**Ответы на вопросы к коллоквиуму**

1. ***Классификация и основные функции СУБД. Архитектура системы баз данных***

Функции СУБД:

1. Описание структуры БД. Подобными средствами являются язык описания данных, язык манипулирования данными и язык создания запросов (SQL).

2. Создание, обновление и извлечение информации из БД. Средством извлечения информации из БД является язык обработки данных.

3. Защита данных. Использование системы разрешается лишь пользователям, имеющим на это право.

4. Целостность данных. При выполнении пользователем операций над данными поддерживается согласованность хранящихся данных.

5. Независимость данных. При использовании данных изменение одних не приводит к изменению других.

6. Восстановление БД после сбоев. В случае аппаратных или программных сбоев система должна возвращаться к некоторому согласованному состоянию данных.

Классификация СУБД может быть произведена по различным признакам, среди которых выделяют:

1. По форме представления информации: фактографические, документальные, мультимедийные, которые соответствуют цифровой, символьной и др. формам представления информации. К ним относят картографические, видео-, аудио-, графические и другие БД.

2. По типу используемой модели данных: иерархические, сетевые, реляционные.

3. По типологии хранения данных: локальные и распределённые (удалённые) БД.

4. По типологии доступа и характеру использования: специализированные и интегрированные.

5. По функциональному назначению (характеру решаемых задач): операционные и справочно-информационные.

6. По сфере возможного применения: универсальные и специализированные (или проблемно-ориентированные) системы.

7. По степени доступности: общедоступные и с ограниченным доступом пользователей.

Архитектура системы бд:

*Централизованная архитектура*

При использовании этой технологии база данных, СУБД и прикладная программа (приложение) располагаются на одном компьютере (рисунок 1). Для такого способа организации не требуется поддержки сети и все сводится к автономной работе. Многопользовательская технология работы обеспечивалась либо режимом мультипрограммирования, либо режимом разделения времени.

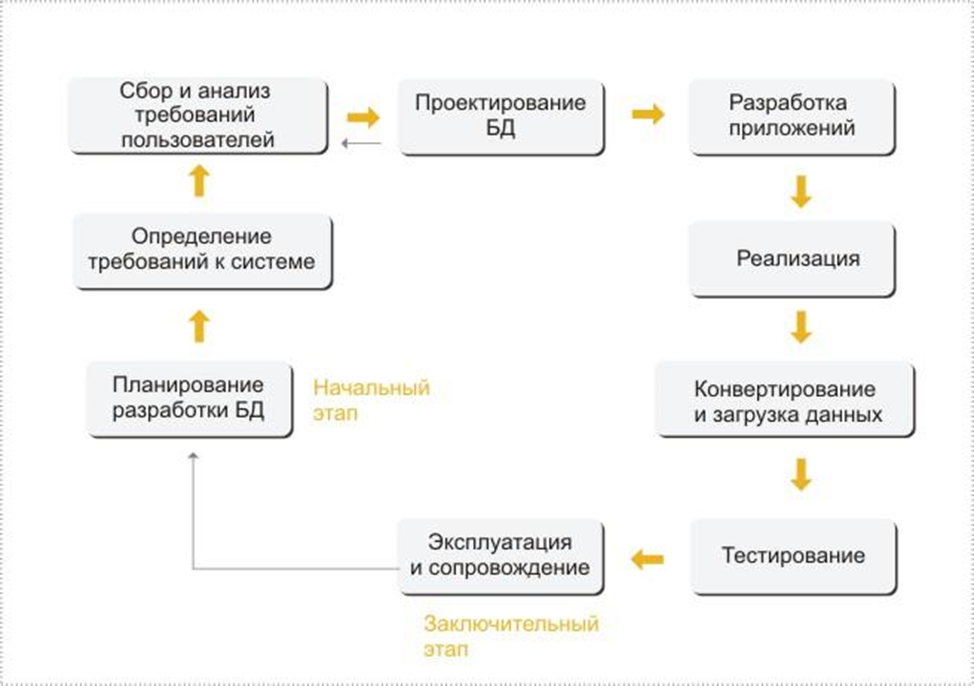
*Архитектура "файл-сервер"*

Эта архитектура баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы базы данных. В соответствии с запросами пользователей файлы с файл-сервера передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных

*Архитектура "клиент – сервер"*

Использование технологии "клиент – сервер" предполагает наличие некоторого количества компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (является сервером сети). Центральная машина (сервер базы данных) помимо хранения базы данных обеспечивает выполнение основного объема обработки данных. Запрос клиента (рабочей станции) порождает поиск и извлечение данных на сервере, которые затем транспортируются по сети к клиенту.

1. ***Основные этапы проектирования БД. Жизненный цикл БД***

Как и любой программный продукт, база данных обладает собственным *жизненным циклом* (ЖЦБД). Главной составляющей в жизненном цикле БД является создание единой базы данных и программ, необходимых для ее работы. ЖЦБД включает в себя следующие основные этапы:

Проектирование базы данных;

Полный цикл разработки базы данных включает *концептуальное*, *логическое* и *физическое* ее проектирование.

*Концептуальное проектирование базы данных*

Цель этого концептуального проектирования – создание концептуальной модели данных исходя из представлений пользователей о предметной области.

В построении общей концептуальной модели данных выделяют ряд этапов.

- Выделение локальных представлений, соответствующих обычно относительно независимым данным. Каждое такое представление проектируется как подзадача.

- Формулирование сущностей, описывающих локальную предметную область проектируемой БД, и описание атрибутов, составляющих структуру каждой сущности.

- Выделение ключевых атрибутов.

- Спецификация связей между сущностями. Удаление избыточных связей.

- Анализ и добавление неключевых атрибутов.

- Объединение локальных представлений.

*Логическое проектирование базы данных*

Цель этапа логического проектирования – преобразование концептуальной модели на основе выбранной модели данных в логическую модель, не зависимую от особенностей используемой в дальнейшем СУБД для физической реализации базы данных.

*Физическое проектирование базы данных*

Цель этапа физического проектирования – описание конкретной реализации базы данных, размещаемой во внешней памяти компьютера.

1. ***Концептуальное проектирование реляционной БД. Модель "Сущность - cвязь". Элементы ER-модели***

С точки зрения теории реляционных БД, основные принципы реляционной модели на концептуальном уровне можно сформулировать следующим образом:

• все данные представляются в виде упорядоченной структуры, определенной в виде строк и столбцов и называемой отношением;

• все значения являются скалярами. Это означает, что для любой строки и столбца любого отношения существует одно и только одно значение;

• все операции выполняются над целым отношением, и результатом их выполнения также является целое отношение. Этот принцип называется замыканием.

Модель "Сущность - Связь": Эта модель предметной области, которая используется на этапе концептуального проектирования. Диаграмма "сущности-связи" служит для описания схемы базы на концептуальном уровне проектирования. Все такие диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

Элементами описания реляционной модели данных на концептуальном уровне являются сущности, атрибуты, домены и связи.

1) Сущность - некоторый обособленный объект или событие, информацию о котором необходимо сохранять в базе данных, имеющий определенный набор свойств - атрибутов.

2) 2) Атрибут - это поименованная характеристика сущности, которая принимает значения из некоторого множества значений.

3) Домен - это набор всех допустимых значений, которые может содержать атрибут.

4) Связи - на концептуальном уровне представляют собой простые ассоциации между сущностями. Например, утверждение "Покупатели приобретают продукты" указывает, что между сущностями "ПОКУПАТЕЛЬ" и "ПРОДУКТ" существует связь.

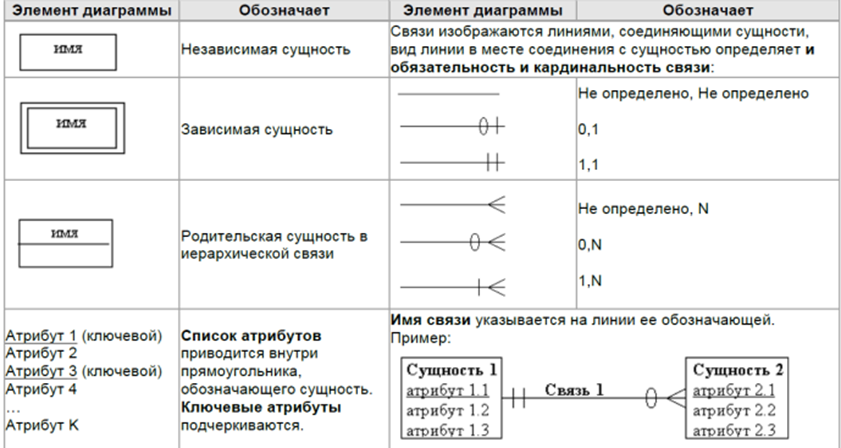
1. ***Построение ER-диаграммы для схемы базы данных (нотации Чена, Мартина, нотация IDEF1X, нотация Баркера)***

Диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

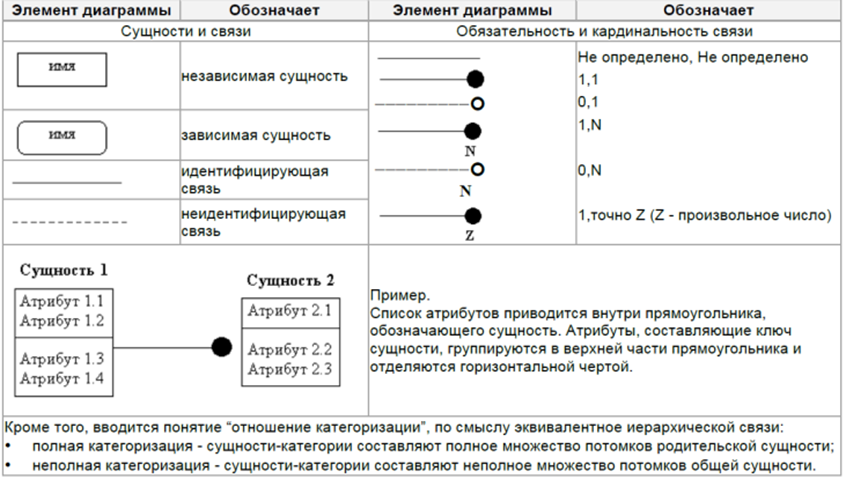
1) Нотация Чена.



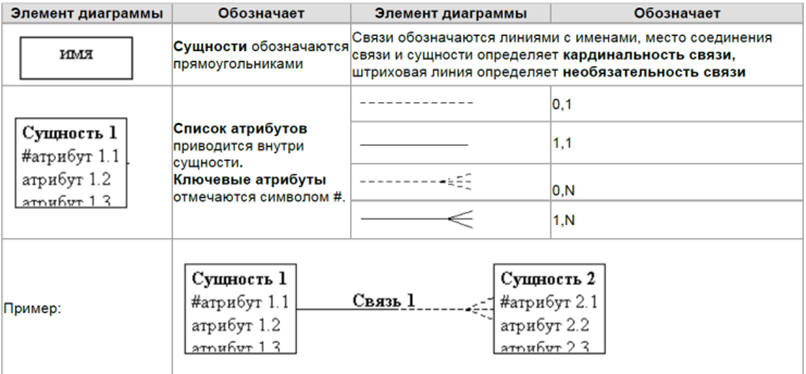
2) Нотация Мартина



3) Нотация IDEF1X



4) Нотация Баркера

****

1. *Логическое проектирование реляционных баз данных. Проектирование реляционных баз данных на основе нормализации*

Реляционная база данных представляет собой множество взаимосвязанных двумерных таблиц – реляционных таблиц, называемых также отношениями, в каждой из которых содержатся сведения об одной сущности автоматизируемой предметной области.

Логическую структуру реляционной базы данных образует совокупность реляционных таблиц, между которыми установлены связи.

В таблицах базы должны сохраняться все данные, необходимые для решения задач предметной области. Причем каждый элемент данных должен храниться только в одном экземпляре. Для создания таблиц, соответствующих реляционной модели данных, используется процесс, называемый нормализацией данных.

*Нормализация* – это удаление из таблиц повторяющихся данных путем их переноса в новые таблицы, записи которых не содержат повторяющихся значений.

Отношения реляционной базы данных содержат как структурную, так и семантическую (смысловую) информацию. Структурная информация задается схемой отношения, а семантическая выражается функциональными связями между атрибутами схемы.

Группировка атрибутов должна быть рациональной и удовлетворять следующим требованиям:

• выбранные для отношения первичные ключи должны быть минимальными;

• выбранный состав отношений должен отличаться минимальной избыточностью атрибутов;

• между атрибутами не должно быть нежелательных функциональных зависимостей и они должны обеспечивать минимальное дублирование данных;

• не должно быть трудностей при выполнении операций включения, удаления и модификации (аномалии);

• перестройка набора отношений при введении новых типов должна быть минимальной.

Таким образом, основная цель логического проектирования базы данных - сокращение избыточности хранимых данных и устранение возможных потенциальных аномалий работы с базами данных.

Для удовлетворения вышеотмеченных требований предложен аппарат нормализации отношений.

*Нормализация отношений* - это пошаговый обратимый процесс композиции или декомпозиции исходных отношений в отношения, обладающие лучшими свойствами при включении, изменении и удалении данных, назначение им ключей по определенным правилам нормализации и выявление всех возможных функциональных зависимостей.

1. *Нормальные формы (НФ). Понятие 1НФ, 2НФ, 3НФ*

Нормализация отношений - это пошаговый обратимый процесс композиции или декомпозиции исходных отношений в отношения, обладающие лучшими свойствами при включении, изменении и удалении данных, назначение им ключей по определенным правилам нормализации и выявление всех возможных функциональных зависимостей.



• каждая следующая нормальная форма в некотором смысле является более ограниченной, но более лучшей, чем предыдущая;

• при переходе к следующей нормальной форме положительные свойства предыдущих нормальных свойств сохраняются

**Нормальная форма** — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами

*- Требования первой нормальной формы (1NF)*

Требование первой нормальной формы (1NF) , заключается в том, чтобы таблицы соответствовали реляционной модели данных и соблюдали определённые реляционные принципы.

• В таблице не должно быть дублирующих строк

• В каждой ячейке таблицы хранится атомарное значение (одно не составное значение)

• В столбце хранятся данные одного типа

• Отсутствуют массивы и списки в любом виде

*- Требования второй нормальной формы (2NF)*

Чтобы база данных находилась во второй нормальной форме (2NF), необходимо чтобы ее таблицы удовлетворяли следующим требованиям:

• Таблица должна находиться в первой нормальной форме

• Таблица должна иметь ключ

• Все неключевые столбцы таблицы должны зависеть от полного ключа (в случае если он составной)

Главное правило второй нормальной формы (2NF) звучит следующим образом Таблица должна иметь правильный ключ, по которому можно идентифицировать каждую строку.

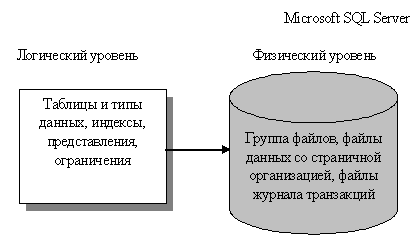
*- Требование третьей нормальной формы (3NF)*

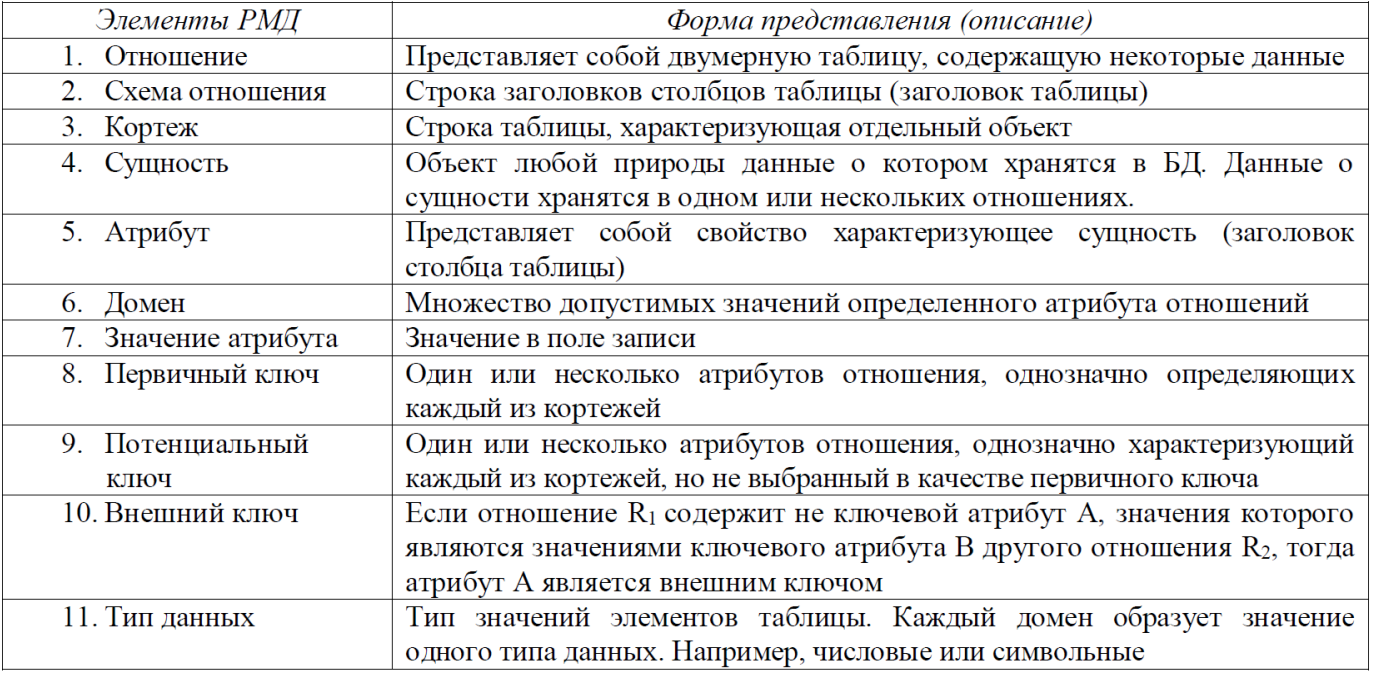
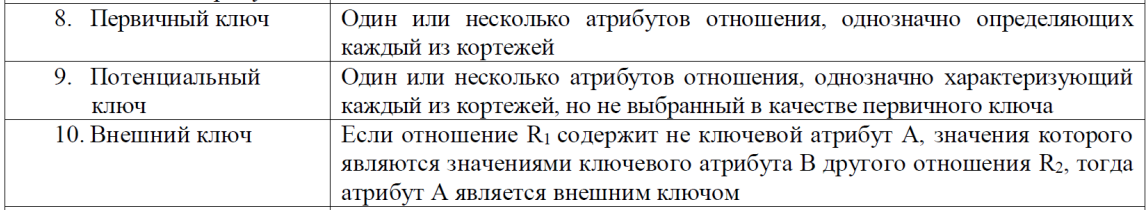
Требование третьей нормальной формы (3NF) заключается в том, чтобы в таблицах отсутствовала транзитивная зависимость.

*Транзитивная зависимость* – это когда неключевые столбцы зависят от значений других неключевых столбцов.

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

1. ***Преобразование ER-модели в реляционную модель. Правила преобразования ER-диаграмм в реляционные таблицы*  
     
   Получение реляционной схемы из ER-модели   
   Шаг 1.** *Каждая простая сущность превращается в отношение.* Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем отношения. **Шаг 2.** *Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат.* Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут. **Шаг 3.** *Компоненты уникального идентификатора сущности превращаются в первичный ключ отношения.* Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. **Шаг 4.** *Связи "многие к одному" (и "один к одному") становятся внешними ключами.* Для этого делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.  
     
    **Шаг 5.** *В таблицах, построенных на основе ассоциаций, внешние ключи используются для идентификации участников ассоциации, а в таблицах, построенных на основе характеристик и обозначений, внешние ключи используются для идентификации сущностей, описываемых этими характеристиками и обозначениями.* Специфицировать ограничения, связанные с каждым из этих внешних ключей. **Шаг 6.** *Если в концептуальной схеме присутствовали подтипы, то возможны два способа: • все подтипы размещаются в одной таблице (а); • для каждого подтипа строится отдельная таблица (б)*. При применении способа (а) таблица создается для наиболее внешнего супертипа. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА и он становится частью первичного ключа. Для работы с подтипами могут создаваться представления. При использовании метода (б) супертип воссоздается с помощью конструкции UNION.   
     
   **Шаг 7.***Выполнить шаги по нормализации полученных отношений, приведя их к желаемой нормальной форме.*  
   **Шаг 8.** *Указать ограничения целостности проектируемой базы данных и дать (если это необходимо) краткое описание полученных таблиц и их полей.*  
   **Шаг 9.** *Создать индексы для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы и выполнять соединения.* **Правила преобразования ER-диаграмм в реляционные таблицы**Для каждой сущности создается таблица. Причем каждому атрибуту сущности соответствует столбец таблицы. Правила генерации таблиц из ER-диаграмм опираются на два основных фактора – тип связи и класс принадлежности сущности. Изложим их.   
     
   **Правило 1**: *Если связь типа 1:1 и класс принадлежности обеих сущностей является обязательным, то необходима только одна таблица.* Первичным ключом этой таблицы может быть первичный ключ любой из двух сущностей.   
     
   **Правило 2**: *Если связь типа 1:1 и класс принадлежности одной сущности является обязательным, а другой - необязательным, то необходимо построить таблицу для каждой сущности.* Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Первичный ключ сущности, для которой класс принадлежности является необязательным, добавляется как атрибут в таблицу сущности с обязательным классом принадлежности.   
     
   **Правило 3**: *Если связь типа 1:1 и класс принадлежности обеих сущностей необязательный, то необходимо построить три таблицы - по одной для каждой сущности и одну для связи.* Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Таблица для связи среди своих атрибутов должна иметь ключи обеих сущностей.   
     
   **Правило 4**: *Если связь типа 1:М и класс принадлежности сущности на стороне М является обязательным, то необходимо построить таблицу для каждой сущности.* Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Первичный ключ сущности на стороне 1 добавляется как атрибут в таблицу для сущности на стороне М.   
     
   **Правило 5**: *Если связь типа 1:М и класс принадлежности сущности на стороне М является необязательным, то необходимо построить три таблицы - по одной для каждой сущности и одну для связи*. Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Таблица для связи среди своих атрибутов должна иметь ключи обеих сущностей.   
     
   **Правило 6**: *Если связь типа М:N, то необходимо построить три таблицы - по одной для каждой сущности и одну для связи.* Первичный ключ сущности должен быть первичным ключом соответствующей таблицы. Таблица для связи среди своих атрибутов должна иметь ключи обеих сущностей.
2. ***Архитектура MS SQL Server. Логический уровень SQL Server. Физический уровень SQL Server***  
   Под архитектурой (структурой) базы данных конкретной СУБД будем понимать основные модели представления данных, используемые в соответствующей СУБД а также взаимосвязи между этими моделями.  
     
   Логический уровень *(уровень модели данных СУБД) - средство представления концептуальной модели*. Здесь каждая СУБД имеет некоторые отличия, но они являются не очень значительными. Отметим, что у разных СУБД существенно отличаются механизмы перехода от логического к физическому уровню представления.

Физический уровень *(внутреннее представление данных в памяти ЭВМ - физическая структура базы данных)*. Данный уровень рассмотрения подразумевает изучение базы данных на уровне файлов, хранящихся на жестком диске. Структура этих файлов – особенность каждой конкретной СУБД, в т.ч. и Microsoft SQL Server.  


1. ***Модели данных. Классификация моделей.*   
   Модель данных** – это совокупность функциональных характеристик объектов и особенностей представления информации.   
   **Модель данных** – это совокупность, трёх составляющих:   
   - набора типов данных;   
   - набора операторов или правил вывода, предназначенных чтобы находить, выдавать или преобразовывать информацию, содержащуюся в любых частях структуры;  
    - набор правил целостности, которые определяют множество непротиворечивых состояний базы данных.   
   **Модель данных** - схема (порядок, совокупность принципов, система) организации данных в единое целое для создания, накопления, обработки и управления.  
   Более простое определение: **модель данных** - совокупность структур данных и операций по их обработке.   
     
   **Классификация:**  
     
   **Инфологические модели** данных используются на ранних стадиях проектирования для описания структур данных в процессе разработки приложения, а **даталогические модели** уже поддерживаются конкретной СУБД. **Физические модели** описывают структуры и принципы их хранения во внешней памяти, а также доступа к ним в зависимости от используемых аппаратных средств и программного обеспечения низкого уровня.  
   Среди разновидностей инфологических моделей наибольшее распространение получили **модели сущность-связь**.   
   **Документальные модели** данных соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке.   
   **Тезаурусные модели** основаны на принципе организации словарей, содержат определенные языковые конструкции и принципы их взаимодействия в заданной грамматике. Эти модели эффективно используются в системах-переводчиках, особенно многоязыковых переводчиках.   
   **Дескрипторные модели** — самые простые из документальных моделей, они широко использовались на ранних стадиях использования документальных баз данных  
     
   **Классические модели данных:  
     
   Иерархическая модель данных.** один объект выступает как родительский, а с ним может быть связано множество подчинённых объектов (дочерних).  
      
   **Сетевая модель данных.** Её основой является сеть, т.е. произвольные графа, в вершинах, которых записана соответствующая информация, а рёбра соответствуют связям между ними.  
     
   **Реляционная модель данных**. Представляет собой совокупность данных, содержащихся в двухмерных таблицах, соединённых между собой отношениями.
2. ***Реляционная модель данных. Элементы реляционной модели данных и формы их представления.***Реляционная модель данных предложена сотрудником фирмы IBM Эдгаром Коддом и основывается на понятии отношение (relation).  
      
   **Реляционная модель данных (РМД)** - это способ рассмотрения данных, при котором данные воспринимаются пользователем как взаимосвязанные таблицы и в распоряжении пользователя имеются некоторые операторы, которые генерируют новые таблицы из старых.   
     
   Под **таблицами** понимается структура данных, состоящая из строк и столбцов. В этой структуре каждый столбец содержит данные только одного типа, каждая строка состоит из набора значений составляющих ее столбцов.   
     
   Под **операторами** понимаются операции выборки, группировки, соединения и некоторые другие, результатом которых являются новые таблицы, полученные на основании старых.   
     
   **Реляционная модель данных (РМД)** **некоторой предметной области** представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними.   
     
   **Отношение** представляет собой множество элементов, называемых кортежами.   
     
   **Основные элементы реляционной модели данных и формы их представления:  
     
   **
3. ***Назначение ключевых полей в реляционной базе данных. Понятие ключа. Типы ключей.***Ключи обычно используют для достижения следующих целей:   
   1) Исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются);   
   2) Упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним - возрастание, а по другим - убывание);   
   3) Ускорения работы с кортежами отношения;   
   4) Организации связанных таблиц.   
     
   Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных, называемое ссылочной целостностью. Это означает, что каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях  
     
   Ключ СУБД – это атрибут или набор атрибутов, который помогает идентифицировать строку (кортеж) в отношении (таблице). Они позволяют найти связь между двумя таблицами. Клавиши помогают однозначно идентифицировать строку в таблице по комбинации одного или нескольких столбцов в этой таблице.  
     
   **Типы ключей:  
   **
4. ***Связывание таблиц. Основные виды связей.***Связывание таблиц позволяет повысить достоверность хранимой информации за счет того, что многие СУБД автоматически проводят контроль целостности вводимой в БД информации при связывании, кроме того, связи между таблицами облегчают доступ к данным за счет возможности обращения к произвольным полям связанных записей, что уменьшает количество явных обращений к таблицам данных. Между двумя таблицами в общем случае могут устанавливаться следующие четыре основных вида связи:   
   • один к одному (1:1);   
   • один ко многим (1:М);   
   • многие к одному (М: 1);   
   • многие ко многим (М:М);   
   Число сущностей, которые могут быть ассоциированы через набор связей, называются *степенью связи*. *Набор связей* – это ассоциация между N сущностями, причём N всегда больше или равен 2, каждая из которых относится к некоторому набору сущностей. Если связь осуществляется между двумя сущностями, то связь называется **бинарной**.  
   **Связь вида 1:1** Данный вид связи образуется в случае, когда все поля связи основной и дополнительной таблиц являются ключевыми. Связанные таблицы при этом являются равноправными. На практике связи вида 1:1 используются редко, т.к. хранимую в двух таблицах информацию легко объединить в одну таблицу, которая занимает меньше места в памяти ЭВМ.   
   **Связь вида 1:М** Этот вид связи имеет место в случае, когда одной записи основной таблицы соответствует несколько записей вспомогательной таблицы. Это наиболее распространенный вид связей в РМД.   
   **Связь вида М:1** Является разновидность связи 1:М. Так если в предыдущем примере рассматривать не отношение «Поставщик» - «Товар», а «Товар» - «Поставщик», получим связь вида М:1.   
   **Связь вида М:М**  Возникает в случае, если нескольким записям основной таблицы соответствует несколько записей дополнительной таблицы. Реляционная модель данных эту связь не поддерживает. Поэтому при возникновении данного вида связи в этот вид связи вовлекается более чем 2 таблицы, т.к. это наиболее общий вид связи.  
   Ещё одной важной характеристикой связи является **класс принадлежностей** входящих в неё сущностей (**кардинальность связи**). Если сущность имеет обязательный класс принадлежностей, то этот факт обозначается указанием интервала числа возможных вхождений сущности в связь. В данном случае это одна или много записей. Необязательно класс принадлежностей обозначается числом 0, 1 или много.
5. ***Правила Е.Ф. Кодда для СУБД***Е.Ф. Кодд определил 13 правил реляционной модели, но, счёт начал с 0.   
     
   0. Реляционная СУБД должна быть способна полностью управлять базой данных через ее реляционные возможности.   
     
   1. Информационное правило - вся информация в реляционной БД (включая имена таблиц и столбцов) должна определяться строго как значения в таблицах.   
     
   2. Гарантированный доступ - любое значение в реляционной БД должно быть гарантированно доступно для использования через комбинацию имени таблицы, значения первичного ключа и имени столбца   
     
   3. Поддержка пустых значений (null value) - СУБД должна уметь работать с пустыми значениями (неизвестными или неиспользованными значениями), в отличие от значений по умолчанию и независимо для любых доменов.   
     
   4. Онлайновый реляционный каталог - описание БД и ее содержания должны быть представлены на логическом уровне как таблицы, к которым можно применять запросы, используя язык базы данных.   
     
   5. Исчерпывающий язык управления данными - по крайней мере, один из поддерживаемых языков должен иметь четко определенный синтаксис и быть всеобъемлющим. Он должен поддерживать описание структуры данных и манипулирование ими, правила целостности, авторизацию и транзакции.   
     
   6. Правило обновления представлений (views) - все представления, теоретически обновляемые, могут быть обновлены через систему.   
     
   7. Вставка, обновление и удаление - СУБД поддерживает не только запрос на отбор данных, но и вставку, обновление и удаление   
     
   8. Физическая независимость данных - на программы-приложения и специальные программы логически не влияют изменения физических методов доступа к данным и структур хранилищ данных.   
     
   9. Логическая независимость данных - изменения структур таблиц, в пределах разумного, на программы-приложения и специальные программы логически не влияют.   
     
   10. Независимость целостности - язык БД должен быть способен определять правила целостности. Они должны сохраняться в онлайновом справочнике, и не должно существовать способа их обойти.   
     
   11. Независимость распределения - на программы-приложения и специальные программы логически не влияет, первый раз используются данные или повторно.   
     
   12. Неподрывность - невозможность обойти правила целостности, определенные через язык базы данных, использованием языков низкого уровня.   
     
   Кодд предложил применение реляционной алгебры в СУРБД, для разбиения данных в связанные наборы. Он организовал свою систему БД вокруг теории, основанной на наборах данных.
6. ***Реляционная алгебра и реляционное исчисление. Основные операции реляционной алгебры.***

***Основные свойства реляционной алгебры. Традиционные операции над множествами:***

1) Объединением двух совместимых по типу отношений А и В (A UNION B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих А или В или обоим отношениям.

2) Пересечением двух совместимых по типу отношений А и В (A INTERSECT B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих одновременно обоим отношениям A и B.

3) Вычитанием двух совместимых по типу отношений А и В (A MINUS B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих отношению A и не принадлежащих отношению B.

4) Произведение. В математике декартово произведение (или для краткости произведение) двух множеств является множеством всех таких упорядоченных пар элементов, что первый элемент в каждой паре берется из первого множества, а второй элемент в каждой паре берется из второго множества.

5) Выборка.

6) Проекция.

7) Естественное соединение.

8) Деление.

***Существует пять основных операций РА и три вспомогательных, которые могут быть выражены через основные.***

1. Проекция (project). Это унарная операция (выполняемая над одним отношением), служащая для выбора подмножества атрибутов (столбцов) из отношения R. Она уменьшает арность результирующего отношения (количество столбцов) и может уменьшить мощность результирующего отношения (количество строк) за счёт исключения одинаковых кортежей

2. Селекция (select). Это унарная операция, результатом которой является подмножество кортежей исходного отношения, соответствующих условиям, которые накладываются на значения одного или нескольких атрибутов. Арность результирующего отношения (количество столбцов) в итоге селекции не меняется.

3. Декартово произведение (сartesian product). Это бинарная операция над разносхемными отношениями, соответствующая определению декартова произведения для РМД. Кортежи результирующего отношения состоят из всех атрибутов исходных отношений

4. Объединение (union). Объединение – бинарная операция над односхемными отношениями результатом является отношение, которое включает в себя все кортежи обоих исходных отношений без повторов; причём, имена полей односхемных отношений могут быть разными, достаточно, чтобы совпадало количество полей и типы данных соответствующих полей.

5. Разность (except). Разность – бинарная операция над односхемными отношениями, результатом является отношение включающее множество кортежей R, не входящих в S; причём, имена полей односхемных отношений могут быть разными, достаточно, чтобы совпадало количество полей и типы данных соответствующих полей

***Вспомогательные операции реляционной алгебры.***

6. Пересечение (intersect). Пересечение – бинарная операция над односхемными отношениями. Пересечение односхемных отношений R и S есть подмножество кортежей, принадлежащих обоим отношениям; причём, имена полей односхемных отношений могут быть разными, достаточно, чтобы совпадало количество полей и типы данных соответствующих полей

7. Соединение (join). Соединения – бинарная операция над разносхемными отношениями. Эта операция определяет подмножество декартова произведения двух разносхемных отношений. Кортеж декартова произведения входит в результирующее отношение, если выполняется условие соединения F, которое задаёт соотношение значений атрибутов разных таблиц

8. Деление (division). Деление – бинарная операция над разносхемными отношениями. Пусть отношение R содержит атрибуты{r1,r2,...,rk, rk+1,...,rn}, а отношение S – атрибуты {rk+1,...,rn}. Тогда результирующее отношение R/S содержит атрибуты {r1,r2,...,rk}. Кортеж отношения R включается в результиру

1. ***Структура языка T-SQL. Идентификаторы. Операторы. Константы и***

***переменные. Типы данных.***

***T-SQL (Transact-SQL)*** - это язык программирования, который используется для работы с реляционными базами данных Microsoft SQL Server.

***Структура языка T-SQL***:

* Комментарии начинаются с символов "--" или "/" и заканчиваются символами "/".
* Идентификаторы используются для именования объектов в базе данных, таких как таблицы, столбцы, процедуры и т.д. Идентификаторы должны начинаться с буквы и могут содержать буквы, цифры и символы подчеркивания. Идентификаторы могут быть заключены в кавычки, если они содержат специальные символы или начинаются с цифры.
* Операторы используются для выполнения различных операций, таких как выборка данных из таблицы, вставка, обновление и удаление данных и т.д.
* Константы используются для задания фиксированных значений, которые не изменяются в процессе выполнения запроса. Константы могут быть числовыми, строковыми или логическими.
* Переменные используются для хранения значений, которые могут изменяться в процессе выполнения запроса. Переменные должны начинаться с символа "@" и могут быть любого типа данных.
* Типы данных определяют тип значений, которые могут быть хранимы в базе данных или в переменных. Некоторые из наиболее распространенных типов данных в T-SQL включают в себя целочисленные, десятичные, строковые, даты и временные, логические и т.д.

# ***Создание, удаление Команда CREATE DATABASE. Команда ALTER DATABASE. Команда DROP DATABASE.***

CREATE DATABASE создает базу данных с указанным именем. Ошибка происходит, если база данных уже существует, и Вы не определяли IF NOT EXISTS.

CREATE DATABASE [IF NOT EXISTS] db\_name

Базы данных в MySQL выполнены как каталоги, содержащие файлы, которые соответствуют таблицам в базе данных. Поскольку нет никаких таблиц в базе данных, когда она первоначально создана, инструкция CREATE DATABASE только создает каталог под каталогом данных MySQL.

DROP DATABASE [IF EXISTS] db\_name

DROP DATABASE удаляет все таблицы в базе данных и удаляет саму базу данных. Если выполнили DROP DATABASE на символически связанной базе данных, связь и первоначальная база данных будут удалены. Будьте ОЧЕНЬ осторожны с этой командой!

DROP DATABASE возвращает число файлов, которые были удалены из каталога баз данных. Обычно это кратное трем число потому, что каждая таблица соответствует файлам .MYD, .MYI и .frm.

Команда DROP DATABASE удаляет из заданного каталога базы данных все найденные файлы со следующими расширениями:

.BAK

.DAT

.HSH

.ISD

.ISM

.ISM

.MRG

.MYD

.MYI

.db

.frm

*ALTER {DATABASE | SCHEMA} [db\_name]*

*alter\_option ...*

*alter\_option: {*

*[DEFAULT] CHARACTER SET [=] charset\_name*

*| [DEFAULT] COLLATE [=] collation\_name*

*| [DEFAULT] ENCRYPTION [=] {'Y' | 'N'}*

*| READ ONLY [=] {DEFAULT | 0 | 1}*

*}*

ALTER DATABASE позволяет изменять общие характеристики базы данных. Эти характеристики хранятся в словаре данных. Эта инструкция требуется ALTER привилегия на база данных. ALTER SCHEMA является синонимом ALTER

DATABASE.

Если имя базы данных опущено, инструкция применяется к базе данных по умолчанию. В этом случае возникает ошибка, если нет базы данных по умолчанию.

Для любого alter\_option, исключенного из инструкции, база данных сохраняет свое текущее значение параметра, за исключением того, что изменение набор**а символов может изменить параметры сортировки и наоборот.**

1. ***Изменение БД. Команда ALTER DATABASE.***

ALTER {DATABASE | SCHEMA} [*db\_name*]

*alter\_option* ...

*alter\_option*: {

[DEFAULT] CHARACTER SET [=] *charset\_name*

| [DEFAULT] COLLATE [=] *collation\_name*

| [DEFAULT] ENCRYPTION [=] {'Y' | 'N'}

| READ ONLY [=] {DEFAULT | 0 | 1}

}

[ALTER DATABASE](https://dev.mysql.com/doc/refman/8.0/en/alter-database.html) позволяет изменять общие характеристики базы данных. Эти характеристики хранятся в словаре данных. Эта инструкция требуется [ALTER](https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_alter) привилегия на база данных. [ALTER SCHEMA](https://dev.mysql.com/doc/refman/8.0/en/alter-database.html) является синонимом [ALTER DATABASE](https://dev.mysql.com/doc/refman/8.0/en/alter-database.html).

Если имя базы данных опущено, инструкция применяется к базе данных по умолчанию. В этом случае возникает ошибка, если нет базы данных по умолчанию.

Для любого ***alter\_option***, исключенного из инструкции, база данных сохраняет свое текущее значение параметра, за исключением того, что изменение набора символов может изменить параметры сортировки и наоборот.

# ***Роли и объекты значений по умолчанию.***

**Объект по умолчанию и роль** – это объекты, которые могут ограничивать одно или несколько полей или типы, определенные пользователем, делая возможным создавать их однажды и использовать регулярно.

**Объект по умолчанию работает также как и ограничение**, но только этот объект принадлежит базе данных, а не таблице.

Если значение не указано, когда вы вставляете данные, для него будет использовано значение по умолчанию объекта, связанного с полем.

Объект значения по умолчанию создается следующим образом:

**CREATE DEFAULT имя**

**AS выражение константы**

После создания значения по умолчанию, вы должны связать его с колонкой или типом данных определенным пользователем с помощью вызова системной процедуры **sp\_bindefault.**

Для отключения значения по умолчанию выполните системную процедуру **sp\_unbindefault**.

Следующий пример помещает шаблон номера телефона, если не указано реальное значение:

**CREATE DEFAULT default\_phone\_no**

**AS ‘(000)00-00-00’**

Следующая команда связывает созданный объект с полем “Phone” таблицы TestTable:

**EXEC sp\_bindefault default\_ phone\_no, ‘TestTable.Phone’**

**Объекты значений по умолчанию имеют свои ограничения:**

• значения по умолчанию не применяются к уже существующим данным, а только к вновь введенным;

• можно использовать только простую логику с одной константой или функцией. Никакой логики здесь не может быть;

• тип данных значения должен соответствовать типу поля. Проверка этого произойдет только во время добавления новой строки, а не во время создания объекта, поэтому необходимо быть в этом моменте очень внимательным;

• у каждого поля может быть только одно значение по умолчанию. Это логично, ведь поле не может принять два значения. Если попытаться задать второе значение, то оно перезапишет уже существующее;

• не может быть обращения к другим таблицам на выборку данных;

• значение по умолчанию добавляется до проверок, поэтому если значение нарушает одно из ограничений, добавление записей станет невозможным;

• для удаления значения по умолчанию, его сначала нужно отвязать от объектов базы данных, к которым оно привязано.

**Теперь переходим к ролям.**

**Роли указывают доступные значения, которые вы можете вставить в колонку.**

Они гарантируют, что данные подпадают под определенный ряд значений, соответствуют определенному шаблону, или соответствуют определенному списку.

Объявление роли может содержать любые выражения, которые действительны для оператора WHERE, который мы будем внимательно рассматривать.

В общем виде роль выглядит следующим образом:

**CREATE RULE имя**

**AS выражение условия**

После создания роли, вы должны связать его с колонкой или типом данных определенным пользователем с помощью **вызова системной процедуры sp\_bindrule**.

Для отключения правила выполните системную процедуру **sp\_unbinrule.**

В следующем примере, правило гарантирует, что поле "Pol" может содержать только букву М или Ж:

***CREATE RULE rule\_pol***

***AS @Pol IN ('М', 'Ж'***)

Следующий пример связывает созданную роль с полем " Pol" таблицы TestTable:

***EXEC sp\_bindrule rule\_pol, 'TestTable.Pol'***

Для удаления значения по умолчанию из базы данных используйте оператор DROP:

***DROP DEFAULT имя [,…] DROP RULE имя [,…]***

Например, созданную ранее роль можно удалить командой:

***DROP RULE rule\_pol***

**Правила являются достаточно мощным решением, но при этом они обладают достаточно большим количеством ограничений:**

• новое правило не может повлиять на уже существующие значения, потому что оно срабатывает только при добавлении или изменении строк данных;

• в правилах можно использовать только не сложные вычисления с константами и функциями MS SQL Server;

• в отличии от ограничений, нельзя сравнивать значения полей;

• в правиле нельзя обращаться к таблицам для выборки данных. Чтобы была возможность для выборки, стоит обратить внимание на триггеры, которые мы будем рассматривать в 3-й главе;

• с одним столбцом можно связать только одно правило. Если бы можно было связывать два правила, то у нас появилась хоть какая-то возможность создавать более сложные запреты, но пока этого нет. Если попытаться связать правило со столбцом, у которого уже есть правило, то старое значение будет заменено новым;

• вполне логичное ограничение – тип данных, используемый в правиле должен совпадать с типом поля;

• связанное правило нельзя удалить. Необходимо сначала удалить связь.

1. ***T-SQL. Создание таблиц. Команда CREATE TABLE.***

Для создания таблиц применяется команда CREATE TABLE. С этой командой можно использовать ряд операторов, которые определяют столбцы таблицы и их атрибуты. И кроме того, можно использовать ряд операторов, которые определяют свойства таблицы в целом. Одна база данных может содержать до 2 миллиардов таблиц.

**Общий синтаксис создания таблицы выглядит следующим образом:**

CREATE TABLE название\_таблицы

(название\_столбца1 тип\_данных атрибуты\_столбца1,

название\_столбца2 тип\_данных атрибуты\_столбца2,

................................................

название\_столбцаN тип\_данных атрибуты\_столбцаN,

атрибуты\_таблицы

)

После команды CREATE TABLE идет название создаваемой таблицы. Имя таблицы выполняет роль ее идентификатора в базе данных, поэтому оно должно быть уникальным. Имя должно иметь длину не больше 128 символов. Имя может состоять из алфавитно-цифровых символов, а также символов $ и знака подчеркивания.Причем первым символом должна быть буква или знак подчеркивания.

Имя объекта не может включать пробелы и не может представлять одно из ключевых слов языка Transact-SQL. Если идентификатор все же содержит пробельные символы, то его следует заключать в кавычки. Если необходимо в качестве имени использовать ключевые слова, то эти слова помещаются в квадратные скобки.

После имени таблицы в скобках указываются параметры всех столбцов и в самом конце атрибуты, которые относятся ко всей таблице. Атрибуты столбцов и атрибуты таблицы являются необязательными компонентами, и их можно не указывать.

В самом просто виде команда CREATE TABLE должна содержать как минимум имя таблицы, имена и типы столбцов.

Таблица может содержать от 1 до 1024 столбцов. Каждый столбец должен иметь уникальное в рамках текущей таблицы имя, и ему должен быть назначен тип данных.

1. ***T-SQL. Первичный и внешний ключи. Резервные ключи. Ограничения PRIMARY KEY, NULL, NOT NULL, UNIQUE, FOREIGN KEY, CHECK.***

При создании таблицы желательно, чтобы она имела уникальный столбец или же совокупность столбцов, которая уникальна для каждой ее строки – по данному уникальному значению можно однозначно идентифицировать запись. Такое значение называется **первичным ключом таблицы**. Ограничение первичного ключа (PRIMARY KEY) обеспечивает уникальность строк и запрещает хранить значения NULL в соответствующих столбцах. Каждый уникальный набор значений в атрибутах, составляющих ограничение, может появиться в таблице только один раз — не более чем в одной строке. СУБД пресекает любые попытки определить ограничения первичного ключа для столбцов, которые способны содержать пустые значения. Если первичный ключ состоит из нескольких полей, то эти поля необходимо перечислить в скобках через запятую. **Внешний ключ (foreign key)** - это столбец (или группа столбцов таблицы), содержащий значения, совпадающие со значениями первичного ключа в этой же или другой таблице. Внешний ключ определяется с помощью предложения **FOREIGN KEY** в комбинации с предложением **REFERENCES.**Ограничение UNIQUE обеспечивает уникальность строк и реализует такую концепцию реляционной модели, как **резервные ключи.** В отличие от первичных ключей в одной таблице возможно присутствие нескольких ограничений UNIQUE. Кроме того, уникальность распространяется на все столбцы, включая те, которые могут иметь пустые значения.

**CONSTRAINT** Необязательное ключевое слово, указывающее на начало определения ограничения PRIMARY KEY, NOT NULL, UNIQUE, FOREIGN KEY или CHECK.

constraint\_name Имя ограничения.

* Имена ограничений должны быть уникальными в пределах схемы, к которой принадлежит таблица. **NULL | NOT NULL** Определяет, допустимы ли для столбца значения NULL. Параметр NULL не является ограничением в строгом смысле слова, но может быть указан так же, как и NOT NULL. Ограничение NOT NULL может быть указано для вычисляемых столбцов только в случае, если одновременно указан параметр PERSISTED.
* PRIMARY KEY Ограничение, которое обеспечивает целостность сущностей для указанного столбца или столбцов с помощью уникального индекса. Можно создать только одно ограничение PRIMARY KEY для таблицы
* . UNIQUE Ограничение, которое обеспечивает целостность сущностей для указанного столбца или столбцов с помощью уникального индекса. В таблице может быть несколько ограничений UNIQUE.
* Ограничения FOREIGN KEY требуют, чтобы каждое значение в столбце существовало в соответствующем связанном столбце или столбцах в связанной таблице. Ограничения FOREIGN KEY могут ссылаться только на столбцы, являющиеся ограничениями PRIMARY KEY или UNIQUE в связанной таблице или на столбцы, на которые имеются ссылки в индексе UNIQUE INDEX связанной таблицы. Внешние ключи в вычисляемых столбцах должны быть также помечены как PERSISTED. [ [ schema\_name. ] referenced\_table\_name ] Имя таблицы, на которую ссылается ограничение FOREIGN KEY, и схема, к которой она принадлежит. ( ref\_column [ ,... n ] ) Столбец или список столбцов из таблицы, на которую ссылается ограничение FOREIGN KEY.
* CHECK Ограничение, обеспечивающее целостность домена путем ограничения возможных значений, которые могут быть введены в столбец или столбцы. Ограничения CHECK в вычисляемых столбцах должны быть также помечены как PERSISTED.**Важно! Рекомендуется указывать параметр NOT NULL для столбца секционирования секционированных таблиц, а также для несекционированных таблиц, являющихся источниками или целями для операций ALTER TABLE...SWITCH. В результате любые ограничения CHECK, заданные для столбцов секционирования, не будут проверять значения NULL.**

1. ***Указание значений по умолчанию для столбца.***

**DEFAULT** Указывает значение, присваиваемое столбцу в случае отсутствия явно заданного значения при вставке. Определения DEFAULT могут применяться к любым столбцам, кроме имеющих тип timestamp или обладающих свойством IDENTITY. Если значение по умолчанию указывается для столбца определяемого пользователем типа, этот тип должен поддерживать неявное преобразование выражения constant\_expression в определяемый пользователем тип. Определения DEFAULT удаляются, когда таблица удаляется из памяти. В качестве значения по умолчанию могут использоваться только константы (например, символьные строки), скалярные функции (системные, определяемые пользователем или функции CLR) или значение NULL

(не особо уверена,что это надо,но пускай будет). DEFAULT, определения Столбец может иметь только определение DEFAULT. Ограничение DEFAULT может содержать значения констант, функции, стандартные функции без параметров SQL или значение NULL. В следующей таблице приведены функции без параметров и возвращаемые ими по умолчанию значения в процессе выполнения инструкции INSERT. Функция без параметров SQL-92 Возвращенное значение CURRENT\_TIMESTAMP Текущие дата и время. CURRENT\_USER Имя пользователя, выполняющего вставку. SESSION\_USER Имя пользователя, выполняющего вставку. SYSTEM\_USER Имя пользователя, выполняющего вставку. ПользовательИмя пользователя, выполняющего вставку. Выражение constant\_expression в определении DEFAULT не может ссылаться на другой столбец таблицы, а также на другие таблицы, представления или хранимые процедуры. Определения DEFAULT нельзя создавать для столбцов с типом данных timestamp или столбцов со свойством IDENTITY. Определения DEFAULT нельзя создавать для столбцов с псевдонимами типов данных, если такой тип привязан к определенному по умолчанию объекту.

1. ***Установка ограничений на уровне столбца, таблицы.***

**PRIMARY KEY, ограничения:**

* В таблице возможно наличие только одного ограничения по первичному ключу.
* Индекс, формируемый ограничением PRIMARY KEY, не может привести к выходу количества индексов в таблице за пределы в 999 некластеризованных индексов и 1 кластеризованный. Если для ограничения PRIMARY KEY не указан параметр CLUSTERED или NONCLUSTERED, применяется параметр CLUSTERED, если для ограничения UNIQUE не определено кластеризованных индексов.
* Все столбцы с ограничением PRIMARY KEY должны иметь признак NOT NULL. Если допустимость значения NULL не указана, то для всех столбцов c ограничением PRIMARY KEY устанавливается признак NOT NULL.

**Ограничения UNIQUE**

* Если для ограничения UNIQUE не указан параметр CLUSTERED или NONCLUSTERED, по умолчанию применяется параметр NONCLUSTERED. Каждое ограничение уникальности создает индекс.
* Количество ограничений UNIQUE не может привести к выходу количества индексов в таблице за пределы в 999 некластеризованных индексов и 1 кластеризованный.
* Если ограничение уникальности определено на столбце определяемого пользователем типа данных CLR, реализация этого типа должна поддерживать двоичную сортировку или сортировку на основе оператора.

**Ограничения FOREIGN KEY**

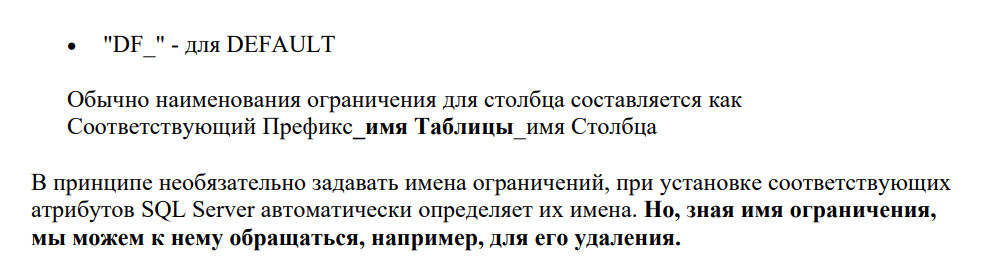
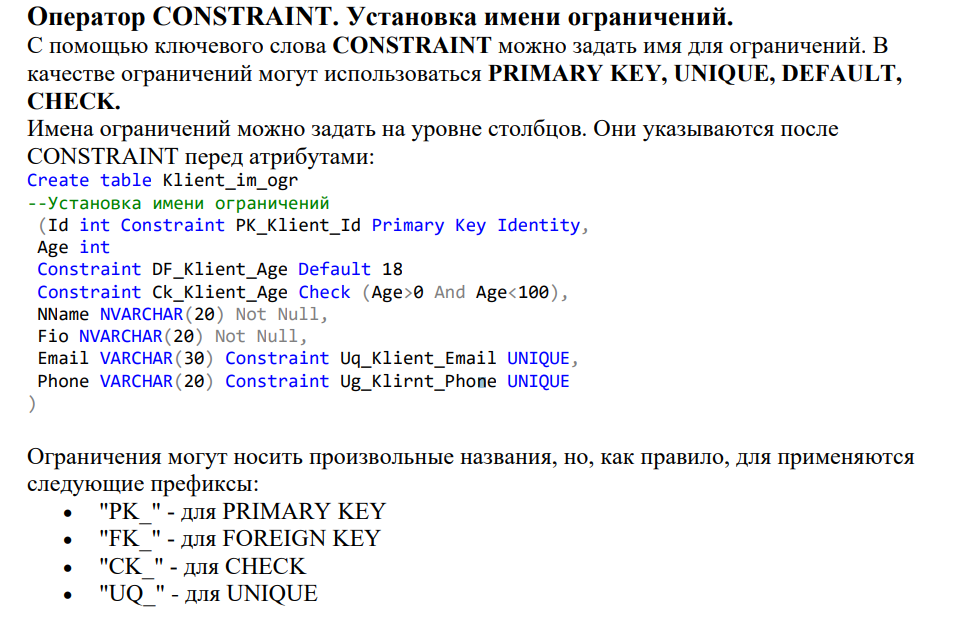
* Если столбцу, имеющему ограничение внешнего ключа, задается значение, отличное от NULL, такое же значение должно существовать и в указываемом столбце; в противном случае будет возвращено сообщение о нарушении внешнего ключа.
* Если не указаны исходные столбцы, ограничения FOREIGN KEY применяются к предшествующему столбцу.
* Ограничения FOREIGN KEY могут ссылаться только на таблицы в пределах той же базы данных на том же сервере. Межбазовую ссылочную целостность необходимо реализовать посредством триггеров.
* Ограничения FOREIGN KEY могут ссылаться на другие столбцы той же таблицы. Это называется самовызовом.
* Компонент Database Engine не имеет стандартного предела на количество ограничений FOREIGN KEY, содержащихся в таблице, ссылающейся на другие таблицы, или на количество ограничений FOREIGN KEY в других таблицах, ссылающихся на указанную таблицу. Тем не менее фактическое количество ограничений FOREIGN KEY, доступных для использования, ограничивается конфигурацией оборудования, базы данных и приложения. Рекомендуется, чтобы таблица содержала не более 253 ограничений FOREIGN KEY, а также чтобы на нее ссылалось не более 253 ограничений FOREIGN KEY. Предел эффективности в конкретном случае может более или менее зависеть от приложения и оборудования. При разработке базы данных и приложений следует учитывать стоимость принудительных ограничений FOREIGN KEY.
* Ограничения FOREIGN KEY не применяются к временным таблицам.
* Ограничения FOREIGN KEY могут ссылаться только на столбцы с ограничениями PRIMARY KEY или UNIQUE в таблице, на которую указывает ссылка, или на столбцы уникального индекса (UNIQUE INDEX) такой таблицы. Если внешний ключ определен на столбце определяемого пользователем типа данных CLR, реализация этого типа должна поддерживать двоичную сортировку. Столбцы, участвующие в связи по внешнему ключу, должны иметь одинаковую длину и масштаб.

**CHECK, ограничение**

* Столбец может содержать любое количество ограничений CHECK, а условие может включать несколько логических выражений, соединенных операторами AND и OR. При указании нескольких ограничений CHECK для столбца их проверка производится в порядке создания.
* Условие поиска должно возвращать логическое выражение и не может ссылаться на другую таблицу.
* Ограничение CHECK уровня столбца может ссылаться только на ограничиваемый столбец, а ограничение CHECK уровня таблицы — только на столбцы этой таблицы.
* Правила и ограничения CHECK выполняют одну и ту же функцию проверки данных при выполнении инструкций INSERT и UPDATE.
* Если для столбца или столбцов задано правило либо одно или несколько ограничений CHECK, применяются все ограничения.
* Ограничения CHECK нельзя определять для столбцов типов text, ntext или image.

**Правила допустимости значения NULL в рамках определения таблицы** Допустимость значений NULL для столбца определяет, разрешает ли столбец использовать такие значения для своих данных. Значение NULL не является нулевым или пустым. NULL означает, что никаких данных не вводилось либо значение NULL было указано явно. Обычно оно подразумевает, что значение неизвестно или неприменимо.

1. ***Имена ограничений. Отключение ограничений.***

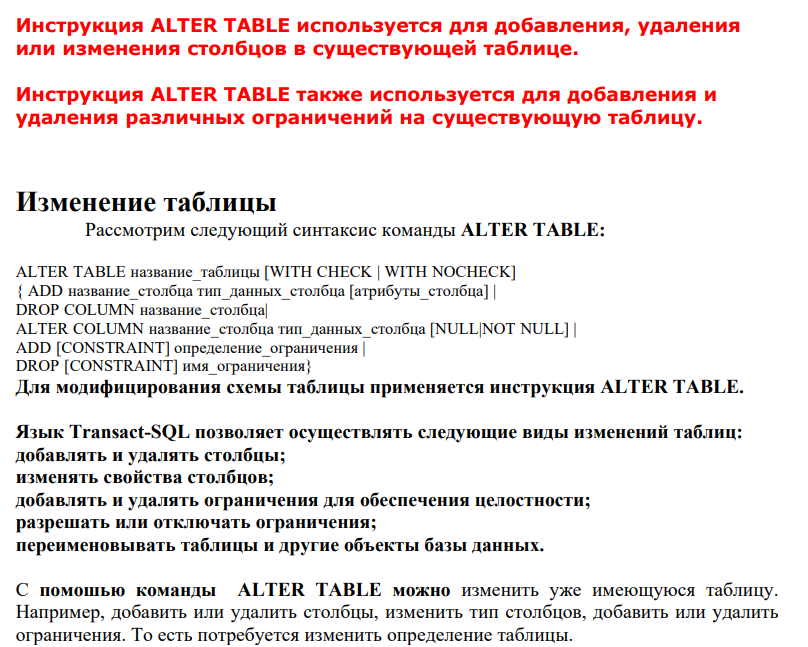
****

1. ***Создание индексов в SQL Server.***

INDEX index\_name [ CLUSTERED | NONCLUSTERED ] ( column\_name [ ASC | DESC ] [ ,... n ] ) Задает создание индекса для таблицы. Это может быть кластеризованный или некластеризованный индекс. Индекс будет содержать указанные столбцы и сортировать данные по возрастанию или убыванию. **INDEX index\_name CLUSTERED COLUMNSTORE** Задает сохранение всей таблицы в виде столбцов с кластеризованным индексом columnstore. Сюда всегда входят все столбцы в таблице. Данные не сортируются в алфавитном или числовом порядке, поскольку организация строк позволяет использовать преимущества сжатия columnstore.

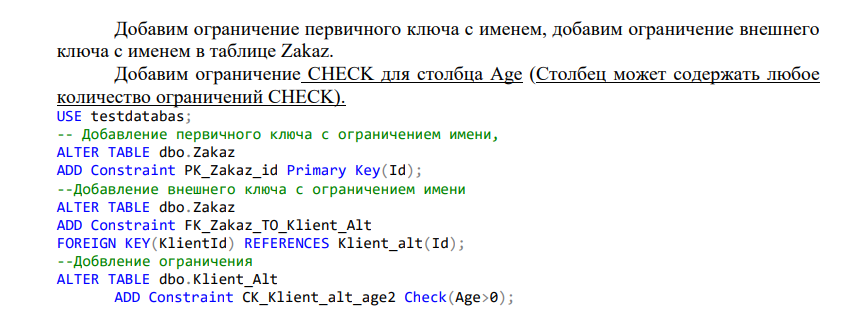
**INDEX index\_name [ NONCLUSTERED ] COLUMNSTORE** ( column\_name [ ,... n ] ) Задает создание некластеризованного индекса columnstore в таблице. Базовая таблица может быть кучей rowstore или кластеризованным индексом или кластеризованным индексом columnstore. В любом случае при создании некластеризованного индекса columnstore в таблице сохраняется вторая копия данных для столбцов в индексе.

1. ***Изменение таблицы. Команда ALTER TABLE***

****

1. ***Добавить, удалить ограничения. Добавление ограничения CHECK.***

**Добавление ограничений с именами**

При добавлении ограничений мы можем указать для них имя, используя оператор CONSTRAINT, после которого указывается имя ограничения:

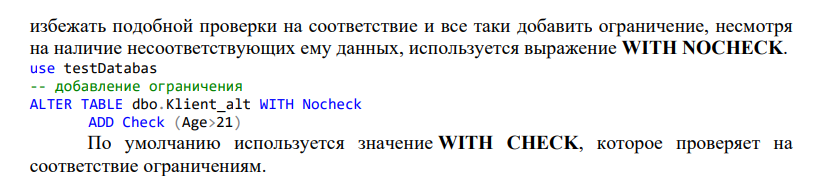
**Удаление ограничений**

Для удаления ограничений необходимо знать их имя. Если мы точно не знаем имя ограничения, то его можно узнать через SQL Server Management Studio Напоминание: Названия ограничений внешних ключей начинаются с "FK". Названия ограничений CHECK начинаются с "CK", а ограничений DEFAULT - с "DF". Далее выполним команду sp\_help Names Команда sp\_help (точнее это процедура SQL сервера) отображает подробную информацию о указанной таблице, в данном случае это таблица Names.

USE testdatabas;

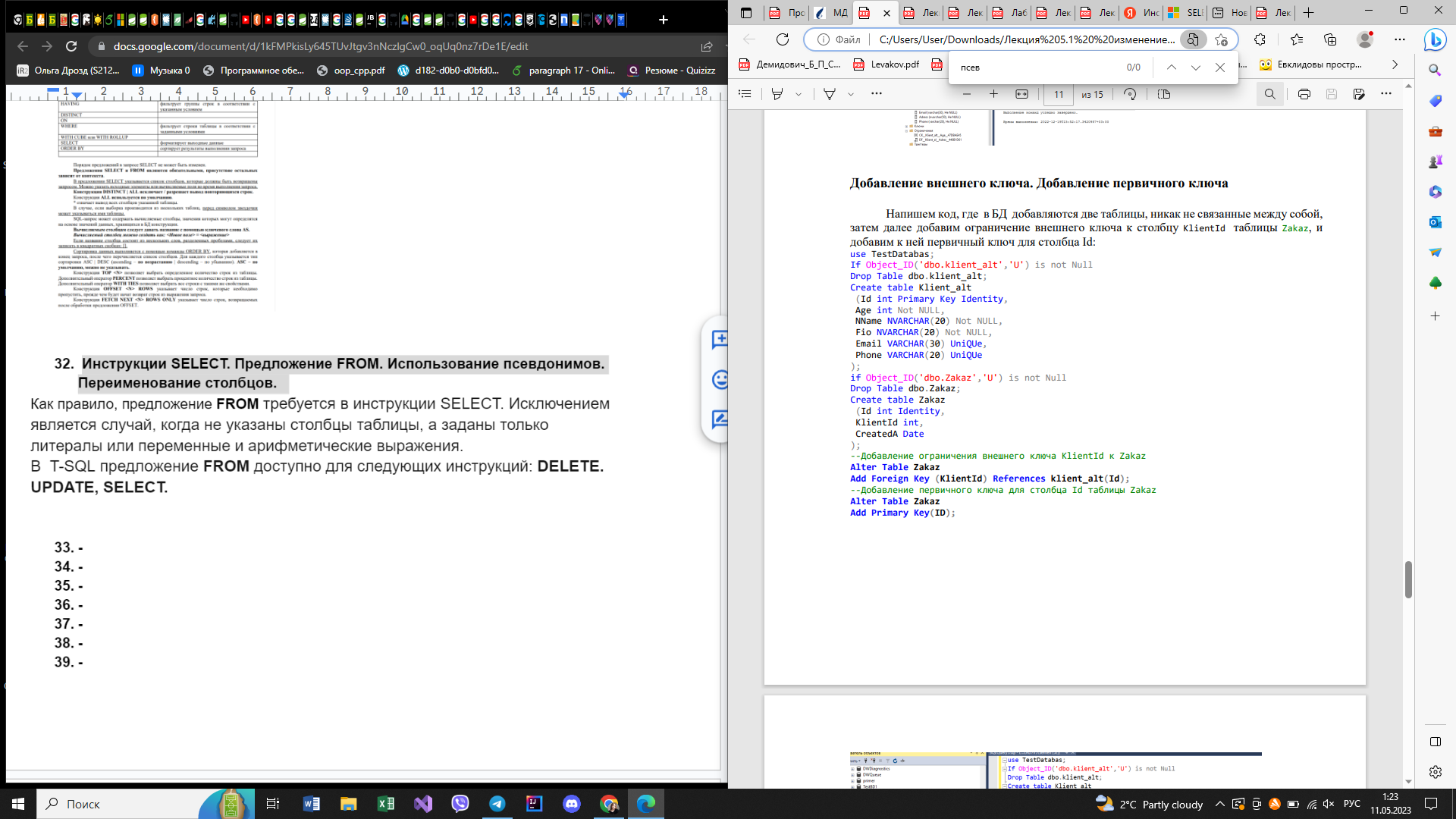
exec sp\_help zakaz



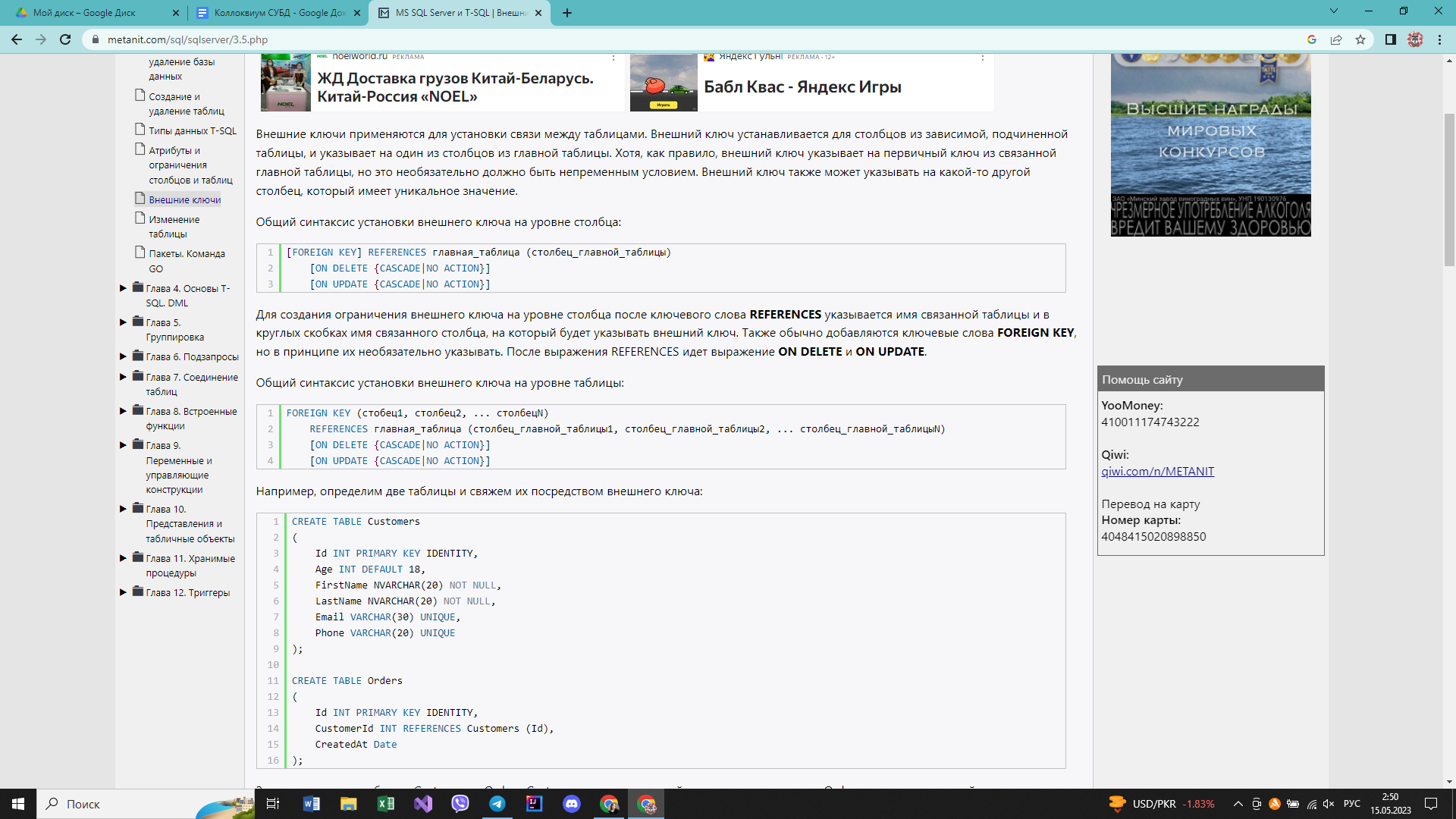


1. ***Добавление внешнего ключа. Добавление первичного ключа***

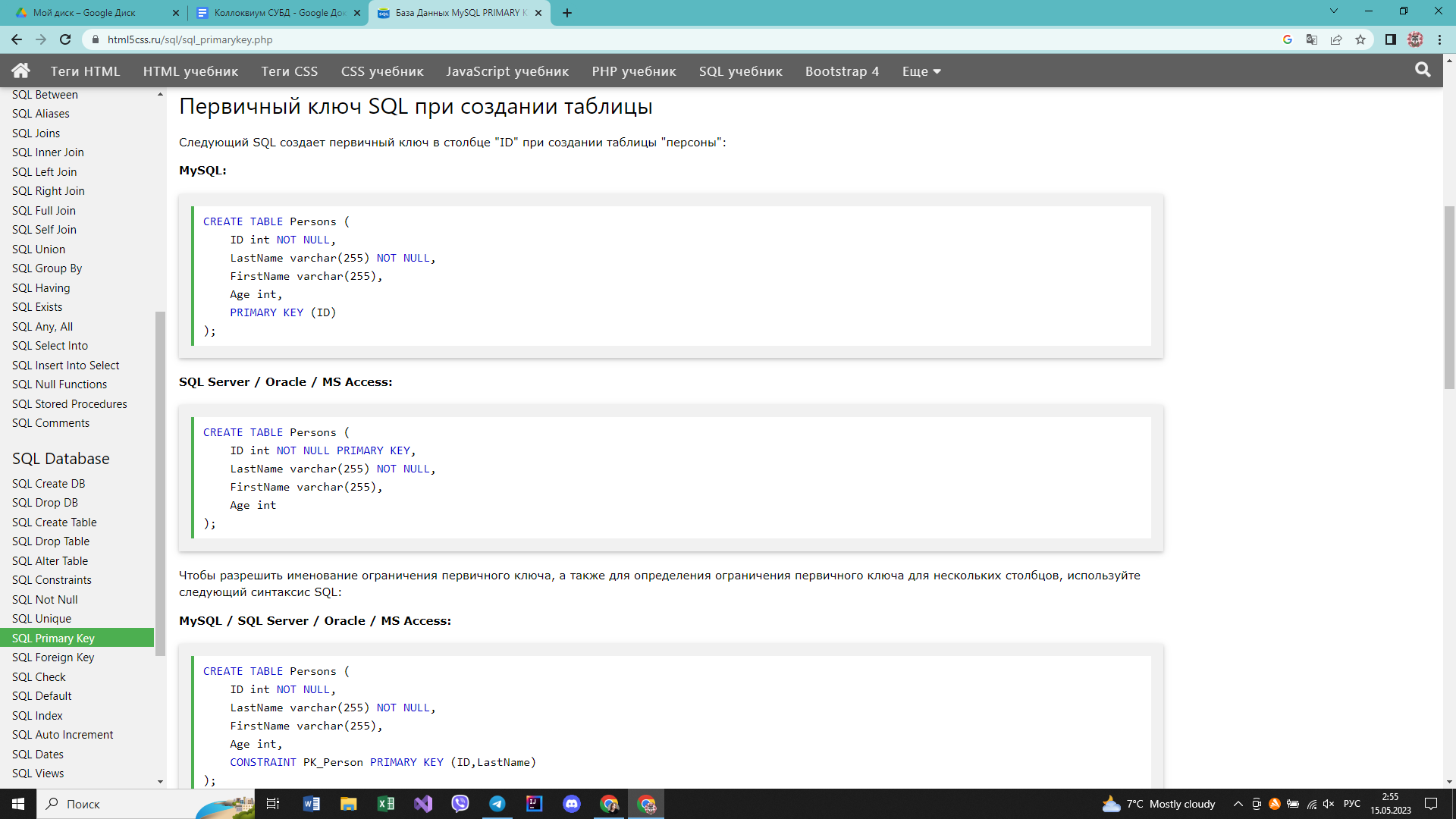
(смотреть последние 6 строк, с учетом комментов)

****

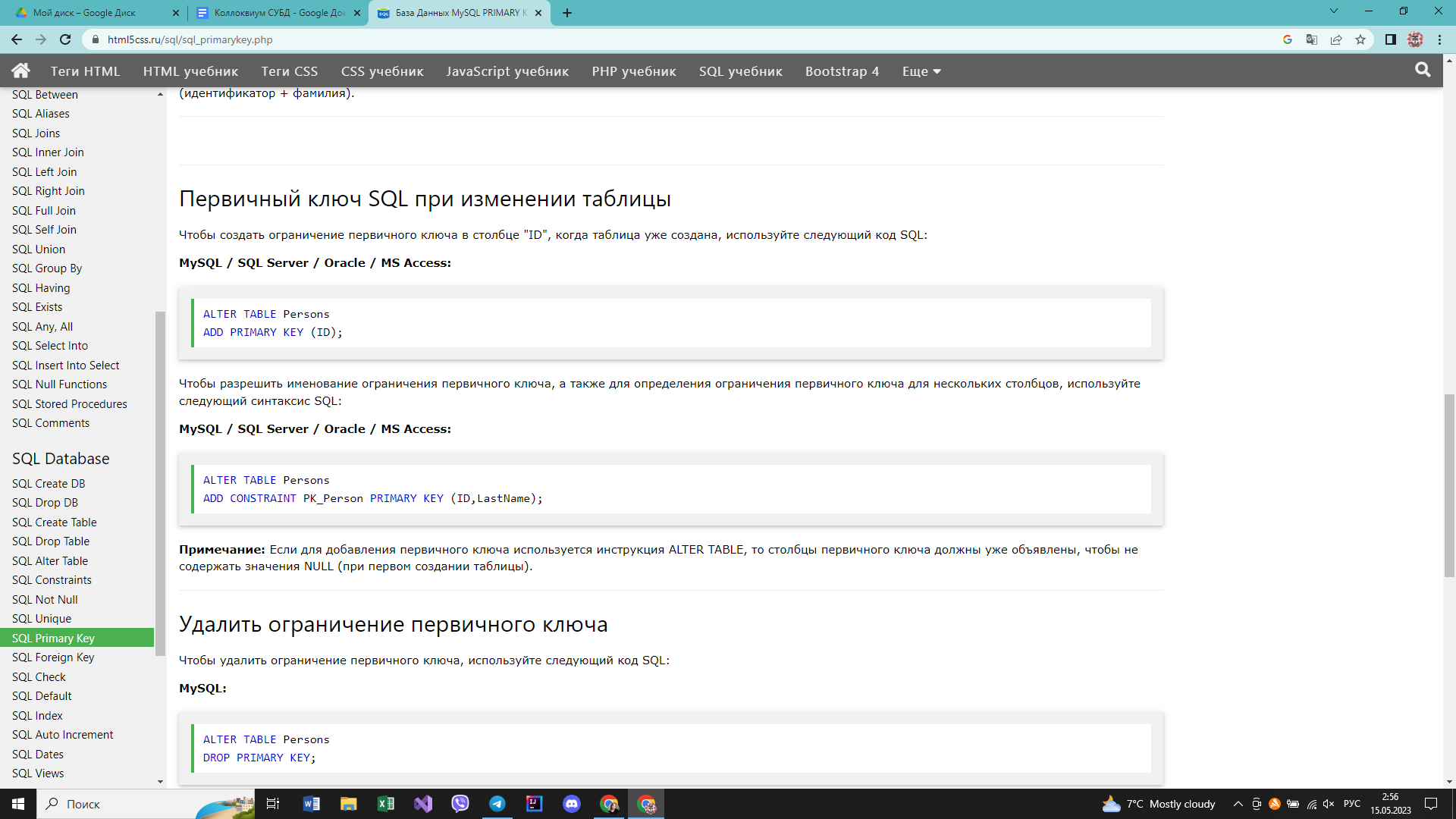
**Либо:**

****

Первичный ключ при создании таблицы:

****

Первичный ключ при изменении таблицы:

****

1. ***Обеспечение целостности данных: целостность таблицы, ссылочная целостность.***

**Типы целостности данных:**

**• Целостность полей –** указывает набор значений данных, которые являются правильными для поля, и определяет, возможно ли использование нулевого значения. Целостность полей лучше всего обеспечивается с помощью ограничения CHECK, формата (с помощью шаблона) или региона возможных значений для поля.

**• Целостность таблицы –** необходимо, чтобы все строки в таблице имели уникальный идентификатор, называемый первичным ключом. Может ли первичный ключ изменяться, или может ли строка удаляться, зависит от уровня целостности.

**• Целостность ссылок –** подразумевает отношения между первичным ключом (таблицы, на которую ссылаются) и внешним ключом (таблицы, которая ссылается на другую) всегда защищенными. Строка основной таблицы, на которую ссылаются, не может быть удалена и первичный ключ не может быть изменен, если вторичный ключ ссылается на строку, пока не будет уничтожена связь. Иначе связь нарушается и восстановить ее потом становится проблематичным.

**Описанная целостность данных –** критерии, которые данные должны содержать как часть описания объекта и после этого SQL Server автоматически гарантирует, что данные соответствуют критериям. Такая целостность обеспечивается с помощью ограничений CHECK, DEFAULT и внешнего ключа.

**Предшествующая целостность данных –** это программа, которая определяет критерии, которым должны соответствовать данные. Этот метод обеспечивается с помощью процедур и триггеров , которые могут выполняться на сервере или с помощью кода программ в клиентском приложении.

**Ограничение –** это основной метод обеспечения целостности данных. Они обеспечивают правильность данных, введенных в поле, и какие отношения обеспечиваются между таблицами. Ограничения могут создаваться во время создания таблицы (CREATE TABLE) или редактирования (ALTER TABLE). Если ограничение назначается отдельному полю, оно называется ограничения уровня поля. Если ограничение ссылается на несколько полей, оно называется ограничением уровня таблицы, даже если оно ссылается не на все колонки таблицы. Имена ограничений должны быть уникальными для базы данных. Если не указывать имена, то сервер сгенерирует значение самостоятельно.

**Ограничение DEFAULT** помещает значение в колонку, когда оно не было указано в операторе INSERT. Оно относится только к оператору добавления записи (INSERT) и не срабатывает во время изменения полей (оператор UPDATE).

**Ограничение CHECK** ограничивает данные, которые пользователь может ввести в определенную колонку указанными значениями.

**Ограничение PRIMARY KEY** определяет первичный ключ таблицы, который уникально идентифицирует строку. Это гарантирует целостность таблицы.

**Ограничение FOREIGN KEY (внешний ключ)** гарантирует ссылочную целостность. Ограничение внешнего ключа определяет ссылку на колонку с первичным ключом или уникальную колонку в этой же или другой таблице. С помощью такого ключа обеспечивается целостность связей между таблицами.

**Ограничение внешнего ключа включает опцию CASCADE, которая позволяет любые изменения сделанные в уникальной колонке или первичном ключе автоматически переносить в значение внешнего ключа. Такое действие называется целостностью каскадных ссылок.**

**Опция REFERENCE команд CREATE TABLE и ALTER TABLE поддерживаю опции ON DELETE и ON UPDATE. Эти опции позволят вам указать опции CASCADE и NO ACTION: [ ON DELETE { CASCADE | NO ACTION } ] [ ON UPDATE { CASCADE | NO ACTION } ] NO ACTION указывает что любые попытки удалить или обновить ключ, на который ссылается вторичный ключ в другой таблице заканчиваются ошибкой, и изменения откатываются. Это значение по умолчанию и без особой надобности не стоит включать каскадных действий.**

**Ограничение UNIQUE (уникальность)** указывает, что две строки в колонке не могут содержать одно и тоже значение. Это ограничение обеспечивает целостность таблицы с уникальным индексом.

1. ***Модификация данных. Добавление данных в таблицу средствами TSQL. Использование инструкции INSERT.***

Если мы хотим с помощью TSQL добавить в таблицу данные, то используем функцию INSERT.

Общий синтаксис: **INSERT [INTO] имя\_таблицы [(список\_столбцов)] VALUES (значение1, значение2, ….,значениеN)**

Вначале идет выражение **INSERT [INTO]** (into необязательное слово), потом в скобках можно указать список столбцов через запятую, либо же название таблицы, потом слово VALUES и в скобках перечисляем добавляемые для столбцов значения. Эти значения для столбцов должны передаваться в таблицу по порядку их объявления.

При использовании данной формы оператора INSERT **список VALUES должен содержать количество значений, равное количеству полей таблицы**. И **тип данных каждого из значений**, указываемых в списке VALUES, **должен совпадать** с типом данных поля, соответствующего этому значению.

Значения, относящиеся к символьным типам и датам должны быть в ‘’(кавычках/апострофах).

1. ***Изменение и удаление данных в таблице средствами T-SQL. Использование инструкций UPDATE, DELETE.***

**Для изменения данных в записях таблицы используется оператор UPDATE.**

Общая форма**:UPDATE table\_name {SET column\_1 = {expression \ DEFAULT \ NULL} [,.. n] [FROM table\_name1 [,....n]] [WHERE condition]**

Строки таблицы table\_name выбираются для изменения **в соответствии с условием** в предложении WHERE.

Значения столбцов каждой изменяемой строки **изменяются** с помощью предложения **SET** инструкции UPDATE, которое соответствующему столбцу присваивает выражение (обычно) или константу.

Если предложение **WHERE** отсутствует, то инструкция **UPDATE** модифицирует все строки таблицы.

Для изменения уже имеющихся строк в таблице следующий формальный синтаксис:

**UPDATE имя\_таблицы SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN [FROM выборка AS псевдоним\_выборки] [WHERE условие\_обновления]**

Для удаления применяется команда **DELETE**:

**DELETE [FROM] имя\_таблицы WHERE условие\_удаления**.

Если необходимо вовсе удалить все строки вне зависимости от условия, то условие можно не указывать.

Если мы говорим об удалении из связанных таблиц, то нельзя удалять сразу из двух таблиц, может быть только одна. После оператора DELETE необходимо явно указать, из какой из двух происходит удаление

1. ***Извлечение данных средствами T-SQL. Общий синтаксис инструкции SELECT и алгоритм ее выполнения.***

Возвращает строки из базы данных и позволяет делать выборку одной или нескольких строк или столбцов из одной или нескольких таблиц в SQL Server. Полный синтаксис инструкции SELECT сложен, однако основные предложения можно кратко описать следующим образом:

**[ WITH { [ XMLNAMESPACES ,]<common\_table\_expression> ] } ]**

**SELECT выбранный\_список [ INTO новая\_таблица ]**

**[ FROM источник\_таблицы ] [ WHERE условие\_поиска ]**

**[ GROUP BY выражение\_группирования ]**

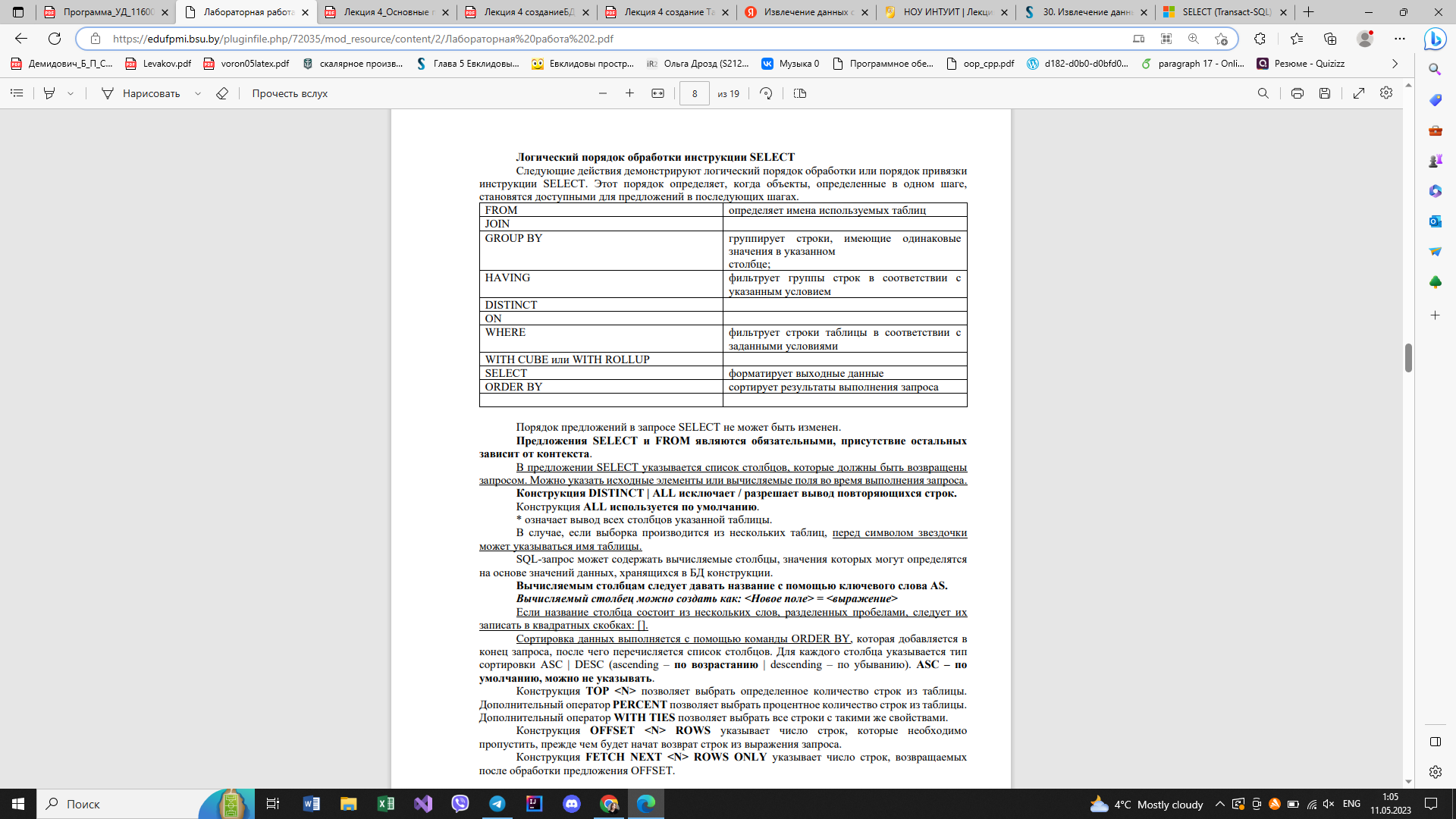
**[ WINDOW выражение\_окна]**

**[ HAVING условие\_поиска ]**

**[ ORDER BY выражение\_упорядочения [ ASC | DESC ]**

**Операторы UNION, EXCEPT и INTERSECT можно использовать между запросами, чтобы сравнить их результаты или объединить в один результирующий набор.**

Порядок предложений в инструкции SELECT имеет значение. Любое из необязательных предложений может быть опущено; но если необязательные предложения используются, они должны следовать в определенном порядке.

****

1. ***Инструкции SELECT. Предложение FROM. Использование псевдонимов. Переименование столбцов.***

Как правило, предложение **FROM** требуется в инструкции SELECT. Исключением является случай, когда не указаны столбцы таблицы, а заданы только литералы или переменные и арифметические выражения.

В T-SQL предложение **FROM** доступно для следующих инструкций: **DELETE. UPDATE, SELECT.**

Имена столбцов, указанные в предложении SELECT, можно переименовать. Это делает результаты более читабельными, поскольку имена полей в таблицах часто сокращают с целью упрощения набора. Для переименования используется ключевое слово AS, хотя согласно стандарту его можно и опустить, так как оно неявно подразумевается.

Переименование особенно желательно при использовании в предложении SELECT выражений для вычисления значения. Эти выражения позволяют получать данные, которые не находятся непосредственно в таблицах. Если выражение содержит имена столбцов таблицы, указанной в предложении FROM, то выражение подсчитывается для каждой строки выходных данных.

1. ***Инструкции SELECT. Выборка с добавлением (SELECT INTO).***

**Оператор SELECT INTO используется для создания новой таблицы на основе результатов запроса SELECT. Синтаксис оператора SELECT INTO выглядит следующим образом:**

**SELECT column1, column2, ...**

**INTO new\_table**

**FROM source\_table**

**WHERE condition;**

**Здесь column1, column2, ... - это список столбцов, которые необходимо выбрать из таблицы source\_table. new\_table - это имя новой таблицы, которая будет создана на основе результатов запроса. condition - это условие, которое определяет, какие строки из таблицы source\_table будут выбраны.**

**Пример использования оператора SELECT INTO:**

**SELECT first\_name, last\_name, email**

**INTO customers\_backup**

**FROM customers**

**WHERE status = 'inactive';**

**В этом примере мы выбираем имена, фамилии и адреса электронной почты всех неактивных клиентов из таблицы customers и сохраняем результаты в новую таблицу customers\_backup.**

**Важно отметить, что оператор SELECT INTO создает новую таблицу с теми же типами данных и ограничениями, что и исходная таблица. Если необходимо изменить типы данных или ограничения, необходимо создать таблицу вручную и использовать оператор INSERT INTO для добавления данных.**

1. ***Инструкции SELECT. Выбор уникальных строк (SELECT DISTINCT).***

**Оператор SELECT DISTINCT используется для выбора уникальных значений из столбца или комбинации столбцов в таблице. Синтаксис оператора SELECT DISTINCT выглядит следующим образом:**

**SELECT DISTINCT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

**Здесь column1, column2, ... - это список столбцов, для которых необходимо выбрать уникальные значения. table\_name - это имя таблицы, из которой необходимо выбрать данные. condition - это условие, которое определяет, какие строки из таблицы будут выбраны.**

**Пример использования оператора SELECT DISTINCT:**

**SELECT DISTINCT department**

**FROM employees**

**WHERE salary > 50000;**

**В этом примере мы выбираем уникальные значения из столбца department таблицы employees, где зарплата сотрудника больше 50000.**

**Важно отметить, что оператор SELECT DISTINCT может замедлить выполнение запроса, особенно если таблица содержит большое количество данных. Поэтому его следует использовать с осторожностью и только тогда, когда это действительно необходимо.**

1. ***Извлечение диапазона строк. Фильтры TOP и OFFSET-FETCH***

**Операторы TOP и OFFSET-FETCH используются для извлечения диапазона строк из таблицы в SQL Server.**

**Оператор TOP позволяет выбрать определенное количество строк из начала результата запроса. Синтаксис оператора TOP выглядит следующим образом:**

**SELECT TOP number|percent column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

**Здесь number - это количество строк, которые необходимо выбрать, а percent - это процент строк, которые необходимо выбрать. column1, column2, ... - это список столбцов, которые необходимо выбрать. table\_name - это имя таблицы, из которой необходимо выбрать данные. condition - это условие, которое определяет, какие строки из таблицы будут выбраны.**

**Пример использования оператора TOP:**

**SELECT TOP 10 first\_name, last\_name**

**FROM employees**

**ORDER BY hire\_date DESC;**

**В этом примере мы выбираем 10 последних нанятых сотрудников из таблицы employees и выводим их имена и фамилии.**

**Оператор OFFSET-FETCH позволяет выбрать диапазон строк из таблицы, начиная с определенной позиции. Синтаксис оператора OFFSET-FETCH выглядит следующим образом:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition**

**ORDER BY column\_name**

**OFFSET offset\_row\_count {ROW | ROWS}**

**FETCH {FIRST | NEXT} fetch\_row\_count {ROW | ROWS} ONLY;**

**Здесь column1, column2, ... - это список столбцов, которые необходимо выбрать. table\_name - это имя таблицы, из которой необходимо выбрать данные. condition - это условие, которое определяет, какие строки из таблицы будут выбраны. column\_name - это имя столбца, по которому необходимо отсортировать результаты. offset\_row\_count - это количество строк, которые необходимо пропустить. fetch\_row\_count - это количество строк, которые необходимо выбрать.**

**Пример использования оператора OFFSET-FETCH:**

**SELECT first\_name, last\_name**

**FROM employees**

**ORDER BY hire\_date DESC**

**OFFSET 10 ROWS**

**FETCH NEXT 5 ROWS ONLY;**

**В этом примере мы выбираем 5 сотрудников из таблицы employees, начиная с 11-го по дате найма, и выводим их имена и фамилии.**

1. ***Инструкции SELECT. Сортировка. Предложение ORDER BY***

**Команда SELECT в SQL используется для выборки данных из таблицы. Предложение ORDER BY используется для сортировки результатов запроса по одному или нескольким столбцам.**

**Синтаксис предложения ORDER BY выглядит следующим образом:**

**SELECT column1, column2, ...**

**FROM table\_name**

**ORDER BY column1, column2, ... ASC|DESC;**

**Здесь column1, column2, ... - это столбцы, по которым нужно отсортировать результаты запроса. table\_name - это имя таблицы, из которой нужно выбрать данные. ASC и DESC - это ключевые слова, которые указывают направление сортировки (по возрастанию или убыванию).**

**Например, если мы хотим выбрать все записи из таблицы employees и отсортировать их по фамилии в алфавитном порядке, мы можем использовать следующий запрос:**

**SELECT \***

**FROM employees**

**ORDER BY last\_name ASC;**

**Этот запрос выберет все записи из таблицы employees и отсортирует их по фамилии в алфавитном порядке по возрастанию.**

**Мы также можем использовать несколько столбцов для сортировки. Например, если мы хотим отсортировать записи из таблицы employees сначала по фамилии, а затем по имени, мы можем использовать следующий запрос:**

**SELECT \***

**FROM employees**

**ORDER BY last\_name ASC, first\_name ASC;**

**Этот запрос выберет все записи из таблицы employees и отсортирует их сначала по фамилии в алфавитном порядке по возрастанию, а затем по имени в алфавитном порядке по возрастанию.**

1. ***Инструкции SELECT. Фильтрация с помощью предложения WHERE.***

**Предложение WHERE в SQL используется для фильтрации данных, которые возвращаются в результате запроса SELECT. Оно позволяет выбирать только те строки, которые удовлетворяют определенным условиям.**

**Синтаксис предложения WHERE выглядит следующим образом:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE condition;**

**Здесь column1, column2, ... - это столбцы, которые нужно выбрать. table\_name - это имя таблицы, из которой нужно выбрать данные. condition - это условие, которое должны удовлетворять строки, чтобы быть выбранными.**

**Условие может содержать операторы сравнения, такие как =, <>, <, >, <=, >=, а также логические операторы AND, OR и NOT. Можно также использовать функции, такие как LIKE, IN и BETWEEN, чтобы создавать более сложные условия.**

**Например, если мы хотим выбрать все записи из таблицы employees, где фамилия равна "Smith", мы можем использовать следующий запрос:**

**SELECT \***

**FROM employees**

**WHERE last\_name = 'Smith';**

**Этот запрос выберет все записи из таблицы employees, где фамилия равна "Smith".**

**Мы также можем использовать операторы сравнения и логические операторы для создания более сложных условий. Например, если мы хотим выбрать все записи из таблицы employees, где фамилия равна "Smith" и зарплата больше 50000, мы можем использовать следующий запрос:**

**SELECT \***

**FROM employees**

**WHERE last\_name = 'Smith' AND salary > 50000;**

**Этот запрос выберет все записи из таблицы employees, где фамилия равна "Smith" и зарплата больше 50000.**

1. ***Извлечение данных средствами T-SQL. Выборка записей из одной таблицы по диапазону значений. Операторы IN и BETWEEN***

**Для выборки записей из одной таблицы по диапазону значений можно использовать оператор BETWEEN. Он позволяет выбрать все записи, у которых значение столбца находится в заданном диапазоне.**

**Синтаксис оператора BETWEEN выглядит следующим образом:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column BETWEEN value1 AND value2;**

**Здесь column1, column2, ... - это столбцы, которые нужно выбрать. table\_name - это имя таблицы, из которой нужно выбрать данные. column - это столбец, значения которого нужно проверить на нахождение в заданном диапазоне. value1 и value2 - это граничные значения диапазона.**

**Например, если мы хотим выбрать все записи из таблицы employees, у которых зарплата находится в диапазоне от 50000 до 70000, мы можем использовать следующий запрос:**

**SELECT \***

**FROM employees**

**WHERE salary BETWEEN 50000 AND 70000;**

**Этот запрос выберет все записи из таблицы employees, у которых зарплата находится в диапазоне от 50000 до 70000.**

**Также для выборки записей из одной таблицы можно использовать оператор IN. Он позволяет выбрать все записи, у которых значение столбца совпадает с одним из заданных значений.**

**Синтаксис оператора IN выглядит следующим образом:**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE column IN (value1, value2, ...);**

**Здесь column1, column2, ... - это столбцы, которые нужно выбрать. table\_name - это имя таблицы, из которой нужно выбрать данные. column - это столбец, значения которого нужно проверить на совпадение с заданными значениями. value1, value2, ... - это значения, с которыми нужно сравнить значения столбца.**

**Например, если мы хотим выбрать все записи из таблицы employees, у которых должность является "Manager" или "Director", мы можем использовать следующий запрос:**

**SELECT \***

**FROM employees**

**WHERE job\_title IN ('Manager', 'Director');**

**Этот запрос выберет все записи из таблицы employees, у которых должность является "Manager" или "Director".**

1. ***Инструкции SELECT. Выборка записей из одной таблицы по шаблону. Оператор LIKE.***

**LIKE**

**Оператор LIKE используется в предложении WHERE для поиска указанного шаблона в столбце.**

**Есть два подстановочных знака, часто используемых в сочетании с оператором LIKE:**

**% - Знак процента представляет собой ноль, один или несколько символов**

**\_ - Подчеркивание представляет собой один символ**

**Примечание: MS Access использует звездочку (\*) вместо знака процента (%) и вопросительный знак (?) вместо подчеркивания (\_).**

**Знак процента и подчеркивание также могут использоваться в комбинациях!**

**Синтаксис LIKE**

**SELECT column1, column2, ...**

**FROM table\_name**

**WHERE columnN LIKE pattern;**

**Определяет, совпадает ли указанная символьная строка с заданным шаблоном. Шаблон может включать обычные символы и символы-шаблоны. Во время сравнения с шаблоном необходимо, чтобы его обычные символы в точности совпадали с символами, указанными в строке. Символы-шаблоны могут совпадать с произвольными элементами символьной строки. Использование подстановочных знаков делает оператор более гибким LIKE , чем использование = операторов сравнения строк и != .**

**SELECT \* FROM Customers**

**WHERE CustomerName LIKE 'a%';**

1. ***Запросы, связанные со значением NULL.***

**NULL - это специальное значение, которое используется в SQL для обозначения отсутствия**

**данных. Оно отличается от пустой строки или нулевого значения, так как NULL означает**

**отсутствие какого-либо значения в ячейке таблицы.**

**функция IS NULL Определяет, может ли указанное выражение быть NULL**

**синтаксис**

**expression IS [ NOT ] NULL**

**expression**

**Любое допустимое выражение expression.**

**NOT**

**Задает отрицание логического результата.**

**Предикат меняет возвращаемые выражением значения на обратные, возвращая TRUE,**

**если значение не равно NULL и FALSE, если значение равно NULL.**

**Типы результата**

**Boolean**

**Значения кода возврата**

**Если значение expression равно NULL, IS NULL возвращает TRUE; в противном случае возвращается значение FALSE.**

**Если значение expression равно NULL, IS NOT NULL возвращает FALSE; в противном случае возвращается значение TRUE.**

**Примечания**

**Для определения, имеет ли выражение значение NULL, используйте IS NULL или IS NOT NULL**

**вместо сравнения операторов (например = или !=). Сравнение операторов возвращает UNKNOWN, если хотя бы один аргумент или они оба равны NULL.**

**Примеры**

**USE AdventureWorks2012;**

**GO**

**SELECT Name, Weight, Color**

**FROM Production.Product**

**WHERE Weight < 10.00 OR Color IS NULL**

**ORDER BY Name;**

**GO**

1. ***Т-SQL Агрегатные функции (AVG, SUM, MIN, MAX, COUNT).***

**Агрегатные функции выполняют вычисления над значениями в наборе**

**строк. Агрегатные функции оперируют значениями столбцов множества**

**строк.**

**В T-SQL имеются следующие агрегатные функции:**

**AVG**: находит среднее значение

**SUM**: находит сумму значений

**MIN**: находит наименьшее значение

**MAX**: находит наибольшее значение

**COUNT**: находит количество строк в запросе

В качестве аргумента все агрегатные функции принимают выражение, которое

представляет критерий для определения значений. Зачастую, в качестве выражения

выступает название столбца, над значениями которого надо проводить вычисления.

Выражения в функциях AVG и SUM должно представлять числовое значение.

Выражение в функциях MIN, MAX и COUNT может представлять числовое или строковое значение или дату.

Все агрегатные функции за исключением COUNT(\*) игнорируют значения NULL.

Для функций COUNT() можно указать аргумент \* для подсчета всех строк

без исключения.

**All и Distinct**

По умолчанию все агрегатные функции учитывают все строки выборки для вычисления результата. Но выборка может содержать повторяющие

значения. Если необходимо выполнить вычисления только над уникальными , то применяется оператор DISTINCT.

По умолчанию вместо DISTINCT применяется оператор ALL, который выбирает

все строки: Так как этот оператор неявно подразумевается при отсутствии DISTINCT, то его можно не указывать.

1. ***Группировка данных. Предложение GROUP BY.***

**Для группировки данных в T-SQL применяются операторы GROUP BY и HAVING**, они имеют следующий формальный синтаксис:

SELECT столбцы

FROM таблица

[WHERE условие фильтрации строк]

[GROUP BY столбцы для группировки]

[HAVING условие фильтрации групп]

[ORDER BY столбцы для сортировки]

Агрегатные функции можно применить не только на всю таблицу, но также на

группу значений. Для этого применяется команда GROUP BY, которая пишется после WHERE.

После команды GROUP BY перечисляется название столбцов, по которым следует группировать данные.

Предложение GROUP BY указывает, что результаты запроса следует разделить

на группы, применить агрегатную функцию по отдельности к каждой группе и

получить для каждой группы одну строку результатов.

В качестве элемента группировки должен выступать любой возвращаемый элемент, указанный в предложении SELECT, кроме значений агрегатных функций.

Оператор GROUP BY может выполнять группировку по множеству столбцов.

Если столбец, по которому производится группировка, содержит значение NULL, то строки со значением NULL составят отдельную группу.

Стоит учитывать, что любой столбец, который используется в выражении

SELECT (не считая столбцов, которые хранят результат агрегатных функций), должны быть указаны после оператора GROUP BY.

И если в выражении SELECT производится выборка по одному или нескольким

столбцам и также используются агрегатные функции, то необходимо использовать выражение GROUP BY.

Также нужно помнить, что выражение GROUP BY должно идти после выражения WHERE, но до выражения ORDER BY.

1. ***Группировка данных. Фильтрация групп. Предложение HAVING.***

Оператор HAVING определяет, какие группы будут включены в выходной

результат, то есть фильтрует полученные группы(результаты) и убирает группы, не подходящие под заданные условия.

Применение HAVING во многом аналогично применению WHERE. Только

WHERE применяется для фильтрации строк, а HAVING используется для фильтрации групп.

Команда HAVING <условие> применяется для фильтрации строк,

возвращаемых при использовании оператора GROUP BY.

HAVING пишется после GROUP BY и имеет такой же формат, как WHERE,

но в качестве значения используется результаты, возвращаемые агрегатными

функциями.

1. ***Разворачивание данных. Табличный оператор PIVOT.***

**В Transact-SQL для написания перекрестных запросов или кросс табличных выражений существует специальный оператор PIVOT**

**PIVOT поворачивает возвращающее табличное значение выражение,**

**преобразуя уникальные значения одного столбца выражения в несколько выходных**

**столбцов. В случае необходимости PIVOT также объединяет оставшиеся**

**повторяющиеся значения столбца и отображает их в выходных данных.**

**PIVOT – это оператор Transact-SQL, который поворачивает результирующий набор**

**данных, т.е. происходит транспонирование таблицы, при этом используются агрегатные**

**функции, и данные соответственно группируются.**

**Синтаксис PIVOT имеет следующий вид:**

**SELECT < столбцы для группировки>, <пивотируемые столбцы > FROM**

**(<запрос возвращающий данных>)**

**AS <псевдоним>**

**PIVOT**

**(<агрегирующая функция>(<столбец>)**

**FOR [<столбец, значения которого будут заголовками>]**

**IN (<список пивотируемых столбцов>)**

**) AS <псевдоним для пивот-таблицы>**

1. ***Отмена разворачивания данных. Табличный оператор UNPIVOT.***

**Оператор UNPIVOT**

**UNPIVOT – это оператор Transact-SQL, который выполняет действия,**

**обратные PIVOT, он разворачивает таблицу в обратную сторону, но в отличие от**

**оператора PIVOT он ничего не агрегирует и уж тем более не разагрегирует.**

**UNPIVOT требуется еще реже, чем PIVOT.**

**Другими словами UNPIVOT преобразует столбцы**

**возвращающего табличное значение выражения в значения столбца.**

**Синтаксис UNPIVOT имеет следующий вид:**

**SELECT <список столбцов>**

**FROM <таблица>**

**UNPIVOT**

**(<столбец значения строк> FOR [<столбец, значения заголовок>]**

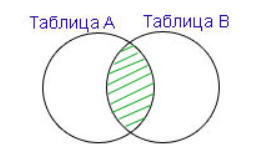
**IN (<список анпивотируемых столбцов>)**

**) AS <псевдоним для анпивот-таблицы>**

1. ***Соединение нескольких таблиц в запросе. Неявное соединение таблиц.***

***Явное соединение таблиц (оператор JOIN). Внутреннее соединение INNER [OUTER] JOIN.***

Запрос с оператором INNER JOIN предназначен для соединения таблиц и вывода результирующей таблицы, в которой данные полностью пересекаются по условию, указанному после ON.

****

После оператора JOIN идет название второй таблицы, из которой надо добавить данные в выборку. Перед JOIN может использоваться необязательное ключевое слово INNER. Его наличие или отсутствие ни на что не влияет. Затем после ключевого слова ON указывается условие соединения. Это условие устанавливает, как две таблицы будут сравниваться. В большинстве случаев для соединения применяется первичный ключ главной таблицы и внешний ключ зависимой таблицы

Inner Join объединяет строки из дух таблиц при соответствии условию. Если одна из таблиц содержит строки, которые не соответствуют этому условию, то данные строки не включаются в выходную выборку.

Также перед оператором JOIN может указываться ключевое слово OUTER, но его применение необязательно

1. ***Соединение нескольких таблиц в запросе. Внешнее левое соединение***

***Внешнее правое соединение. Полное внешнее соединение***

***Перекрестное соединение.***

Перед оператором JOIN указывается одно из ключевых слов LEFT, RIGHT или FULL, которые определяют тип соединения:

LEFT: выборка будет содержать все строки из первой или левой таблицы

Left Join выбирает все строки первой таблицы и затем присоединяет к ним строки правой таблицы.

RIGHT: выборка будет содержать все строки из второй или правой таблицы.

RIGHT JOIN – это такое же объединение как и Left join только будут выводиться все данные из правой таблицы и только те данные из левой таблицы в которых есть ключ объединения

FULL: выборка будет содержать все строки из обоих таблиц.

Cross Join или перекрестное соединение создает набор строк, где каждая строка из одной таблицы соединяется с каждой строкой из второй таблицы. CROSS JOIN – это объединение SQL по которым каждая строка одной таблицы объединяется с каждой строкой другой таблицы.

1. ***Объединение результатов двух запросов в один результирующий***

***набор. Оператор UNION [ALL]***

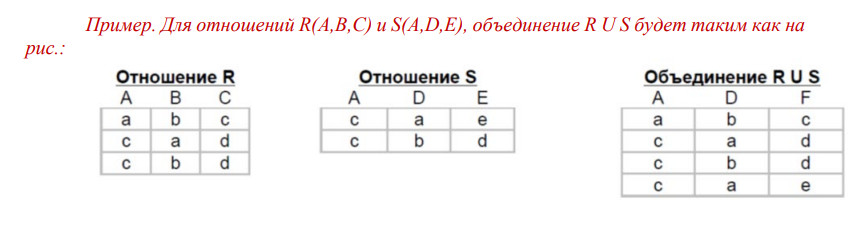
Объединением двух совместимых по типу отношений А и В (A UNION B) называется отношение с тем же заголовком, как и в отношениях А и В, и с телом, состоящим из множества всех кортежей, принадлежащих А или В или обоим отношениям.

****

Для объединения результатов двух или более запросов в одну таблицу используется команда UNION. Команда UNION объединяет вывод двух или более запросов в единый набор строк и столбцов и имеет вид:

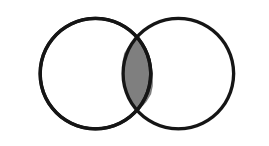
Первый запрос UNION [ALL] Второй запрос

Для объединения результатов нескольких запросов с помощью UNION, они должны соответствовать следующим требованиям: − содержать одинаковое количество столбцов; − типы данных столбцов должны совпадать во всех запросах; − в промежуточных запросах нельзя использовать сортировку ORDER BY.



1. ***Пересечение и разность результатов двух запросов. Операторы NTERSECT, EXCEPT***

Пересечением двух множеств (назовем их A и B) является множество всех элементов, которые принадлежат как A, так и B.



В языке T-SQL оператор INTERSECT возвращает пересечение результирующих наборов, полученных при выполнении двух запросов; в результат попадают только те строки, которые содержатся в обоих входящих наборах.

Команда INTERSECT позволяет найти общие строки в результатах двух запросов, то есть данный оператор выполняет операцию пересечения множеств. Команда INTERSECT имеет следующий вид: Первый запрос INTERSECT Второй запрос. Требования к использованию команды INTERSECT такие же, как к командам UNION

В языке T-SQL разность множеств реализована в виде оператора работы с на борами под названием EXCEPT; он принимает два входящих запроса и возвращает строки, которые присутствуют в первом из них, но отсутствуют во втором.

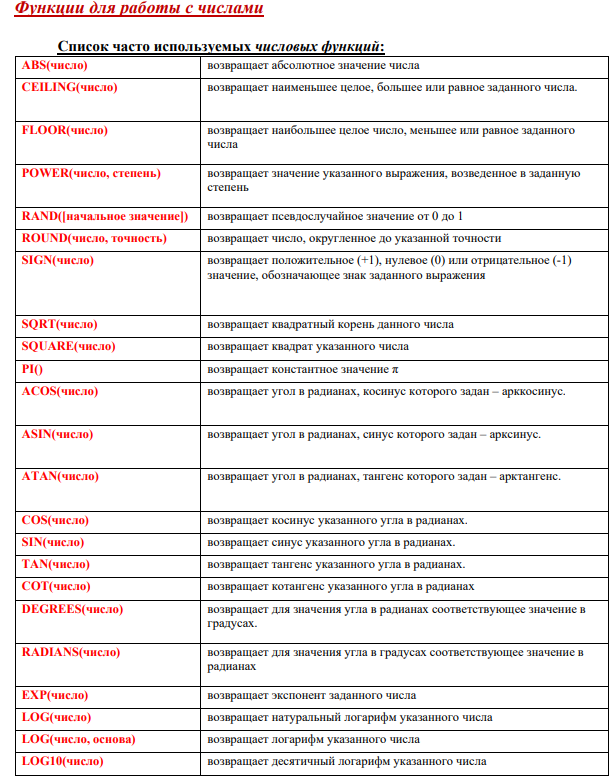
Первым делом оператор EXCEPT убирает дубликаты из двух входящих мультимножеств (превращая их тем самым в настоящие множества), после чего возвращает строки, которые содержатся в первом наборе, но которых нет во втором. SELECT\_выражение1 EXCEPT SELECT\_выражение2

1. ***Подзапросы (вложенные запросы) в основных командах Т-SQL.***

Язык SQL поддерживает написание вложенных запросов. Запрос, который возвращает конечный результат, называют внешним. Внутри себя он может использовать результат выполнения других запросов. Внутренние запросы вызываются во время выполнения кода и являются аналогами выражений, основанных на переменных или константах. В отличие от выражений их результат может изменяться в зависимости от содержимого таблицы. Использование вложенных запросов избавляет от необходимости хранить промежуточные результаты в отдельных переменных. Вложенные запросы могут быть либо автономными, либо коррелирующими. Первые, в отличие от вторых, никак не связаны со своим внешним кодом. Результат вложенного запроса может состоять из одного или нескольких значений или же вовсе являться таблицей

Подзапросы (вложенные запросы), возвращающие единственное значение, наиболее часто применяются в случаях, когда значение определённого столбца в основном запросе требуется сравнить с некоторым единственным значением при помощи одного из операторов сравнения (=, <, >, <=, >=). Значение, с которым производится сравнение, как раз и возвращается подзапросом (вложенным запросом). Верные признаки того, что подзапрос вернёт одно единственное значение: • в подзапросе применяется одна из агрегатных функций (COUNT, SUM, AVG, MAX, MIN); • подзапрос извлекает значение уникального идентификатора, например, первичного ключа.

1. ***Встроенные функции. Функции для работы со строками. Функции для работы с числами. Функции по работе с датами и временем.***

****

****

****

1. ***Встроенные функции преобразование данных.***

1. CAST: используется для преобразования одного типа данных в другой. Например, CAST('123' AS INT) вернет целочисленное значение 123.

2. CONVERT: похож на CAST, но позволяет указать стиль преобразования. Например, CONVERT(DATETIME, '2020-01-01', 120) вернет дату 1 января 2020 года в формате yyyy-mm-dd hh:mm:ss.

3. DATEPART: возвращает определенную часть даты или времени. Например, DATEPART(YEAR, GETDATE()) вернет текущий год.

4. GETDATE: возвращает текущую дату и время.

5. ISNULL: используется для замены NULL на другое значение. Например, ISNULL(NULL, 'Значение не определено') вернет строку "Значение не определено".

6. LEN: возвращает длину строки. Например, LEN('Привет, мир!') вернет значение 12.

7. LOWER: преобразует все символы строки в нижний регистр. Например, LOWER('Привет, Мир!') вернет строку "привет, мир!".

8. UPPER: преобразует все символы строки в верхний регистр. Например, UPPER('Привет, Мир!') вернет строку "ПРИВЕТ, МИР!".

9. SUBSTRING: возвращает подстроку из строки. Например, SUBSTRING('Привет, Мир!', 8, 3) вернет строку "Мир".

10. REPLACE: заменяет все вхождения одной строки на другую. Например, REPLACE('Привет, Мир!', 'Мир', 'мир') вернет строку "Привет, мир!".

1. ***Функция IIF. Функции NEWID, ISNULL и COALESCE.***

Функция IIF в Ms SQL - это условная функция, которая возвращает одно значение, если условие истинно, и другое значение, если условие ложно. Ее синтаксис выглядит следующим образом:

IIF(условие, значение\_если\_истина, значение\_если\_ложь)

Пример использования:

*SELECT IIF(1=1, 'истина', 'ложь')*

Результат: "истина"

Функция NEWID в Ms SQL - это функция, которая генерирует уникальный идентификатор (GUID) типа uniqueidentifier. Ее синтаксис выглядит следующим образом:

NEWID()

Пример использования:

*SELECT NEWID()*

Результат: "1C4D4B8C-9FCA-4E5E-B7A7-7B7A8B7C6D5E"

Функция ISNULL в Ms SQL - это функция, которая возвращает первый аргумент, если он не является NULL, и второй аргумент, если первый аргумент равен NULL. Ее синтаксис выглядит следующим образом:

ISNULL(значение, замена)

Пример использования:

*SELECT ISNULL(NULL, 'замена')*

Результат: "замена"

Функция COALESCE в Ms SQL - это функция, которая возвращает первый не-NULL аргумент из списка аргументов. Ее синтаксис выглядит следующим образом:

COALESCE(значение1, значение2, ..., значениеN)

Пример использования:

*SELECT COALESCE(NULL, 'значение1', 'значение2')*

Результат: "значение1"

1. ***Переменные в Т-SQL. Локальные, глобальные переменные. Объявление и инициализация переменной Команда присваивания SET. Переменные в запросах.***

Переменные в T-SQL используются для хранения значений, которые могут быть использованы в различных операциях. В T-SQL есть два типа переменных: локальные и глобальные.

Локальные переменные объявляются внутри хранимой процедуры или блока кода и имеют ограниченную область видимости. Они могут быть использованы только внутри того блока кода, в котором были объявлены. Локальные переменные объявляются с помощью ключевого слова DECLARE и инициализируются с помощью оператора SET.

Пример объявления и инициализации локальной переменной:

*DECLARE @myVariable INT;*

*SET @myVariable = 10;*

Глобальные переменные объявляются вне хранимых процедур или блоков кода и могут быть использованы в любом месте в базе данных. Они объявляются с помощью ключевого слова DECLARE и опционально могут быть инициализированы с помощью оператора SET.

Пример объявления и инициализации глобальной переменной:

*DECLARE @@myGlobalVariable INT = 20;*

Переменные могут быть использованы в запросах, например, для передачи параметров в хранимые процедуры или для хранения промежуточных результатов.

Пример использования переменной в запросе:

*DECLARE @myVariable INT = 10;*

*SELECT \* FROM myTable WHERE myColumn = @myVariable;*

В целом, переменные в T-SQL позволяют упростить и оптимизировать работу с данными в базе данных, а также повысить безопасность и защиту от SQL-инъекций.

1. ***Управляющая конструкции BEGIN…END в Т-SQL. Команды ветвления, циклов.***

BEGIN...END - это блок кода, который позволяет группировать несколько команд в один блок. Этот блок может использоваться в командах ветвления и циклов.

Команды ветвления позволяют изменять порядок выполнения команд в зависимости от условий. В T-SQL есть несколько команд ветвления:

- IF...ELSE - позволяет выполнить один блок команд, если условие истинно, и другой блок, если условие ложно.

- CASE - позволяет выполнить различные блоки команд в зависимости от значения выражения.

- COALESCE - позволяет выбрать первое значение из списка, которое не является NULL.

Циклы позволяют повторять выполнение блока команд несколько раз. В T-SQL есть несколько команд циклов:

- WHILE - позволяет повторять выполнение блока команд до тех пор, пока условие истинно.

- REPEAT...UNTIL - позволяет повторять выполнение блока команд до тех пор, пока условие ложно.

- FOR - позволяет повторять выполнение блока команд заданное количество раз.

Пример использования BEGIN...END в команде IF:

*IF @x > 0*

*BEGIN*

*PRINT 'Переменная @x больше нуля'*

*END*

*ELSE*

*BEGIN*

*PRINT 'Переменная @x меньше или равна нулю'*

*END*

Пример использования WHILE:

*DECLARE @i INT = 1*

*WHILE @i <= 10*

*BEGIN*

*PRINT 'Значение переменной @i: ' + CAST(@i AS VARCHAR(2))*

*SET @i = @i + 1*

*END*

1. ***Представления в Т-SQL. Создание, редактирование, изменение и удаление представления.***

Представление (View) в T-SQL - это виртуальная таблица, которая создается на основе одной или нескольких таблиц и используется для упрощения запросов к базе данных. Представление может содержать любой допустимый запрос SELECT, включая операторы JOIN, WHERE и GROUP BY.

Создание представления:

CREATE VIEW [имя представления]

AS

SELECT [столбцы]

FROM [таблицы]

WHERE [условие]

Пример:

*CREATE VIEW [dbo].[EmployeesView]*

*AS*

*SELECT [EmployeeID], [LastName], [FirstName], [Title]*

*FROM [dbo].[Employees]*

*WHERE [City] = 'Seattle'*

Редактирование представления:

ALTER VIEW [имя представления]

AS

SELECT [столбцы]

FROM [таблицы]

WHERE [условие]

Пример:

*ALTER VIEW [dbo].[EmployeesView]*

*AS*

*SELECT [EmployeeID], [LastName], [FirstName], [Title]*

*FROM [dbo].[Employees]*

*WHERE [City] = 'Redmond'*

Изменение представления:

ALTER VIEW [имя представления]

WITH CHECK OPTION

Пример:

*ALTER VIEW [dbo].[EmployeesView]*

*WITH CHECK OPTION*

Удаление представления:

DROP VIEW [имя представления]

Пример:

*DROP VIEW [dbo].[EmployeesView]*

1. ***Табличные переменные. Временные локальные и глобальные таблицы.***

Табличные переменные в T-SQL - это временные таблицы, которые создаются в памяти сервера и используются для хранения временных результатов запросов. Они могут содержать любые данные и использоваться в качестве аргументов для хранимых процедур и функций.

Создание табличной переменной:

DECLARE @имя\_переменной TABLE (

[столбцы]

)

Пример:

*DECLARE @myTableVariable TABLE (*

*[ID] INT,*

*[Name] VARCHAR(50),*

*[Age] INT*

*)*

Вставка данных в табличную переменную:

INSERT INTO @имя\_переменной

VALUES ([значения])

Пример:

*INSERT INTO @myTableVariable*

*VALUES (1, 'John', 25),*

*(2, 'Kate', 30),*

*(3, 'Mike', 35)*

Выборка данных из табличной переменной:

SELECT [столбцы]

FROM @имя\_переменной

Пример:

*SELECT [ID], [Name], [Age]*

*FROM @myTableVariable*

Временные таблицы - это таблицы, которые создаются в базе данных и используются для хранения временных результатов запросов. Они могут быть локальными или глобальными.

Создание временной локальной таблицы:

CREATE TABLE #имя\_таблицы (

[столбцы]

)

Пример:

*CREATE TABLE #tempTable (*

*[ID] INT,*

*[Name] VARCHAR(50),*

*[Age] INT*

*)*

Создание временной глобальной таблицы:

CREATE TABLE ##имя\_таблицы (

[столбцы]

)

Пример:

*CREATE TABLE ##tempTable (*

*[ID] INT,*

*[Name] VARCHAR(50),*

*[Age] INT*

*)*

Вставка данных в временную таблицу:

INSERT INTO #имя\_таблицы

VALUES ([значения])

Пример:

*INSERT INTO #tempTable*

*VALUES (1, 'John', 25),*

*(2, 'Kate', 30),*

*(3, 'Mike', 35)*

Выборка данных из временной таблицы:

SELECT [столбцы]

FROM #имя\_таблицы

Пример:

*SELECT [ID], [Name], [Age]*

*FROM #tempTable*

Удаление временных таблиц:

DROP TABLE #имя\_таблицы

DROP TABLE ##имя\_таблицы

Пример:

*DROP TABLE #tempTable*

*DROP TABLE ##tempTable*

1. ***Хранимые процедуры. Типы хранимых процедур в Т-SQL. Создание, выполнение процедуры, удаление хранимых процедур .***

Хранимая процедура - это предварительно скомпилированный блок кода, который может быть вызван из другого кода или приложения. Она может содержать SQL-запросы, управляющие конструкции, переменные и параметры.

Типы хранимых процедур в T-SQL:

1. Процедуры без параметров - не принимают входные параметры.

2. Процедуры с параметрами - принимают входные параметры.

3. Процедуры с выходными параметрами - возвращают выходные параметры.

4. Процедуры с параметрами ввода-вывода - принимают и возвращают параметры.

**Создание хранимой процедуры:**

CREATE PROCEDURE [имя\_процедуры]

[параметры]

AS

BEGIN

[тело процедуры]

END

Пример:

*CREATE PROCEDURE GetCustomers*

*AS*

*BEGIN*

*SELECT \* FROM Customers*

*END*

**Выполнение хранимой процедуры:**

EXEC [имя\_процедуры] [параметры]

Пример:

*EXEC GetCustomers*

**Удаление хранимой процедуры:**

DROP PROCEDURE [имя\_процедуры]

Пример:

*DROP PROCEDURE GetCustomers*