**#1.** **Понятие типа данных, их классификация**

Тип данных определяет диапазон значений и операций, которые могут быть применены к этим значениям. Классификация типов данных Согласно стандартной классификации, *типы данных* бывают следующие 1. Простые. 1.1. Числовые. Хранятся числа. Могут применяться обычные арифметические операции. 1.1.1. Целочисленные: со знаком, то есть могут принимать как положительные, так и отрицательные значения; и без знака, то есть могут принимать только неотрицательные значения. 1.1.2. Вещественные: с фиксированной точкой, то есть хранятся знак и цифры целой и дробной частей и с плавающей точкой, то есть число приводится к виду m\*2e, где m — мантисса, e — экспонента причем 1/2<=m<=1, а e - целое число и хранятся знак, и числа m и e. 1.2. Символьный тип. Хранит один символ. Могут использоваться различные кодировки.1.3. Строковый тип. Хранит строку символов. Может применяться операция конкатенация (сложение строк). Вообще говоря, может рассматриваться как массив символов, но как правило выделяется в качестве простого. 1.4. Логический тип. Имеет два значения: истина(true) и ложь(false). Могут применяться логические операции. Используется в операторах ветвления и циклах. В некоторых языках является подтипом числового типа, при этом false=0, true=1. 1.5. Перечислимый тип. Может хранить только те значения, которые прямо указаны в его описании.2.

*Составные*. Формируются на основе комбинаций простых типов. 2.1. Массив. Является индексированным набором элементов одного типа. Одномерный массив — вектор, двумерный массив — таблица. 2.2. Запись. Набор различных элементов (полей записи), хранимый как единое целое. Возможен доступ к отдельным полям записи. 2.3. Множество (тип данных). В основном совпадает с обычным математическим понятием множества. Допустимы стандартные операции с множествами и проверка на принадлежность элемента множеству. 3. Другие типы данных. Если описанные выше типы данных представляли какие-либо объекты реального мира, то рассматриваемые здесь типы данных представляют объекты компьютерного мира, то есть являются исключительно компьютерными терминами. 3.1. Указатель (тип данных). Хранит адрес в памяти компьютера, указывающий на какую-либо информацию, как правило - указатель на переменную. 3.2. Ссылки (тип данных)

**2.** **Понятие модели данных.**

Модель данных - это совокупность структур данных и операций их обработки. Рассмотрим три основных типа моделей данных: иерархическую, сетевую и реляционную. Иерархическая модель представляет собой совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих перевернутое по структуре дерево (граф). К основным понятиям иерархической структуры относятся уровень, узел и связь. Узел - это совокупность атрибутов данных, описывающих некоторый объект. На схеме иерархического дерева узлы представляются вершинами графа. Каждый узел на более низком уровне связан только с одним узлом, находящимся на более высоком уровне. Иерархическое дерево имеет только одну вершину, не подчиненную никакой другой вершине и находящуюся на самом верхнем - первом уровне. Зависимые (подчиненные) узлы находятся на втором, третьем и т. д. уровнях. Количество деревьев в базе данных определяется числом корневых записей. К каждой записи базы данных существует только один иерархический путь от корневой записи.

В сетевой структуре при тех же основных понятиях (уровень, узел, связь) каждый элемент может быть связан с любым другим элементом

Реляционная модель данных объекты и связи между ними представляет в виде таблиц, при этом связи тоже рассматриваются как объекты. Все строки, составляющие таблицу в реляционной базе данных, должны иметь первичный ключ. Все современные средства СУБД поддерживают реляционную модель данных. Эта модель характеризуются простотой структуры данных, удобным для пользователя табличным представлением и возможностью использования формального аппарата алгебры отношений и реляционного исчисления для обработки данных.

**3. Иерархическая модель данных.**

Одной из первых моделей данных была иерархическая модель, предложенная компанией IBM в конце 1960-х гг. в СУБД IMS (Information Management System). Такая модель данных представляется набором деревьев, связанных друг с другом по принципу построения иерархических структур. В современных информационных системах такими иерархическими моделями данных пользуются при составлении структур в формате XML, применяя при передаче данных и работе с данными в интернет- системах. Организация данных в иерархической модели предполагает обязательное сочетание родительского и дочернего объектов данных. Эти объекты данных характеризуются следующими структурными элементами:

• атрибут — представляется наименьшей единицей элемента данных, обозначающей функциональную сущность описываемого свойства объекта;

• запись — представляется группой атрибутов, обозначающей конкретный экземпляр описываемого объекта;

• групповое отношение [1] — представляется связью между записями разных типов, которая отражает взаимодействие родительской и дочерней записей;

• ключевой элемент — представляется атрибутом, который может содержать только уникальные значения для каждой записи.

Учитывая уникальность значений ключевых элементов в иерархической модели, для объекта данных некорневого уровня она содержит ключевой элемент, определяемый уникальностью значений в рамках группового отношения, что позволяет однозначно выделить уникальную запись дочернего объекта в пределах конкретной записи родительского объекта. Для групповых отношений характерно важное свойство, которое предполагает, что для хранения любой некорневой записи обязательно должна существовать родительская запись. При этом если удаляется родительская запись, то автоматически должны быть удалены все дочерние записи. Эти свойства накладывают очень существенное ограничение на использование модели в реальных информационных системах. Например, когда нужно только накапливать данные, то иерархическая модель может быть вариантом представления соответствующей структуры. Также иерархическая модель используется при наличии иерархически организованных данных.

\_Примером такой организации может быть интернет-страница или структура передаваемого сообщения, что зачастую организуется с помощью иерархической схемы XML.

К преимуществам иерархической модели можно отнести относительную простоту восприятия логической структуры базы данных человеком. Система обеспечивает высокое быстродействие при транзакционной обработке, когда номенклатура типов запросов фиксирована.

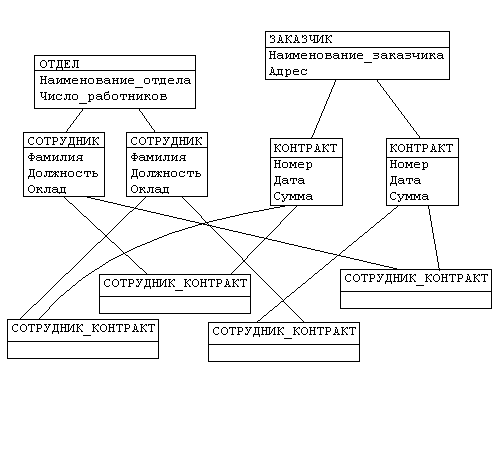
Наряду с существенными достоинствами иерархической модели выделяется несколько недостатков, среди которых отметим следующие:

— в реализации иерархической модели возникает дублирование сведений (создание парных записей[2]), при этом в модели не предусмотрена поддержка соответствия таких записей;

— в иерархической модели реализуется только связь типа 1:JV (один — ко — многим), причем только в направлении от родительского элемента к дочернему, не предоставляя возможности создания связей с указанием нескольких родительских элементов.

Иерархическая структура базы данных – это основа функционирования семейства ОС Windows. Для того чтобы убедиться в этом, достаточно воспользоваться традиционной опцией «Проводник». Кликнув по ней, вы попадает в корень операционной системы, в которой расположены крупнейшие структурные компоненты – «Этот компьютер», «Загрузки», «Изображения», «Музыка», «Рабочий стол» и др. Двигаясь в направлении от корня до других структурных единиц, вы будете переходить от одного уровня гигантской иерархической системы до другого. На практике это выглядит примерно так: «Этот компьютер»→«Локальный диск (С)»→«Пользователи»→«User»→«Документы»→«Файл». Таким образом, переходя от одной папки к другой, вы, наконец, находите необходимую вам информацию. В данном случае иерархический вид базы данных выступает в качестве вашего путеводителя, обеспечивающего быстрый и удобный доступ ко всем компонентам системы.

**4. Сетевая модель данных.**

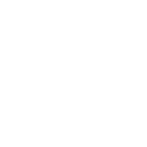
Сетевая модель, как и иерархическая, была 

разработана в 1960-х гг. Одним из первых вариантов ее реализации (начало

1970-х гг.) стала система CODASYL (Conference on Data System Languages).

В ее основе лежат те же понятия, что и в иерархической модели, т.е. вершина – уровень связь но существенным отличием является то, что любая вершина может быть связана с любой другой вершиной, независимо от уровня иерархии последней.

Если использовать язык теории графов, то иерархическая структура отображается графом типа дерево, в сетевая – произвольным графом, т.е. графом допускающим существование контуров.

Достоинства: предоставляет большие возможности в смысле допустимости образования произвольных связей.

* Недостатки: высокая сложность и жесткость схемы БД, сложность понимания и выполнения обработки информации.

**5. Реляционная модель данных.**

Реляционная модель данных - созданная Эдгаром Коддом

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы). Цели создания реляционной модели данных:

обеспечение более высокой степени независимости от данных;

создание прочного фундамента для решения семантических вопросов и проблем непротиворечивости и избыточности данных;

расширение языков управления данными за счёт включения операций над множествами.

Реляционная модель данных предусматривает структуру данных, обязательными объектами которой являются: отношение; атрибут; домен; кортеж; степень; кардинальность; первичный ключ.

Отношение - это плоская (двумерная) таблица, состоящая из столбцов и строк:

**Атрибут -** это поименованный столбец отношения.

Домен - это набор допустимых значений для одного или нескольких атрибутов.

Кортеж - это строка отношения.

Степень определяется количеством атрибутов, которое оно содержит

Кардинальность - это количество кортежей, которое содержит отношение.

Первичный ключ - это уникальный идентификатор для таблицы.

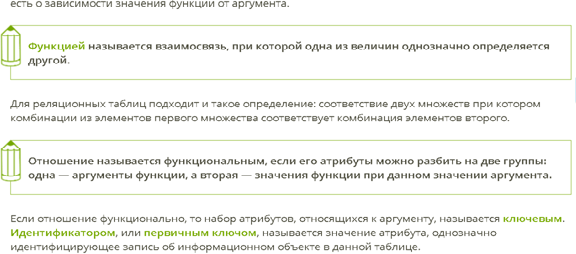
Достоинства реляционной модели данных:

1. **Наглядность, простота, гибкость** структуры данных.
2. Относительная простота практической реализации.

Недостатки:

1. Некоторая ограниченность и предопределенность набора атрибутов и доменов, что можно устранить при переходе к объектно-реляционной модели.

**6. Понятие функционального отношения, 1-я нормальная форма**



Используемые термины

Атрибут — свойство некоторой сущности. Часто называется полем таблицы.

Домен атрибута — множество допустимых значений, которые может принимать атрибут.

Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).

Отношение — конечное множество кортежей (таблица).

Схема отношения — конечное множество атрибутов, определяющих некоторую сущность. Иными словами, это структура таблицы, состоящей из конкретного набора полей.

Проекция — отношение, полученное из заданного путём удаления и (или) перестановки некоторых атрибутов.

Функциональная зависимость между атрибутами (множествами атрибутов) X и Y означает, что для любого допустимого набора кортежей в данном отношении: если два кортежа совпадают по значению X, то они совпадают по значению Y.

Первая нормальная форма--- Отношение находится в первой нормальной форме (сокращённо 1НФ), если все его атрибуты атомарны, то есть если ни один из его атрибутов нельзя разделить на более простые атрибуты, которые соответствуют каким-то другим свойствам описываемой сущности

**7. Нормализация отношений, 2-я нормальная форма**

Нормализация представляет процесс разделения данных по отдельным связанным таблицам.

Нормализация имеет своей целью избавиться от избыточности в отношениях и модифицировать их структуру таким образом, чтобы процесс работы с ними не был обременён различными посторонними сложностями. При игнорировании такого подхода эффективность проектирования стремительно снижается, что вкупе с прочими подобными вольностями может привести к критическим последствиям.

--Как правило, нормализация преимущественно применяется при восходящем подходе проектировании базы данных, то есть когда мы все атрибуты, которые надо сохранить в бд, группируем по сущностям, для которых затем создаются таблицы. Однако при нисходящем подходе, когда вначале выявляются сущности, а затем их атрибуты и связи между ними, нормализация также может применяться, например, для проверки корректности спроектированных таблиц.

--В ненормализованной форме таблица может хранить информацию о двух и более сущностях. Также она может содержать повторяющиеся столбцы. Также столбцы могут хранить повторяющиеся значения. В нормализованной же форме каждая таблица хранит информацию только об одной сущности.

--Вторая нормальная форма (2NF) предполагает, что каждый столбец, не являющийся ключом, должен зависеть от первичного ключа.

Требования второй нормальной формы (2NF)Чтобы база данных находилась во второй нормальной форме (2NF), необходимо чтобы ее таблицы удовлетворяли следующим требованиям: 1 Таблица должна находиться в первой нормальной форме 2Таблица должна иметь ключ 3 Все неключевые столбцы таблицы должны зависеть от полного ключа (в случае если он составной) Ключ – это столбец или набор столбцов, по которым гарантировано можно отличить строки друг от друга, т.е. ключ идентифицирует каждую строку таблицы.

**8. 3-я нормальная форма.**

Переменная отношения находится в 3НФ, когда она находится во 2НФ, и отсутствуют транзитивные функциональные зависимости не ключевых атрибутов от ключевых, то есть любой столбец таблицы должен зависеть только от ключевого столбца. Она расширяет две предыдущие нормальные формы, неся в себе два правила: – таблица должна соответствовать второй нормальной форме; – все столбцы, не входящие в полный первичный ключ, должны зависеть от него и не должны зависеть друг от друга

Чтобы нормализовать базу данных до третьей нормальной формы, необходимо сделать так, чтобы в таблицах отсутствовали не ключевые столбцы, которые зависят от других не ключевых столбцов.

Иными словами, не ключевые столбцы не должны пытаться играть роль ключа в таблице, т.е. они действительно должны быть не ключевыми столбцами, такие столбцы не дают возможности получить данные из других столбцов, они дают возможность посмотреть на информацию, которая в них содержится, так как в этом их назначение. Главное правило третьей нормальной форме (3NF) звучит следующим образом:

Таблица должна содержать правильные не ключевые столбцы

Чтобы определить, находится ли эта таблица в третьей нормальной форме, мы должны проверить все не ключевые столбцы, каждый из них должен зависеть только от первичного ключа, и никаким образом к другим не ключевым столбцам он не должен относиться.

**9. 4-я нормальная форма.**

Четвертая нормальная форма (4NF) применяется для устранения многозначных зависимостей (multivalued dependencies) - таких зависимостей, где столбец с первичным ключом имеет связь один-ко-многим со столбцом, который не является ключом. Эта нормальная форма устраняет некорректные отношения многие-ко-многим. Обычно во всех источниках приводится два основных глобальных преимущества:

1.Устранение аномалий 2. Повышение производительности

Если с устранением аномалий все ясно, т.е. в полностью нормализованной базе данных их не будет и это хорошо, то с повышением производительности не все так однозначно.

Да, нормализация повышает производительность, но только где-то до 3 нормальной формы. Начиная с 4 нормальной формы, производительность увеличиваться не будет, более того, с каждой новой формой производительность будет значительно снижаться, не говоря уже о том, что с нормализованной базой данных до 5 или 6 нормальной формы будет крайне сложно и неудобно работать и сопровождать ее, ведь с каждой новой формой мы значительно увеличиваем количество таблиц в базе данных.

Поэтому процесс нормализации не является строго обязательным, т.е. не нужно нормализовать базу данных, только для того чтобы она была нормализована.

В процессе проектирования базы данных необходимо следовать здравому смыслу и найти баланс между отсутствием аномалий и приемлемой производительностью.

Полностью нормализованная база данных – это плохая база данных.

Хорошая база данных – это база, которая достаточно нормализована, чтобы не создавать аномалии для пользователей этой базы данных, и в то же время она имеет хорошую производительность.

**10.** **Система управления базой данных (СУБД), назначение, функции.**

Система управления базами данных (СУБД) – это комплекс программно-языковых средств, позволяющих создать базы данных и управлять данными. Иными словами, СУБД — это набор программ, позволяющий организовывать, контролировать и администрировать базы данных. Большинство сайтов не могут функционировать без базы данных, поэтому СУБД используется практически повсеместно.

Основные функции СУБД:   
● Создание баз данных, изменение, удаление и объединение их по определённым признакам.   
● Хранение данных, в том числе больших массивов, в структурированном виде и нужном формате.   
● Защита данных от взлома и нежелательных изменений при помощи распределённого доступа: когда разным группам пользователей доступны разный объём и сегменты данных.   
● Выгрузка и сортировка данных по заданным фильтрам при помощи SQL-запросов.   
● Поддержка целостности баз данных, резервное копирование и восстановление после сбоёв.

**11. СУБД: типы, состав.**

СУБД (система управления базами данных) — это программное обеспечение, предназначенное для создания, управления и обработки баз данных. Существуют различные типы СУБД, которые могут быть классифицированы по различным критериям. Одним из таких критериев является модель данных, которая определяет, как организованы данные в базе данных. Вот некоторые распространенные типы СУБД:  
  
1. Реляционные СУБД (RDBMS): Реляционные СУБД основаны на модели данных, известной как реляционная модель. В этой модели данные представлены в виде таблиц (отношений), состоящих из строк (кортежей) и столбцов (атрибутов). Каждая таблица представляет собой сущность, а отношения между таблицами устанавливаются с помощью ключей. Примеры реляционных СУБД включают MySQL, Oracle, Microsoft SQL Server.  
  
2. Иерархические СУБД (HDBMS): Иерархические СУБД организуют данные в виде иерархической структуры, состоящей из уровней, где каждый уровень содержит связанные между собой записи. В этой модели данные представлены в виде дерева, где каждая запись может иметь одного или несколько родителей и ноль или несколько детей. Примеры иерархических СУБД включают IBM's Information Management System (IMS).  
  
3. Сетевые СУБД (NDBMS): Сетевые СУБД организуют данные в сетевую структуру, где каждая запись может иметь несколько связей с другими записями. Это позволяет представлять сложные отношения между данными. Однако сетевые СУБД менее распространены и использовались главным образом в прошлом. Примеры сетевых СУБД включают Integrated Data Store (IDS) и Integrated Database Management System (IDMS).  
  
4. Объектно-ориентированные СУБД (OODBMS): Объектно-ориентированные СУБД предназначены для работы с объектно-ориентированными моделями данных. Они позволяют сохранять объекты, их свойства и взаимосвязи между ними, что обеспечивает более натуральное представление данных. Примеры объектно-ориентированных СУБД включают MongoDB и Apache Cassandra.  
  
Каждый тип СУБД имеет свои преимущества и недостатки, и выбор определенного типа зависит от требований и характеристик конкретного проекта или приложения.

Структура СУБД

**Ядро**. Это основа всей системы, которая отвечает за хранение и обработку баз данных. В ядре фиксируются все изменения: добавление, удаление или исправление целых баз и отдельных ячеек.

● **Процессор, или компилятор.** Обрабатывает запросы к базам данных на внутренних языках и SQL, преобразуя их в нужные команды и передавая результаты.   
● **Программные средства, или утилиты**. С их помощью пользователи вводят запросы, а администраторы баз данных настраивают доступ и другие параметры.   
● **Базы данных.** То, где хранятся данные, организованные особым образом, иногда — в зашифрованном виде. Если это реляционные базы, то данные представлены в виде таблиц, связанных с друг другом. Если объектные — в виде объектов: блоков информации с определёнными свойствами и параметрами.

**12. Модель сущность-связь, приложения для проектирования ER-моделей**

Модель «сущность – связь» (ER-модель) была предложена П. Ченом в 1976 году [5] как средство «ручного» проектирования баз данных. Для модели «сущность – связь» базовыми являются понятия: сущность;− связь;− атрибут. Модель «сущность – связь» основана на диаграммной технике. Для представления различных аспектов структуры данных (объектов, свойств объектов, связей между объектами, свойств связей и других) используются графические средства. Сущность – это абстрактный объект, который в конкретном контексте имеет независимое существование. Сущность имеет семантически значимое имя, как правило, имя существительное, например: «Студент», «Преподаватель», «Сотрудник», «Отдел», «Компьютер, «Книга» и т. Д Связь – это ассоциация сущностей или отношение между сущностями Связь имеет семантически значимое имя, как правило, в форме глагола, например: «Преподаватель» «Ведет» «Дисциплину», «Отдел» «Включает» «Сотрудника» и т. Д Связи обладают свойствами: вид (категориальная, идентифицирующая, неидентифицирующая);− степень (унарная, бинарная, тернарная, N-арная);− кардинальность;− внешний ключ сущности.− Атрибут – это свойство сущности или связи. Атрибутам как спецификаторам свойств сущностей или связей присваиваются семантически значимые имена, как правило, в форме существительного. Атрибутами сущности «Сотрудник» являются: табельный номер, фамилия, имя, отчество, дата рождения и другие персональные характеристики.

**13. Язык Jet-SQL: типы данных.**

: Access 2013, Office 2013

BINARY 1 байт на каждый символ В поле этого типа может храниться любой тип данных.

BIT 1 байт Значения "Да" и "Нет" и поля, содержащие только одно из двух значений.

TINYINT 1 байт Целое число от 0 до 255.

MONEY 8 байтов Масштаб целого числа от 922 337 203 685 477,5808 до 922 337 203 685 477,5807.

DATETIME (см. DOUBLE) 8 байтов Значение даты и времени между годами от 100 до 9999.

REAL 4 байта Значение с плавающей запятой одинарной точности с диапазоном от –3,402823E38 до –1,401298E-45 для отрицательных значений, от 1,401298E-45 до 3,402823E38 для положительных значений, а также 0. FLOAT 8 байтов Значение с плавающей запятой двойной точности с диапазоном от –1,79769313486232E308 до –4,94065645841247E-324 для отрицательных значений, от 4,94065645841247E-324 до 1,79769313486232E308 для положительных значений, а также 0.

SMALLINT 2 байта Короткое целое число в диапазоне от –32 768 до 32 767. INTEGER 4 байта Длинное целое число в диапазоне от –2 147 483 648 до 2 147 483 647.

DECIMAL 17 байтов Тип точных числовых данных, содержащих значения от 1028 – 1 до –1028 – 1

TEXT 2 байта на каждый символ Значение от 0 до 2,14 гигабайт.

IMAGE В соответствии с требованием Значение от 0 до 2,14 гигабайт. Используется для объектов OLE.

CHARACTER 2 байта на каждый символ Значение от 0 до 255 символов.

**14. Язык Jet-SQL: создание, удаление таблицы.**

Чтобы создать новую таблицу в Access с помощью Access SQL, необходимо присвоить таблице имя, имя поля и определить тип данных, которые будут содержаться в этих полях. Используйте инструкцию [**CREATE TABLE**](https://learn.microsoft.com/ru-ru/office/client-developer/access/desktop-database-reference/create-table-statement-microsoft-access-sql) для определения таблицы в SQL

Так мы сначала указываем название новой таблицы, затем в скобках перечисляем столбцы, которые будем создавать, причем их названия не могут повторяться в пределах одной таблицы. После названий столбцов указывается тип данных для каждого поля (**CHAR (10)**), затем отмечаем может ли поле содержать пустые значения (**NULL**или **NOT NULL**), а также нужно указать поле, которое будет первичным ключом (**Primary key**).

Чтобы удалить таблицу из базы данных, используйте инструкцию [**DROP TABLE**](https://learn.microsoft.com/ru-ru/office/client-developer/access/desktop-database-reference/drop-statement-microsoft-access-sql) .

**15. Язык Jet-SQL: создание, удаление ограничений.**

* Ограничения — это правила, применяемой на столбцы данных в таблицах. Они используются для ограничения типа данных, которые могут добавлены в таблицу. Это гарантирует точность и достоверность данных в базе данных. Ограничения могут быть либо на уровне столбца или уровня таблицы. Ограничения на уровне столбца применяются только к одной колонке, в то время как ограничения уровня таблицы применяются ко всей таблице.
* [Ограничение NOT NULL](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-konstanta-not-null/) — гарантирует , что столбец не может иметь нулевое значение.
* [Ограничение DEFAULT](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-konstanta-default/) — обеспечивает значение по умолчанию для столбца, если он не указан.
* [Ограничение UNIQUE](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-konstanta-unique/) — гарантирует , что все значения в столбце различны.
* [PRIMARY Key](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-primary-key-pervichnyj-klyuch/) — идентифицирует уникальность для каждой строки/записи в таблице базы данных.
* [FOREIGN Key](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-foreign-key-vneshnij-klyuch/) — идентифицирует уникальность строки/записи в данной таблицы базы данных.
* [Ограничение CHECK](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-ogranichenie-check/) — проверочное ограничение гарантирует, что все значения в столбце удовлетворяют определенным условиям.
* [INDEX](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-konstanta-index/) — используется для очень быстрого создания и извлечения данных из базы данных.

Ограничения могут быть указаны, если таблица создается с помощью заявление [CREATE TABLE](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-sozdat-tablicu/), или вы можете использовать оператор [ALTER TABLE](https://andreyex.ru/bazy-dannyx/uchebnoe-posobie-po-sql/sql-komanda-alter-table/) для создания ограничений даже после того, как таблица была создана. Любое ограничение, которое вы определили может быть удалено с помощью команды ALTER TABLE, с опцией DROP CONSTRAINT.

ALTER TABLE tblCustomers DROP CONSTRAINT PK\_tblCustomers

**16. Язык Jet-SQL: Редактирование структуры таблицы.**

После создания и заполнения таблицы может потребоваться изменить ее структуру. Для этого используйте инструкцию [**ALTER TABLE**](https://learn.microsoft.com/ru-ru/office/vba/api/overview/access) . Имейте в виду, что изменение структуры существующей таблицы может привести к потере некоторых данных.  С помощью инструкции **ALTER TABLE** можно добавить, удалить или изменить столбец (или поле), а также добавить или удалить ограничение. Чтобы добавить поле с инструкцией **ALTER TABLE** , используйте предложение **ADD COLUMN** с именем поля, его типом данных и размером типа данных, если это необходимо. ALTER TABLE имя таблицы ADD COLUMN Address TEXT(30) Чтобы изменить тип данных или размер поля, используйте предложение **ALTER COLUMN** с именем поля, требуемым типом данных и требуемым размером типа данных, если это необходимо. ALTER TABLE tblCustomers ALTER COLUMN Address TEXT(40) Если вы хотите изменить имя поля, необходимо удалить поле, а затем повторно создать его. Чтобы удалить поле, используйте предложение **DROP COLUMN** только с именем поля. ALTER TABLE tblCustomers DROP COLUMN Address Обратите внимание, что при использовании этого метода будут устранены существующие данные для поля.

**17. Понятие индекса, создание индексов средствами языка Jet-SQL.**

*Индексом* называется особый справочник значения поля (или нескольких полей одновременно), где в упорядоченном по значениям полей виде хранятся ссылки на записи.

Чтобы создать индекс в существующей таблице, используйте команду CREATE INDEX. Команда CREATE INDEX имеет следующий синтаксис:

CREATE INDEX index\_name  
 ON table (field1 [DESC][, field2 [DESC], ...])  
Требуются только команды CREATE INDEX, имя индекса, аргумент ON, имя таблицы с полями, которые требуется индексировать, и список полей, которые нужно включить в индекс.

Индекс является самостоятельным объектом базы данных, поэтому имеет имя. В скобках можно указывать несколько полей через запятую. Порядок задаётся ключевыми словами ASC (по возрастанию) и DESC (по убыванию), порядок определяется для каждого поля, а не для всех сразу. В индексах существует ограничение: поле не может быть включено в индекс дважды - поэтому для названия книги мы создали два отдельных индекса. Избыток индексов может привести к снижению производительности работы с базой данных.

Удалить индекс также не сложно

| DROP INDEX index\_name |
| --- |

**18. Язык Jet-SQL: Команды модификации данных**

оператор **INSERT** используется для вставки (добавления) строк в таблицу базы данных. Добавление можно осуществить несколькими способами:- добавить одну полную строку- добавить часть строки- добавить результаты запроса. Выполнять запрос на вставку можно любое количество раз. Каждый раз в таблице будет создаваться новая запись, даже если данные не поменялись

*INSERT* (INSERT INTO <имя\_таблицы> [(<имя\_столбца>,...)]

VALUES (<значение>,...) – вставка данных в таблицу;

Заметим, что при вставке строки в таблицу проверяются все ограничения, наложенные на данную таблицу. Это могут быть ограничения первичного ключа или уникального индекса, проверочные ограничения типа **CHECK**, ограничения ссылочной целостности. В случае нарушения какого-либо ограничения вставка строки будет отклонена.

UPDATE  можно выполнять безопасно любое количество раз. После первого запроса дальнейшие его вызовы не сделают никаких изменений.

За одну операцию можно обновить несколько полей. Для этого достаточно перечислить каждое присваивание через запятую в части SET. Порядок, в котором изменяются поля, — не важен. Часть WHERE можно вообще не указывать:

*UPDATE* (UPDATE <имя\_таблицы> SET <имя\_столбца>=<значение>,...

         [WHERE <условие>]– обновление данных в таблице БД;

DELETE с его помощью можно удалить данные из таблицы: Этот запрос работает практически так же, как и UPDATE. Только здесь ничего не надо указывать после DELETE.

*DELETE* (DELETE FROM<имя\_таблицы>

**19. Язык Jet-SQL: оператор SELECT: конструкции раздела SELECT.**

Для извлечения записей из таблиц в SQLопределён оператор SELECT. С помощью этой команды осуществляется не только операция реляционной алгебры *«выборка»* (горизонтальное подмножество), но и *предварительное соединение (join)*двух и более таблиц. Это наиболее сложное и мощное средство SQL. *Полный синтаксис оператора SELECT имеет следующий вид:*

SELECT [ALL/DISTINCT] <список\_выбора>

FROM <имя\_таблицы>,..

[WHERE <условие> ]

[GROUP BY <имя\_столбца>,..]

[HAVING <условие>]

[ORDER BY <имя\_столбца>[ASC/DESC],…]

Данный оператор всегда начинается с ключевого слова SELECT и состоит из 6 разделов (предложений): *SELECT, FROM, WHERE, GROUPBY, HAVING и ORDERBY.* Только два из них *– SELECT и FROM – являются обязательными*

**20. Язык Jet-SQL: оператор SELECT: раздел WHERE (операции BETWEEN, LIKE).**

Оператор BETWEEN выбирает значения в заданном диапазоне. Эти значения могут быть числами, текстом или датами.Оператор BETWEEN является инклюзивным: включаются начальные и конечные значения.

Синтаксис BETWEEN

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name*BETWEEN *value1* AND *value2;*

Оператор Like используется в предложении WHERE для поиска заданного шаблона в столбце.

В сочетании с оператором LIKE используются два подстановочных знака:

* % - Знак процента представляет ноль, один или несколько символов
* ? - Знак вопросительный знак представляет один символ

SELECT *column1, column2, ...*  
FROM *table\_name*  
WHERE *columnN* LIKE *pattern*;

WHERE CustomerName LIKE 'a%' Находит любые значения, которые начинаются с "a"

WHERE CustomerName LIKE '%a' Находит любые значения, которые заканчиваются "a"

WHERE CustomerName LIKE '%or%'   
Находит любые значения, которые имеют значение "or" в любом положении

WHERE CustomerName LIKE '?r%' Находит все значения, которые имеют значение "r" во второй позиции

WHERE CustomerName LIKE 'a?%?%' Находит любые значения, которые начинаются с "a" и длиной не менее 3 символов

WHERE ContactName LIKE 'a%o' Находит любые значения, которые начинаются с "a" и заканчиваются на "o"

**21. Язык Jet-SQL: оператор SELECT: раздел WHERE (операции IN, NULL).**

Оператор IN позволяет указать несколько значений в предложении WHERE.Оператор IN- это сокращенное выражение для нескольких условий OR. Синтаксис IN

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name* IN (*value1*,*value2*, ...);

Следующая инструкция SQL выбирает всех клиентов, которые находятся в "Germany", "France" или "UK":SELECT \* FROM Customers  
WHERE Country IN ('Germany', 'France', 'UK');

## [IS NULL](https://sql-academy.org/ru/guide/is-null-between-in-operators#is-null)

## Оператор IS NULL позволяет узнать равно ли проверяемое значение NULL, т.е. пустое ли значение.

Для примера выведем всех преподавателей, у кого отсутствует отчество: SELECT \* FROM Teacher

WHERE отчество IS NULL; middle\_name(отчество)

Для использования отрицания, то есть, если мы хотим найти все записи, где поле не равно NULL, мы должны использовать следующий синтаксис: SELECT \* FROM Teacher

WHERE middle\_name IS NOT NULL;

**22. Язык Jet-SQL: агрегатные функции.**

С помощью агрегатных функций SQL вы можете определить различные статистические данные для наборов значений

**Avg(***expr***)** Вычисляет среднее арифметическое для набора значений, содержащихся в указанном поле в запросе . Функция **Avg** не включает в вычисление поля **NULL** .

**Count(***expr***)** Вычисляет количество записей, возвращенных запросом. Функция **Count** не учитывает записи, содержащие поля **Null**, если *expr* не является подстановочным знаком в виде звездочки (\*). Если используется звездочка, функция **Count** вычисляет общее количество записей, включая записи, содержащие поля **Null**. **Count (** \* **)** значительно быстрее, чем **Count (** [ *Имя столбца* ] **)**. Звездочку не нужно заключать в кавычки (' ').

**First**( *expr* ) **Last**( *expr* ) Возвращает значение поля из первой или последней записи в результирующем наборе, возвращенном запросом.

Они просто возвращают значение указанного поля в первой или последней записи, соответственно, результирующий набор, возвращенный запросом.

**Min(***expr***)** **Max(***expr***)** Возвращает минимальный или максимальный набор значений, содержащихся в указанном поле запроса. Используйте **min** и **max** , чтобы определить наименьшие и наибольшие значения в поле на основе указанной агрегации или группировки. Например, эти функции можно использовать для возврата наименьших и самых высоких затрат на транспортировку

**Sum(***expr***)** Возвращает сумму набора значений, содержащихся в указанном поле запроса. Функция **Sum** суммирует значения в поле. Например, можно использовать функцию **Sum** для определения общей стоимости расходов на доставку.Функция **Sum** игнорирует записи, содержащие поля **NULL** .

**Var(***expr***)** **VarP(***expr***)** Возвращает оценки дисперсии для совокупности или выборки, представленной в виде набора значений, содержащихся в указанном поле в запросе. Функция **VarP** оценивает популяцию, а функция **Var** — выборку совокупности.

Если базовый запрос содержит менее двух записей, функции **Var** и **VarP** возвращают значение **NULL** , указывающее, что отклонение не может быть вычислено.

**23. Язык Jet-SQL: раздел FROM, объединения таблиц.**

В системе реляционных баз данных, такой как Access, часто требуется извлекать сведения из нескольких таблиц за раз. Это можно сделать с помощью инструкции SQL [**JOIN**](https://learn.microsoft.com/ru-ru/office/vba/access/concepts/structured-query-language/join-microsoft-access-sql-reserved-word)

[**Inner JOIN**](https://learn.microsoft.com/ru-ru/office/vba/api/overview/access), также известный как равное соединение, является наиболее часто используемым типом соединения. Это соединение используется для получения строк из двух или более таблиц путем сопоставления значения поля, которое является общим для таблиц. Поля, к которых вы присоединяете, должны иметь аналогичные типы данных Чтобы создать **инструкцию INNER JOIN** , используйте **ключевые слова INNER JOIN** в предложении [**FROM**](https://learn.microsoft.com/ru-ru/office/vba/access/concepts/structured-query-language/from-clause-microsoft-access-sql)

SELECT поля\_таблиц

FROM таблица\_1

[INNER] | [[LEFT | RIGHT | FULL][OUTER]] JOIN таблица\_2

ON условие\_соединения

[**OUTER JOIN**](https://learn.microsoft.com/ru-ru/office/vba/api/overview/access) используется для получения записей из нескольких таблиц при сохранении записей из одной из таблиц, даже если в другой таблице нет соответствующей записи. Ядро СУБД Access поддерживает два типа **OUTER JOIN** : **LEFT OUTER JOIN** и **RIGHT OUTER JOIN**.

Представьте себе две таблицы, которые находятся рядом друг с другом: таблицу слева и таблицу справа. **LEFT OUTER JOIN** выбирает все строки в правой таблице, соответствующие условиям реляционного сравнения, а также выбирает все строки из левой таблицы, даже если в правой таблице не существует совпадений. **RIGHT OUTER JOIN** — это просто обратная часть **левого внешнего соединения**; вместо этого сохраняются все строки в правой таблице.

**24. Язык Jet-SQL: раздел FROM, использование псевдонимов.**

Для обращения к таблице в инструкции SELECT можно использовать ее псевдоним, указав его в предложении FROM. Псевдоним таблицы — это имя, назначенное источнику данных в запросе при использовании выражения в качестве источника данных или для упрощения ввода и прочтения инструкции SQL. Такая возможность полезна, если имя источника данных слишком длинное или его трудно вводить, особенно если есть несколько полей с одинаковыми именами из разных таблиц.

Например, если необходимо выбрать данные из двух полей с именем ID, одно из которых содержится в таблице tblCustomer, а другое — в таблице tblOrder, предложение SELECT может выглядеть следующим образом:

SELECT [tblCustomer].[ID], [tblOrder].[ID]

Используя псевдонимы таблиц в предложении FROM, можно упростить ввод запроса. Предложение FROM с псевдонимами таблиц может выглядеть следующим образом:

FROM [tblCustomer] AS [C], [tblOrder] AS [O]

Эти псевдонимы таблиц можно использовать в предложении SELECT следующим образом: SELECT [C].[ID], [O].[ID]

**25.** **Язык Jet-SQL: группировка.**

Группировка данных позволяет разделить все данные на логические наборы, благодаря чему становится возможным выполнение статистических вычислений отдельно в каждой группе.

Оператор SQL GROUP BY служит для распределения строк - результата запроса - по группам, в которых значения некоторого столбца, по которому происходит группировка, являются одинаковыми. Группировку можно производить как по одному столбцу, так и по нескольким.

Оператор GROUP BY имеет следующий синтаксис:

SELECT ИМЕНА\_СТОЛБЦОВ

FROM ИМЯ\_ТАБЛИЦЫ

[WHERE УСЛОВИЕ]

GROUP BY ИМЕНА\_СТОЛБЦОВ

**26. Язык Jet-SQL: сортировка.**

Суть процесса сортировки заключается к приведению последовательности к определенному порядку.

Для сортировки данных, получаемых в результате запроса в языке SQL используется ключевое слово **ORDER BY**.

База данных (далее – БД) сортирует данные по указанному параметру по убыванию, либо по возрастанию, в зависимости от условия указанного в запросе (ASC / DESC – соответственно).

Мы также можем использовать более одной колонки при использовании элемента **ORDER BY**.

Данный запрос имеет следующий общий вид:

**SELECT колонка1, колонка2 ... колонкаN**

**FROM имя\_таблицы**

**[WHERE условие]**

**[ORDER BY колонка1, колонка2 ... колонкаN] [ASC | DESC];**

Порядок сортировки по умолчанию возрастает (от A до Z, от 0 до 9

ASC

Чтобы выполнить сортировку в порядке убывания (от Z до A, от 9 до 0), добавьте зарезервированное слово DESC в конец каждого поля, которое нужно отсортировать в порядке убывания.

**27.Язык Jet-SQL: подзапросы.**

Подзапрос – это запрос на выборку данных, вложенный в другой запрос. Простые подзапросы характеризуются тем, что они формально никак не связаны с содержащими их внешними запросами. Это позволяет сначала выполнить подзапрос, результат которого затем используется для выполнения внешнего запроса.

Простые подзапросы можно разделить на три категории:

1. Подзапросы, возвращающие единственное значение. Тип возвращаемой подзапросом таблицы определяет, как можно ее использовать и какие операторы можно применять. Сам подзапрос действует как временная таблица, областью видимости для которой является выражение.

В качестве примера используем подзапрос для связи двух таблиц.

SELECT Name

FROM spr\_Publish

WHERE ID\_City=(SELECT ID FROM spr\_City where Name='Минск'); Выбрав с помощю подзапроса первичный ключ из справочника spr\_City по внешнему ключу ID\_City таблицы spr\_Publish, выбираем нужные данные

Подзапросы, возвращающие список значений из одного столбца таблицы. Подзапрос может возвращать и несколько записей. Чтобы в этом случае в условии WHERE внешнего запроса можно было использовать операторы сравнения, требующие единственного значения, используются кванторы, такие как ALL (все) и SOME или ANY (некоторый).

Подзапрос в качестве источника выборки Подзапрос можно вставлять в оператор FROM. При этом подзапрос будет выступать в качестве источника данных. В таком запросе таблице, возвращаемой подзапросом в операторе FROM, можно присвоить псевдоним, а во внешнем запросе использовать данные этой таблицы, используя псевдоним.

SELECT (Tab.Name+ Tab.Tel) as Визитка FROM(SELECT \* FROM spr\_Author WHERE (Tel like "\*(29)\*")) Tab;

**28 .Язык Jet-SQL: соотнесённые подзапросы.**

Соотнесенным называется подзапрос, который ссылается на данные внешнего запроса

Подзапрос также можно использовать в секции Where. Здесь в отличие от предыдущих вариантов связанный (коррелированный) подзапрос выполняется каждый раз для каждой строки основного запроса.

Например, следующий запрос использует связанный подзапрос для подсчета количества публикаций у каждого автора. Затем основной запрос выбирает тех авторов, у которых больше одной публикации.

SELECT a.Name

FROM spr\_Author a

WHERE 1 <(select count(\*)from [tab\_Books]b where b.ID\_Author=a.id

Ссылка на a.Id в самом конце подзапроса – это то, что делает этот подзапрос связанным. Чтобы подзапрос мог выполняться, основной запрос должен поставлять значения для a.Id. В данном случае основной запрос извлекает из таблицы spr\_Author все строки и выполняет по одному подзапросу для всех авторов, передавая в него соответствующий Id автора при каждом выполнении. Если подзапрос возвращает значение большее одного, условие фильтрации выполняется и строка добавляется в результирующий набор.

В Jet SQL, подзапросы могут использоваться для создания более сложных запросов путем вложения одного запроса внутри другого. Ниже приведены некоторые примеры соотнесенных подзапросов в Jet SQL:  
  
1. Простой подзапрос с использованием ключевого слова "IN":  
```sql  
SELECT column1, column2  
FROM table1  
WHERE column1 IN (SELECT column1 FROM table2);  
```  
Этот запрос выбирает значения column1 и column2 из table1, где column1 присутствует в результате подзапроса, который выбирает column1 из table2.  
  
2. Подзапрос с оператором сравнения:  
```sql  
SELECT column1, column2  
FROM table1  
WHERE column1 > (SELECT AVG(column1) FROM table2);  
```  
Этот запрос выбирает значения column1 и column2 из table1, где column1 больше среднего значения column1 из table2, полученного в подзапросе.  
  
3. Подзапрос с использованием связанного оператора EXISTS:  
```sql  
SELECT column1, column2  
FROM table1  
WHERE EXISTS (SELECT column1 FROM table2 WHERE table2.column1 = table1.column1);  
```  
Этот запрос выбирает значения column1 и column2 из table1, где существует соответствующая запись в table2 с одинаковым значением column1.  
  
4. Подзапрос с использованием оператора ANY/ALL:  
```sql  
SELECT column1, column2  
FROM table1  
WHERE column1 > ANY (SELECT column1 FROM table2);  
```  
Этот запрос выбирает значения column1 и column2 из table1, где column1 больше любого значения column1 из table2.

**29. Язык Jet-SQL: объединение UNION**

В Jet SQL (структурированном запросовом языке) объединение (union) используется для комбинирования результатов двух или более запросов в один результат. Результирующий набор данных будет содержать все уникальные строки, полученные из каждого запроса. Вот пример использования объединения union в Jet SQL:  
  
```sql  
SELECT column1, column2 FROM table1  
UNION  
SELECT column1, column2 FROM table2;  
```  
  
В этом примере `SELECT column1, column2 FROM table1` представляет первый запрос, а `SELECT column1, column2 FROM table2` - второй запрос. Оба запроса должны иметь одинаковое число столбцов и соответствующие типы данных для каждого столбца.  
  
Результатом объединения union будет результирующий набор данных, содержащий все уникальные строки, полученные из обоих запросов. Если есть повторяющиеся строки, они будут удалены из результата.  
  
Обратите внимание, что при использовании объединения union, порядок строк в результирующем наборе будет определен внутренней логикой Jet SQL и не обязательно будет соответствовать порядку строк в исходных таблицах.  
  
Также в Jet SQL существует альтернативный оператор объединения union all, который объединяет результаты запросов, включая все строки, включая повторяющиеся строки. Для использования union all просто замените оператор UNION на UNION ALL в запросе.  
  
```sql  
SELECT column1, column2 FROM table1  
UNION ALL  
SELECT column1, column2 FROM table2;  
```  
  
Таким образом, результатом объединения union all будет результирующий набор данных, содержащий все строки из обоих запросов, включая повторяющиеся строки.

**30. СУБД MS Access: состав, возможности.**

Программа MS Access представляет собой систему управления реляционными базами данных и входит в пакет MS Office фирмы Microsoft. C помощью этой программы пользователь может решать следующие задачи

1) организация данных.

Эта задача предполагает возможность создания таблиц и предоставление пользователю возможности работать с данными, хранящимися в них;

2) ввод и редактирование данных. MS Access позволяет работать с данными не только в режиме таблицы, но и позволяет пользователю разработать и использовать удобные формы для ввода, редактирования и просмотра данных. Это удобно при большом объеме таблиц, особенно когда она содержит большое количество полей;

3) связывание таблиц. Означает возможность создания запросов, форм и отчетов для одновременного отображения сведений из нескольких таблиц;

4) отбор данных. Данная СУБД предоставляет пользователю широкие возможности отбора необходимых данных с помощью запросов или фильтрации;

5) представление данных. Эта функция обеспечивает создание отчетов, представляющих всевозможные итоговые данные таблиц в удобном и наглядном виде, которые можно просматривать, печатать или публиковать в Интернет. В конце концов, именно отчеты являются целью любой системы учета данных

База данных, создаваемая и обрабатываемая с помощью MS Access представляет собой один файл с расширением .mdb. Этот файл может включать в себя объекты нескольких типов: таблицы, запросы, формы, отчеты, страницы, макросы и модули.

СУБД MS Access имеет три основных рабочих режима:

• начальный режим. В этом режиме Access функционирует после запуска, до того как будет открыта какая-либо база данных;

• режим конструктора. Этот режим предназначен для создания и изменения структуры таблиц и запросов, разработки форм и страниц доступа к данным и форматирования отчетов для печати;

• режим выполнения или просмотра. В этом режиме пользователь работает с данными таблиц, форм и отчетов, открытых в отдельных окнах.

**31. СУБД MS SQL Server: состав, возможности.**

Microsoft SQL Server (MS SQL Server) — это реляционная система управления базами данных (СУБД), разработанная компанией Microsoft. MS SQL Server предлагает широкий набор функций и возможностей для эффективного хранения, управления и обработки данных. Вот некоторые из основных компонентов и возможностей MS SQL Server:  
  
1. Движок базы данных: MS SQL Server использует свой собственный движок базы данных для обработки и хранения данных. Этот движок поддерживает ACID (атомарность, согласованность, изолированность, долговечность) для обеспечения надежности и целостности данных.  
  
2. Язык запросов: MS SQL Server поддерживает Transact-SQL (T-SQL), расширение стандартного языка SQL, для написания запросов и управления базой данных. T-SQL предоставляет богатый набор операторов и функций для манипулирования данными.  
  
3. Управление данными: MS SQL Server предоставляет мощные инструменты для управления данными, включая создание, изменение и удаление таблиц, индексов, представлений, хранимых процедур, триггеров и других объектов базы данных.  
  
4. Транзакции и согласованность: MS SQL Server обеспечивает поддержку транзакций, позволяя выполнить группу операций как единое целое. Это гарантирует согласованность данных при обработке множественных изменений.  
  
5. Масштабируемость и производительность: MS SQL Server обладает возможностями для горизонтального и вертикального масштабирования, позволяющими обрабатывать большие объемы данных и обеспечивать высокую производительность. Включает такие функции, как партиционирование таблиц, кластеризацию индексов, оптимизацию запросов и другие.  
  
6. Безопасность: MS SQL Server предлагает многоуровневую систему безопасности для защиты данных. Это включает аутентификацию и авторизацию пользователей, управление доступом на уровне объектов, шифрование данных и аудит действий пользователей.  
  
7. Репликация и высокая доступность: MS SQL Server поддерживает функции репликации, которые позволяют создавать дублирующиеся копии данных для распределенных сред и обеспечивать отказоустойчивость. Также предоставляются возможности для настройки кластеров высокой

**32. Понятие транзакции, управление транзакциями средствами языка Transact-SQL.**

Транзакция в контексте базы данных представляет собой логическую операцию или группу операций, которые должны быть выполнены как единое целое. Транзакции обеспечивают атомарность, согласованность, изолированность и долговечность (ACID) данных в базе данных.  
  
Управление транзакциями средствами языка Transact-SQL (T-SQL), используемого в Microsoft SQL Server, осуществляется с помощью следующих операторов:  
  
1. BEGIN TRANSACTION: Этот оператор начинает новую транзакцию. Все операции, выполняемые после этого оператора, считаются частью транзакции до ее фиксации или отката.  
  
2. COMMIT: Этот оператор фиксирует текущую транзакцию, сохраняя все изменения в базе данных, сделанные в рамках этой транзакции. После фиксации транзакции изменения становятся постоянными и видимыми для других пользователей.  
  
3. ROLLBACK: Этот оператор отменяет текущую транзакцию и откатывает все изменения, сделанные в рамках этой транзакции. Откат возвращает базу данных к состоянию, предшествовавшему началу транзакции.  
  
4. SAVEPOINT: Этот оператор создает точку сохранения внутри текущей транзакции. Точка сохранения позволяет откатиться к определенной точке внутри транзакции, вместо полного отката.  
  
5. SET TRANSACTION ISOLATION LEVEL: Этот оператор устанавливает уровень изоляции для текущей транзакции. Уровень изоляции определяет степень видимости изменений, сделанных другими транзакциями, для текущей транзакции.  
  
Пример использования операторов управления транзакциями в T-SQL:  
  
```  
BEGIN TRANSACTION;  
  
-- Выполнение операций в рамках транзакции  
  
IF (/\* условие \*/)  
    COMMIT;  
ELSE  
    ROLLBACK;  
```  
В этом примере операторы BEGIN TRANSACTION и COMMIT/ROLLBACK определяют границы транзакции. В зависимости от условия, транзакция может быть либо зафиксирована (COMMIT), либо откачена (ROLLBACK).

**33. Язык  Transact-SQL: типы данных, доступ к объектам базы данных.**

В языке Transact-SQL (T-SQL), используемом в Microsoft SQL Server, доступны различные типы данных и операторы для работы с объектами базы данных. Вот некоторые из наиболее распространенных типов данных и операторов доступа к объектам базы данных в T-SQL:  
  
Типы данных:  
1. Числовые типы данных:  
   - INT: Целочисленный тип данных для хранения целых чисел.  
   - DECIMAL(p, s): Фиксированная точность и масштаб числового типа данных.  
   - FLOAT: Числовой тип данных с плавающей точкой.  
   - MONEY: Тип данных для хранения денежных значений.  
  
2. Строковые типы данных:  
   - VARCHAR(n): Строковый тип переменной длины с максимальной длиной n символов.  
   - NVARCHAR(n): Строковый тип Unicode переменной длины с максимальной длиной n символов.  
   - CHAR(n): Строковый тип фиксированной длины с длиной n символов.  
   - NCHAR(n): Строковый тип фиксированной длины Unicode с длиной n символов.  
  
3. Типы данных даты и времени:  
   - DATE: Тип данных для хранения даты.  
   - TIME: Тип данных для хранения времени.  
   - DATETIME: Тип данных для хранения даты и времени.  
   - TIMESTAMP: Автоматически генерируемый уникальный метка времени.  
  
4. Логический тип данных:  
   - BIT: Логический тип данных, который может принимать значения TRUE, FALSE или NULL.  
  
Операторы доступа к объектам базы данных:  
1. SELECT: Используется для извлечения данных из таблицы или представления.  
2. INSERT: Используется для добавления новых записей в таблицу.  
3. UPDATE: Используется для обновления существующих записей в таблице.  
4. DELETE: Используется для удаления записей из таблицы.  
5. CREATE: Используется для создания новых объектов базы данных, таких как таблицы, представления, хранимые процедуры и т.д.  
6. ALTER: Используется для изменения структуры или свойств существующих объектов базы данных.  
7. DROP: Используется для удаления существующих объектов базы данных.

**34. Язык  Transact-SQL: управляющие конструкции.**

Transact-SQL (T-SQL) - это расширение языка SQL, используемое для управления данными в Microsoft SQL Server и Azure SQL Database. В T-SQL есть несколько управляющих конструкций, которые позволяют управлять логикой выполнения запросов и процедур. Вот некоторые из них:  
  
1. Условные операторы:  
   - IF...ELSE: Позволяет выполнять блок кода, если заданное условие истинно, и другой блок кода, если условие ложно.  
   - CASE: Позволяет выполнять различные действия в зависимости от значения выражения или условий.  
  
2. Циклы:  
   - WHILE: Выполняет блок кода, пока заданное условие истинно.  
   - CURSOR: Используется для обхода результатов запроса по одной строке за раз.  
  
3. Итерации:  
   - FOR: Выполняет блок кода заданное количество раз.  
   - LOOP: Выполняет блок кода до тех пор, пока не будет достигнуто условие выхода.  
  
4. Обработка ошибок:  
   - TRY...CATCH: Позволяет обрабатывать исключения (ошибки) в блоке кода и выполнить альтернативные действия в случае возникновения ошибки.  
  
5. Управление транзакциями:  
   - BEGIN TRANSACTION: Начинает новую транзакцию.  
   - COMMIT: Фиксирует изменения, сделанные в транзакции.  
   - ROLLBACK: Откатывает транзакцию и отменяет все сделанные изменения.

**35. Язык  Transact-SQL: операторы GO, PRINT, USE.**

Операторы `GO`, `PRINT` и `USE` являются часто используемыми в Transact-SQL (T-SQL). Вот их описание:  
  
1. `GO`:  
  - `GO` не является T-SQL оператором, а скорее командой среды выполнения (например, SQL Server Management Studio или sqlcmd), которая указывает на разделение пакета T-SQL на отдельные блоки.  
  - `GO` используется для разделения скрипта на отдельные части, которые выполняются по отдельности. Каждый раз, когда встречается `GO`, предыдущий блок кода выполняется и компилируется, а затем выполняется следующий блок кода.  
  - `GO` также используется для разделения пакета T-SQL на отдельные транзакции или пакеты выполнения.  
  
2. `PRINT`:  
  - `PRINT` используется для вывода сообщений или значений в окно результатов.  
  - Он может использоваться для отладки или для вывода информации пользователю в процессе выполнения запросов или хранимых процедур.  
  - Синтаксис: `PRINT 'Текст'` или `PRINT @переменная`.  
  
3. `USE`:  
  - `USE` используется для указания базы данных, с которой вы будете работать в текущем сеансе.  
  - Он изменяет контекст базы данных, что означает, что все последующие операции будут выполняться в указанной базе данных.  
  - Синтаксис: `USE ИмяБазыДанных`.  
  
Примеры использования:  
```sql  
-- Пример использования оператора GO  
SELECT \* FROM Table1  
GO  
SELECT \* FROM Table2  
GO  
  
-- Пример использования uоператора PRINT  
DECLARE @message VARCHAR(50)  
SET @message = 'Привет, мир!'  
PRINT @message  
  
-- Пример использования оператора USE  
USE MyDatabase  
SELECT \* FROM MyTable  
```  
Обратите внимание, что операторы `GO` и `USE` используются в среде выполнения T-SQL, такой как SQL Server Management Studio или sqlcmd, и не являются частью самого языка T-SQL. Оператор `PRINT` является частью T-SQL и может использоваться в любой среде, поддерживающей T-SQL.

**36.** **Язык  Transact-SQL: работа с переменными, типы переменных**

В Transact-SQL (T-SQL) доступны различные типы переменных, которые могут использоваться для хранения и обработки значений в процессе выполнения запросов и процедур. Вот некоторые из наиболее распространенных типов переменных в T-SQL:  
 Работа с переменными

Для создания сложных запросов, выполняемых в несколько этапов, в SQL предусмотрены переменные.

В переменных могут сохраняться результаты вычисления встроенных функций и констант.

Для использования переменной, ее надо предварительно «объявить», выполнив команду **DECLARE** языка SQL. Команда **DECLARE** имеет следующий формат: **DECLARE** *<имя\_переменной> <тип\_переменной>*, где *<имя\_переменной>* - это название объявляемой переменной, а *<тип\_переменной>* - тип данных, хранимых в ней. Названия переменных начинаются с символа «@», после которого могут использоваться буквы латинского алфавита, цифры и знаки «@», «$», «%», «&», «~», «\_».

Поддерживаются следующие основные типы данных:

* *Geometry*- пространственный объект;
* *BigInt* - целочисленное значение в диапазоне от -263-1 до 263 (от - 9,223,372,036,854,775,808 по +9,223,372,036,854,775,807);
* *Int* - целочисленное значение в диапазоне от -231 (-2,147,483,648) to 231-1 (2,147,483,647);
* *SmallInt* - целочисленное значение в диапазоне от -215 (-32,768) to 215-1 (32,767);
* *Double*, *Float*, *Real*- число с плавающей точкой;
* *char*, *varchar*- строковое значение.

Переменным можно присваивать только значения соответствующего типа данных.

Примеры объявления переменных разных типов в T-SQL:  
```sql  
DECLARE @IntVariable INT  
DECLARE @VarcharVariable VARCHAR(50)

**37. Язык  Transact-SQL: операторы управления правами доступа к объектам базы данных.**

В Transact-SQL (T-SQL) для управления правами доступа к объектам базы данных используются операторы и команды, связанные с системой управления базами данных. Некоторые из наиболее распространенных операторов и команд для управления правами доступа в T-SQL включают:  
  
1. `GRANT`:  
   - `GRANT` используется для предоставления прав доступа к объектам базы данных (таким как таблицы, представления, процедуры и т. д.) пользователям или ролям.  
   - Синтаксис: `GRANT Permissions ON Object TO User/Role`.  
  
2. `REVOKE`:  
   - `REVOKE` используется для отзыва ранее предоставленных прав доступа к объектам базы данных.  
   - Синтаксис: `REVOKE Permissions ON Object FROM User/Role`.  
  
3. `DENY`:  
   - `DENY` используется для явного запрещения прав доступа к объектам базы данных для конкретных пользователей или ролей.  
   - Синтаксис: `DENY Permissions ON Object TO User/Role`.  
  
4. `EXECUTE AS`:  
   - `EXECUTE AS` позволяет выполнять процедуру или функцию с контекстом безопасности другого пользователя или роли.  
   - Синтаксис: `EXECUTE AS {USER = 'User' | LOGIN = 'Login' | OWNER | CALLER}`.  
  
5. `ALTER AUTHORIZATION`:  
   - `ALTER AUTHORIZATION` используется для изменения владельца объекта базы данных.  
   - Синтаксис: `ALTER AUTHORIZATION ON Object TO Owner`.  
  
6. `CREATE LOGIN`:  
   - `CREATE LOGIN` используется для создания нового входа в систему (логина).  
   - Синтаксис: `CREATE LOGIN LoginName WITH PASSWORD = 'Password'`.  
  
7. `CREATE USER`:  
   - `CREATE USER` используется для создания нового пользователя базы данных.  
   - Синтаксис: `CREATE USER UserName FOR LOGIN LoginName`.  
  
8. `DROP LOGIN` и `DROP USER`:  
   - `DROP LOGIN` используется для удаления существующего входа в систему (логина).  
   - `DROP USER` используется для удаления существующего пользователя базы данных.  
   - Синтаксис: `DROP LOGIN LoginName` или `DROP USER UserName`.  
  
Обратите внимание, что эти операторы и команды могут различаться в зависимости от конкретной системы управления базами данных, такой как Microsoft SQL Server или другие платформы баз данных. Также учтите, что для выполнения операций по управлению правами доступа вам может потребоваться соответствующая роль или привилегии администратора базы данных.

**38. Язык  Transact-SQL: ограничения.**

В Transact-SQL (T-SQL) для установки ограничений на данные в базе данных можно использовать различные механизмы ограничений. Ниже приведены некоторые из наиболее распространенных ограничений, которые могут быть определены в T-SQL:  
  
1. Ограничения целостности:  
  - `PRIMARY KEY`: Ограничение, которое гарантирует уникальность значений в столбце или наборе столбцов и предотвращает наличие NULL значений.  
  - `FOREIGN KEY`: Ограничение, которое создает связь между столбцами двух таблиц и обеспечивает согласованность данных.  
  - `UNIQUE`: Ограничение, которое гарантирует уникальность значений в столбце или наборе столбцов.  
  - `CHECK`: Ограничение, которое определяет условие, которому должны удовлетворять значения в столбце.  
  
2. Ограничения NULL значений:  
  - `NOT NULL`: Ограничение, которое запрещает наличие NULL значений в столбце.  
  
3. Ограничения диапазона значений:  
  - `CHECK`: Ограничение, которое определяет диапазон значений, которым должны удовлетворять значения в столбце.  
  
4. Ограничения уникальности:  
  - `UNIQUE`: Ограничение, которое гарантирует уникальность значений в столбце или наборе столбцов.  
  
5. Ограничения размера данных:  
  - `VARCHAR(n)`: Ограничение размера переменной длины для строковых данных, где `n` представляет максимальную длину строки.  
  - `CHAR(n)`: Ограничение размера фиксированной длины для строковых данных, где `n` представляет фиксированную длину строки.  
  
Примеры определения ограничений в T-SQL:  
  
```sql  
-- Определение ограничения PRIMARY KEY  
CREATE TABLE MyTable  
(  
   ID INT PRIMARY KEY,  
   Name VARCHAR(50)  
)  
-- Определение ограничения FOREIGN KEY  
CREATE TABLE Orders  
(  
   OrderID INT PRIMARY KEY,  
   CustomerID INT,  
   CONSTRAINT FK\_CustomerID FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
)  
-- Определение ограничения CHECK  
CREATE TABLE Employees  
(  
   EmployeeID INT PRIMARY KEY,  
   Age INT CHECK (Age >= 18)  
)  
-- Определение ограничения NOT NULL  
CREATE TABLE MyTable  
(  
   ID INT PRIMARY KEY,  
   Name VARCHAR(50) NOT NULL  
)  
```

**39.** **Язык  Transact-SQL: курсоры.**

**Курсор** – это особый временный объект SQL, предназначенный для использования в программах и хранимых процедурах. С его помощью можно в цикле пройти по результирующему набору строк запроса, по отдельности считывая и обрабатывая каждую его строку. Курсоры Transact-SQL создаются с помощью команды DECLARE CURSOR. Как объект курсора, так и множество строк, на которое он указывает, должны существовать на сервере.

Реализация курсоров в среде MS SQL Server SQL Server поддерживает три вида курсоров: • курсоры SQL применяются в основном внутри триггеров, хранимых процедур и сценариев;

• курсоры сервера действуют на сервере и реализуют программный интерфейс приложений для ODBC, OLE DB, DB\_Library;

• курсоры клиента реализуются на самом клиенте. Они выбирают весь результирующий набор строк из сервера и сохраняют его локально, что позволяет ускорить операции обработки данных за счет снижения потерь времени на выполнение сетевых операций

**Характеристики курсоров**

• Отражение изменений Способность курсора отражать изменения в данных называется чувствительностью курсора. При создании вашего курсора могут быть независимо определены два вида чувствительности: изменения каких строк включаются во множество (членство множества) и отражение изменений в исходных строках.

• Прокрутка Второй характеристикой курсора является способность осуществления прокрутки как вперед, так и назад, либо только вперед. Здесь имеет место извечная для программирования дилемма: скорость против гибкости. Последовательные курсоры (forward-only) работают значительно быстрее, но имеют меньшую гибкость.

• Обновление Последней характеристикой, используемой для классификации курсоров, является возможность обновления строк курсором. Опять же, курсоры "только чтение" обычно более производительны, но имеют меньшую гибкость

**Типы курсоров**

Transact-SQL поддерживает четыре различных типа курсоров: **статические, ключевые, динамические и курсоры быстрого доступа, или "пожарные" (firehose).** Каждый тип курсора хранит различные данные относительно строк, на которые он указывает, кроме того, каждому типу курсора свойственны различные сочетания характеристик, рассмотренных в предыдущем разделе.

**• Статические курсоры** Статический курсор делает как бы моментальный снимок данных, задаваемых оператором SELECT, и хранит их в базе данных tempdb. Он "не чувствует" изменений в структуре или в значениях данных, а поскольку любые модификации будут отражены только в копии, этот курсор всегда открывается в режиме "только чтение". Статические курсоры, однако, могут быть объявлены как последовательные или как прокручиваемые.

**• Ключевые курсоры** Ключевой курсор копирует в базу tempdb только те столбцы, которые уникально идентифицируют каждую строку. Чтобы иметь возможность объявить ключевой курсор, каждая таблица, входящая в определение оператора SELECT, должна иметь уникальный индекс, который задает копируемый набор – ключ. Ключевые курсоры могут быть как модифицируемыми, так и иметь режим "только для чтения". Они также могут быть прокручиваемыми или последовательными. Членство в ключевом курсоре фиксируется на момент объявления курсора. Если в процессе открытого состояния курсора добавляется строка, удовлетворяющая условию отбора, она не будет добавлена во множество. Типы курсоров

**• Динамические курсоры** Динамический курсор ведет себя так, как если бы при каждом обращении к строке повторно выполнялся оператор SELECT. Динамические курсоры отражают изменения, связанные как с членством, так и со значениями исходных данных, независимо от того, сделаны ли эти изменения внутри курсора, либо внесены другим пользователем.

**• Курсоры быстрого доступа** SQL Server поддерживает специальную оптимизированную форму не прокручиваемого курсора, допускающего только чтение. Этот вид курсора объявляется с использованием ключевого слова FAST\_FORWARD, и чаще всего его называют "пожарным" курсором (firehose). "Пожарные" курсоры очень эффективны, но при их использовании имеются два важных ограничения. Во-первых, если в операторе определения SELECT курсора вы использовали столбцы с типом данных text, ntext или image, а также фразу TOP, SQL Server преобразует курсор в ключевой. Во-вторых, если оператор SELECT, который вы использовали для определения курсора, содержит таблицы, имеющие триггеры, и таблицы, не имеющие триггеров, курсор преобразуется в статический. Триггеры представляют собой сценарии Transact-SQL, которые автоматически исполняются сервером при в

1. Определение курсора:  
  ```sql  
  DECLARE cursor\_name CURSOR FOR  
  SELECT column1, column2, ...  
  FROM table\_name  
  WHERE condition  
  ```  
  Здесь вы определяете курсор с помощью ключевого слова `DECLARE` и присваиваете ему имя `cursor\_name`. Затем указываете SQL-запрос, который определяет набор данных, по которому будет выполняться итерация.  
  
2. Открытие курсора:  
  ```sql  
  OPEN cursor\_name  
  ```  
  С помощью оператора `OPEN` вы открываете курсор и подготавливаете его для использования.  
  
3. Получение данных из курсора:  
  ```sql  
  FETCH NEXT FROM cursor\_name INTO variable1, variable2, ...  
  ```  
  С помощью оператора `FETCH NEXT` вы получаете следующую запись из курсора и сохраняете значения в указанных переменных.  
  
4. Использование данных из курсора:  
  ```sql  
  WHILE @@FETCH\_STATUS = 0  
  BEGIN  
      -- Ваш код для обработки данных  
      ...  
      FETCH NEXT FROM cursor\_name INTO variable1, variable2, ...  
  END  
  ```  
  Вы используете цикл `WHILE` для обработки данных в курсоре. Цикл выполняется, пока `@@FETCH\_STATUS` равен 0, что означает, что в курсоре еще есть доступные записи. Внутри цикла вы можете выполнять необходимые операции с данными, используя значения переменных.  
  
5. Закрытие курсора:  
  ```sql  
  CLOSE cursor\_name  
  ```  
  С помощью оператора `CLOSE` вы закрываете курсор после окончания итераций.  
  
6. Освобождение ресурсов, связанных с курсором:  
  ```sql  
  DEALLOCATE cursor\_name  
  ```  
  С помощью оператора `DEALLOCATE` вы освобождаете ресурсы, связанные с курсором, после завершения его использования.

**40. Язык  Transact-SQL: представления.**

Transact-SQL (T-SQL) предоставляет возможность создавать представления (views) в базе данных SQL Server. Представление - это виртуальная таблица, которая основывается на результатах запроса SELECT и может быть использована в качестве таблицы при выполнении других запросов.

**Типы представлений:**

**Индексированным** называется материализованное представление. Это означает, что определение представления вычисляется, а результирующие данные хранятся точно так же, как и таблица.

**Секционированным** называется представление, соединяющее горизонтально секционированные данные набора таблиц-элементов, находящихся на одном или нескольких серверах

**Системные** представления предоставляют доступ к метаданным каталога. Системные представления можно использовать для получения сведений об экземпляре SQL Server или объектах, определенных в экземпляре. Метаданные — это субканальная информация об используемых данных. субканальная информация- идентификатор определенного объема информации, т. е. данные о данных.

Вот пример создания представления с использованием T-SQL:  
  
```sql  
CREATE VIEW [dbo].[name]  
AS  
SELECT \*  
FROM name\_table  
WHERE условие  
```  
После создания представления можно использовать его в запросах, как если бы это была обычная таблица.   
  
Представления могут быть полезными, когда требуется ограничить доступ к определенным столбцам или строкам, а также для упрощения и повторного использования сложных запросов. Они также позволяют создавать виртуальные таблицы, которые могут быть обновлены, если определены соответствующие триггеры обновления.  
  
Чтобы удалить представление, можно использовать оператор DROP VIEW:  
  
```sql  
DROP VIEW [dbo].[name]  
```

**41. Язык  Transact-SQL: хранимые процедуры.**

Хранимая процедура (stored procedure) в Transact-SQL (T-SQL) представляет собой набор предопределенных SQL-операторов, которые хранятся и выполняются на сервере базы данных. Они используются для группировки и выполнения повторяющихся задач или сложных операций в базе данных.  
  
Вот пример создания хранимой процедуры с использованием T-SQL:  
  
```sql  
CREATE PROCEDURE [dbo].[GetEmployeeByID]  
    @EmployeeID INT  
AS  
BEGIN  
    SELECT EmployeeID, FirstName, LastName, Department  
    FROM Employees  
    WHERE EmployeeID = @EmployeeID  
END  
```  
В этом примере создается хранимая процедура с именем "GetEmployeeByID". Она принимает входной параметр @EmployeeID типа INT и возвращает информацию о сотруднике с указанным идентификатором.  
  
После создания хранимой процедуры ее можно вызвать следующим образом:  
```sql  
EXEC [dbo].[GetEmployeeByID] @EmployeeID = 12345  
```  
В этом примере вызывается хранимая процедура "GetEmployeeByID" и передается значение 12345 в качестве аргумента для параметра @EmployeeID. Результатом будет набор строк, содержащих информацию о сотруднике с указанным идентификатором.  
  
Хранимые процедуры предоставляют ряд преимуществ, таких как повышение производительности, уменьшение сетевого трафика, повышение безопасности и возможность повторного использования кода. Они также позволяют создавать транзакции, использовать логику управления и обработки ошибок.  
  
Чтобы удалить хранимую процедуру, можно использовать оператор DROP PROCEDURE:  
  
```sql  
DROP PROCEDURE [dbo].[GetEmployeeByID]  
```  
Это удалит хранимую процедуру "GetEmployeeByID" из базы данных.

**42. Язык  Transact-SQL: триггеры.**

Триггеры представляют специальный тип хранимой процедуры, которая вызывается автоматически при выполнении определенного действия над таблицей или представлением, в частности, при добавлении, изменении или удалении данных, то есть при выполнении команд **INSERT, UPDATE, DELETE.**

Формальное определение триггера:

CREATE TRIGGER имя\_триггера

ON {имя\_таблицы | имя\_представления}

{AFTER | INSTEAD OF} [INSERT | UPDATE | DELETE]

AS выражения\_sql

Для создания триггера применяется выражение CREATE TRIGGER, после которого идет имя триггера. Как правило, имя триггера отражает тип операций и имя таблицы, над которой производится операция.

Каждый триггер ассоциируется с определенной таблицей или представлением, имя которых указывается после слова ON.

-Затем устанавливается тип триггера. Мы можем использовать один из двух типов:

AFTER: выполняется после выполнения действия. Определяется только для таблиц.

INSTEAD OF: выполняется вместо действия (то есть по сути действие - добавление, изменение или удаление - вообще не выполняется). Определяется для таблиц и представлений

После типа триггера идет указание операции, для которой определяется триггер: INSERT, UPDATE или DELETE.

Для триггера AFTER можно применять сразу для нескольких действий, например, UPDATE и INSERT. В этом случае операции указываются через запятую. Для триггера INSTEAD OF можно определить только одно действие.

И затем после слова AS идет набор выражений SQL, которые собственно и составляют тело триггера.

**Отключение триггера**

Бывает, что мы хотим приостановить действие триггера, но удалять его полностью не хотим. В этом случае его можно временно отключить с помощью команды DISABLE TRIGGER:

А когда триггер понадобится, его можно включить с помощью команды ENABLE TRIGGER:

ENABLE TRIGGER Products\_**INSERT**\_UPDATE ON Products

**Удаление триггера**

Для удаления триггера необходимо применить команду DROP TRIGGER:

1.запуск по объекту(таблица или виюер)

2.запуск по событию по объекту(INSERT, UPDATE, DELETE)

3.когда по времени или (после события до события)

А потом что

Триггеры представляют специальный тип хранимой процедуры, которая

вызывается автоматически при выполнении определенного действия над ...

самый критичный **UPDATE**