL] <имя_курсора>

46. Язык Transact SQL. Мониторинг курсоров. Модификация и удаление строк через курсоры.

Пример.

Значения глобальной переменной @@CURSOR_ROWS:

- -1 - число строк может меняться
- 0 - курсор не открыт или содержит 0 строк
- n - количество строк в курсоре равно n

Значения глобальной переменной @@FETCH_STATUS:

- 0 - команда FETCH выполнена успешно
- 1 - команда выполнена неудачно (выход за пределы курсора)
- 2 - команда выполнена неудачно (обращение к удаленной записи)

```
-- объявление курсора avg_salary_cur
DECLARE avg_salary_cur CURSOR LOCAL
        FORWARD_ONLY STATIC READ_ONLY
FOR SELECT Salary FROM Sotr

-- открытие курсора
OPEN avg_salary_cur

-- объявление локальных переменных
DECLARE @kol INT, @x INT, @salary BIGINT

-- инициализация переменных
SET @kol = 0
SET @salary = 0

-- чтение данных из курсора в локальные переменные
FETCH NEXT FROM avg_salary_cur INTO @x

-- цикл по всем записям курсора
WHILE @@FETCH_STATUS = 0
BEGIN
    SET @salary = @salary + @x
    SET @kol = @kol + 1
    -- переход к следующей записи и чтение данных
    FETCH NEXT FROM avg_salary_cur INTO @x
END

-- вычисление среднего значения
SET @salary = @salary / @kol

-- вывод результатов
PRINT 'Средняя зарплата составляет - ' +
      CAST (@salary AS VARCHAR(8))

-- закрытие курсора
CLOSE avg_salary_cur

-- освобождение курсора
DEALLOCATE avg_salary_cur
```

CAST (AS)

Модификация и удаление строк через курсоры

Предусмотрена специальная форма фразы WHERE, которая поддерживает модификацию и удаление через курсор

А. Позиционное обновление

UPDATE <имя таблицы или представления>

SET <список для модификации>

WHERE CURRENT OF <имя курсора>

Б. Позиционное удаление

DELETE <имя таблицы или представления>

WHERE CURRENT OF <имя_курсора>

47. Хранимые процедуры. Преимущества выполнения в БД хранимых процедур. Типы хранимых процедур.

При создании приложения, которое имеет серверную составляющую, встает вопрос как клиентское приложение будет работать с БД которая расположена где-то на серверах

Взаимодействия:

- На сервере хранятся только таблицы, а остальной код хранится в коде приложения
- На сервере хранятся все процедуры, а в коде приложения только интерфейс

Хранимая процедура - программа, которая объединяет запросы и их процедурную логику (операторы присваивания, ветвления и т.д.), которая хранится в БД

Поскольку процедуры хранятся на сервере, то и выполняются они на сервере, обеспечивая более высокое быстродействие

Одна и та же ХП может быть использована в любом количестве, на любом пользовательском приложением

Преимущества выполнения в БД хранимых процедур по сравнению с отдельными командами Transact SQL:

- хранимые процедуры поддерживают модульное программирование
- хранимые процедуры могут вызывать другие ХП и функции
- хранимые процедуры могут быть вызваны из прикладных программ других типов
- все они прошли этап синтаксического анализа и находятся в исполняемом формате
- хранимые процедуры выполняются быстрее
- хранимые процедуры проще использовать
- ХП может вызвать клиентское приложение, другая ХП и триггер

Типы хранимых процедур:

- системные хранимые процедуры (они предназначены для выполнения административных действий) sp_addlogin или sp_statistics (хранятся в системных БД)
- пользовательские хранимые процедуры
- временные хранимые процедуры; локальные (# имя) и глобальные (## имя)

48. Хранимые процедуры. Создание и вызов хранимых процедур. Пример.

Создание хранимых процедур

CREATE PROCEDURE <имя процедуры>

[{@имя параметра <тип данных> [= значение _по_умолчанию] [OUTPUT]}]

AS <тело процедуры>

В теле процедуры могут применяться практически все команды языка SQL, могут объявляться транзакции, могут устанавливаться блокировки и могут вызываться ХП

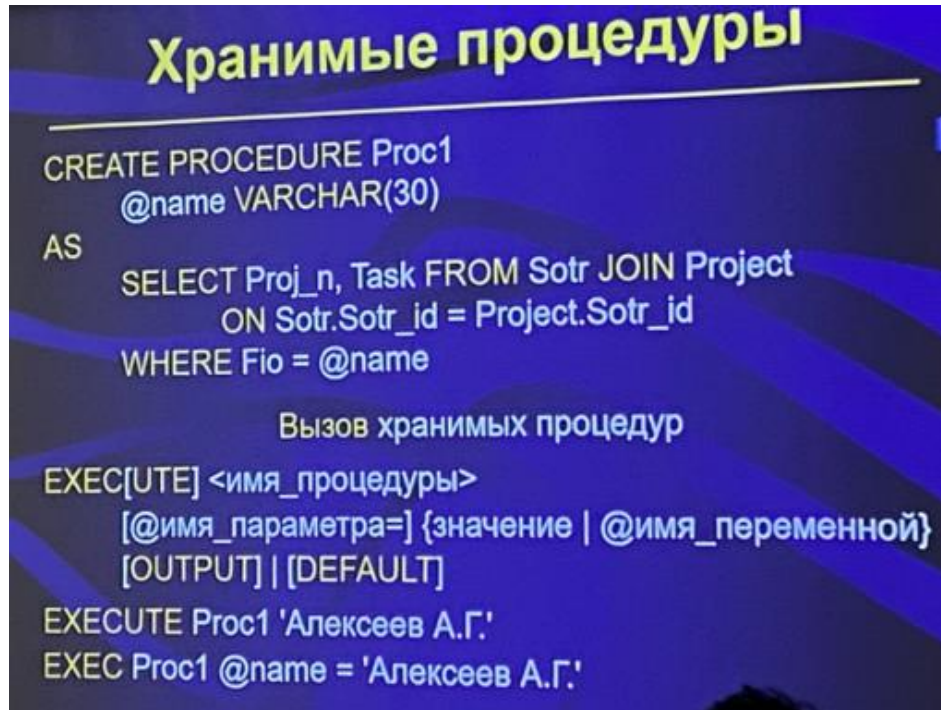
Для преждевременного выхода из процедуры используется команда RETURN

Вызов хранимых процедур

EXEC[UTE] <имя процедуры>

[@имя параметра=] {значение | @имя переменной}

[OUTPUT] I (DEFAULT)



49. Пользовательские функции. Виды пользовательских функций. Отличие пользовательских функций от хранимых процедур.

Пользовательские функции являются программными модулями, которые принимают параметры, выполняют некоторые действия и возвращают результат

Виды пользовательских функций:

- скалярные функции (scalar), могут состоять из любого числа операторов, возвращают не табличные значения
- функции с одним запросом (inline-функции), состоят из одной SELECT команды и возвращают таблицу
- многооператорные табличные функции (multi-statement)

Отличие функций от хранимых процедур:

- В теле функции запрещены изменения глобальных объектов, например таблиц, курсоров и т.д.
- Если при выполнении команды произойдет ошибка в теле процедуры, то текущая команда прерывается, и выполнится следующая команда данной процедуры

В функции такая ошибка приведет к прерыванию выполнения всей функции

50. Пользовательские функции. Создание и вызов скалярных пользовательских функций. Пример.

Пользовательские функции являются программными модулями, которые принимают параметры, выполняют некоторые действия и возвращают результат

скалярные функции (scalar), могут состоять из любого числа операторов, возвращают не табличные значения

Создание скалярной функции

```
CREATE FUNCTION <имя функции>
    ([[@имя параметра стип данных> [= значение _по умолчанию]])
RETURNS скалярный_тип_данных
AS
BEGIN
    Операторы
    RETURN скалярное_выражение
END
```

```
CREATE FUNCTION MaxSotr ()
RETURNS INT
AS
BEGIN
    -- сохраним во временной таблице число
    сотрудников каждого отдела
    DECLARE @temp TABLE (kol INT)
    INSERT INTO @temp SELECT COUNT(*) FROM Sotr
                                GROUP BY Dep
    -- найдем максимальное число среди всех отделов
    DECLARE @m INT
    SELECT @m = MAX(kol) FROM @temp
    RETURN @m
END
DECLARE @Result INT
SELECT @Result = MaxSotr ()
PRINT @Result
```

```
CREATE FUNCTION CountSotrInProj (@proj INT)
RETURNS INT
AS
BEGIN
    DECLARE @c INT
    SELECT @c = COUNT(*) FROM Project WHERE
                                Proj_n = @proj
    RETURN @c
END
PRINT CountSotrInProj (1234)
```

51. Пользовательские функции. Создание и вызов табличных пользовательских функций.

Пример.

Пользовательские функции являются программными модулями, которые принимают параметры, выполняют некоторые действия и возвращают результат

функции с одним запросом (inline-функции), состоит из одной SELECT команды и возвращает таблицу

Функции типа inline

```
CREATE FUNCTION <имя_ функции>
    ([[@имя параметра <тип_данных> [ = значение _по _умолчанию]]])
RETURNS TABLE
AS
RETURN <SELECT-запрос>
```

Все поля SELECT запроса должны иметь имена, если его нет, то задать

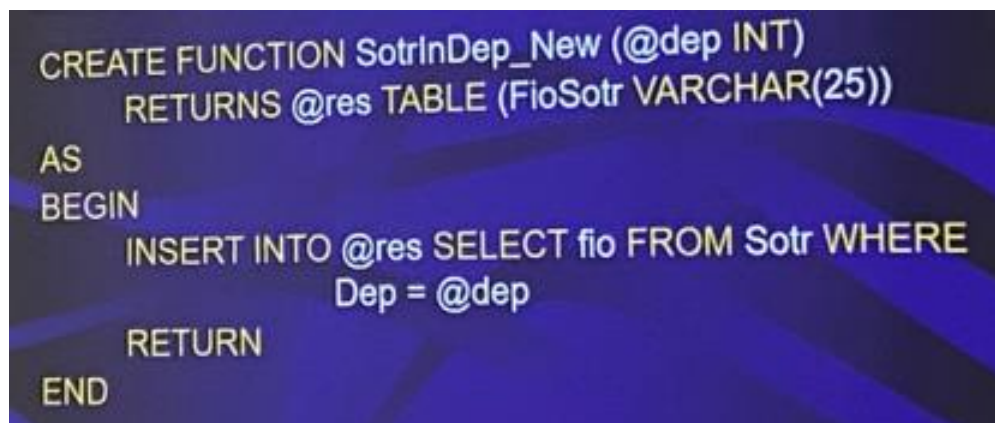


```
CREATE FUNCTION SotrInDep (@dep INT)
RETURNS TABLE
AS
    RETURN
    SELECT fio FROM Sotr WHERE Dep = @dep
SELECT * FROM SotrInDep(4) ORDER BY fio
```

многооператорные табличные функции (multi-statement)

Многооператорные табличные функции

```
CREATE FUNCTION <имя_функции>
    ([[@имя_параметра <тип_данных>[ = значение _по_умолчанию] } ]])
RETURNS @имя_таблицы TABLE <определение_таблицы>
AS
BEGIN
    Операторы
    RETURN
```



```
CREATE FUNCTION SotrInDep_New (@dep INT)
    RETURNS @res TABLE (FioSotr VARCHAR(25))
AS
BEGIN
    INSERT INTO @res SELECT fio FROM Sotr WHERE
        Dep = @dep
    RETURN
END
```

52. Транзакции. Свойства транзакций. Три вида определения транзакций. Действия при разработке транзакций.

Транзакция - это неделимая последовательность операций обработки данных, которая выполняется как единое целое и переводит БД из одного целостного состояния в другое

Свойства транзакций:

- Атомарность - каждая транзакция представляет собой единицу работы СУБД и не может быть разбита на меньшие части, выполняются либо все изменения данных в определённой транзакции либо ни одного
- Согласованность - транзакция не может прервать ни одну из проверок на противоречивость
- Изолированность - транзакции следует изолировать от изменения данных другими транзакциями, никакая другая транзакция не может изменять данные, пока они не зафиксированы
- Устойчивость - после завершения транзакции, изменения данных станут устойчивыми, независимо от

3 вида определения транзакции (только для нашей СУБД)

- автоматическое
- явное
- подразумеваемое

По умолчанию SQL сервер работает в режиме автоматического начала транзакции

При разработке транзакций следует:

- Определить границы транзакции (должна быть настолько маленькой, насколько возможно, но её продолжительность должна быть достаточной чтобы она отвечала определённым требованиям)
- Разработать механизм управления ошибками
- Определить уровень изолированности транзакции, SQL сервер реализует различные варианты изолированности транзакций, дабы избежать простота

53. Автоматические транзакции. Пример. Команды определения явных транзакций. Пример.

Выполняется всё или ничего

Для того чтобы явно указать транзакцию используется первая строчка кода ниже

`BEGIN TRAN [SACTION] [<Имя_транзакции>]`

В журнале транзакций фиксируется первоначальное значение изменения данных и момент начала транзакции

`Commit tran [saction] [<имя_транзакции>]`

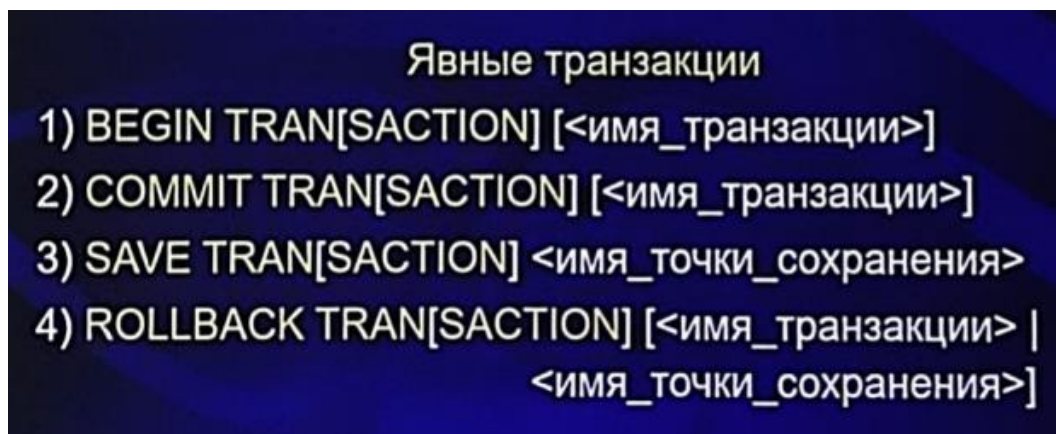
Если транзакций большая, можно сделать точку сохранения

`Save tran [saction] <имя_точки_сохранения>`

Откатить на любое сохранение или начало

`Rollback tran [saction] [<имя_транзакции>|< имя_точки сохранения>]`

Восстанавливается первоначальное состояние системы




```
BEGIN TRAN
    INSERT INTO table1 VALUES (1, 'Первая строка')
    INSERT INTO table1 VALUES (2, NULL)
    INSERT INTO table1 VALUES (3, 'Третья строка')
COMMIT TRAN
SELECT * FROM table1
```

Id	col
1	Первая строка
3	Третья строка

54. Транзакции. Транзакции с обработчиком ошибок. Пример.

Транзакция - это неделимая последовательность операций обработки данных, которая выполняется как единое целое и переводит БД из одного целостного состояния в другое

Обработка ошибок

В блоке begin-end выполняются потенциально опасные программы, если при этом произойдёт ошибка уровня с 11-19, то управление передаётся в блок catch

Уровни ошибок:

1-10 Информационные сообщения

11-19 Относительно серьёзные ошибки

20-25 Очень серьёзные ошибки

```
-- блок команд с возможными ошибками
BEGIN TRY
    BEGIN TRAN
        INSERT INTO table1 VALUES (1, 'Первая строка')
        INSERT INTO table1 VALUES (2, NULL)
        INSERT INTO table1 VALUES (3, 'Третья строка')
    COMMIT TRAN
END TRY
-- обработка ошибок
BEGIN CATCH
    RAISERROR("Ошибка в транзакции!", 14, 1)
    ROLLBACK TRAN
END CATCH
```

55. Триггеры. Цели создания триггеров. Компоненты триггера. Типы триггеров.

Триггер - это предварительно определенное действие или последовательность действий или последовательность действий, автоматически осуществляемых при выполнении операций модификации данных

Ни пользователь ни приложение не может запустить триггер, он выполняется автоматически при изменении данных

Триггеры обеспечивают проверку любых изменений на корректность, прежде чем любые изменения будут выполнены

Цели использования триггеров:

- Проверка на корректность введенных данных и выполнение сложных ограничений целостности данных (Например ограничение на количество студентов в группе)
- Выдача предупреждений о необходимости выполнения некоторых действий с таблицей (Не стоит использовать триггеры, если можно использовать другие способы проверки)

Каждый триггер привязывается к одной конкретной таблице, всё производимые триггером действия, рассматриваются как одна подразумеваемая транзакция без начала и конца, но с возможностью прерывания

Компоненты триггера:

- Ограничения
- Событие - характеризует возникновение ситуации, требующей проверки ограничений (добавление, удаление, изменение)
- Предусмотренное действие

Типы триггеров:

AFTER - триггер запускается только после успешного выполнения модификации данных

INSTEAD OF - запускается вместо операции по модификации

Insert, update, delete - определяет на какую команду отреагирует триггер

56. Триггеры. Создание триггеров. Особенности программирования триггеров. Пример.

Триггер - это предварительно определенное действие или последовательность действий или последовательность действий, автоматически осуществляемых при выполнении операций модификации данных

Ни пользователь ни приложение не может запустить триггер, он выполняется автоматически при изменении данных

Триггеры обеспечивают проверку любых изменений на корректность, прежде чем любые изменения будут выполнены

Создание триггеров:

```
CREATE TRIGGER <имя_триггера>
  ON <имя таблицы_или_представления>
  {AFTER | INSTEAD OF}
  {[INSERT] [,] [UPDATE] [,] [DELETE]}
  AS <операторы_SQL>
```

Программирование триггеров

Специальные таблицы: inserted и deleted

Функция @@ROWCOUNT возвращает количество строк, обработанных последней командой

AFTER - триггер запускается только после успешного выполнения модификации данных

INSTEAD OF - запускается вместо операции по модификации

Insert, update, delete - определяет на какую команду отреагирует триггер

На одну и ту же команду модификации может быть написан не один триггер

Триггеры типа after нельзя создавать для представлений, только для таблиц

Типа instead of и на представления и на таблицы

Программирование триггеров

Специальные таблицы inserted и deleted

Структура этих таблицы идентична для той таблицы на которую пишется триггер, на каждый триггер создаётся пара таблиц inserted, deleted

Команды	Содержимое таблиц	
	<i>inserted</i>	<i>deleted</i>
INSERT	строки, которые пользователь вставляет в таблицу	пусто
DELETE	пусто	строки, которые пользователь пытается удалить
UPDATE	новые значения строк	старые значения строк

ROLLBACK TRANSACTION


```

CREATE TRIGGER Trig_ins_after ON Sotr AFTER INSERT
AS
BEGIN
    -- определить отдел, в который назначен сотрудник
    DECLARE @dep INT
    SELECT @dep = Dep FROM inserted
    -- если это третий отдел, то отменить добавление
    IF @dep = 3
    BEGIN
        PRINT 'Запрещено добавление в третий отдел'
        ROLLBACK TRAN
    END
END

```

Return

End

Если нужно выйти из триггера, надо написать команду return, rollback не окончание работы триггера

```

CREATE TRIGGER Trig_ins_instead ON Sotr
    INSTEAD OF INSERT
AS
BEGIN
    DECLARE @dep INT
    SELECT @dep = Dep FROM inserted
    -- если это не третий отдел, то реализовать добавление
    IF @dep <> 3
        INSERT INTO Sotr SELECT * FROM inserted
    ELSE
        PRINT 'Запрещено добавление в третий отдел'
    END
END

```

В экзаменационный билет входят два теоретических вопроса и одна задача.

Задачи из следующих разделов:

- построение ER-диаграмм;
- операции реляционной алгебры;
- нормализация реляционных баз данных;
- курсоры;
- хранимые процедуры;
- пользовательские функции;
- триггеры.