# **Введение**

База данных (БД) — это организованная структура, предназначенная для хранения, изменения и обработки взаимосвязанной информации, преимущественно больших объемов.

Система управления базами данных (СУБД) — это комплекс программных средств, необходимых для создания структуры новой базы, ее наполнения, редактирования содержимого и отображения информации. Наиболее распространенными СУБД являются Microsoft SQL Server, MySQL, PostgreSQL, Oracle.

В рамках нашего курса мы будем изучать базы данных на примере Microsoft SQL Server.

Для выполнения лабораторных работ Вам понадобится:

1. SQL Express 2017 Express (или новее)
2. Microsoft SQL Server Management Studio

Все эти программы можно скачать на официальном сайте Microsoft.

В таблице 1 представлены темы курсовых работ. Выберите тему и убедитесь, что ее больше никто не выбрал из вашей группы. Все лабораторные работы будут выполняться исходя из темы курсовой. Вариант нельзя менять в течение семестра. Выполненные лабораторные работы лягут в основу курсовой работы.

**Таблица 1 – Темы курсовых работ**

|  |  |
| --- | --- |
| **№ п/п** | **Название** |
| 1 | ИС «Электронная ведомость ВУЗа» |
| 2 | ИС «Пассажирское автопредприятие» |
| 3 | ИС «Таксопарк» |
| 4 | ИС «Электронная регистратура поликлиники» |
| 5 | ИС «Библиотека» |
| 6 | ИС «Гостиница» |
| 7 | ИС «Ресторан» |
| 8 | ИС «Кинотеатр» |
| 9 | ИС «Автосалон» |

Продолжение таблицы 1.

|  |  |
| --- | --- |
| **№ п/п** | **Название** |
| 10 | ИС «Цветочный магазин» |
| 11 | ИС «ЗАГС» |
| 12 | ИС «Отдел кадров предприятия» |
| 13 | ИС «Паспортный стол» |
| 14 | ИС «Складской учет на предприятии» |
| 15 | ИС «Туристическое агентство» |
| 16 | ИС «Автосервис» |
| 17 | ИС «Кондитерское предприятие» |
| 18 | ИС «Ателье мод» |
| 19 | ИС «Электронный школьный дневник» |
| 20 | ИС «Служба доставки» |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» |
| 22 | ИС «Салон красоты» |
| 23 | ИС «Санитарно-курортный комплекс» |

# **Лабораторная работа №1 Создание и удаление базы данных. Общие сведения**

Базу данных MS SQL Server можно создать несколькими способами:

1. Используя интерфейс SQL Management Studio
2. Используя Transact-SQL

Рассмотрим каждый способ создания базы данных подробнее.

**SQL Management Studio**

В SQL Management Studio в контекстном меню элемента ***Базы данных***выбрать пункт ***Создать базу данных***(рис. 1.1).

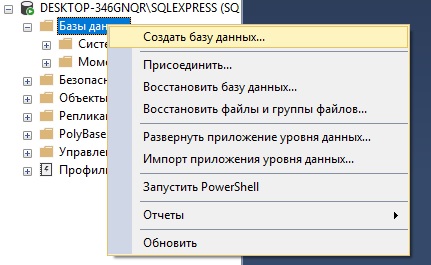


Рисунок 1.1

В открывшемся диалоговом окне необходимо заполнить имя новой базы данных. Остальные настройки можно оставить по умолчанию. Нажать на кнопку ***ОК.***

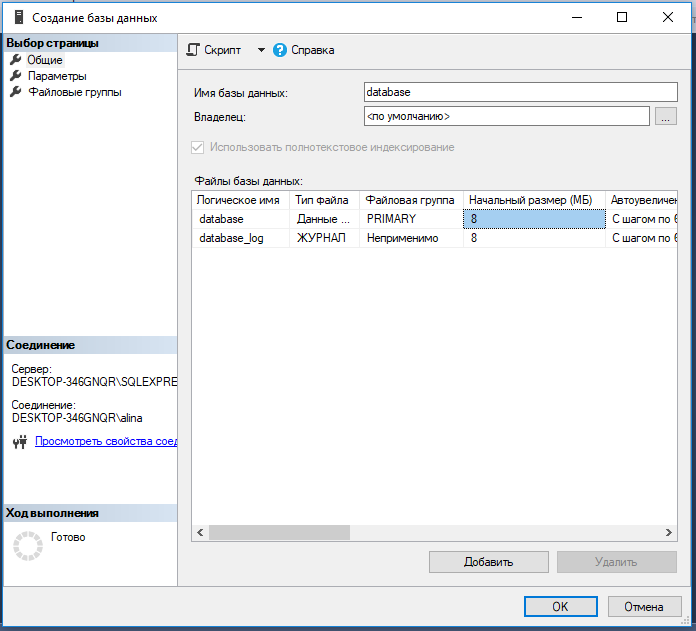


Рисунок 1.2

**Transact-SQL**

Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос на языке SQL посредством специального API. СУБД должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения.

Выделяются две разновидности языка SQL: PL-SQL и T-SQL. PL-SQL используется в таких СУБД как Oracle и MySQL. T-SQL (Transact-SQL) применяется в SQL Server.

**Типы команд T-SQL**

Команды T-SQL (в зависимости от выполняемой задачи) подразделяются на следующие типы:

***DDL (Data Definition Language / Язык определения данных).*** К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. (т.е. определяют данные).

Команды DDL:

***CREATE***: создает объекты базы данных

***ALTER***: изменяет объекты базы данных

***DROP***: удаляет объекты базы данных

***TRUNCATE***: удаляет все данные из таблицы

***DML (Data Manipulation Language / Язык манипуляции данными).*** К этому типу относят команды на выборку данных, их обновление, добавление, удаление – т.е. все те команды, с помощью которых можно управлять данными.

К этому типу относятся следующие команды:

***SELECT***: извлекает данные из БД

***UPDATE***: обновляет данные

***INSERT***: добавляет новые данные

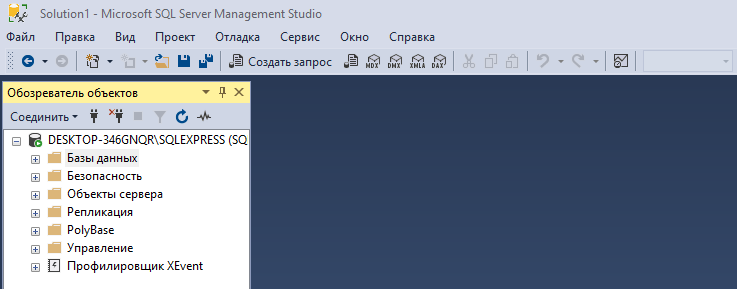
***DELETE***: удаляет данные

***DCL (Data Control Language / Язык управления доступа к данным).*** К этому типу относят команды, которые управляют правами по доступу к данным. В частности, это следующие команды:

***GRANT***: предоставляет права для доступа к данным

***REVOKE***: отзывает права на доступ к данным

Открыть окно запроса можно, нажав на кнопку ***Создать запрос*** на панели инструментов.



Для создания базы данных с помощью выполнения скрипта используется следующий код:

CREATE DATABASE databaseName

Чтобы выполнить запрос необходимо нажать на кнопку ***Выполнить*** на панели инструментов либо нажать F5 на клавиатуре.

Если нужно выполнить только не весь код, а только его часть, то предварительно его нужно выделить и только затем выполнить.

Для того чтобы выполнять различные операции с базой данных (создавать, изменять, удалять объекты; выполнять запросы) необходимо установить базу данных в качестве текущей. Сделать это можно, используя команду **USE**:

USE databaseName

Для удаления базы данных используется следующая команда:

DROP DATABASE databaseName

**Задание**

Создать базу данных с названием, отражающим выбранную тему курсовой работы.

# **Лабораторная работа №2** **Создание и удаление таблиц. Атрибуты и ограничения столбцов и таблиц.**

Для создания таблицы используется команда **CREATE TABLE.** В общем виде синтаксис создания таблицы выглядит следующим образом:

CREATE TABLE table\_name

(название\_столбца1 тип\_данных атрибуты\_столбца1,

название\_столбца2 тип\_данных атрибуты\_столбца2,

………………)

Имя таблицы должно быть уникальным, имя может состоять из алфавитно-цифровых символов, знака подчеркивания и знака $. Первым символом должен быть знак подчеркивания или буква.

Например, пример создания простой таблицы, которая содержит информацию о сотрудниках.

CREATE TABLE Employees

(Id INT,

FirstName NVARCHAR(50),

LastName NVARCHAR(100),

BirthDate DATETIME,

Email VARCHAR(200))

В данном случае таблица *Employees* состоит из 5 столбцов: *Id, FirstName, LastName, BirthDate, Email*. Идентификатор (*Id*) имеет целочисленный тип (Int), Фамилия и имя (*FirstName, LastName*) имеют тип nvarchar, то есть представляют собой строку UNICODE, длинной не более указанной в скобках.

Для удаления таблицы используется команда **DROP:**

DROP TABLE tableName

При создании столбцов в T-SQL могут быть использованы различные ограничения.

**PRIMARY KEY –** атрибут, с помощью которого может быть создан *первичный ключ* таблицы.

Первичный ключ уникально идентифицирует строку в таблице. Как правило, в роли первичного ключа выступают столбцы с типом INT, но это необязательное условие и любой столбец может быть первичным ключом.

Например, создадим таблицу *Employee* с первичным ключом *Id:*

CREATE TABLE Employees

(Id INT PRIMARY KEY,

FirstName NVARCHAR(50),

LastName NVARCHAR(100),

BirthDate DATETIME,

Email VARCHAR(200))

Также можно установить первичный ключ на уровне таблицы – результат будет тот же:

CREATE TABLE Employees

(Id INT,

FirstName NVARCHAR(50),

LastName NVARCHAR(100),

BirthDate DATETIME,

Email VARCHAR(200),

PRIMARY KEY(Id))

Первичный ключ может состоять из нескольких столбцов. Такой ключ может быть полезен, если несколько столбцов должны уникально идентифицировать строку в таблице. При этом по отдельности столбцы могут быть не уникальны.

Например, создадим для таблицы *Employee* первичный ключ, который будет включать в себя столбцы *FirstName, SecondName, BirthDate:*

CREATE TABLE Employees

(FirstName NVARCHAR(50),

LastName NVARCHAR(100),

BirthDate DATETIME,

Email VARCHAR(200),

PRIMARY KEY(FirstName, LastName, BirthDate))

Допустим, в таблице содержится запись о сотруднике: Крамской Иван 27.05.1837г. Первичный ключ не позволит добавить в таблицу сотрудника с одновременно такими же фамилией, именем и датой рождения. Но позволит, например, добавить с такими же фамилией и датой рождением, но другим именем. Например, Крамской Николай 27.05.1837г.

Обратите внимание, при добавлении составного первичного ключа в таблице был убран столбец Id, который являлся *искусственным (суррогатным) первичным ключом. Искусственный ключ –* это поле, значение которого не образуется на основе каких-либо других данных из таблицы и базы в целом, а генерируется искусственно и его единственной целью является служить первичным ключом.

**IDENTITY –** атрибут, с помощью которого можно сделать столбец идентификатором. Этот атрибут может применять к столбцам числовых типов (INT, SMALLINT, BIGINT, DECIMAL, NUMERIC). При добавлении новой записи в таблицу не обязательно указывать для этого столбца значение, т.к. SQL Server присвоит его автоматически, и оно будет равно предыдущему значению плюс указанный инкремент, который по умолчанию равен единице. Как правило, в роли идентификатора выступает тот же столбец, который является первичным ключом, но это необязательно.

Полная форма атрибута выглядит следующим образом:

IDENTITY(seed, increment),

где *seed* – начальное значение (по умолчанию единица),

*increment* – значение, на которое будет увеличено значение при добавлении новой записи (по умолчанию единица). Но можно использовать краткую форму (IDENTITY), которая эквивалентна выражению IDENTITY(1, 1):

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50),

LastName NVARCHAR(100),

BirthDate DATETIME,

Email VARCHAR(200))

Обратите внимание, что только один столбец в таблице может применять этот атрибут.

**UNIQUE –** атрибут, который используется, если значения в столбце должны быть уникальными.

Например, *Email* в таблице должен быть уникальным и не может повторяться более одного раза:

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50),

LastName NVARCHAR(100),

BirthDate DATETIME,

Email VARCHAR(200) UNIQUE)

**NULL / NOT NULL –** атрибуты, которые указывают, могут ли значения в данном столбце отсутствовать (быть NULL). По умолчанию, если атрибут не был указан при создании таблицы, столбец будет допускать значения NULL. Единственным исключением является первичный ключ – он не может принимать значения NULL.

Например, столбцы *FirstName* и *LastName* обязательно должны иметь значения:

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME,

Email VARCHAR(200) UNIQUE)

**DEFAULT –** атрибут, который задает значение по умолчанию. Указанное значение будет использовано, если при добавлении записи явно не будет указано другое значение.

Например, установим дату рождения по умолчанию 01.01.1990:

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

Email VARCHAR(200) UNIQUE)

**CHECK –** атрибут, который задает ограничение для значений, которые могут храниться в столбце. Для этого необходимо в скобках указать условие, которому должен соответствовать столбец или несколько столбцов. Например, дата рождения сотрудников не может быть ранее 1920 года и позже 2002 года, а значение *Email* не может быть пустой строкой. *Обратите внимание, что пустая строка не равна значению NULL.*

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01' CHECK

(BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE CHECK (Email != ''))

Для соединения условий используется ключевое слово AND. Условия можно задать в виде операций сравнения больше (>), меньше (<), не равно (!=).

**CONSTRAINT** – ключевое слово, с помощью которого можно задать имя для ограничения (PRIMARY KEY, FOREIGN KEY, DEFAULT, UNIQUE, CHECK).

Ограничения могут носить любые название, но, как правило, используются следующие префиксы:

PK\_ - для первичного ключа (PRIMARY KEY)

FK\_ - для внешнего ключа (FOREIGN KEY)

DF\_ - для значения по умолчания (DEFAULT)

UQ\_ - для уникальности значения (UNIQUE)

CK\_ - для проверки значения (CHECK)

Ниже указан пример использования **CONSTRAINT.**

CREATE TABLE Employees

(Id INT CONSTRAINT PK\_Employee\_Id PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME

CONSTRAINT DF\_Employee\_BirthDate DEFAULT '1990-01-01',

CONSTRAINT CK\_Employee\_BirthDate CHECK

(BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) CONSTRAINT UQ\_Employee\_Email UNIQUE)

Задавать имена для ограничений необязательно, SQL Server делает это автоматически, если имя не указано.

**Внешние ключи**

*Внешние ключи (Foreign Keys)* применяются для установки связи между таблицами.

Внешний ключ устанавливается для столбцов зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Как правило, внешний ключ указывает на первичный ключ из главной таблицы, но это необязательное условие. Внешний ключ также может указывать на другой столбец, который имеет уникальное значение (UNIQUE).

Общий синтаксис установки внешнего ключа на уровне столбца:

FOREIGN KEY (child\_col1, child\_col2, ... child\_col\_n)

REFERENCES parent\_table (parent\_col1, parent\_col2, ... parent\_col\_n)

[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]

Для создания ограничения внешнего ключа на уровне столбца после ключевого слова REFERENCES указывается имя связанной таблицы и в круглых скобках имя связанного столбца, на который будет указывать внешний ключ. После выражения REFERENCES идет выражение ON DELETE и ON UPDATE.

Рассмотрим следующий пример: каждый сотрудник в организации принадлежит к какому-либо подразделению. Добавим таблицу *Departments*, которая будет хранить данные о подразделениях и в таблице *Employees* добавим внешний ключ к новой таблице *Departments*.

CREATE TABLE Departments

(Id INT PRIMARY KEY IDENTITY,

Name NVARCHAR(200) NOT NULL,

Description NVARCHAR(500) NOT NULL)

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

CHECK (BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE,

DepartmentId INT References Departments (Id))

Теперь таблица *Employees* содержит внешний ключ к таблице *Departments*. То есть в столбце *DepartmentId* в таблице *Employees* будет содержаться соответствующее значение столбца *Id* таблицы *Departments*.

Внешний ключ можно также указать на уровне таблицы после объявления всех столбцов, используя ключевое слово FOREIGN KEY:

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

CHECK (BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE,

DepartmentId INT,

FOREIGN KEY (DepartmentId) References Departments (Id))

Используя оператор CONSTRAINT, можно задать имя для ограничения внешнего ключа. Как правило, имя начинается с префикса "FK\_":

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

CHECK (BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE,

DepartmentId INT,

CONSTRAINT FK\_Employees\_Departments FOREIGN KEY (DepartmentId) References Departments (Id))

**ON DELETE** / **ON UPDATE**

С помощью выражений **ON DELETE** и **ON UPDATE** можно установить действия, которые будут выполняться соответственно при удалении и изменении связанной строки из главной таблицы. И для определения действия доступны следующие опции:

**CASCADE**: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.

**NO ACTION**: предотвращает какие-либо действия в зависимой таблице при удалении или изменении связанных строк в главной таблице.

**SET NULL**: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL. Это возможно только если в зависимой таблице для данного столбца разрешены NULL-значения.

**SET DEFAULT**: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение по умолчанию, которое задается с помощью атрибуты DEFAULT. Если для столбца не задано значение по умолчанию, то в качестве него применяется значение NULL.

**CASCADE (Каскадное удаление)**

По умолчанию, если на строку из главной таблицы по внешнему ключу ссылается какая-либо строка из зависимой таблицы, то эту строку нельзя удалить из главной таблицы. Вначале необходимо будет удалить все связанные строки из зависимой таблицы. Если при удалении строки из главной таблицы необходимо, чтобы были удалены все связанные строки из зависимой таблицы, то применяется каскадное удаление, то есть опция CASCADE:

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

CHECK (BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE,

DepartmentId INT,

FOREIGN KEY (DepartmentId) References Departments (Id) ON DELETE CASCADE)

Выражение **ON UPDATE CASCADE** работает аналогично. При изменении значения первичного ключа автоматически изменится значение связанного с ним внешнего ключа. Но первичные ключи, как правило, изменяются очень редко, и не рекомендуется использовать в качестве первичных ключей столбцы с изменяемыми значениями. Поэтому на практике выражение **ON UPDATE** используется редко.

**SET NULL (Установка NULL)**

При установки для внешнего ключа опции SET NULL необходимо, чтобы столбец внешнего ключа допускал значение NULL, иначе произойдет ошибка:

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

CHECK (BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE,

DepartmentId INT,

FOREIGN KEY (DepartmentId) References Departments (Id) ON DELETE SET NULL)

**SET DEFAULT (Установка значения по умолчанию)**

CREATE TABLE Employees

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(100) NOT NULL,

BirthDate DATETIME DEFAULT '1990-01-01',

CHECK (BirthDate > '1920-01-01' AND BirthDate < '2002-01-01'),

Email VARCHAR(200) UNIQUE,

DepartmentId INT,

FOREIGN KEY (DepartmentId) References Departments (Id) ON DELETE SET DEFAULT)

**Задание к лабораторной работе**

Выполните задание, соответствующее выбранному варианту. Вы можете изменить состав полей, связи и таблицы, если считаете нужным сделать по-другому. Главное создать связь между таблицами через внешний ключ, добавить все ограничения (PRIMARY KEY, IDENTITY, UNIQUE, DEFAULT, NULL / NOT NULL, CHECK) и аргументировать свой выбор.

Внимание: лабораторная работа №2 сдается вместе с лабораторной работой №3.

**Вариант 1 - ИС «Электронная ведомость ВУЗа»**

Создайте две таблицы – *Дисциплины* и *Экзаменаторы*.

Таблица *Дисциплины* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки,
* Описание – строковое поле, не более 1000 символов,
* Экзаменатор – внешний ключ к таблице *Экзаменаторы* с опцией SET NULL.

Таблица *Экзаменаторы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное поле,
* Ученая степень экзаменатора – строковое поле, не более 300 символов, по умолчанию «не имеет».
* Ученое звание – строковое поле, не более 200 символов, может содержать только следующие значения: «доцент», «профессор», «» (пустое значение).

**Вариант 2 - ИС «Пассажирское автопредприятие»**

Создайте две таблицы – *Транспортные средства* и *Модели транспортных средств*.

Таблица *Транспортные средства* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Дата начала эксплуатации – не может быть ранее 01.01.2000г.,
* Модель - внешний ключ к таблице *Модели транспортных средств* с опцией каскадного удаления, не допускаются NULL-значения,
* Имя водителя – строковое поле, не более 300 символов, может быть NULL.

Таблица *Модели транспортных средств* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Марка – строковое поле, не более 200 символов, не допускаются значения NULL и пустые строки,
* Модель – строковое поле, не более 200 символов, не допускаются значения NULL, уникальное значение,
* Дата выпуска – значение по умолчанию 01.01.2005 г. ,
* Пробег.

**Вариант 3 - ИС «Таксопарк»**

Создайте две таблицы – *Транспортные средства* и *Классы (например, Эконом, Комфорт, Люкс и т.д.)*.

Таблица *Транспортные средства* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Марка – строковое поле, не более 200 символов, не допускаются значения NULL и пустые строки,
* Модель – строковое поле, не более 200 символов, не допускаются значения NULL, уникальное значение,
* Дата выпуска – не ранее 01.01.2008 г.,
* Дата начала эксплуатации – по умолчанию 01.01.2010 г.,
* Класс - внешний ключ к таблице *Классы* с опцией SET DEFAULT, класс по умолчанию «Эконом»,
* Имя водителя – строковое поле, не более 300 символов.

Таблица *Классы* должна содержать следующие поля:

* Название – первичный ключ,
* Описание – строковое поле, не более 500 символов,
* Цена – в рублях, должна находиться в диапазоне 100 – 2000 руб.

**Вариант 4 - ИС «Электронная регистратура поликлиники»**

Создайте две таблицы – *Сотрудники* и *Должность*.

Таблица *Сотрудники* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата рождения – не может быть ранее 01.01.2001 г.,
* Специализация – строковое поле, не более 300 символов,
* Дата начала работы – по умолчанию 10.09.2010 г.
* Должность - внешний ключ к таблице *Должности* с опцией SET NULL.

Таблица *Должности* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 500 символов, уникальное значение,
* Описание – строковое поле, не более 1000 символов.

**Вариант 5 - ИС «Библиотека»**

Создайте две таблицы – *Книги* и *Авторы*.

Таблица *Книги* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 500 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата издания – не позднее 01.01.2020 г.,
* Жанр – строковое поле, не более 300 символов, допускаются значения NULL.
* Автор - внешний ключ к таблице *Авторы* с опцией каскадного удаления.

Таблица *Авторы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО – строковое поле, не более 500 символов, уникальное значение,
* Дата рождения – по умолчанию 01.01.1960 г.,
* Место рождения – строковое поле, не более 1000 символов.

**Вариант 6 - ИС «Гостиница»**

Создайте две таблицы – *Номера* и *Горничные*.

Таблица *Номера* должна содержать следующие поля:

* Идентификатор (номер) – первичный ключ,
* Количество комнат – по умолчанию 1,
* Количество мест – по умолчанию 2, не может быть меньше 1,
* Горничная – внешний ключ к таблице *Горничные* с опцией SET NULL.

Таблица *Горничные* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата начала работы,
* Дата рождения – по умолчанию 01.01.1960 г.,
* Паспортные данные – строковое поле, не более 200 символов.

**Вариант 7 - ИС «Ресторан»**

Создайте две таблицы – *Официанты и Столики* .

Таблица *Официанты* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата начала работы – не ранее 03.06.2015 г.

Таблица *Столики* должна содержать следующие поля:

* Номер стола – первичный ключ,
* Количество мест – по умолчанию 2, с проверкой не менее 2 и не более 10,
* Номер зала – по умолчанию 1,
* Официант – внешний ключ к таблице *Официанты* с опцией SET NULL.

**Вариант 8 - ИС «Кинотеатр»**

Создайте две таблицы – *Фильмы и Залы* .

Таблица *Фильмы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата начала показа,
* Дата окончания показа,
* Зал – внешний ключ к таблице Залы с опцией SET NULL.

Таблица *Залы* должна содержать следующие поля:

* Номер зала – первичный ключ,
* Количество мест – по умолчанию 80, с проверкой не менее 80 и не более 340,
* Тип зала (2D, 3D) – по умолчанию 2D.

**Вариант 9 - ИС «Автосалон»**

Создайте две таблицы – *Машины и Покупатели*.

Таблица *Машины* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Марка – строковое поле, не более 200 символов, не допускаются значения NULL и пустые строки,
* Модель – строковое поле, не более 200 символов, не допускаются значения NULL, уникальное значение,
* Дата выпуска – по умолчанию 01.01.2010 г.,
* Пробег,
* Цена – не может быть меньше 0,
* Покупатель – внешний ключ к таблице Покупатели с опцией SET NULL.

Таблица *Покупатели* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата рождения – не может быть ранее 01.01.2001 г.
* Паспортные данные – строковое поле, не более 200 символов.

**Вариант 10 - ИС «Цветочный магазин»**

Создайте две таблицы – *Цветы и Поставщики*.

Таблица *Цветы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки,
* Цена поставщика – не может быть меньше 0,
* Цена продажи – не может быть меньше 0,
* Поставщик – внешний ключ к таблице *Поставщики* с опцией SET NULL.

Таблица *Поставщики* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название поставщика - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Страна – строковое поле, не более 200 символов, по умолчанию «Россия»
* Город – строковое поле, не более 200 символов, по умолчанию «Москва»
* Реквизиты для оплаты.

**Вариант 11 - ИС «ЗАГС»**

Создайте две таблицы – *Заявления и Заявители*.

Таблица *Заявления* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Супруг – внешний ключ к таблице *Заявители* с опцией каскадного удаления,
* Супруга – внешний ключ к таблице *Заявители* с опцией каскадного удаления,
* Дата подачи заявления,
* Дата регистрации,

Таблица *Заявители* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата рождения – не ранее 01.01.2001 г.
* Паспортные данные - строковое поле, не более 300 символов.
* Страна – строковое поле, не более 200 символов, по умолчанию «Россия»
* Город – строковое поле, не более 200 символов, по умолчанию «Москва»

**Вариант 12 - ИС «Отдел кадров предприятия»**

Создайте две таблицы – *Отделы и Сотрудники*.

Таблица *Отделы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 250 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Описание – строковое поле, не более 500 символов,
* Страна – строковое поле, не более 200 символов, по умолчанию «Россия»,
* Город – строковое поле, не более 200 символов, по умолчанию «Москва»

Таблица *Сотрудники* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата рождения – не ранее 01.01.2001 г.
* Паспортные данные - строковое поле, не более 300 символов,
* Отдел – внешний ключ к таблице *Отделы* с опцией SET NULL.

**Вариант 13 - ИС «Паспортный стол»**

Создайте две таблицы – *Паспорта и Статус*.

Таблица *Паспорта* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Серия – строковое поле, не более 4 символов, не допускаются значения NULL и пустые строки. Добавить ограничение уникальности на это поле и поле «Номер», т.е. по отдельности поля могут быть повторяться, но в составе серия-номер значение должно быть уникальным.
* ФИО – строковое поле, не более 700 символов,
* Дата рождения,
* Номер - строковое поле, не более 4 символов, не допускаются значения NULL и пустые строки,
* Код подразделения – по умолчанию 770-001
* Дата подачи заявления,
* Дата выдачи,
* Место выдачи.
* Статус – внешний ключ к таблице *Статусы* с опцией SET NULL.

Таблица *Статусы* (В обработке, Готов к выдаче, Просрочен и т.д.)должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название - строковое поле, не более 200 символов, не допускаются значения NULL и пустые строки, уникальное значение.

**Вариант 14 - ИС «Складской учет на предприятии»**

Создайте две таблицы – *Товары и Накладные*.

Таблица *Товары* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки.
* Цена – не может быть меньше 0,
* Количество – не может быть меньше 0,
* Номер накладной – внешний ключ к таблице *Накладные* с опцией каскадного удаления.

Таблица *Накладные* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Дата поставки,
* Сумма – по умолчанию 0,
* Отправитель – строковое поле, не более 300 символов.

**Вариант 15 - ИС «Туристическое агентство»**

Создайте две таблицы – *Путевки и Туристы*.

Таблица *Путевки* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Страна – по умолчанию «Россия»,
* Город,
* Дата покупки – не ранее 01.01.2010 г.,
* Дата поездки,
* Цена – не может быть менее 0 руб.,
* Турист – внешний ключ к таблице *Туристы* с опцией каскадного удаления.

Таблица *Туристы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата рождения – не ранее 01.01.2001 г.
* Паспортные данные - строковое поле, не более 300 символов,

**Вариант 16 - ИС «Автосервис»**

Создайте две таблицы – *Транспортные средства* и *Заказчик.*

Таблица *Транспортные средства* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Марка – строковое поле, не более 200 символов, не допускаются значения NULL и пустые строки,
* Модель – строковое поле, не более 200 символов, не допускаются значения NULL, уникальное значение,
* Год выпуска,
* Причина обращения – значение по умолчанию «Тех.осмотр»,
* Цена за услугу – не может быть меньше 0,
* Заказчик - внешний ключ к таблице *Заказчики* с опцией каскадного удаления.

Таблица *Заказчики* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Телефон – не допускаются значения NULL и пустые строки.

**Вариант 17 - ИС «Кондитерское предприятие»**

Создайте две таблицы – *Кондитерское изделие* и *Витрины.*

Таблица *Кондитерские изделия* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Количество в наличии – не может быть меньше 0, по умолчанию 0,
* Цена – не может быть меньше 0,
* Витрина - внешний ключ к таблице *Витрины* с опцией SET NULL.

Таблица *Витрины* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Описание - строковое поле, не более 300 символов.

**Вариант 18 - ИС «Ателье мод»**

Создайте две таблицы – *Изделия* и *Заказчики.*

Таблица *Изделия* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Название – строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки,
* Цена – не может быть меньше 0,
* Готовность – булево значение (0 – не готово, 1 - готово). По умолчанию 0.
* Заказчик - внешний ключ к таблице *Заказчики* с опцией каскадного удаления.

Таблица *Заказчики* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Телефон – не допускаются значения NULL и пустые строки.

**Вариант 19 - ИС «Электронный школьный дневник»**

Создайте две таблицы – *Классы* и *Школьники.*

Таблица *Классы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Классный руководитель - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки,
* Кабинет – не может быть меньше 0.
* Этаж – по умолчанию 1.

Таблица *Школьники* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Телефон родителя – не допускаются значения NULL и пустые строки,
* Класс – внешний ключ к таблице *Классы* с опцией SET NULL,
* Дата поступления – по умолчанию 01.01.2010 г.

**Вариант 20 - ИС «Служба доставки»**

Создайте две таблицы – *Заказы* и *Заказчики.*

Таблица *Заказы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Состав заказа - строковое поле, не более 500 символов, не допускаются значения NULL и пустые строки,
* Сумма заказа – не может быть меньше 0,
* Промокод,
* Адрес доставки – строковое поле, не более 500 символов, не допускаются значения NULL и пустые строки,
* Дата заказа – не может быть ранее 01.01.2015 г.
* Дата доставки,
* Способ оплаты – по умолчанию «Карта»,
* Заказчик – внешний ключ к таблице *Заказчики* с опцией каскадного удаления.

Таблица *Заказчики* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Имя - строковое поле, не более 200 символов, не допускаются значения NULL и пустые строки,
* Телефон – не допускаются значения NULL и пустые строки.

**Вариант 21 - ИС «Сервисный центр по ремонту компьютерной техники»**

Создайте две таблицы – *Заказы* и *Виды техники.*

Таблица *Заказы* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Вид техники – внешний ключ к таблице *Виды техники* с опцией SET NULL,
* Сумма заказа – не может быть меньше 0,
* Дата заказа – не может быть ранее 01.01.2015 г.
* Способ оплаты – по умолчанию «Карта»

Таблица *Виды техники* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Наименование - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Модель – строковое значение, не более 300 символов.

**Вариант 22 - ИС «Салон красоты»**

Создайте две таблицы – *Услуги* и *Мастера.*

Таблица *Услуги* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Наименование услуги - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Цена (руб) – не может быть меньше 0, по умолчанию 500,
* Среднее время выполнения (мин) – не может быть меньше 0,
* Мастер – внешний ключ к таблице *Мастера* с опцией SET NULL.

Таблица *Мастера* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* ФИО - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Дата начала работы – не может быть ранее 05.06.2017 г.

**Вариант 23 - ИС «Санитарно-курортный комплекс»**

Создайте две таблицы – *Процедуры* и *Кабинеты.*

Таблица *Процедуры* должна содержать следующие поля:

* Идентификатор – первичный ключ,
* Наименование процедуры - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Среднее время выполнения (мин) – не может быть меньше 0,
* Количество мест – по умолчанию 1,
* Кабинет – внешний ключ к таблице *Кабинеты* с опцией SET NULL.

Таблица *Кабинеты* должна содержать следующие поля:

* Идентификатор (номер) – первичный ключ,
* Примечание - строковое поле, не более 300 символов, не допускаются значения NULL и пустые строки, уникальное значение,
* Ответственный (ФИО) – строковое поле, не более 300 символов.

# **Лабораторная работа №3** **Добавление, изменение и удаление данных.**

**Добавление данных**

Для добавления данных применяется команда INSERT, которая имеет следующий вид:

INSERT INTO имя\_таблицы [(поле1, поле2, …, полеN)] VALUES (значение1, значение2, ... значениеN)

Вначале идет выражение INSERT INTO и имя таблицы, в которую необходимо вставить данные. Затем в скобках можно указать список столбцов через запятую, в которые нужно добавить данные. Если не указывать эти значения в скобках, то новые данные будут вставляться в порядке по умолчанию. После этого в конце после слова VALUES в скобках перечислить добавляемые значения.

Например, добавим в таблицу *Employees* записи, в скобках после INSERT INTO имя\_таблицы укажем названия столбцов, в которые будем вставлять значения:

INSERT INTO Employees (FirstName, LastName, BirthDate, Email)

VALUES(N'Игорь', N'Ларин', '1990-01-01', 'i.larin@gmail.com')

Обратите внимание, что перед вставкой значения nvarchar необходимо указать N, например:

N'Игорь'

N указывает константную строку Unicode.

Аналогично можно добавить новые значения, не перечисляя список столбцов:

INSERT INTO Employees VALUES(N'Игорь', N'Ларин', '1990-01-01', 'i.larin@gmail.com')

Для неуказанных столбцов будет добавляться значение по умолчанию, если задан атрибут DEFAULT, или значение NULL. При этом неуказанные столбцы должны допускать значение NULL или иметь атрибут DEFAULT.

Например, добавим нового сотрудника, не указав дату рождения:

INSERT INTO Employees (FirstName, LastName, Email)

VALUES(N'Игорь', N'Ларин', 'i.larin@gmail.com')

Либо указав ключевое слово DEFAULT. В этом случае будет вставлено значение по умолчанию, а если его нет, то NULL, при условии, что в столбце допускаются NULL-значения:

INSERT INTO Employees (FirstName, LastName, BirthDate, Email)

VALUES(N'Игорь', N'Ларин', DEFAULT, 'i.larin@gmail.com')

Также можно добавить несколько новых записей в одной команде:

INSERT INTO Employees (FirstName, LastName, BirthDate, Email)

VALUES(N'Дмитрий', N'Берин', '1990-05-01', 'd.berin@gmail.com'),

(N'Мария', N'Кузнецова', '1986-03-21', 'mariya.k@gmail.com'),

('John', 'Newman', '1992-11-11', 'j.newman@gmail.com')

**Изменение данных**

Для изменения уже имеющихся строк в таблице применяется команда UPDATE. Она имеет следующий синтаксис:

UPDATE имя\_таблицы

SET столбец1 = значение1, столбец2 = значение2, ... столбецN = значениеN

[FROM выборка AS псевдоним\_выборки]

[WHERE условие\_выборки]

Ключевое слово WHERE служит для указания условий выборки из таблицы. Более подробно оно будет рассмотрено в следующих лабораторных работах.

Изменим дату рождения у всех сотрудников на ‘1992-04-04’:

UPDATE Employees SET BirthDate = '1992-04-04'

Изменения будут применены ко всем записям в таблице, т.к. не указано условие выборки с помощью ключевого слова WHERE.

Теперь изменим дату рождения только для сотрудника с именем Дмитрий Берин:

UPDATE Employees SET BirthDate = '1991-04-01'

WHERE FirstName = N'Дмитрий' AND LastName = N'Берин'

Рассмотрим более сложный запрос: изменим дату рождения только для первых двух сотрудников:

UPDATE Employees SET BirthDate = '1994-06-05'

FROM

(SELECT TOP 2 \* FROM Employees) AS Selected

WHERE Employees.Id = Selected.Id

С помощью подзапроса после ключевого слова FROM производится выборка первых двух строк. Для этой выборки определен псевдоним Selected. Псевдоним указывается после оператора AS.

Далее идет условие обновления Employees.Id = Selected.Id. То есть фактически мы работаем с двумя таблицами - Employees и Selected (которая является производной от Employees). В Selected находятся две первые строки. В Employees - все строки. И обновление производится только для тех строк, которые есть в выборке Selected. То есть если в таблице Employees десятки сотрудников, то обновление коснется только двух первых из них.

**Удаление данных**

Для удаления данных используется ключевое слово DELETE.

DELETE [FROM] имя\_таблицы

WHERE условие\_удаления

Например, удалим сотрудников, у которых дата рождения равна ‘1994-06-05’:

DELETE FROM Employees

WHERE BirthDate = '1994-06-05'

Если необходимо удалить все записи из таблицы вне зависимости от условий, то достаточно выполнить следующий запрос:

DELETE FROM Employees

Либо более короткая версия:

DELETE Employees

**Задание к лабораторной работе**

Заполнить данными таблицы, созданные в лабораторной работе №2:

1. Вставить одну запись в запросе.
2. Вставить несколько записей в запросе.
3. Вставить данные, указав список столбцов.
4. Вставить данные, не указывая значения для столбцов со значениями по умолчанию.
5. Дополните таблицы данными, как минимум, до 20 записей в одной таблице и как минимум 15 записей в другой. Используйте только валидные данные, избегая случайно набранных (например, «врфыов», «hdsasd» и т.д.)

Обновить данные в таблице:

1. Все записи в таблице.
2. По заданному условию в соответствии с вариантом:

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Название** | **Условие** |
| 1 | ИС «Электронная ведомость ВУЗа» | Установить ученое звание = «доцент» для всех преподавателей, у которых ученая степень не равна «не имеет» |
| 2 | ИС «Пассажирское автопредприятие» | Установить дату выпуска = 1 января 2000 г. для всех транспортных средств, у которых пробег больше 10000км |
| 3 | ИС «Таксопарк» | Установить дату выпуска = 13 апреля 2006 г. для всех транспортных средств, у которых марка = ‘Mazda’ и модель = ‘CX-9’ |
| 4 | ИС «Электронная регистратура поликлиники» | Установить специализацию = «Эндокринология» для всех сотрудников, у которых дата начала работы позднее 4 апреля 2010 года |
| 5 | ИС «Библиотека» | Установить жанр = «Детектив» для всех книг, у которых дата издания ранее 15 мая 1965 года |
| 6 | ИС «Гостиница» | Установить количество комнат = 3 для всех номеров, у которых количество мест больше 6 |
| 7 | ИС «Ресторан» | Установить количество мест = 4 для всех столиков, которые находятся в зале №2 |
| 8 | ИС «Кинотеатр» | Установить количество мест = 120 для всех залов, типом которых является 3D |
| 9 | ИС «Автосалон» | Установить цену = 600000 для всех машин, у которых пробег более 10000 км |
| 10 | ИС «Цветочный магазин» | Установить цену продажи = цена\_поставщика + 400руб для всех цветов, у которых цена более 1000 руб. |
| 11 | ИС «ЗАГС» | Установить дату регистрации = дате подачи заявления для всех заявлений, у которых дата подачи заявления ранее 1 января 2000 г. |
| 12 | ИС «Отдел кадров предприятия» | Установить страну = «Белоруссия» для всех сотрудников, у которых город = «Минск» |
| 13 | ИС «Паспортный стол» | Установить место выдачи = «Астрахань» для всех паспортов, у которых код подразделений = «300-000» |
| 14 | ИС «Складской учет на предприятии» | Установить сумму = 1000 руб для всех накладных, у которых дата поставки ранее 1 января 2006 года. |
| 15 | ИС «Туристическое агентство» | Установить цену = 20000 руб для всех путевок, у которых город = «Москва» |
| 16 | ИС «Автосервис» | Установить цену = 1000 руб для всех транспортных средств, у которых услуга = «тех.осмотр» |
| 17 | ИС «Кондитерское предприятие» | Установить цену = 250 руб для эклеров. |
| 18 | ИС «Ателье мод» | Установить цену = 100 руб для всех неготовых изделий. |
| 19 | ИС «Электронный школьный дневник» | Установить этаж = 2 для всех классов, у которых кабинет = 201. |
| 20 | ИС «Служба доставки» | Указать промокод = ‘JANUARY20’ для всех заказов, у которых дата заказа была в январе 2019г. |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» | Снизить сумму заказа на 100 руб для всех заказов, у которых способ оплаты = «Карта» |
| 22 | ИС «Салон красоты» | Увеличить цену услуги на 200 руб для всех услуг, у которых среднее время выполнения больше 90 минут |
| 23 | ИС «Санитарно-курортный комплекс» | Установить количество мест = 3 для всех процедур, у которых среднее время выполнения меньше 60 минут |

Удалить записи из таблицы в соответствии с заданным условием:

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Название** | **Условие** |
| 1 | ИС «Электронная ведомость ВУЗа» | Удалить всех преподавателей, у которых ученая степень равна «не имеет» |
| 2 | ИС «Пассажирское автопредприятие» | Удалить все транспортные средства с пробегом более 15000 км |
| 3 | ИС «Таксопарк» | Удалить все классы, цена которых менее 300 руб. |
| 4 | ИС «Электронная регистратура поликлиники» | Удалить должность «Офтальмолог» |
| 5 | ИС «Библиотека» | Удалить все книги с жанром «Фэнтези» |
| 6 | ИС «Гостиница» | Удалить все номера, у которых количество комнат менее 2. |
| 7 | ИС «Ресторан» | Удалить все столики, у которых больше 8 мест. |
| 8 | ИС «Кинотеатр» | Удалить все залы с количеством мест меньше 100. |
| 9 | ИС «Автосалон» | Удалить все автомобили, у которых дата выпуска ранее 1 января 2005 года. |
| 10 | ИС «Цветочный магазин» | Удалить все цветы, у которых цена поставщика более 3000 руб. |
| 11 | ИС «ЗАГС» | Удалить все заявления, у которых дата подачи заявления ранее 1 января 2000 года. |
| 12 | ИС «Отдел кадров предприятия» | Удалить все отделы города Самара. |
| 13 | ИС «Паспортный стол» | Удалить все паспорта, который были выданы ранее 1 января 2001 года. |
| 14 | ИС «Складской учет на предприятии» | Удалить все накладные с суммой менее 500 руб. |
| 15 | ИС «Туристическое агентство» | Удалить все туристические путевки в Кисловодск. |
| 16 | ИС «Автосервис» | Удалить все транспортные средства с ценой за услугу менее 750 руб. |
| 17 | ИС «Кондитерское предприятие» | Удалить все кондитерские изделия, которых нет в наличии. |
| 18 | ИС «Ателье мод» | Удалить все готовые изделия с ценой менее 300 руб. |
| 19 | ИС «Электронный школьный дневник» | Удалить всех школьников, которые поступали ранее 1 сентября 2007 года. |
| 20 | ИС «Служба доставки» | Удалить все заказы, у которых дата доставки ранее 12 октября 2015 года. |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» | Удалить все заказы, у которых способ оплаты = «Наличные» и сумма заказа менее 250 руб. |
| 22 | ИС «Салон красоты» | Удалить все услуги, у которых среднее время выполнения менее 20 мин. |
| 23 | ИС «Санитарно-курортный комплекс» | Удалить все процедуры, у которых количество мест более 5. |

# **Лабораторная работа №4 Выборка данных. Фильтрация.**

Для получения данных применяется команда SELECT. В упрощенном виде она имеет следующий синтаксис:

SELECT список\_столбцов FROM имя\_таблицы

Для выбора абсолютно всех столбцов используется знак звездочка (\*).

Например, выберем все данные из таблицы *Employees*.

SELECT \* FROM EMPLOYEES

Если необходимо получить не все столбцы, а только определенные, необходимо перечислить их названия после ключевого слова SELECT.

Например, выберем из таблицы *Employees* только имя, фамилию и email:

SELECT FirstName, LastName, Email FROM EMPLOYEES

Если название столбца совпадает с каким-либо ключевым словом (например, User, Table и т.д.) необходимо заключить его в квадратные скобки (например, [User], [Table]).

С помощью оператора **AS** можно изменить название выходного столбца или задать его псевдоним.

Например, зададим псевдонимы для столбцов FirstName и LastName, Имя и Фамилия соответственно.

select FirstName as 'Имя', LastName as 'Фамилия' from Employees

Оператор DISTINCT позволяет выбрать уникальные значения. Он убирает все одинаковые записи по выбранным столбцам.

SELECT DISTINCT FirstName, LastName, BirthDate, Email FROM Employees

Оператор ORDER BY позволяет отсортировать извлекаемые значения в определенном порядке. По умолчанию значения сортируются по возрастанию (ASC).

Например, отсортируем извлекаемые значения из таблицы Employees по полю FirstName, LastName.

SELECT FirstName, LastName, Email

FROM EMPLOYEES

ORDER BY FirstName, LastName

В данном случае данные будут отсортированы по возрастанию, сначала по полю FirstName, затем по полю LastName.

Для сортировки по убыванию необходимо указать ключевое слово DESC.

SELECT FirstName, LastName, Email

FROM EMPLOYEES

ORDER BY FirstName DESC

Также можно для каждого столбца указать способ сортировки.

Например, отсортируем по полю FirstName по возрастанию, а по полю LastName по убыванию.

SELECT FirstName, LastName, Email

FROM EMPLOYEES

ORDER BY FirstName ASC, LastName DESC

Оператор **TOP** позволяет выбрать определенное количество строк из выборки. В скобках после слова **TOP** указывается количество.

Например, выберем первые две строки из таблицы *Employees.*

SELECT TOP(2) FirstName, LastName, Email

FROM EMPLOYEES

ORDER BY FirstName ASC, LastName DESC

Дополнительный оператор **PERCENT** позволяет выбрать процентное количество строк из таблицы. Например, выберем 50% строк:

SELECT TOP 50 PERCENT FirstName, LastName, Email

FROM EMPLOYEES

ORDER BY FirstName ASC, LastName DESC

Если необходимо извлечь строки из любого места используются операторы **OFFSET** и **FETCH.** Эти операторы применяются только в отсортированных выборках с помощью ключевого слова **ORDER BY**.

SELECT \* FROM EMPLOYEES ORDER BY FirstName OFFSET 2 ROWS

В данном случае будет пропущено 2 строки.

Теперь выберем только 3 строки, начиная с третьей.

SELECT \* FROM EMPLOYEES

ORDER BY FirstName

OFFSET 2 ROWS

FETCH NEXT 3 ROWS ONLY

После оператора **FETCH** указывается ключевое слово **FIRST** или **NEXT** (какое именно в данном случае не имеет значения) и затем указывается количество строк, которое надо получить.

**Фильтрация данных**

Для фильтрации данных используется ключевое слово **WHERE**. После ключевого слова указывается условие.

WHERE условие

Если условие истинно, то строка попадает в результирующую выборку. В качестве можно использовать операции сравнения. Эти операции сравнивают два выражения. В T-SQL можно применять следующие операции сравнения:

**=**: сравнение на равенство

<>: сравнение на неравенство

<: меньше чем

>: больше чем

!<: не меньше чем

!>: не больше чем

<=: меньше чем или равно

>=: больше чем или равно

Например, найдем всех сотрудников, которых зовут «Роман»:

SELECT \* FROM EMPLOYEES WHERE FirstName = N'Роман'

Регистр в данном случае не имеет значения.

Для рассмотрения дальнейших примеров создадим таблицу *Smartphones* и заполним ее данными*.*

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

Например, выберем из таблицы *Smartphones* все смартфоны, цена которых больше 50000.

SELECT \* FROM SMARTPHONES WHERE Price >= 50000

Для того чтобы объединить несколько условий, используются логические операторы.

**AND** – логическое И. В результирующую выборку попадут только те данные, которые удовлетворяют всем условиям.

Например, выберем смартфоны, цена которых больше 50000 и в наличии имеется более 3 штук.

SELECT \* FROM SMARTPHONES WHERE Price >= 50000 AND Count >= 3

**OR –** логическое ИЛИ. В результирующую выборку попадут те данные, которые удовлетворяют хотя бы одному условию.

Например, выберем смартфоны, цена которых больше 50000 или меньше 25000.

SELECT \* FROM SMARTPHONES WHERE Price >= 50000 OR Price < 25000

**NOT –** логическое отрицание. Если выражение ложно, то общее условие истинно.

Например, выберем смартфоны, производителем которых не является компания Apple.

SELECT \* FROM SMARTPHONES WHERE NOT Manufacturer = 'Apple'

Но в большинстве случаев данное условие лучше написать следующим образом:

SELECT \* FROM SMARTPHONES WHERE NOT Manufacturer <> 'Apple'

Также можно использовать несколько операторов в одном условие. Например, нужно выбрать те смартфоны, которые стоят больше 50000 и которых в наличии 3 штуки, либо смартфоны, которые стоят меньше 25000 и которых в наличии более 7 штук.

SELECT \* FROM SMARTPHONES

WHERE (Price >= 50000 AND Count = 3) OR (Price < 25000 AND Count > 7)

Ряд столбцов может допускать значение NULL. Это значение не эквивалентно пустой строке. NULL представляет полное отсутствие какого-либо значения. И для проверки на наличие подобного значения применяется оператор IS NULL.

Например, найдем все смартфоны, у которых не указана цена.

SELECT \* FROM SMARTPHONES WHERE Price IS NULL

**Операторы фильтрации**

Оператор **IN** позволяет определить набор значений, который будет в результирующей выборке.

WHERE выражение [NOT] IN (выражение)

Выражение в скобках после **IN** определяет набор значений.

Например, выберем все смартфоны, производителями которых являются Apple или Samsung.

SELECT \* FROM SMARTPHONES WHERE Manufacturer IN ('Apple', 'Samsung')

Это выражение можно заменить следующим:

SELECT \* FROM SMARTPHONES

WHERE Manufacturer = 'Apple' OR Manufacturer = 'Samsung'

Используя оператор NOT можно найти все смартфоны, производителями которых НЕ являются Apple и Samsung.

SELECT \* FROM SMARTPHONES

WHERE Manufacturer NOT IN ('Apple', 'Samsung')

Оператор **BETWEEN** определяет диапазон значений с помощью начального и конечного значения, которому должно соответствовать выражение:

WHERE выражение [NOT] BETWEEN начальное\_значение AND конечное\_значение

Например, найдем все смартфоны, цена которых находится между 15000 и 30000.

SELECT \* FROM SMARTPHONES WHERE Price BETWEEN 15000 AND 30000

Используя оператор NOT можно найти все смартфоны, цена которых находится за пределами промежутка 15000 и 30000.

SELECT \* FROM SMARTPHONES WHERE Price NOT BETWEEN 15000 AND 30000

Оператор **LIKE** принимает шаблон строки, которому должно соответствовать выражение.

Для определения шаблона могут применяться ряд специальных символов подстановки:

%: соответствует любой подстроке, которая может иметь любое количество символов, при этом подстрока может и не содержать ни одного символа

\_: соответствует любому одиночному символу

[ ]: соответствует одному символу, который указан в квадратных скобках

[ - ]: соответствует одному символу из определенного диапазона

[ ^ ]: соответствует одному символу, который не указан после символа ^

Примеры использования подстановок:

WHERE Name LIKE 'iPhone%'

Соответствует таким значениям как "iPhone X" или "iPhone XS"

WHERE ProductName LIKE iPhone X\_'

Соответствует таким значениям как "iPhone XS" или "iPhone XR"

WHERE Name LIKE 'iPhone [78]'

Соответствует таким значениям как "iPhone 7" или "iPhone8"

WHERE Name LIKE 'iPhone [6-8]'

Соответствует таким значениям как "iPhone 6", "iPhone 7" или "iPhone 8"

WHERE Name LIKE 'iPhone [^7]%'

Соответствует таким значениям как "iPhone 6", "iPhone 6S" или "iPhone8". Но не соответствует значениям "iPhone 7" и "iPhone 7S"

WHERE ProductName LIKE 'iPhone [^1-6]%'

Соответствует таким значениям как "iPhone 7", "iPhone 7S" и "iPhone 8". Но не соответствует значениям "iPhone 5", "iPhone 6" и "iPhone 6S"

**Задание к лабораторной работе**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Название** | **Условие** |
| 1 | ИС «Электронная ведомость ВУЗа» | 1. Выбрать все данные из таблицы *Дисциплины*. 2. Выбрать всех экзаменаторов (поля *ФИО*, *Ученое звание*), у которых ученое звание = «доцент», отсортировать по полю ФИО по возрастанию. Для поля ФИО задать псевдоним «Фамилия Имя Отчество» 3. Выбрать первые 5 дисциплин (поля *Название, Описание*), которые содержатся в списке «Экономика», «Педагогика», «Иностранный язык». 4. Выбрать всех преподавателей, у которых ученая степень = «доктор технических наук», а ученое звание = «профессор» или «доцент». 5. Выбрать все дисциплины (поля *Название, Описание*), которые содержат подстроку «матем». |
| 2 | ИС «Пассажирское автопредприятие» | 1. Выбрать все данные из таблицы *Транспортные средства*. 2. Выбрать все модели транспортных средств (поля *Модель, Марка, Пробег*), у которых пробег меньше 5000 км, отсортировать по полю *Марка* по возрастанию, а по полю по убыванию. 3. Выбрать первые 7 транспортных средств (поля *Модель, Дата начала эксплуатации*), у которых дата начала эксплуатации позже 1 января 2017 года. Для поля «Дата начала эксплуатации» задать псевдоним «Дата». 4. Выбрать все модели транспортных средств, у которых марка = BMW и дата выпуска ранее 25 декабря 2015 года или марка = Toyota и дата выпуска позднее 30 марта 2016 года. 5. Выбрать все транспортные средства, ФИО водителя которых начинается на букву ‘А’. |
| 3 | ИС «Таксопарк» | 1. Выбрать все данные из таблицы *Транспортные средства*. 2. Выбрать все модели транспортных средств (поля *Марка, Модель, Дата начала эксплуатации*), у которых дата начала эксплуатации ранее 20 февраля 2015 года, отсортировать по полю *Дата выпуска* по убыванию. 3. Выбрать первые 2 класса (поля *Название, Цена*), цена которых находится между 300 и 1000 руб. Для поля *Название* задать псевдоним «Название класса». 4. Выбрать все транспортные средства, у которых марка = “Mazda” и модель = “CX 7”. 5. Выбрать все транспортные средства, ФИО водителя которых содержит подстроку ‘Иван’. |
| 4 | ИС «Электронная регистратура поликлиники» | 1. Выбрать все данные из таблицы *Должность*. 2. Выбрать всех сотрудников (поля *ФИО, Дата рождения, Специализация*), у которых дата рождения ранее 1 января 1960 года, отсортировать по полю ФИО по возрастанию. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 3. Выбрать всех сотрудников (поля *ФИО, Дата рождения, Дата начала работы* ), у которых дата начала работы попадает в период с 25 марта 2015 года по 25 марта 2019 года. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 4. Выбрать всех сотрудников, у которых ФИО содержит слово «Дмитрий», специализация = «Хирургия» и дата начала работы ранее 1 января 2000г. 5. Выбрать все должности, у которых название содержит слово «Заведующий». |
| 5 | ИС «Библиотека» | 1. Выбрать все данные из таблицы *Авторы*. 2. Выбрать все книги (поля *Название, Дата издания*), изданные ранее 2 февраля 1950 года, отсортировать по полю *Название* по убыванию. 3. Выбрать все книги (поля *Название, Жанр*), которые попадают в следующий список жанров «Фантастика», «Роман», «Детектив», отсортировать по ФИО автора. Для поля *Жанр* задать псевдоним «Жанр книги». 4. Выбрать все книги, у которых *Дата издания* находится в промежутке с 1 января 1990 года по 1 января 2010 года и жанр = «Фантастика» или название книги содержит «Поттер». 5. Выбрать всех авторов, ФИО которых содержит «Толстой». |
| 6 | ИС «Гостиница» | 1. Выбрать все данные из таблицы *Горничные*. 2. Выбрать всех горничных (поля *ФИО, Дата начала работы*), которые начали работать позже 15 мая 2017 года, отсортировать по ФИО. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 3. Выбрать все номера (поля *Идентификатор, Количество комнат, Количество мест*), в которых количество комнат меньше 3 и количество мест больше или равно 4. 4. Выбрать всех горничных, которые родились раньше 11 ноября 1970 года и начали работать раньше 1 января 2000г. 5. Выбрать всех горничных, у которых в поле *Паспортные данные* содержится «Астрахань». |
| 7 | ИС «Ресторан» | 1. Выбрать все данные из таблицы *Официанты*. 2. Выбрать всех официантов (поля *ФИО, Дата начала работы*), которые начали работать позже 30 июня 2014 года, отсортировать по ФИО. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 3. Выбрать все столики (поля *Номер стола, Количество мест*), в которых количество мест больше или равно 4. 4. Выбрать все столики, которые находятся в 1 или 2 зале и могут уместить от 3 до 7 гостей. 5. Выбрать всех официантов, у которых *ФИО* начинается на ‘Б’. |
| 8 | ИС «Кинотеатр» | 1. Выбрать все данные из таблицы Залы. 2. Выбрать все фильмы (поля *Название, Дата начала показа, Дата окончания показа*), которые поступили в прокат 20 декабря 2018 года, отсортировать по названию. Для поля *Название* задать псевдоним «Название фильма». 3. Выбрать все залы (поля *Номер зала, Количество мест*), в которых количество мест больше 80. 4. Выбрать все фильмы, которые поступили в прокат позже 20 января 2019 года и содержат в названии «Джон». 5. Выбрать все 2D залы с количеством мест меньше 70. |
| 9 | ИС «Автосалон» | 1. Выбрать все данные из таблицы Покупатели. 2. Выбрать все машины (поля *Марка, Модель, Пробег, Цена*), которые стоят больше миллиона, отсортировать по полю Пробег по возрастанию. Для поля *Цена* задать псевдоним «Цена, руб.». 3. Выбрать всех покупателей (поля ФИО, Дата рождения), которые родились позже 1 января 1990 года. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 4. Выбрать все машины, которые были выпущены позже 1 марта 2015 года и стоят меньше 1 400 000 или пробег которых меньше 7000. 5. Выбрать все машины, у которых модель содержит «CX». |
| 10 | ИС «Цветочный магазин» | 1. Выбрать все данные из таблицы Поставщики. 2. Выбрать все цветы (поля *Название, Цена поставщика, Цена продажи*), которые стоят от 200руб до 1000руб за штуку. Для поля *Название* задать псевдоним «Название цветка». 3. Выбрать всех поставщиков (Название, Страна) из России. Для поля *Название* задать псевдоним «Название поставщика». 4. Выбрать все цветы, название которых содержит слово «роза», а цена не находится в диапазоне от 400 руб до 700 руб. 5. Выбрать всех поставщиков, страна которых начинается на букву «А». |
| 11 | ИС «ЗАГС» | 1. Выбрать все данные из таблицы Заявители. 2. Выбрать все заявления (поля *Идентификатор, Дата подачи заявления*), которые были поданы ранее 1 января 2003 года. Для поля *Идентификатор* задать псевдоним «Номер заявления». 3. Выбрать всех заявителей (ФИО, Дата рождения, Страна, Город), которые родились в период с 31 декабря 1985 года по 1 января 2001 года. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 4. Выбрать все заявления, которые были поданы позже 1 января 2016 года или дата регистрации которых позже 1 марта 2016 года. 5. Выбрать всех заявителей, ФИО супруга или супруги которых содержит «Иванов». |
| 12 | ИС «Отдел кадров предприятия» | 1. Выбрать все данные из таблицы Сотрудники. 2. Выбрать все отделы (поля *Название, Страна*), из Белоруссии, отсортировать по названию. Для поля *Название* задать псевдоним «Название отдела». 3. Выбрать всех сотрудников (ФИО, Дата рождения), которые родились раньше 20 января 1970 года. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 4. Выбрать всех сотрудников, которые получили паспорт в Москве и родились в период с 1 января 1990 года по 1 января 2001 года. 5. Выбрать все отделы, в название которых входит слово «управ». |
| 13 | ИС «Паспортный стол» | 1. Выбрать все данные из таблицы Статусы. 2. Выбрать все паспорта (поля *Серия, Номер*), серия которых равна 1206, отсортировать по номеру. 3. Выбрать все паспорта (ФИО, Дата рождения), которые были выданы в Санкт-Петербурге. Для поля *ФИО* задать псевдоним «Фамилия Имя Отчество». 4. Выбрать все паспорта, которые были сделаны в Астрахани в период с 1 января 2005 года по 1 января 2010 года. 5. Выбрать все паспорта, ФИО которых содержит фамилию «Петров». |
| 14 | ИС «Складской учет на предприятии» | 1. Выбрать все данные из таблицы Накладные. 2. Выбрать все товары (поля *Название, Цена, Количество*), которых нет в наличии. 3. Выбрать товары (Название, Цена), которые стоят в пределах 400-3000 руб. 4. Выбрать все накладные (Идентификатор, Дата поставки, Сумма, Отправитель), которые были доставлены в период с 1 марта 2019 года по 1 июня 2019 года, сумма которых превышала 30000 и имя отправителя содержало «Иван». Для поля Идентификатор задать псевдоним «Номер накладной». 5. Выбрать все товары, которые начинаются на «М». |
| 15 | ИС «Туристическое агентство» | 1. Выбрать все данные из таблицы Туристы. 2. Выбрать все путевки (поля *Идентификатор, Страна, Город*), в Россию, цена которых меньше 30000, отсортировать по полю Цена. Для поля Идентификатор задать псевдоним «Номер путевки». 3. Выбрать путевки (Идентификатор, Страна, Цена), которые стоят в пределах от 10000 до 40000 руб. 4. Выбрать все путевки (Идентификатор, Страна, Город, Цена, Дата поездки, Дата покупки) не в России, которые были куплены в период с 1 декабря 2018 года по 31 декабря 2018 года или в период с 1 декабря 2017 года по 31 декабря 2017 года. Для поля Идентификатор задать псевдоним «Номер путевки». 5. Выбрать всех туристов с именем «Маргарита». |
| 16 | ИС «Автосервис» | 1. Выбрать все данные из таблицы Заказчики. 2. Выбрать все транспортные средства (поля *Марка, Модель, Причина обращения*), с причиной обращения «Тех.осмотр». 3. Выбрать всех заказчиков с телефоном, начинающимся на 896134. 4. Выбрать все транспортные средства (Идентификатор, Марка, Модель, Цена за услугу), которые были обслужены в период с 1 июня 2019 года по 31 августа 2019 года, и услуга которым стоила более 3000 руб. Для поля Идентификатор задать псевдоним «Номер обращения». 5. Выбрать всех заказчиков с фамилией «Ларин». |
| 17 | ИС «Кондитерское предприятие» | 1. Выбрать все данные из таблицы Витрины. 2. Выбрать все кондитерские изделия (поля *Название, Количество в наличии, Цена*), которых нет в наличии. 3. Выбрать все кондитерские изделия (поля *Идентификатор, Название,* *Количество в* *наличии, Цена*), цена которых меньше 200 руб, а количество больше 10 штук. Отсортировать по цене. Для поля Идентификатор задать псевдоним «Идентификатор изделия». 4. Выбрать все кондитерские изделия, которые в названии содержат «без глютена», цена которых от 200 руб до 500 руб, а в наличии не менее 5 штук. 5. Выбрать все витрины, в названии которых содержится «Сломана». |
| 18 | ИС «Ателье мод» | 1. Выбрать все данные из таблицы Заказчики. 2. Выбрать все готовые изделия (поля *Название, Цена*). 3. Выбрать все неготовые изделия (поля *Идентификатор, Название, Цена*), цена которых больше 10000руб. Отсортировать по цене в порядке убывания. Для поля Идентификатор задать псевдоним «Идентификатор изделия». 4. Выбрать всех заказчиков, у которых номер телефона начинается со следующей комбинации 8927865. 5. Выбрать все изделия, в названии которых нет слова «платье». |
| 19 | ИС «Электронный школьный дневник» | 1. Выбрать все данные из таблицы Классы. 2. Выбрать всех школьников (ФИО, Класс, Дата поступления), которые поступили позже 1 сентября 2008 года. Для поля ФИО задать псевдоним «Фамилия Имя Отчество». 3. Выбрать классы, которые находятся на 2 или 3 этаже и классным руководителем которых является преподаватель с фамилией «Круглова». 4. Выбрать всех школьников, которые поступили в период с 1 сентября 2002 года по 1 сентября 2008 года и у которых телефон родителя содержит комбинацию 891765. 5. Выбрать всех школьников с фамилией «Иванов». |
| 20 | ИС «Служба доставки» | 1. Выбрать все данные из таблицы Заказчики. 2. Выбрать все заказы (поля Идентификатор, Сумма заказа, Дата заказа), сумма которых была менее 1000 руб. Для поля *Идентификатор* задать псевдоним «Номер заказа». 3. Выбрать все заказы (поля Идентификатор, Сумма заказа, Состав заказа, Адрес доставки, Дата заказа), которые были заказаны в период с 1 сентября 2019 года по 30 сентября 2019 года, содержали в составе «пицца» и были доставлены на улицу «Ленина». Для поля *Идентификатор* задать псевдоним «Номер заказа». 4. Выбрать все заказы, которые были оплачены картой. 5. Выбрать все заказы с промокодом «SEPT20», отсортировать по полю *Сумма заказа* по убыванию. |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» | 1. Выбрать все данные из таблицы *Виды техники*. 2. Выбрать все заказы (поля Идентификатор, Сумма заказа, Дата заказа), сумма которых была более 3000 руб. Для поля *Идентификатор* задать псевдоним «Номер заказа». 3. Выбрать все заказы (поля Идентификатор, Сумма заказа, Дата заказа), которые были заказаны в период с 1 июня 2019 года по 31 августа 2019 года на сумму от 5000 до 10000руб. Для поля *Идентификатор* задать псевдоним «Номер заказа». 4. Выбрать все заказы, которые были оплачены картой. 5. Выбрать все виды техники, у которых модель содержит «Canon», отсортировать по полю *Наименование.* |
| 22 | ИС «Салон красоты» | 1. Выбрать все данные из таблицы *Мастера*. 2. Выбрать все услуги (поля Идентификатор, Наименование, Цена), которые стоят от 150руб до 500руб. Для поля *Идентификатор* задать псевдоним «Номер услуги». 3. Выбрать все услуги (поля Идентификатор, Наименование, Цена, Среднее время выполнения), которые длятся более 90минут и стоят меньше 1200руб или которые длятся менее 30 минут и стоят не более 500 руб. Для поля *Идентификатор* задать псевдоним «Номер услуги». Отсортировать по наименованию. 4. Выбрать всех мастеров, которые начала работать до 1 января 2019 года. 5. Выбрать все услуги, в название которых входит подстрока «ногт». |
| 23 | ИС «Санитарно-курортный комплекс» | 1. Выбрать все данные из таблицы *Кабинеты*. 2. Выбрать все процедуры (поля Идентификатор, Наименование, Количество мест), которые содержат количество мест больше 1. Для поля *Идентификатор* задать псевдоним «Номер процедуры». 3. Выбрать все процедуры (поля Идентификатор, Наименование, Среднее время выполнения, Количество мест), которые длятся более 90минут и мест на одного или которые длятся менее 30 минут и мест на 2. Для поля *Идентификатор* задать псевдоним «Номер процедуры». Отсортировать по наименованию. 4. Выбрать всех кабинеты, за которые отвечает «Петров». 5. Выбрать все процедуры, в название которых входит «массаж» и среднее время выполнения которых от 30 до 60 минут. |

# **Лабораторная работа №5 Группировка данных.**

**Агрегатные функции**

Агрегатные функции выполняют вычисления над значениями в наборе 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.

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

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

Функция **AVG** возвращает среднее значение в диапазоне значений столбца таблицы.

Найдем среднюю цену смартфонов:

SELECT AVG (Price) AS 'Средняя цена' FROM Smartphones

Также можно применить фильтрацию и найти среднюю цену только смартфонов Samsung.

SELECT AVG (Price) AS 'Средняя цена' FROM Smartphones

WHERE Manufacturer = 'Samsung'

Функция **COUNT** вычисляет количество строк в выборке.

Можно использовать двумя способами. Первый способ COUNT(\*) подсчитывает количество строк в выборке:

SELECT COUNT(\*) FROM Smartphones

Второй способ вычисляет количество строк по выбранному столбцу, при этом строки со значением NULL игнорируются:

SELECT COUNT(Manufacturer) FROM Smartphones

Функции **MIN** и **MAX** возвращают соответственно минимальное и максимальное значение по столбцу. Например, найдем минимальную цену среди смартфонов:

SELECT MIN(Price) FROM Smartphones

Максимальную цену:

SELECT MAX(Price) FROM Smartphones

Данные функции также игнорируют значения NULL и не учитывают их при подсчете.

Функция SUM вычисляет сумму значений столбца. Например, подсчитаем общее количество смартфонов:

SELECT SUM(Count) FROM Smartphones

По умолчанию все рассмотренные функции во время вычисления результата учитывают все строки выборки. Если необходимо учитывать только уникальные значения, используется ключевое слово **DISTINCT**.

SELECT AVG(DISTINCT Price) FROM Smartphones

По умолчанию вместо **DISTINCT** применяется оператор **ALL**.

SELECT AVG(ALL Price) FROM Smartphones

Но указание этого оператора необязательно, т.к. **ALL** применяется по умолчанию.

**Операторы GROUP BY и HAVING**

Для группировки данных в T-SQL применяются операторы **GROUP BY** и **HAVING**, для использования которых применяется 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] |

Оператор **GROUP BY** определяет, как строки будут группироваться.

Например, сгруппируем товары по производителю:

SELECT Manufacturer, COUNT(\*) FROM Smartphones

GROUP BY Manufacturer

Данное выражение выберет всех производителей и подсчитает для каждого количество смартфонов.

Обратите внимание, что любой столбец в выражении SELECT, кроме тех, что содержатся в агрегатных функциях, должен содержаться в выражении 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**. Так, следующий пример работать не будет, так как он не содержит выражение группировки:

SELECT Manufacturer, COUNT(\*) FROM Smartphones

Другой пример, добавим группировку по количеству товаров:

SELECT Name, Manufacturer, COUNT(\*) FROM Smartphones

GROUP BY Name, Manufacturer

Следует учитывать, что выражение **GROUP BY** должно идти после выражения **WHERE**, но до выражения **ORDER BY**:

SELECT Manufacturer, COUNT(\*) AS SmartphonesCount FROM Smartphones

WHERE Price > 30000

GROUP BY Manufacturer

ORDER BY ManufacturerCount

Оператор **HAVING** определяет, какие группы будут включены в выходной результат, то есть выполняет фильтрацию групп.

Применение **HAVING** во многом аналогично применению **WHERE**. Только если **WHERE** применяется к фильтрации строк, то **HAVING** используется для фильтрации групп.

Например, найдем все группы смартфонов по производителям, которых более 5 штук.

SELECT Manufacturer, COUNT(\*) AS SmartphonesCount FROM Smartphones

GROUP BY Manufacturer

HAVING COUNT(\*) > 5

При этом в одной команде мы можем использовать выражения **WHERE** и **HAVING**:

SELECT Manufacturer, COUNT(\*) AS SmartphonesCount FROM Smartphones

WHERE Price > 30000

GROUP BY Manufacturer

HAVING COUNT(\*) > 5

То есть в данном случае сначала фильтруются строки: выбираются те смартфоны, стоимость которых больше 30000. Затем выбранные смартфоны группируются по производителям. И далее фильтруются сами группы - выбираются те группы, в которых смартфонов больше 5.

**Задание к лабораторной работе**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Название** | **Условие** |
| 1 | ИС «Электронная ведомость ВУЗа» | 1. Посчитать всех экзаменаторов, у которых ученое звание = «доцент». 2. Выбрать все ученые звания и посчитать для них количество преподавателей. 3. Дополнить предыдущий запрос следующим условием: вывести только те звания, количество преподавателей с которыми больше 5. |
| 2 | ИС «Пассажирское автопредприятие» | 1. Найти минимальное значение пробега для всех моделей транспортных средств, у которых пробег меньше 5000 км. 2. Выбрать все марки и посчитать для каждой среднее значение пробега. 3. Дополнить предыдущий запрос следующим условием: вывести только те марки, среднее значение пробега у которых меньше 7000. |
| 3 | ИС «Таксопарк» | 1. Найти дату выпуска самого нового транспортного средства, у которого дата начала эксплуатации ранее 20 февраля 2015 года. 2. Выбрать все марки и для каждой марки найти самую раннюю дату выпуска. 3. Дополнить предыдущий запрос следующим условием: вывести только те марки, ранняя дата которых позже 1 января 2000 года. |
| 4 | ИС «Электронная регистратура поликлиники» | 1. Найти дату прихода самого нового сотрудника. 2. В таблице Сотрудники выбрать все должности и для каждой вычислить количество сотрудников. 3. Дополнить предыдущий запрос следующим условием: вывести только те должности, количество сотрудников в которой более 3. |
| 5 | ИС «Библиотека» | 1. Найти дату издания самой новой книги. 2. Выбрать все жанры и для каждого жанра посчитать количество книг. 3. Дополнить предыдущий запрос следующим условием: вывести только те жанры, количество книг в которых менее 10. |
| 6 | ИС «Гостиница» | 1. Выбрать самое максимально возможное количество комнат. 2. Из таблицы Номера выбрать всех горничных и для каждой посчитать количество номеров. 3. Дополнить предыдущий запрос следующим условием: вывести только тех горничных, количество номеров у которых меньше 3. |
| 7 | ИС «Ресторан» | 1. Выбрать самое максимально возможное количество мест за столом. 2. Из таблицы Столики выбрать все официантов и для каждого посчитать количество столиков. 3. Дополнить предыдущий запрос следующим условием: вывести только тех официантов, количество столиков у которых больше 2. |
| 8 | ИС «Кинотеатр» | 1. Найти дату самого раннего фильма, когда-либо показанного в кинотеатре. 2. Из таблицы фильмы выбрать Зал и для каждого посчитать количество фильмов. 3. Дополнить предыдущий запрос следующим условием: вывести только те залы, количество фильмов у которых меньше 2. |
| 9 | ИС «Автосалон» | 1. Найти среднее значение пробега для автомобилей, выпущенных позже 1 января 20014 года. 2. Выбрать все марки и для каждой посчитать максимальную цену автомобиля, представленного в таблице. 3. Дополнить предыдущий запрос следующим условием: вывести только те марки, максимальная цена которых меньше 1000000. |
| 10 | ИС «Цветочный магазин» | 1. Найти среднюю цену поставки цветов. 2. Выбрать всех поставщиков из таблицы Цветы и для каждого посчитать количество цветов. 3. Дополнить предыдущий запрос следующим условием: вывести только тех поставщиков, количество цветов у которых меньше 5. |
| 11 | ИС «ЗАГС» | 1. Найти дату самого первого заявления в период с 1 января 1990 года по 1 января 2000 года. 2. Из таблицы заявления выбрать всех заявителей, указанных в качестве Супруга и посчитать дл каждого количество заявлений. 3. Дополнить предыдущий запрос следующим условием: вывести только тех заявителей, количество заявлений у которых больше 1. |
| 12 | ИС «Отдел кадров предприятия» | 1. Найти самого молодого сотрудника с фамилией «Петров». 2. Из таблицы Сотрудники выбрать все отделы и для каждого посчитать количество сотрудников. 3. Дополнить предыдущий запрос следующим условием: вывести только те отделы, количество сотрудников в которых больше 7. |
| 13 | ИС «Паспортный стол» | 1. Найти дату подачи самого раннего паспорта в период с 1 января 2003 года по 1 января 2008 года. 2. Из таблицы Паспорта выбрать все статусы и для каждого посчитать количество паспортов. 3. Дополнить предыдущий запрос следующим условием: вывести только те статусы, количество паспортов в которых больше 10. |
| 14 | ИС «Складской учет на предприятии» | 1. Посчитать среднюю цену товаров, количество которых более 3. 2. Из таблицы Товары выбрать все накладные и для каждой накладной посчитать количество товаров. 3. Дополнить предыдущий запрос следующим условием: вывести только те накладные, количество товаров в которых меньше 10. |
| 15 | ИС «Туристическое агентство» | 1. Найти максимальную цену путевки в Россию. 2. Из таблицы Путевки выбрать всех туристов и для каждого туриста посчитать количество путевок. 3. Дополнить предыдущий запрос следующим условием: вывести только тех туристов, количество путевок которых больше 2. |
| 16 | ИС «Автосервис» | 1. Найти среднюю цену за услугу для машин марки «BMW». 2. Из таблицы *Транспортные средства* выбрать всех заказчиков и для каждого заказчика посчитать количество обращений. 3. Дополнить предыдущий запрос следующим условием: вывести только тех заказчиков, которые обращались более одного раза. |
| 17 | ИС «Кондитерское предприятие» | 1. Найти максимальную цену за кондитерское изделие, количество которого более 5 штук. 2. Из таблицы *Кондитерские изделия* выбрать все витрины и для каждой посчитать количество кондитерских изделий. 3. Дополнить предыдущий запрос следующим условием: вывести только те витрины, в которых количество кондитерских изделий больше 5. |
| 18 | ИС «Ателье мод» | 1. Найти минимальную цену среди неготовых изделий. 2. Из таблицы *Изделия* выбрать всех заказчиков и для каждого заказчика посчитать количество изделий. 3. Дополнить предыдущий запрос следующим условием: вывести только тех заказчиков, в которых количество изделий больше 3. |
| 19 | ИС «Электронный школьный дневник» | 1. Найти самую раннюю дату поступления среди школьников с фамилией «Петров». 2. Из таблицы Школьники выбрать все классы и для каждого класса посчитать количество школьников. 3. Дополнить предыдущий запрос следующим условием: вывести только те классы, в которых количество школьников больше 13. |
| 20 | ИС «Служба доставки» | 1. Посчитать среднюю сумму заказа в период с 31 декабря 2018 года по 10 января 2019 года. 2. Из таблицы *Заказы* выбрать всех заказчиков и для каждого заказчика посчитать количество заказов. 3. Дополнить предыдущий запрос следующим условием: вывести только те заказы, в которых количество заказов больше 5. |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» | 1. Найти максимальную сумму заказа в период с 1 января 2018 года по 31 декабря 2018 года. 2. Из таблицы заказы выбрать все виды техники и для каждого вида техники посчитать количество заказов. 3. Дополнить предыдущий запрос следующим условием: вывести только те виды техники, в которых количество заказов меньше 5. |
| 22 | ИС «Салон красоты» | 1. Найти максимальное среднее время выполнения услуги в ценовом диапазоне от 1000 до 5000 руб. 2. Из таблицы *Услуги* выбрать всех мастеров и для каждого мастера посчитать количество услуг. 3. Дополнить предыдущий запрос следующим условием: вывести только тех мастеров, у которых количество услуг больше 2. |
| 23 | ИС «Санитарно-курортный комплекс» | 1. Найти максимальное количество мест для процедуры со средним временем выполнения от 30мин до 60мин. 2. Из таблицы *Процедуры* выбрать все кабинеты и для каждого кабинета посчитать количество процедур. 3. Дополнить предыдущий запрос следующим условием: вывести только те кабинеты, в которых проводятся более 2 процедур. |

# **Лабораторная работа №6** **Нормализация баз данных.**

Процесс проектирования базы данных с использованием метода нормальных форм (НФ) является итерационным и заключается в последовательном переводе отношения из первой нормальной формы (1НФ) в нормальную форму более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивается определенным типом функциональных зависимостей и устранением соответствующих аномалий при выполнении операций над отношениями базы данных, а также сохранении свойств предшествующих нормальных форм.

Нормализация базы данных позволяет избежать избыточности и дублирования данных.

**Термины**

**Атрибут** — свойство некоторой сущности. Часто называется полем таблицы.

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

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

**Отношение** — конечное множество кортежей (таблица).

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

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

**Функциональная зависимость** между атрибутами (множествами атрибутов) X и Y означает, что для любого допустимого набора кортежей в данном отношении: если два кортежа совпадают по значению X, то они совпадают по значению Y. Например, если значение атрибута «Название компании» — Яндекс, то значением атрибута «Страна» в таком кортеже всегда будет «Россия». Обозначение: {X} -> {Y}.

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

**Метод нормальных форм (НФ)** состоит в сборе информации об объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений.

**Цель нормализации**: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей (строк таблицы).

**Аномалией** называется такая ситуация в таблице базы данных, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.

**Аномалии-модификации** проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.

**Аномалии-удаления** — при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.

**Аномалии-добавления** возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

**Первая нормальная форма**

Отношение находится в 1НФ, если все его атрибуты являются простыми (атомарными), все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Например, есть таблица *Смартфоны*:

|  |  |  |
| --- | --- | --- |
| **Id** | **Производители** | **Модели** |
| 1 | Samsung | Galaxy S7, Galaxy S8 Plus, Galaxy S9, Galaxy S10 |
| 2 | iPhone | 7, 8, X, XS, XR, 11 |
| 3 | Huawei | Honor 8, P20, P30 |

Нарушение нормализации 1НФ происходит в моделях, т.к. в одной ячейке содержится список из нескольких элементов. Например, в моделях Samsung находится список из 4 моделей: Galaxy S7, Galaxy S8 Plus, Galaxy S9, Galaxy S10, т.е. он не является атомарным. Преобразуем таблицу к 1НФ:

|  |  |  |
| --- | --- | --- |
| **Id** | **Производители** | **Модели** |
| 1 | Samsung | Galaxy S7 |
| 2 | Samsung | Galaxy S8 Plus |
| 3 | Samsung | Galaxy S9 |
| 4 | Samsung | Galaxy S10 |
| 5 | iPhone | 7 |
| 6 | iPhone | 8 |
| 7 | iPhone | X |
| 8 | iPhone | XS |
| 9 | iPhone | XR |
| 10 | iPhone | 11 |
| 11 | Huawei | Honor 8 |
| 12 | Huawei | P20 |
| 13 | Huawei | P30 |

Рассмотрим второй пример. Допустим, у нас есть таблица, которая хранит данные о сотрудниках:

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **ФИО** | **Дата рождения** | **Паспортные данные** |
| 1 | Ларин Сергей Петрович | 01.01.1982 | 13 02 111111 выдан 02.07.2015, код подразделения 300-001 |
| 2 | Морозов Алексей Сергеевич | 11.03.1991 | 11 10 555555 выдан 21.11.2003, код подразделения 300-002 |
| 3 | Дмитриев Константин Александрович | 23.04.1988 | 12 03 434343 выдан 17.07.2017, код подразделения 300-003 |

По таблице видно, что поле *ФИО* следует разбить на три отдельных поля: *Фамилия, Имя* и *Отчество*. Это повысит удобство при написании запросов. Поле *Паспортные данные*  также не является атомарным и его можно разбить на поля *Серия, Номер, Дата выдачи, Код подразделения.*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Id | Фамилия | Имя | Отчество | Дата рождения | Серия | Номер | Дата выдачи | Код подразделения |
| 1 | Ларин | Сергей | Петрович | 01.01.1982 | 1302 | 111111 | 02.07.2015 | 300-001 |
| 2 | Морозов | Алексей | Сергеевич | 11.03.1991 | 1110 | 555555 | 21.11.2003 | 300-002 |
| 3 | Дмитриев | Константин | Александрович | 23.04.1988 | 1203 | 434343 | 17.07.2017 | 300-003 |

**Вторая нормальная форма**

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа (ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

Рассмотрим следующие таблицы. Допустим у нас есть таблицы Поставщики, Клиенты, Сотрудники. Каждая из таблиц имеет поле Адрес.

Таблица *Поставщики*:

|  |  |  |
| --- | --- | --- |
| **Id** | **Наименование** | **Адрес** |
| 1 | ООО «Первый луч» | г. Москва, ул. Космонавтов 34, 3 |
| 2 | АО «Виктория» | г. Астрахань ул. Бэра 7 к2, 41 |
| 3 | ООО «Росинка» | г. Краснодар, ул. Школьная 13, 6 |

Таблица *Клиенты*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Фамилия** | **Имя** | **Отчество** | **Адрес** |
| 1 | Ларин | Игорь | Сергеевич | г. Астрахань, ул. Татищева 10А, 13 |
| 2 | Киров | Артем | Дмитриевич | г. Москва ул. Ленина 20, 22 |
| 3 | Дмитриев | Егор | Петрович | г. Астрахань, ул. Кирова 39, 9 |

Таблица *Сотрудники*:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Id** | **Фамилия** | **Имя** | **Отчество** | **Адрес** |
| 1 | Одинцов | Роман | Алексеевич | г. Москва, ул. Ленская 15, 6 |
| 2 | Зорин | Владислав | Владимирович | г. Астрахань, ул. Савушкина 5, 3 |
| 3 | Шишкин | Алексей | Игоревич | г. Астрахань, ул. Бакинская 79, 1 |

В рамках приведения таблиц к первой нормальной форме необходимо разделить адрес на атомарные части: индекс, город, улица, дом, корпус, квартира.

Например, таблица *Поставщики* приобретет следующий вид:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **Наименование** | **Индекс** | **Id города** | **Город** | **Улица** | **Дом** | **Корпус** | **Квартира** |
| 1 | ООО «Первый луч» | 101000 | 1 | Москва | Космонавтов | 34 |  | 3 |
| 2 | АО «Виктория» | 414000 | 2 | Астрахань | Бэра | 7 | 2 | 41 |
| 3 | ООО «Росинка» | 350000 | 3 | Краснодар | Школьная | 13 |  | 6 |

Аналогично можно привести к 1 нормальной форме остальные таблицы.

Видно, что таблицы имеют информацию об адресе. В рамках второй нормальной формы эту информацию можно выделить в отдельную таблицу *Адреса,* а в таблицы *Поставщики*, *Клиенты* и *Сотрудники* добавить внешний ключ к таблице *Адреса.*

Таблица *Поставщики* приобретет следующий вид:

|  |  |  |
| --- | --- | --- |
| **Id** | **Наименование** | **Адрес** |
| 1 | ООО «Первый луч» | 1 |
| 2 | АО «Виктория» | 2 |
| 3 | ООО «Росинка» | 3 |

Таблица *Адреса* будет выглядеть следующим образом:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **Индекс** | **Id города** | **Город** | **Улица** | **Дом** | **Корпус** | **Квартира** |
| 1 | 101000 | 1 | Москва | Космонавтов | 34 |  | 3 |
| 2 | 414000 | 2 | Астрахань | Бэра | 7 | 2 | 41 |
| 3 | 350000 | 3 | Краснодар | Школьная | 13 |  | 6 |
| 4 | 101000 | 1 | Москва | Центральная | 5 | 3 | 54 |

**Третья нормальная форма**

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Рассмотрим предыдущую таблицу *Адреса*:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Id** | **Индекс** | **Id города** | **Город** | **Улица** | **Дом** | **Корпус** | **Квартира** |
| 1 | 101000 | 1 | Москва | Космонавтов | 34 |  | 3 |
| 2 | 414000 | 2 | Астрахань | Бэра | 7 | 2 | 41 |
| 3 | 350000 | 3 | Краснодар | Школьная | 13 |  | 6 |
| 4 | 101000 | 1 | Москва | Центральная | 5 | 3 | 54 |

В данном случае данные в таблице *Адреса* зависят от города по Id и названию. Это избыточно и не соответствует третьей нормальной форме. Необходимо адреса вынести в отдельную таблицу-справочник *Города,* а в таблице Адреса оставить внешний ключ к новой таблице.

Таблица *Адреса* приобретет следующий вид:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Id** | **Индекс** | **Id города** | **Улица** | **Дом** | **Корпус** | **Квартира** |
| 1 | 101000 | 1 | Космонавтов | 34 |  | 3 |
| 2 | 414000 | 2 | Бэра | 7 | 2 | 41 |
| 3 | 350000 | 3 | Школьная | 13 |  | 6 |
| 4 | 101000 | 1 | Центральная | 5 | 3 | 54 |

Таблица *Города* будет иметь следующий вид:

|  |  |
| --- | --- |
| **Id города** | **Город** |
| 1 | Москва |
| 2 | Астрахань |
| 3 | Краснодар |
| 1 | Москва |

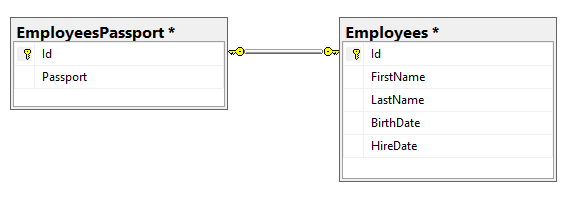
**Связывание двух таблиц**

**Один-к-одному**

Связь один к одному образуется, когда ключевой столбец (идентификатор) присутствует в другой таблице, в которой тоже является ключом либо свойствами столбца задана его уникальность (одно и тоже значение не может повторяться в разных строках).

На практике связь «один к одному» наблюдается не часто. Например, она может возникнуть, когда требуется разделить данных одной таблицы на несколько отдельных таблиц с целью безопасности.

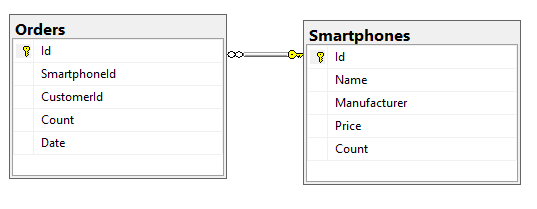
Например, в базе данных есть таблица *Сотрудники*, которой пользуются несколько отделов. Таблица большая и есть данные, к которым должны иметь доступ менеджеры, но не должны иметь доступ документоведы.



**Один-ко-многим**

Это наиболее часто встречаемый тип связей. В этом типе связей несколько строк из дочерний таблицы зависят от одной строки в родительской таблице.

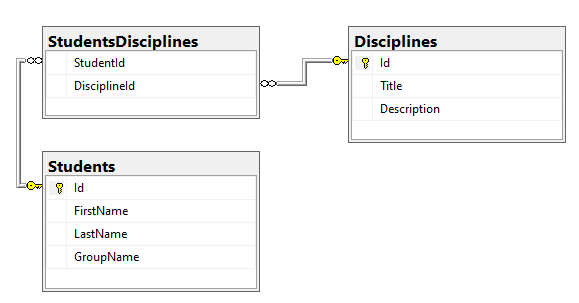
Например, рассмотрим таблицы *Смартфоны* и *Заказы.* Например, один смартфон может быть в нескольких заказах.



**Многие-ко-многим**

При этом типе связей одна строка из таблицы А может быть связана с множеством строк из таблицы В. В свою очередь одна строка из таблицы В может быть связана с множеством строк из таблицы А.

Типичный пример - студенты и курсы: один студент может посещать несколько курсов, и соответственно на один курс могут записаться несколько студентов.



**Задание к лабораторной работе**

Выделить основные сущности и привести схему к третьей нормальной форме. В результате должна получится такая полноценная база данных, которая будет использоваться в курсовой работе.

# **Лабораторная работа №7 Подзапросы, вложенные запросы.**

T-SQL поддерживает функциональность подзапросов (subquery), то есть таких запросов, которые могут быть встроены в другие запросы.

Например, создадим таблицы для смартфонов, покупателей и заказов:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

CREATE TABLE Orders

(Id INT PRIMARY KEY IDENTITY,

[SmartphoneId] INT NOT NULL REFERENCES Smartphones(Id) ON DELETE CASCADE,

[CustomerId] INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,

Count INT,

Date DATETIME)

Таблица *Orders* содержит ссылки на две другие таблицы через поля (внешние ключи) *SmartphoneId* и *CustomerId*.

Добавим в таблицы некоторые данные:

INSERT INTO Smartphones VALUES('Samsung Galaxy S8', 'Samsung', 35000, 3)

INSERT INTO Smartphones VALUES('Samsung Galaxy S8 plus', 'Samsung', 35000, 4)

INSERT INTO Smartphones VALUES('Samsung Galaxy S9', 'Samsung', 50000, 6)

INSERT INTO Smartphones VALUES('iPhone X', 'Apple', 40000, 5)

INSERT INTO Smartphones VALUES('iPhone XR', ' Apple', 50000, 7)

INSERT INTO Smartphones VALUES('iPhone XS', ' Apple', 56000, 5)

INSERT INTO Smartphones VALUES('iPhone 11', ' Apple', 65000, 10)

INSERT INTO Customers VALUES(N'Королев Ренат Александрович', '125412-565214', '89614528745', 100)

INSERT INTO Customers VALUES(N'Орлова Мария Викторовна', '418759-325698', '89174852503', 584)

INSERT INTO Customers VALUES(N'Дмитриев Николай Николаевич', '512586-023652', '89294158754', 397)

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone X'),

(SELECT Id FROM Customers WHERE CardNumber='125412-565214'),

2, '2019-04-01')

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone XR'),

(SELECT Id FROM Customers WHERE CardNumber='418759-325698'),

3, '2019-05-25')

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='Samsung Galaxy S9'),

(SELECT Id FROM Customers WHERE CardNumber='512586-023652'),

4, '2019-08-27')

Обратите внимание на добавление элементов в таблицу *Orders.* Например, первый заказ был сделан покупателем с картой 125412-565214 и товаром *iPhone X*. Для добавления записи об этом в таблицу *Orders* нам необходимы идентификаторы покупателя (CustomerId) и смартфона (SmartphoneId). Но на момент добавления этой записи мы можем не знать этих идентификаторов. Здесь могут помочь подзапросы.

Подзапрос выполняет команду SELECT и заключается в скобки. В данном же случае при добавлении одного заказа выполняется два подзапроса. Каждый подзапрос возвращает одного скалярное значение, например, числовой идентификатор.

В данном случае подзапросы выполнялись к другой таблице, но могут выполняться и к той же, к которой вызывается основной запрос. Например, найдем смартфоны (таблица Smartphones), которые имеют максимальную цену:

SELECT \* FROM Smartphones WHERE Price=(SELECT MAX(Price) FROM Smartphones)

Или найдем смартфоны, цена которых ниже средней:

SELECT \* FROM Smartphones WHERE Price<(SELECT AVG(Price) FROM Smartphones)

**Коррелирующие подзапросы**

Подзапросы бывают **коррелирующими** и **некоррелирующими**. В примерах выше команды SELECT выполняли фактически один подзапрос для всей команды, например, подзапрос возвращает максимальную или среднюю цену, которая не изменится, сколько бы мы строк не выбирали в основном запросе. То есть результат подзапроса не зависел от строк, которые выбираются в основном запросе. И такой подзапрос выполняется один раз для всего внешнего запроса.

Но также существуют коррелирующие подзапросы (correlated subquery), результаты которых зависят от строк, которые выбираются в основном запросе.

Например, выберем все заказы из таблицы *Orders*, добавив к ним информацию о смартфоне:

SELECT Date, Count,(SELECT Name FROM Smartphones WHERE Smartphones.Id = Orders.SmartphoneId) AS Smartphone

FROM Orders

Здесь для каждой строки из таблицы *Orders* будет выполняться подзапрос, результат которого зависит от столбца *SmartphoneId*. И каждый подзапрос может возвращать различные данные.

Коррелирующий подзапрос может выполняться и для той же таблицы, к которой выполняется основной запрос. Например, выберем из таблицы *Smartphones* те смартфоны, стоимость которых выше средней цены смартфонов для данного производителя:

SELECT Name, Manufacturer, Price,

(SELECT AVG(Price) FROM Smartphones AS SubSmartphones WHERE SubSmartphones.Manufacturer=MainSmartphones.Manufacturer) AS AvgPrice

FROM Smartphones AS MainSmartphones

WHERE Price > (SELECT AVG(Price) FROM Smartphones AS SubSmartphones

WHERE SubSmartphones.Manufacturer=MainSmartphones.Manufacturer)

В данном случае определено два коррелирующих подзапроса. Первый подзапрос определяет спецификацию столбца *AvgPrice*. Он будет выполняться для каждой строки, извлекаемой из таблицы *Smartphones*. В подзапрос передается производитель смартфона и на его основе выбирается средняя цена для смартфонов именно этого производителя. И так как производитель у товаров может отличаться, то и результат подзапроса в каждом случае также может отличаться.

Второй подзапрос аналогичен, только он используется для фильтрации извлекаемых из таблицы *Smartphones*. И также он будет выполняться для каждой строки.

Чтобы избежать двойственности при фильтрации в подзапросе при сравнении производителей (SubSmartphones.Manufacturer=MainSmartphones.Manufacturer) для внешней выборки установлен псевдоним MainSmartphones, а для выборки из подзапросов определен псевдоним SubSmartphones.

Следует учитывать, что коррелирующие подзапросы выполняются для каждой отдельной строки выборки, то выполнение таких подзапросов может замедлять выполнение всего запроса в целом.

**Подзапросы в командах SELECT**

В выражении SELECT мы можем вводить подзапросы четырьмя способами:

1. Использовать в условии в выражении WHERE
2. Использовать в условии в выражении HAVING
3. Использовать в качестве таблицы для выборки в выражении FROM
4. Использовать в качестве спецификации столбца в выражении SELECT

Рассмотрим некоторые из этих случаев. Например, получим все смартфоны, у которых цена выше средней:

SELECT \* FROM Smartphones WHERE Price>(SELECT AVG(Price) FROM Smartphones)

Чтобы получить нужные товары, нам вначале надо выполнить подзапрос на получение средней цены товара: SELECT AVG(Price) FROM Smartphones.

Или выберем всех покупателей из таблицы *Customers*, у которых нет заказов в таблице *Orders*:

SELECT \* FROM Customers WHERE Id NOT IN (SELECT CustomerId FROM Orders)

Хотя в данном случае подзапросы прекрасно справляются со своей задачей, стоит отметить, что это не самый эффективный способ для извлечения данных из других таблиц, так как в рамках T-SQL для сведения данных из разных таблиц можно использовать оператор JOIN, который будет рассмотрен в следующей лабораторной работе.

**Получение набора значений**

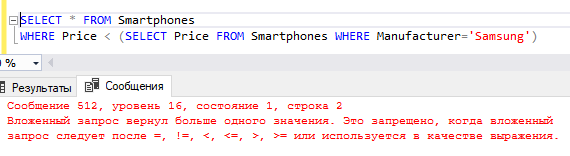
При использовании в операторах сравнения подзапросы должны возвращать одно скалярное значение. Но иногда возникает необходимость получить набор значений. Чтобы при использовании в операторах сравнения подзапрос мог возвращать набор значений, перед ним необходимо использовать один из операторов: ALL, SOME или ANY.

При использовании ключевого слова ALL условие в операции сравнения должно быть верно для всех значений, которые возвращаются подзапросом. Например, найдем все товары, цена которых меньше, чем у любого смартфона компании Samsung:

SELECT \* FROM Smartphones

WHERE Price < ALL(SELECT Price FROM Smartphones WHERE Manufacturer='Samsung')

Если бы мы в данном случае опустили бы ключевое слово ALL, то мы столкнулись бы с ошибкой.



Допустим, если подзапрос возвращает значения val1, val2 и val3, то условие фильтрации фактически было бы аналогично объединению этих значений через оператор AND:

WHERE Price < val1 AND Price < val2 AND Price < val3

В то же время подобный запрос гораздо проще переписать другим образом:

SELECT \* FROM Smartphones

WHERE Price <

(SELECT MIN(Price) FROM Smartphones WHERE Manufacturer='Samsung')

При применении ключевых слов ANY и SOME условие в операции сравнения должно быть истинным для хотя бы одного из значений, возвращаемых подзапросом. По действию оба этих оператора аналогичны, поэтому можно применять любое из них. Например, в следующем случае получим товары, которые стоят меньше самого дорогого смартфона компании Samsung:

SELECT \* FROM Smartphones

WHERE Price < ANY(SELECT Price FROM Smartphones WHERE Manufacturer='Samsung')

Этот запрос можно также упростить:

SELECT \* FROM Smartphones

WHERE Price <

(SELECT MAX(Price) FROM Smartphones WHERE Manufacturer='Samsung')

**Подзапрос как спецификация столбца**

Результат подзапроса может представлять отдельный столбец в выборке. Например, выберем все заказы и добавим к ним информацию о названии смартфона:

SELECT Date, Count,(SELECT Name FROM Smartphones WHERE Smartphones.Id = Orders.SmartphoneId) AS Smartphone FROM Orders

**Подзапросы в команде INSERT**

В команде INSERT подзапросы могут применяться для определения значения, которое вставляется в один из столбцов:

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone X'),

(SELECT Id FROM Customers WHERE CardNumber='125412-565214'),

2, '2019-04-01')

**Подзапросы в команде UPDATE**

В команде UPDATE подзапросы могут применяться:

1. В качестве устанавливаемого значения после оператора SET
2. Как часть условия в выражении WHERE

Так, увеличим количество купленных смартфонов на 3 в тех заказах, где номер карты покупателя = 125412-565214:

UPDATE Orders SET Count = Count + 2

WHERE CustomerId = (SELECT Id FROM Customers WHERE

CardNumber='125412-565214')

**Подзапросы в команде DELETE**

В команде DELETE подзапросы также применяются как часть условия. Так, удалим все заказы на Galaxy S8, которые сделал клиент с картой покупателя '125412-565214':

DELETE FROM Orders

WHERE CustomerId = (SELECT Id FROM Customers WHERE

CardNumber='125412-565214') AND

SmartphoneId = (SELECT Id FROM Smartphones WHERE

Name = 'Samsung Galaxy S8')

**Оператор EXISTS**

Оператор EXISTS позволяет проверить, возвращает ли подзапрос какое-либо значение. Как правило, этот оператор используется для индикации того, что какая-либо строка удовлетворяет условию. То есть фактически оператор EXISTS не возвращает строки, а лишь указывает, что в базе данных есть как минимум одна строка, которые соответствует данному запросу. Поскольку возвращения набора строк не происходит, то подзапросы с подобным оператором выполняются довольно быстро.

Применение оператора имеет следующий формальный синтаксис:

WHERE [NOT] EXISTS(подзапрос)

Например, найдем всех покупателей из таблицы Customers, которые когда-либо делали заказы:

SELECT \* FROM Customers

WHERE EXISTS (SELECT \* FROM Orders WHERE Orders.Id=Customers.Id)

Для получения аналогичного результата можно использовать оператор IN:

SELECT \* FROM Customers

WHERE Id IN (SELECT CustomerId FROM Orders WHERE Orders.Id=Customers.Id)

**Задание к лабораторной работе**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Название** | **Условие** |
| 1 | ИС «Электронная ведомость ВУЗа» | 1. Выбрать все дисциплины, у которых экзаменатор имеет ученое звание «доцент». 2. Выбрать все дисциплины с именем соответствующего экзаменатора. 3. Выбрать всех экзаменаторов, которые ведут хотя бы одну дисциплину. 4. Добавить новую дисциплину (Id экзаменатора выбрать подзапросом по полю ФИО) 5. Изменить ученое звание на «профессор» для всех экзаменаторов, у которых более 2 дисциплин. |
| 2 | ИС «Пассажирское автопредприятие» | 1. Выбрать все транспортные средства с моделью «Mazda CX7». 2. Выбрать все транспортные средства с названием соответствующей марки и модели. 3. Выбрать все транспортные средства, у которых пробег меньше среднего. 4. Добавить новое транспортное средство (Id модели выбрать подзапросом по полю *Марка* и *Модель*) 5. Увеличить пробег в два раза для всех транспортных средств, которые начали эксплуатировать раньше 1 января 2005 года. |
| 3 | ИС «Таксопарк» | 1. Выбрать все транспортные средства с классом «Комфорт». 2. Выбрать все транспортные средства с названием соответствующего класса. 3. Выбрать все транспортные средства, цена за поездку в которых больше средней. 4. Добавить новое транспортное средство (Id класса выбрать подзапросом по полю *Название*) 5. Увеличить цену в полтора раза для всех классов, транспортные средства которых начали эксплуатировать позже 1 января 2018 года. |
| 4 | ИС «Электронная регистратура поликлиники» | 1. Выбрать всех сотрудников с должностью «Медсестра». 2. Выбрать всех сотрудников с названием должности. 3. Выбрать все должности, которые не занимает ни один из сотрудников. 4. Добавить нового сотрудника (Id должности выбрать подзапросом по полю *Название*) 5. Изменить дату начала работы на 20 марта 2019 года для всех сотрудников, которые занимают должность «Заведующий отделением». |
| 5 | ИС «Библиотека» | 1. Выбрать все книги, автором которых является Пушкин Александр Сергеевич. 2. Выбрать все книги с именем автора. 3. Выбрать всех авторов, для которых нет ни одной книги. 4. Добавить новую книгу (Id автора выбрать подзапросом по полю *ФИО*). 5. Изменить дату издания на 30 августа 2006 года для всех книг, автором которых является Достоевский. |
| 6 | ИС «Гостиница» | 1. Выбрать все номера, горничные которых начали работать после 3 мая 2008 года. 2. Выбрать все номера с ФИО соответствующих горничных. 3. Выбрать всех горничных, которые не убирают ни одной комнаты. 4. Добавить новый номер (Id горничной выбрать подзапросом по полю *ФИО*). 5. Увеличить количество комнат в номере на 1 для всех номеров, горничные которых начали работать до 1 января 2018 года. |
| 7 | ИС «Ресторан» | 1. Выбрать все столики, официанты которых начали работать до 31 января 2019 года. 2. Выбрать все столики с ФИО соответствующих официантов. 3. Выбрать всех официантов, которые не обслуживают ни один столик. 4. Добавить новый столик (Id официанта выбрать подзапросом по полю *ФИО*). 5. Увеличить количество мест за столиком на 2 для всех столиков, официанты которых начали работать после 1 января 2018 года. |
| 8 | ИС «Кинотеатр» | 1. Выбрать все фильмы, которые показывают в 3D-залах. 2. Выбрать все фильмы с названием соответствующего зала. 3. Выбрать все залы, в которых не показывают ни одного фильма. 4. Добавить новый фильм (Id зала выбрать подзапросом по полям *Тип зала* и *Количество мест*). 5. Изменить дату окончания показа = 1 января 2020 года для всех фильмов, которые показывают в 2D залах с количество мест больше 60. |
| 9 | ИС «Автосалон» | 1. Выбрать все машины, покупатели которых родились после 1 января 1990 года. 2. Выбрать все машины с ФИО соответствующего покупателя. 3. Выбрать все машины, которые никто не купил. 4. Добавить новую машину (Id покупателя выбрать подзапросом по полю *ФИО*). 5. Изменить дату выпуска машин = 20 июля 2010 года для всех покупателей, которые родились после 1 января 1995 года. |
| 10 | ИС «Цветочный магазин» | 1. Выбрать все цветы, поставщики которых не из России. 2. Выбрать все цветы с названием соответствующего поставщика. 3. Выбрать все цветы, у которых нет поставщика. 4. Добавить новый цветок (Id поставщика выбрать подзапросом по полю *Название поставщика*). 5. Изменить страну поставщика на «Китай», цветы которых стоят дороже 2000 руб. |
| 11 | ИС «ЗАГС» | 1. Выбрать все заявления, в которых хотя бы один из заявителей родился позже 1 января 1990 года. 2. Выбрать все заявления с ФИО соответствующих супруги и супруга. 3. Выбрать все заявления, супруги которых из Санкт-Петербурга. 4. Добавить новое заявление (Id супруги и супруга выбрать подзапросом по полю *ФИО*). 5. Изменить дату подачи заявления на 1 февраля 2005 года, супруг или супруга которого родились раньше 1 мая 1990 года в городе Волгоград. |
| 12 | ИС «Отдел кадров предприятия» | 1. Выбрать всех сотрудников из отдела в городе Москва. 2. Выбрать всех сотрудников с названием соответствующих отделов. 3. Выбрать все отделы, в которых не работает ни один сотрудник. 4. Добавить нового сотрудника (Id отдела выбрать подзапросом по полю *Название, Страна и Город*). 5. Изменить город для всех отделов на «Ставрополь» для всех сотрудников, у которых поле *Паспортные данные* содержит «Ставрополь». |
| 13 | ИС «Паспортный стол» | 1. Выбрать все просроченные паспорта. 2. Выбрать все паспорта с названием соответствующего статуса. 3. Выбрать все паспорта без статуса. 4. Добавить новый паспорт (Id статуса выбрать подзапросом по полю *Название*). 5. Изменить место выдачи = «Москва» для всех паспортов, у которых поле *Статус* равен «Готов к выдаче». |
| 14 | ИС «Складской учет на предприятии» | 1. Выбрать все товары, накладные которых составлены на сумму больше 5000. 2. Выбрать все товары с датой поставки и отправителем соответствующих накладных. 3. Выбрать все товары, накладные которых составлены на сумму больше средней суммы всех накладных. 4. Добавить новый товар (Id накладной выбрать подзапросом по полям *Дата поставки* и *Отправитель*). 5. Увеличить в два раза цену товаров, которые были поставлены после 1 июня 2016 года. |
| 15 | ИС «Туристическое агентство» | 1. Выбрать все путевки, туристы которых родились позже 1 января 1993 года. 2. Выбрать все путевки с ФИО соответствующих туристов. 3. Выбрать всех туристов, которые купили путевки на сумму более 50000 руб. 4. Добавить новую путевку (Id туриста выбрать подзапросом по полю *ФИО*). 5. Уменьшить в два раза цену путевок, для всех туристов, которые родились раньше 1 января 1950 года. |
| 16 | ИС «Автосервис» | 1. Выбрать все транспортные средства, заказчики которых имеют телефонные номера, начинающиеся на «8927» 2. Выбрать все транспортные средства с ФИО соответствующих заказчиков. 3. Выбрать всех заказчиков, которые заплатили за услугу более 3000 руб. 4. Добавить новое транспортное средство (Id заказчика выбрать подзапросом по полю *ФИО*). 5. Уменьшить в 2 раза цену за услугу на транспортные средства для всех заказчиков, которых зовут «Иван». |
| 17 | ИС «Кондитерское предприятие» | 1. Выбрать все кондитерские изделия, которые находятся в витринах с описанием «2 этаж». 2. Выбрать все кондитерские изделия с описанием соответствующих витрин. 3. Выбрать все витрины, которые содержат кондитерские изделия на сумму более 5000руб. 4. Добавить новое кондитерское изделие (Id витрины выбрать подзапросом по полю *Описание*). 5. Уменьшить в 2 раза цену на кондитерские изделия, которые находятся в витринах, содержащие в описании «со скидкой». |
| 18 | ИС «Ателье мод» | 1. Выбрать все изделия, заказчики которых имеют номера, начинающиеся на «8917». 2. Выбрать все изделия с ФИО соответствующих заказчиков. 3. Выбрать всех заказчиков, которые заказали изделия на сумму более 5000 руб. 4. Добавить новое изделие (Id заказчика выбрать подзапросом по полю *ФИО*). 5. Увеличить в 2 раза цены на изделия, которые стоят меньше средней цены всех изделий. |
| 19 | ИС «Электронный школьный дневник» | 1. Выбрать всех школьников, которые учатся на 3 этаже. 2. Выбрать всех школьников с ФИО соответствующих классных руководителей. 3. Выбрать все классы, которые не закреплены ни за одним школьником. 4. Добавить нового школьника (Id класса выбрать подзапросом по полю *Классный руководитель*). 5. Изменить дату поступления на 1 сентября 2019 года для всех школьников, которые учатся в классах на 1 этаже. |
| 20 | ИС «Служба доставки» | 1. Выбрать все заказы, заказчики которых имеют телефоны, начинающиеся на «8961». 2. Выбрать все заказы с именем соответствующих заказчиков. 3. Выбрать всех заказчиков, которые сделали заказы на сумму более 2500руб. 4. Добавить новый заказ (Id заказчика выбрать подзапросом по полю *Имя*). 5. Изменить промокод на «NOVEMBER30» для всех заказов, сумма которых равна максимальной сумме среди всех заказов. |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» | 1. Выбрать все заказы, выполненные для вида техники «Принтер». 2. Выбрать все заказы с наименованием соответствующих видов техники. 3. Выбрать все заказы, для которых не установлено ни одного вида техники. 4. Добавить новый заказ (Id вида техники выбрать подзапросом по полю *Наименование*). 5. Увеличить сумму заказа в два раза для всех заказов, сумма которых меньше средней суммы среди всех заказов. |
| 22 | ИС «Салон красоты» | 1. Выбрать все услуги, мастера которых работают раньше 1 марта 2015 года. 2. Выбрать все услуги с ФИО соответствующих мастеров. 3. Выбрать все услуги, которые не оказывает ни один мастер. 4. Добавить новую услугу (Id мастера выбрать подзапросом по полю *ФИО*). 5. Увеличить цену на услугу в 2 раза, если среднее время выполнения равно максимальному среднему времени выполнения среди всех услуг. |
| 23 | ИС «Санитарно-курортный комплекс» | 1. Выбрать все процедуры, которые проводятся в кабинетах с ответственным «Роман». 2. Выбрать все процедуры с именами ответственных соответствующих процедурам. 3. Выбрать все процедуры, которым не назначен ни один из кабинетов. 4. Добавить новую процедуру (Id кабинета выбрать подзапросом по полю *Ответственный*). 5. Увеличить количество мест на 2, если среднее время выполнения равно максимальному среднему времени выполнения среди всех процедур. |

# **Лабораторная работа №8 Выборки из нескольких таблиц. Соединение таблиц.**

**Неявное соединение таблиц**

Для сведения данных из разных таблиц мы можем использовать стандартную команду SELECT.

Допустим, у нас есть база данных с таблицами для смартфонов, покупателей и заказов:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

CREATE TABLE Orders

(Id INT PRIMARY KEY IDENTITY,

[SmartphoneId] INT NOT NULL REFERENCES Smartphones(Id) ON DELETE CASCADE,

[CustomerId] INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,

Count INT,

Date DATETIME)

Таблицы *Smartphones* и *Customers* связаны с таблицей *Orders* связью один ко многим.

Добавим в таблицы некоторые данные:

INSERT INTO Smartphones VALUES('Samsung Galaxy S8', 'Samsung', 35000, 3)

INSERT INTO Smartphones VALUES('Samsung Galaxy S8 plus', 'Samsung', 35000, 4)

INSERT INTO Smartphones VALUES('Samsung Galaxy S9', 'Samsung', 50000, 6)

INSERT INTO Smartphones VALUES('iPhone X', 'Apple', 40000, 5)

INSERT INTO Smartphones VALUES('iPhone XR', ' Apple', 50000, 7)

INSERT INTO Smartphones VALUES('iPhone XS', ' Apple', 56000, 5)

INSERT INTO Smartphones VALUES('iPhone 11', ' Apple', 65000, 10)

INSERT INTO Customers VALUES(N'Королев Ренат Александрович', '125412-565214', '89614528745', 100)

INSERT INTO Customers VALUES(N'Орлова Мария Викторовна', '418759-325698', '89174852503', 584)

INSERT INTO Customers VALUES(N'Дмитриев Николай Николаевич', '512586-023652', '89294158754', 397)

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone X'),

(SELECT Id FROM Customers WHERE CardNumber='125412-565214'),

2, '2019-04-01')

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone XR'),

(SELECT Id FROM Customers WHERE CardNumber='418759-325698'),

3, '2019-05-25')

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='Samsung Galaxy S9'),

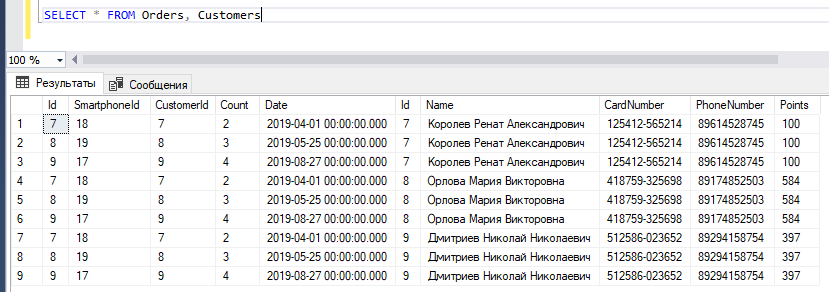
(SELECT Id FROM Customers WHERE CardNumber='512586-023652'),

4, '2019-08-27')

Теперь соединим две таблицы Orders и Customers:

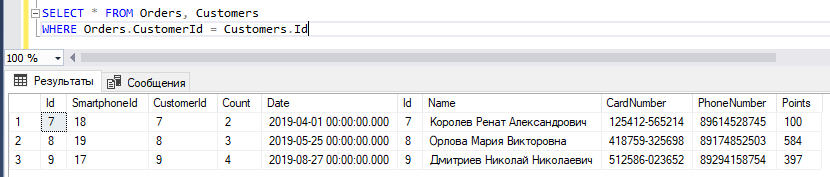
SELECT \* FROM Orders, Customers

При такой выборке каждая строка из таблицы *Orders* будет совмещаться с каждой строкой из таблицы *Customers*. То есть, получится перекрестное соединение. Например, в таблице *Orders* три строки, в таблице *Customers* тоже три строки, значит мы получим 3 \* 3 = 9 строк:

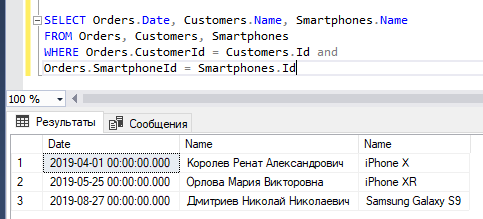


То есть в данном случае мы получаем прямое (декартово) произведение двух групп. Но этот результат едва может быть полезным. Потому что каждый заказ из таблицы *Orders* связан с конкретным покупателем из таблицы *Customers*, а не со всеми возможными покупателями.

Чтобы решить задачу, необходимо использовать выражение WHERE и фильтровать строки при условии, что поле *CustomerId* из таблицы *Orders* соответствует полю *Id* из таблицы *Customers*:



Также можно соединить три таблицы. Ключевой таблицей останется таблица *Orders,* из которой извлекаются все заказы, а затем к ней присоединяются данные по клиенту по условию *Orders.CustomerId = Customers.Id* и данные по смартфонам *Orders.SmartphoneId = Smartphones.Id.*



**Явное соединение таблиц (JOIN)**

Для явного соединения таблиц используется оператор JOIN. Соединение бывает внутреннее (INNER) и внешнее (OUTER).

**INNER JOIN**

В общем виде синтаксис выглядит следующим образом:

SELECT столбцы

FROM таблица1

    [INNER] JOIN таблица2

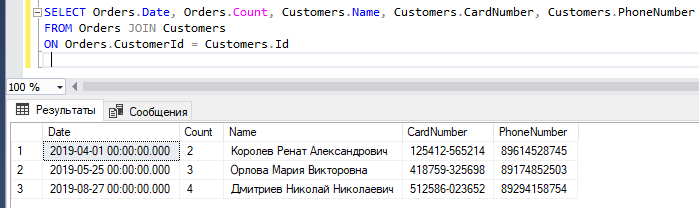
    ON условие1

    [[INNER] JOIN таблица3

    ON условие2]

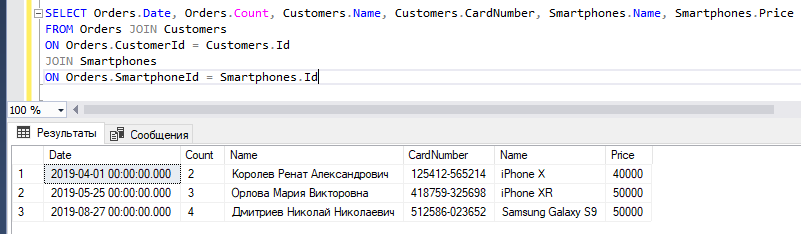
После оператора JOIN идет название второй таблицы, из которой надо добавить данные в выборку. Перед JOIN может использоваться необязательное ключевое слово INNER. Его наличие или отсутствие ни на что не влияет, т.к. по умолчанию выполняется INNER JOIN. Затем после ключевого слова ON указывается условие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

Используя JOIN, выберем все заказы и добавим к ним информацию о товарах:

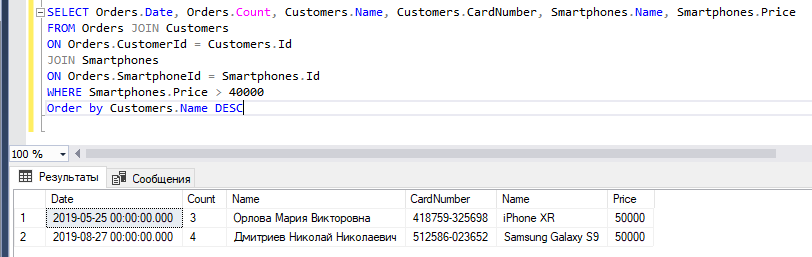


Поскольку таблицы могут содержать столбцы с одинаковыми названиями, то при указании столбцов для выборки указывается их полное имя вместе с именем таблицы, например, "Orders.Count" или "Customers.CardNumber".

Аналогично можно присоединять и остальные таблицы. Например, добавим информацию о смартфонах.



Благодаря соединению таблиц мы можем использовать их столбцы для фильтрации выборки или ее сортировки:



При использовании оператора JOIN следует учитывать, что процесс соединения таблиц может быть ресурсоемким, поэтому следует соединять только те таблицы, данные из которых действительно необходимы. Чем больше таблиц соединяется, тем больше снижается производительность.

**Внешнее соединение (OUTER)**

В отличие от внутреннего соединение внешнее соединение возвращает все строки одной или двух таблиц, которые участвуют в соединении.

OUTER JOIN имеет следующий формальный синтаксис:

SELECT столбцы

FROM таблица1

    {LEFT|RIGHT|FULL} [OUTER] JOIN таблица2 ON условие1

    [{LEFT|RIGHT|FULL} [OUTER] JOIN таблица3 ON условие2]...

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
* **RIGHT**: выборка будет содержать все строки из второй или правой таблицы
* **FULL**: выборка будет содержать все строки из обеих таблиц

Также перед оператором **JOIN** может указываться ключевое слово **OUTER**, но его применение необязательно. Далее после **JOIN** указывается присоединяемая таблица, а затем идет условие соединения.

Например, соединим таблицы *Orders* и *Customers.*

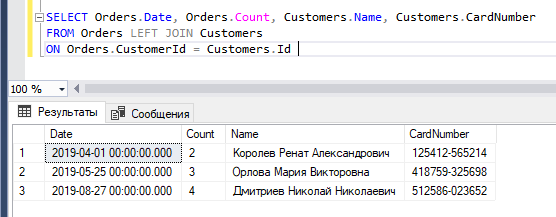
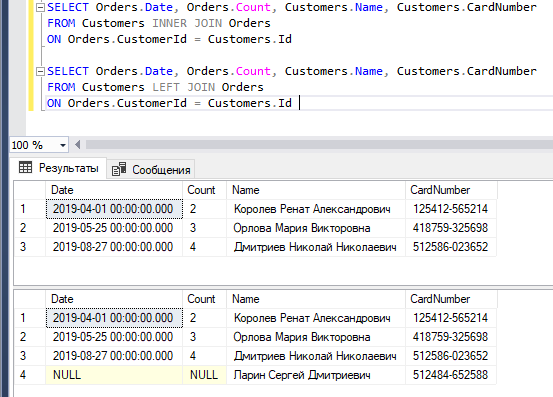
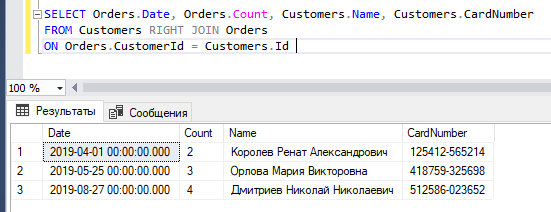


Таблица *Orders* является первой или левой таблицей, а таблица *Customers* - правой таблицей. Поэтому, так как здесь используется выборка по левой таблице, то вначале будут выбираться все строки из таблицы *Orders*, а затем к ним по условию Orders.CustomerId = Customers.Id будут добавляться связанные строки из *Customers*.

По вышеприведенному результату может показаться, что левостороннее соединение аналогично INNER JOIN, но это не так. INNER JOIN объединяет строки из двух таблиц при соответствии условию. Если одна из таблиц содержит строки, которые не соответствуют этому условию, то данные строки не включаются в выходную выборку. LEFT JOIN выбирает все строки первой таблицы и затем присоединяет к ним строки правой таблицы. К примеру, возьмем таблицу *Customers* и добавим к покупателям информацию об их заказах:

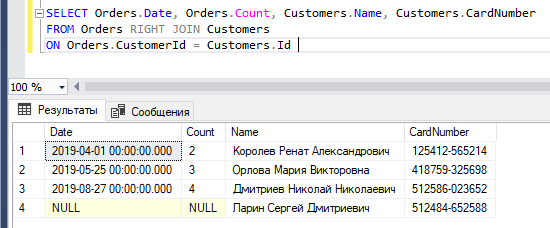


Изменим запрос, используя RIGHT JOIN.



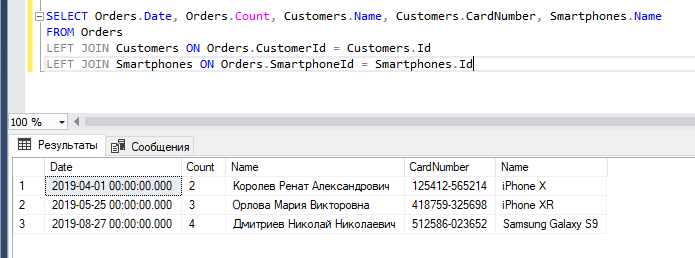
В этом случае берутся все данные из таблицы *Orders* и им присоединяются данные из таблицы *Customers.*

Поменяв местами *Customers* и *Orders,* получим следующий результат:

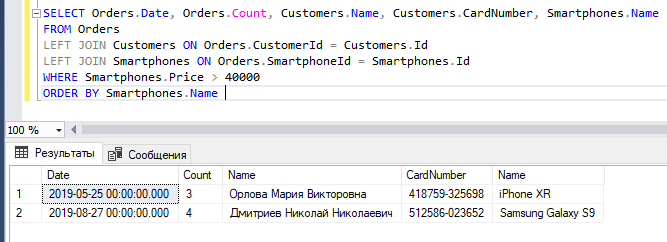


Теперь берутся все данные из таблицы *Customers* и им присоединяются данные из таблицы *Orders*. Поскольку один из покупателей из таблицы *Customers* не имеет связанных заказов из таблицы *Orders*, то соответствующие столбцы, которые берутся из *Orders*, будут иметь значения NULL.

Также можно соединять более двух таблиц:

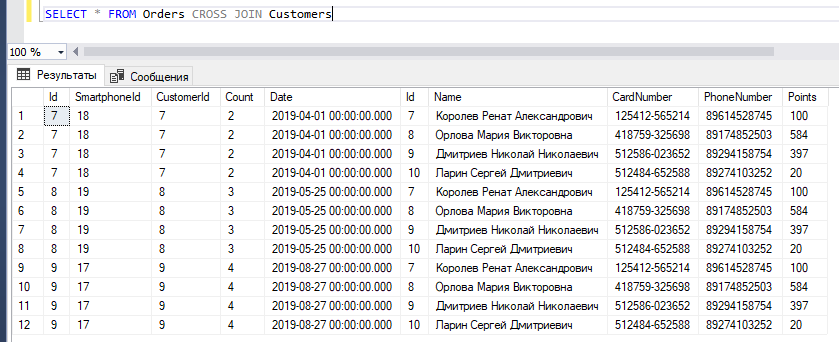


В запросе с соединениями можно также использовать фильтрацию и сортировку.



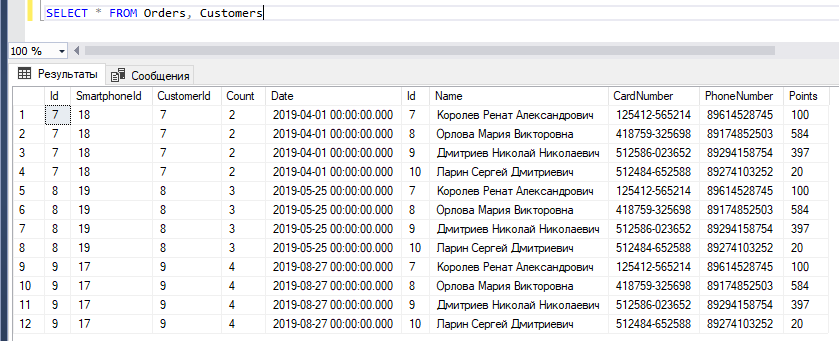
**CROSS JOIN**

Это перекрестное соединение создает набор строк, где каждая строка из одной таблицы соединяется с каждой строкой из другой таблицы. Например, соединим таблицы *Orders* и *Customers.*

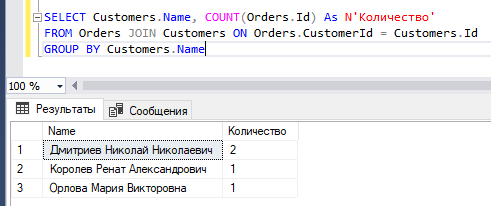


Если в таблице *Orders* 3 строки, а в таблице *Customers* четыре строки, то в результате перекрестного соединения создается 3 \* 4 = 12 строк вне зависимости, связаны ли данные строки или нет.

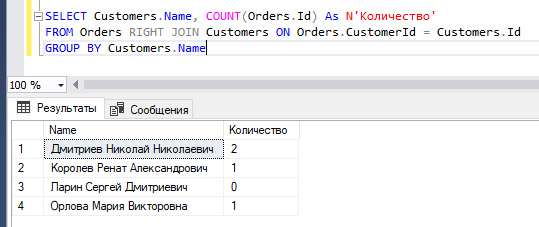
Это аналогично неявному соединению.



В выражениях INNER/OUTER JOIN также можно использовать группировку. Например, выведем для каждого пользователя количество заказов, которые он сделал:



Если необходимо вывести даже тех покупателей, у которых нет заказов, то применяется OUTER JOIN:



**Оператор UNION**

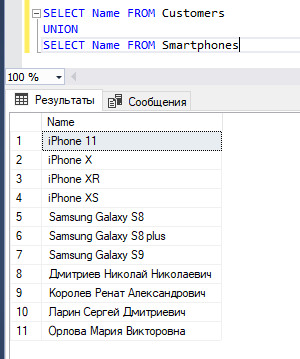
Оператор UNION подобно INNER JOIN или OUTER JOIN позволяет соединить две таблицы. Но в отличие от INNER/OUTER JOIN объединения соединяют не столбцы разных таблиц, а два однотипных набора в один. Формальный синтаксис объединения:

SELECT\_выражение1

UNION [ALL] SELECT\_выражение2

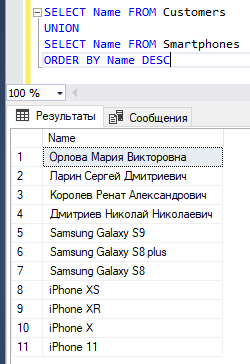
[UNION [ALL] SELECT\_выражениеN]

Выберем данные из таблиц *Smartphones* и *Customers* по общему полю *Name.*

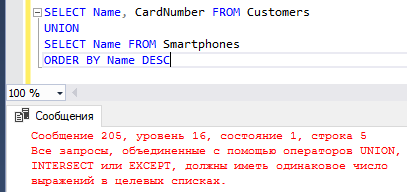


В данном случае из первой таблицы *Customers* выбирается одно значение – название клиента. Из второй таблицы *Smartphones* также выбирается одно значение – название смартфона. То есть при объединении количество выбираемых столбцов и их тип совпадают для обеих выборок.

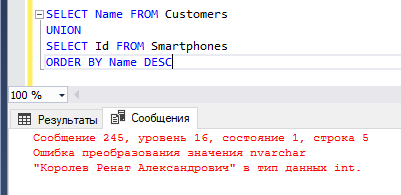
Если мы захотим при этом еще произвести сортировку, то в выражениях ORDER BY необходимо ориентироваться именно на названия столбцов первой выборки:



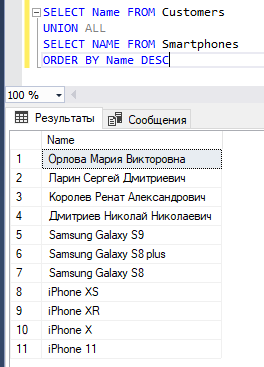
Если же в одной выборке больше столбцов, чем в другой, то они не смогут быть объединены. Например, в следующем случае объединение завершится с ошибкой:



Если типы столбцов не совпадают, то запрос также завершится ошибкой.



Если оба объединяемых набора содержат в строках идентичные значения, то при объединении повторяющиеся строки удаляются. Если же необходимо при объединении сохранить все, в том числе повторяющиеся строки, то для этого необходимо использовать оператор **ALL**:



**Оператор EXCEPT**

Оператор EXCEPT позволяет найти разность двух выборок, то есть те строки которые есть в первой выборке, но которых нет во второй. Для его использования применяется следующий формальный синтаксис:

SELECT\_выражение1

EXCEPT SELECT\_выражение2

Допустим, у нас есть две таблицы *Customers* и *Suppliers.*

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

CREATE TABLE Suppliers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

INSERT INTO Customers VALUES(N'Королев Ренат Александрович', '125412-565214', '89614528745', 100)

INSERT INTO Customers VALUES(N'Орлова Мария Викторовна', '418759-325698', '89174852503', 584)

INSERT INTO Customers VALUES(N'Дмитриев Николай Николаевич', '512586-023652', '89294158754', 397)

INSERT INTO Customers VALUES(N'Ларин Сергей Дмитриевич', '512484-652588', '89274103252', 20)

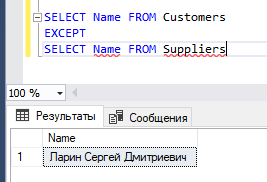
INSERT INTO Suppliers VALUES(N'Королев Ренат Александрович', '125412-565214', '89614528745', 100)

INSERT INTO Suppliers VALUES(N'Орлова Мария Викторовна', '418759-325698', '89174852503', 584)

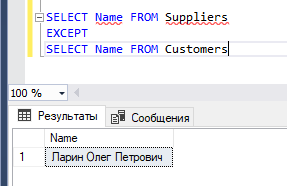
INSERT INTO Suppliers VALUES(N'Дмитриев Николай Николаевич', '512586-023652', '89294158754', 397)

INSERT INTO Suppliers VALUES(N'Ларин Олег Петрович', '741520-336520', '89175420236', 205)

Допустим, нам нужны все клиенты за исключением поставщиков.



Подобным образом можно получить всех поставщиков, которые не являются клиентами.

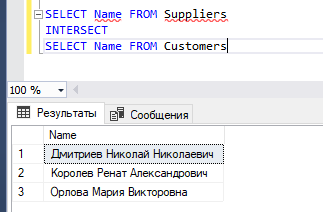


**Оператор INTERSECT**

Оператор INTERSECT позволяет найти общие строки для двух выборок, то есть данный оператор выполняет операцию пересечения множеств. Для его использования применяется следующий формальный синтаксис:

SELECT\_выражение1

INTERSECT SELECT\_выражение2



**Задание к лабораторной работе**

**Вариант 1 - ИС «Электронная ведомость ВУЗа»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Дисциплины* и соответствующие им значения из таблицы *Экзаменаторы.* Вывести поля: Название дисциплины, Описание, ФИО, ученая степень и ученое звание экзаменатора.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все дисциплины и соответствующих им экзаменаторов.
4. Добавьте новую таблицу *Студенты* с полями Идентификатор, ФИО nvarchar(300), Группа, Дата зачисления. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините ФИО из таблицы *Экзаменаторы* и ФИО из таблицы *Студенты.*
6. Выведите на экран имена всех студентов, которые не являются преподавателями.
7. Выведите на экран имена всех студентов, которые одновременно являются преподавателями (или имеют одинаковые имена).

**Вариант 2 - ИС «Пассажирское автопредприятие»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Транспортные средства* и соответствующие им значения из таблицы *Модели.* Вывести поля: Дата начала эксплуатации, Имя водителя, Марка, Модель, Пробег.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все транспортные средства и соответствующих им модели.
4. Добавьте новую таблицу *Клиенты* с полями Идентификатор, ФИО nvarchar(300), Телефон, Email. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имя водителя из таблицы *Транспортные средства* и ФИО из таблицы *Клиенты.*
6. Выведите на экран имена всех клиентов, которые не являются водителями.
7. Выведите на экран имена всех водителей, которые одновременно являются клиентами (или имеют одинаковые имена).

**Вариант 3 - ИС «Таксопарк»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Транспортные средства* и соответствующие им значения из таблицы *Классы.* Вывести поля: Дата выпуска, Дата начала эксплуатации, Марка, Модель, Название класса и Описание класса.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все транспортные средства и соответствующих им классы.
4. Добавьте новую таблицу *Клиенты* с полями Идентификатор, ФИО nvarchar(300), Телефон, Email. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имя водителя из таблицы *Транспортные средства* и ФИО из таблицы *Клиенты.*
6. Выведите на экран имена всех клиентов, которые не являются водителями.
7. Выведите на экран имена всех водителей, которые одновременно являются клиентами (или имеют одинаковые имена).

**Вариант 4 - ИС «Электронная регистратура поликлиники»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Сотрудники* и соответствующие им значения из таблицы *Должности.* Вывести поля: ФИО, Специализация, Дата начала работы, Название должности.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите всех сотрудников и соответствующие им должности.
4. Добавьте новую таблицу *Пациенты* с полями Идентификатор, ФИО nvarchar(300), Телефон, Email, Номер страхового полиса, Резидент (булево). Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имя сотрудника из таблицы *Сотрудники* и имя пациента из таблицы *Пациенты.*
6. Выведите на экран имена всех пациентов, которые не являются сотрудниками поликлиники.
7. Выведите на экран имена всех сотрудников, которые одновременно являются пациентами (или имеют одинаковые имена).

**Вариант 5 - ИС «Библиотека»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Книги* и соответствующие им значения из таблицы *Авторы.* Вывести поля: Название книги, Дата издания, Жанр, ФИО Автора, Дата рождения автора.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все книги и соответствующих им авторов.
4. Добавьте новую таблицу *Читатели* с полями Идентификатор, ФИО nvarchar(300), Телефон, Email. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имя автора из таблицы *Авторы* и имя читателя из таблицы *Читатели.*
6. Выведите на экран имена всех читателей, которые не являются авторами книг.
7. Выведите на экран имена всех читателей, которые одновременно являются авторами книг (или имеют одинаковые имена).

**Вариант 6 - ИС «Гостиница»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Номера* и соответствующие им значения из таблицы *Горничные.* Вывести поля: Количество комнат, Количество мест, ФИО горничной, Дата начала работы горничной.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все номера и соответствующих им горничных.
4. Добавьте новую таблицу *Гости (Постояльцы)* с полями Идентификатор, ФИО nvarchar(300), Телефон, Email, Паспортные данные. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имя горничной из таблицы *Горничные* и имя гостя из таблицы *Гости.*
6. Выведите на экран имена всех горничных, которые не являются гостями гостиницы.
7. Выведите на экран имена всех горничных, которые одновременно являются гостями гостиницы (или имеют одинаковые имена).

**Вариант 7 - ИС «Ресторан»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Столики* и соответствующие им значения из таблицы *Официанты.* Вывести поля: Номер столика, Количество мест, Номер зала, ФИО официанта, Дата начала работы.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все столики и соответствующих им официантов.
4. Добавьте новую таблицу *Постоянные клиенты* с полями Идентификатор, ФИО nvarchar(300), Телефон, Email, Номер скидочной карты. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имя официанта из таблицы *Официанты* и имя постоянного клиента из таблицы *Постоянные клиенты.*
6. Выведите на экран имена всех официантов, которые не являются постоянными клиентами ресторана.
7. Выведите на экран имена всех официантов, которые одновременно являются постоянными клиентами (или имеют одинаковые имена).

**Вариант 8 - ИС «Кинотеатр»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Фильмы* и соответствующие им значения из таблицы *Залы.* Вывести поля: Название фильма, Дата начала показа, Номер зала, Тип зала.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все фильмы и соответствующие им залы.
4. Добавьте новую таблицу *Архив* с полями Идентификатор, Название фильма, Дата начала проката, Дата окончания проката. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините названия фильмов из таблицы *Фильмы* и названия фильмов из таблицы *Архив.*
6. Выведите на экран названия всех фильмов, которые еще не попали в архив.
7. Выведите на экран названия всех архивных фильмов, которые сейчас показывают (находятся в таблице *Фильмы*) (или имеют одинаковые имена).

**Вариант 9 - ИС «Автосалон»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Машины* и соответствующие им значения из таблицы *Покупатели.* Вывести поля: Марка, Модель, Дата выпуска, Цена, ФИО покупателя, Дата рождения.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все машины и соответствующих им покупателей.
4. Добавьте новую таблицу *Сотрудники салона* с полями Идентификатор, ФИО nvarchar(300), Дата рождения, Паспортные данные, Должность. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена из таблицы *Покупатели* и имена из таблицы *Сотрудники.*
6. Выведите на экран имена всех покупателей, которые не являются сотрудниками автосалона.
7. Выведите на экран имена всех сотрудников, которые что-либо покупали в автосалоне (находятся в таблице *Покупатели*) (или имеют одинаковые имена).

**Вариант 10 - ИС «Цветочный магазин»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Цветы* и соответствующие им значения из таблицы *Поставщики.* Вывести поля: Название, Цена поставщика, Цена продажи, Название поставщика, Страна, Город.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все цветы и соответствующих им поставщиков.
4. Добавьте новую таблицу *Постоянные покупатели* с полями Идентификатор, ФИО nvarchar(300), Дата рождения, Телефон, Номер скидочной карты. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена из таблицы *Постоянные покупатели* и названия из таблицы *Поставщики.*
6. Выведите на экран имена всех постоянных покупателей, которые не являются поставщиками цветов.
7. Выведите на экран имена всех поставщиков, которые являются постоянными покупателями магазина (находятся в таблице *Постоянные покупатели*) (или имеют одинаковые имена).

**Вариант 11 - ИС «ЗАГС»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Заявления* и соответствующие им значения из таблицы *Заявители.* Вывести поля: Идентификатор заявления, Дата подачи заявления, ФИО супруга, Дата рождения супруга, ФИО супруги, Дата рождения супруги.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все заявления и соответствующих им заявителей.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Дата рождения, Телефон, Паспортные данные. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена из таблицы *Заявители* и имена из таблицы *Сотрудники.*
6. Выведите на экран имена всех сотрудников, которые никогда не выступали в качестве заявителей (по имени).
7. Выведите на экран имена всех сотрудников, которые хотя бы раз подавали заявления (находятся в таблице *Заявители*) (или имеют одинаковые имена).

**Вариант 12 - ИС «Отдел кадров предприятия»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Сотрудники* и соответствующие им значения из таблицы *Отделы.* Вывести поля: ФИО сотрудника, Дата рождения, Название отдела, Страна и Город отдела.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите всех сотрудников и соответствующие им отделы.
4. Добавьте новую таблицу *Филиалы* с полями Идентификатор, Название nvarchar(300), Страна, Город. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините названия из таблицы *Отделы* и названия из таблицы *Филиалы.*
6. Выведите на экран названия всех отделов, которые не совпадают по названию с филиалами.
7. Выведите на экран названия всех отделов, которые совпадают по названию с филиалами.

**Вариант 13 - ИС «Паспортный стол»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Паспорта* и соответствующие им значения из таблицы *Статусы.* Вывести поля: Серия, Номер, Дата подачи заявления, Название статуса.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все паспорта и соответствующие им статусы.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Страна, Город. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена из таблицы *Паспорта* и имена из таблицы *Сотрудники.*
6. Выведите на экран имена всех сотрудников, о которых нет информации о получении паспорта (по полю ФИО).
7. Выведите на экран имена всех сотрудников, которые хоть раз получали паспорт (или имеют одинаковые имена).

**Вариант 14 - ИС «Складской учет на предприятии»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Товары* и соответствующие им значения из таблицы *Накладные.* Вывести поля: Название, Цена, Количество, Идентификатор накладной, Дата поставки.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все товары и соответствующие им накладные.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Страна, Город. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена отправителей из таблицы *Накладные* и имена из таблицы *Сотрудники.*
6. Выведите на экран имена всех сотрудников, которые ни разу не отправляли товар через предприятие (по имени).
7. Выведите на экран имена всех сотрудников, которые хоть раз отправляли товар (или имеют одинаковые имена).

**Вариант 15 - ИС «Туристическое агентство»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Путевки* и соответствующие им значения из таблицы *Туристы.* Вывести поля: Страна, Город, Цена, Дата поездки, ФИО туриста, Дата рождения.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все путевки и соответствующих им туристов.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Страна, Город. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена туристов из таблицы *Туристы* и имена из таблицы *Сотрудники.*
6. Выведите на экран имена всех сотрудников, которые ни разу не покупали путевку (по имени).
7. Выведите на экран имена всех сотрудников, которые хоть раз покупали путевку через агентство (или имеют одинаковые имена).

**Вариант 16 - ИС «Автосервис»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Транспортные средства* и соответствующие им значения из таблицы *Заказчики.* Вывести поля: Марка, Модель, Причина обращения, Цена за услугу, ФИО заказчика, Телефон.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все транспортные средства и соответствующих им заказчиков.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Страна, Город. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена заказчиков из таблицы *Заказчики* и имена из таблицы *Сотрудники.*
6. Выведите на экран имена всех сотрудников, которые ни разу не пользовались услугами автосервиса (по имени).
7. Выведите на экран имена всех сотрудников, которые хоть раз пользовались услугами автосервиса (или имеют одинаковые имена).

**Вариант 17 - ИС «Кондитерское предприятие»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Кондитерские изделия* и соответствующие им значения из таблицы *Витрины.* Вывести поля: Название, Цена, Количество в наличии и Описание витрины.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все кондитерские изделия и соответствующие им витрины.
4. Добавьте новую таблицу *Меню на заказ* с полями Идентификатор, Название изделия nvarchar(300), Цена за штуку. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините название кондитерских изделий из таблицы *Кондитерские изделия* и названия из таблицы *Меню на заказ.*
6. Выведите на экран названия всех изделий, которые не доступны к заказу (по названию).
7. Выведите на экран названия всех изделий, которые доступны как на заказ, так и имеются на витрине (или имеют одинаковые названия).

**Вариант 18 - ИС «Ателье мод»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Изделия* и соответствующие им значения из таблицы *Заказчики.* Вывести поля: Название, Цена, Готовность, ФИО заказчика, Телефон.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все изделия и соответствующих им заказчиков.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Должность. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена заказчиков из таблицы *Заказчики* и имена сотрудников из таблицы *Сотрудники.*
6. Выведите на экран имена всех сотрудников, которые ни разу не делали заказ (по имени).
7. Выведите на экран имена сотрудников, которые хоть раз делали заказ (или имеют одинаковые имена).

**Вариант 19 - ИС «Электронный школьный дневник»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Школьники* и соответствующие им значения из таблицы *Классы.* Вывести поля: ФИО, Дата поступления, Классный руководитель, Кабинет, Этаж.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите всех школьников и соответствующие им классы.
4. Добавьте новую таблицу *Преподаватели* с полями Идентификатор, ФИО nvarchar(300), Телефон, Паспортные данные. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена классных руководителей из таблицы *Классы* и имена преподавателей из таблицы *Преподаватели.*
6. Выведите на экран имена всех преподавателей, которые не являются классными руководителями (по имени).
7. Выведите на экран имена всех преподавателей, которые являются классными руководителями (или имеют одинаковые имена).

**Вариант 20 - ИС «Служба доставки»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Заказы* и соответствующие им значения из таблицы *Заказчики.* Вывести поля: Идентификатор заказа, Сумма, Дата, Имя заказчика, Телефон.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все заказы и соответствующих им заказчиков.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Должность, Паспортные данные. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена сотрудников из таблицы *Сотрудники* и имена заказчиков из таблицы *Заказчики.*
6. Выведите на экран имена всех сотрудников, которые не являются заказчиками (по имени).
7. Выведите на экран имена всех сотрудников, которые хоть раз делали заказ (или имеют одинаковые имена).

**Вариант 21 - ИС «Сервисный центр по ремонту компьютерной техники»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Заказы* и соответствующие им значения из таблицы *Виды техники.* Вывести поля: Идентификатор заказа, Сумма, Дата, Наименование вида техники.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все заказы и соответствующие им виды техники.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Должность, Паспортные данные. А в таблицу *Заказы* добавьте поле ФИО заказчика. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена сотрудников из таблицы *Сотрудники* и имена заказчиков из таблицы *Заказы.*
6. Выведите на экран имена всех сотрудников, которые не являются заказчиками (по имени).
7. Выведите на экран имена всех сотрудников, которые хоть раз делали заказ (или имеют одинаковые имена).

**Вариант 22 - ИС «Салон красоты»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Услуги* и соответствующие им значения из таблицы *Мастера.* Вывести поля: Наименование, Цена, Среднее время выполнения, ФИО мастера, Дата начала работы.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все услуги и соответствующих им мастеров.
4. Добавьте новую таблицу *Клиенты* с полями Идентификатор, ФИО nvarchar(300), Телефон. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена мастеров из таблицы *Мастера* и имена клиентов из таблицы *Клиенты.*
6. Выведите на экран имена всех мастеров, которые не являются клиентами (по имени).
7. Выведите на экран имена всех мастеров, которые хоть раз пользовались услугами салона (или имеют одинаковые имена).

**Вариант 23 - ИС «Санитарно-курортный комплекс»**

1. С помощью неявного соединения таблиц выберите данные из таблицы *Процедуры* и соответствующие им значения из таблицы *Кабинеты.* Вывести поля: Наименование, Среднее время выполнения, Количество мест, Идентификатор кабинета, Имя ответственного.
2. Перепишите предыдущий запрос, используя внутреннее соединение.
3. Используя внешнее соединение (сначала LEFT, затем RIGHT), выберите все процедуры и соответствующие им кабинеты.
4. Добавьте новую таблицу *Сотрудники* с полями Идентификатор, ФИО nvarchar(300), Телефон, Должность. Заполните ее валидными тестовыми данными (не менее 5 значений).
5. Используя оператор UNION, объедините имена сотрудников из таблицы *Сотрудники* и имена ответственных из таблицы *Кабинеты.*
6. Выведите на экран имена всех сотрудников, которые не являются ответственными ни за один кабинет (по имени).
7. Выведите на экран имена всех сотрудников, которые являются ответственными хотя бы за один кабинет (или имеют одинаковые имена).

# **Лабораторная работа №9 Встроенные функции для работы с данными.**

**Функции для работы со строками**

Для работы со строками в T-SQL можно применять следующие функции:

**LEN**: возвращает количество символов в строке. В качестве параметра в функцию передается строка, для которой надо найти длину:

SELECT LEN(Name) AS N 'Длина имени' FROM Customers

SELECT LEN('Samsung') --7 символов

**LTRIM**: удаляет начальные пробелы из строки. В качестве параметра принимает строку:

SELECT LTRIM(' Samsung')

**RTRIM**: удаляет конечные пробелы из строки. В качестве параметра принимает строку:

SELECT RTRIM('Samsung ')

**CHARINDEX**: возвращает индекс, по которому находится первое вхождение подстроки в строке. В качестве первого параметра передается подстрока, а в качестве второго - строка, в которой надо вести поиск:

SELECT CHARINDEX('ms', 'Samsung') – 3

**PATINDEX**: возвращает индекс, по которому находится первое вхождение определенного шаблона в строке:

SELECT PATINDEX('%s\_n%', 'Samsung') --4

**LEFT**: вырезает с начала строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

SELECT LEFT('Samsung', 4) --Sams

**RIGHT**: вырезает с конца строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

SELECT RIGHT('Samsung', 3) --ung

**SUBSTRING**: вырезает из строки подстроку определенной длины, начиная с определенного индекса. Первый параметр функции - строка, второй - начальный индекс для вырезки, и третий параметр - количество вырезаемых символов:

SELECT SUBSTRING('Samsung', 2, 3) -- ams

**REPLACE**: заменяет одну подстроку другой в рамках строки. Первый параметр функции - строка, второй - подстрока, которую надо заменить, а третий - подстрока, на которую надо заменить:

SELECT REPLACE('Samsung Galaxy S8 Plus', 'S8 Plus', 'Note 10') --Samsung Galaxy Note 10

**REVERSE**: переворачивает строку наоборот:

SELECT REVERSE('123456789') – 987654321

**CONCAT**: объединяет две строки в одну. В качестве параметра принимает от 2-х и более строк, которые надо соединить:

SELECT CONCAT('Tom', ' ', 'Smith')  -- Tom Smith

**LOWER**: переводит строку в нижний регистр:

SELECT LOWER('Samsung')   -- Samsung

**UPPER**: переводит строку в верхний регистр:

SELECT UPPER('Samsung')   -- SAMSUNG

Все эти функции можно использовать в выражениях SELECT. Рассмотрим пример. Допустим, у нас есть таблица *Subscribers:*

CREATE TABLE Subscribers

(Id INT PRIMARY KEY IDENTITY,

FirstName NVARCHAR(100),

LastName NVARCHAR(100),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

Заполним таблицу данными:

INSERT INTO Subscribers VALUES(N'Ренат', N'Королев', '125412-565214', '89614528745', 100)

INSERT INTO Subscribers VALUES(N'Мария', N'Орлова', '418759-325698', '89174852503', 584)

INSERT INTO Subscribers VALUES(N'Николай', N'Дмитриев', '512586-023652', '89294158754', 397)

INSERT INTO Subscribers VALUES(N'Сергей', N'Ларин', '512484-652588', '89274103252', 20)

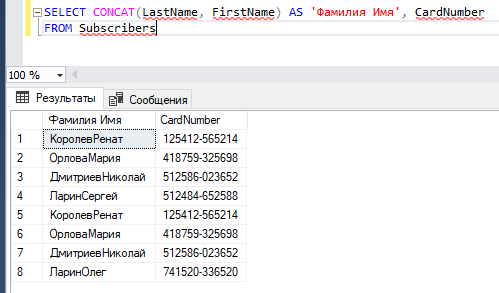
INSERT INTO Subscribers VALUES(N'Ренат', N'Королев', '125412-565214', '89614528745', 100)

INSERT INTO Subscribers VALUES(N'Мария', N'Орлова', '418759-325698', '89174852503', 584)

INSERT INTO Subscribers VALUES(N'Николай', N'Дмитриев', '512586-023652', '89294158754', 397)

INSERT INTO Subscribers VALUES(N'Олег', N'Ларин', '741520-336520', '89175420236', 205)

Имя и фамилия хранятся в разных столбцах, но при выборке их можно вывести в одном поле, используя функцию CONCAT.



**Функции для работы с числовыми данными**

**ROUND**: округляет число. В качестве первого параметра передается число. Второй параметр указывает на длину. Если длина представляет положительное число, то оно указывает, до какой цифры после запятой идет округление. Если длина представляет отрицательное число, то оно указывает, до какой цифры с конца числа до запятой идет округление

SELECT ROUND(836.248, 2)   -- 836.250

SELECT ROUND(836.248, -2)  -- 800.000

**ISNUMERIC**: определяет, является ли значение числом. В качестве параметра функция принимает выражение. Если выражение является числом, то функция возвращает 1. Если не является, то возвращается 0.

SELECT ISNUMERIC(836.248)          -- 1

SELECT ISNUMERIC('836.248')        -- 1

SELECT ISNUMERIC('MS SQL')         -- 0

SELECT ISNUMERIC('29-03-2019')  -- 0

**ABS**: возвращает абсолютное значение числа.

SELECT ABS(-834)    -- 834

**CEILING**: возвращает наименьшее целое число, которое больше или равно текущему значению.

SELECT CEILING(-546.23) -- -546

SELECT CEILING(546.23) -- 547

**FLOOR**: возвращает наибольшее целое число, которое меньше или равно текущему значению.

SELECT FLOOR(-546.23) -- -547

SELECT FLOOR(546.23) -- 546

**SQUARE**: возводит число в квадрат.

SELECT SQUARE(7)        -- 49

**SQRT**: получает квадратный корень числа.

SELECT SQRT(196)        -- 14

**RAND**: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.

SELECT RAND() --0,5729426786005

SELECT RAND() --0,530478516617022

**COS**: возвращает косинус угла, выраженного в радианах

SELECT COS(1.0472)  -- 0.5 - 60 градусов

**SIN**: возвращает синус угла, выраженного в радианах

SELECT SIN(1.5708)  -- 1 - 90 градусов

**TAN**: возвращает тангенс угла, выраженного в радианах

SELECT TAN(0.7854)  -- 1 - 45 градусов

**Функции для работы с датой и временем**

**GETDATE**: возвращает текущую локальную дату и время на основе системных часов в виде объекта datetime

SELECT GETDATE()    -- 2019-08-12 21:34:55.830

**GETUTCDATE**: возвращает текущую локальную дату и время по гринвичу (UTC/GMT) в виде объекта datetime

SELECT GETDATE()    -- 2019-08-12 17:34:55.830

**SYSDATETIME**: возвращает текущую локальную дату и время на основе системных часов, но отличие от **GETDATE** состоит в том, что дата и время возвращаются в виде объекта datetime2

SELECT SYSDATETIME()        -- 2019-08-12 21:02:22.7446744

**SYSUTCDATETIME**: возвращает текущую локальную дату и время по Гринвичу (UTC/GMT) в виде объекта datetime2

SELECT SYSDATETIME()        -- 2019-08-12 17:02:22.7446744

**SYSDATETIMEOFFSET**: возвращает объект datetimeoffset(7), который содержит дату и время относительно GMT

SELECT SYSDATETIMEOFFSET()    -- 2019-08-12 17:02:22.7446744 +03:00

**DAY**: возвращает день даты, который передается в качестве параметра

SELECT DAY(GETDATE())       -- 12

**MONTH**: возвращает месяц даты

SELECT MONTH(GETDATE())       -- 08

**YEAR**: возвращает год из даты

SELECT YEAR(GETDATE())       -- 2019

**DATENAME**: возвращает часть даты в виде строки. Параметр выбора части даты передается в качестве первого параметра, а сама дата передается в качестве второго параметра:

SELECT DATENAME(month, GETDATE())       -- Ноябрь

Для определения части даты можно использовать следующие параметры (в скобках указаны их сокращенные версии):

- year (yy, yyyy): год

- quarter (qq, q): квартал

- month (mm, m): месяц

- dayofyear (dy, y): день года

- day (dd, d): день месяца

- week (wk, ww): неделя

- weekday (dw): день недели

- hour (hh): час

- minute (mi, n): минута

- second (ss, s): секунда

- millisecond (ms): миллисекунда

- microsecond (mcs): микросекунда

- nanosecond (ns): наносекунда

- tzoffset (tz): смешение в минутах относительно Гринвича (для объекта datetimeoffset)

**DATEPART**: возвращает часть даты в виде числа. Параметр выбора части даты передается в качестве первого параметра (используются те же параметры, что и для DATENAME), а сама дата передается в качестве второго параметра:

SELECT DATEPART(month, GETDATE())       -- 7

**DATEADD**: возвращает дату, которая является результатом сложения числа к определенному компоненту даты. Первый параметр представляет компонент даты, описанный выше для функции DATENAME. Второй параметр - добавляемое количество. Третий параметр - сама дата, к которой надо сделать прибавление:

SELECT DATEADD(month, 2, '2019-7-28')      -- 2019-09-28 00:00:00.000

SELECT DATEADD(day, 5, '2019-7-28')     -- 2019-08-02 00:00:00.000

SELECT DATEADD(day, -5, '2019-7-28')       -- 2019-07-23 00:00:00.000

Если добавляемое количество представляет отрицательное число, то фактически происходит уменьшение даты.

**DATEDIFF**: возвращает разницу между двумя датами. Первый параметр - компонент даты, который указывает, в каких единицах стоит измерять разницу. Второй и третий параметры - сравниваемые даты:

SELECT DATEDIFF(year, '2017-7-28', '2018-9-28')     -- разница 1 год

SELECT DATEDIFF(month, '2017-7-28', '2018-9-28')    -- разница 14 месяцев

SELECT DATEDIFF(day, '2017-7-28', '2018-9-28')      -- разница 427 дней

**TODATETIMEOFFSET:** возвращает значение datetimeoffset, которое является результатом сложения временного смещения с другим объектом datetimeoffset

SELECT TODATETIMEOFFSET('2019-07-28 01:10:22', '+03:00')

**SWITCHOFFSET:** возвращает значение datetimeoffset, которое является результатом сложения временного смещения с объектом datetime2

SELECT SWITCHOFFSET(SYSDATETIMEOFFSET(), '+02:30')

**EOMONTH:** возвращает дату последнего дня для месяца, который используется в переданной в качестве параметра дате.

SELECT EOMONTH('2019-02-05')    -- 2019-02-28

SELECT EOMONTH('2019-02-05', 3) -- 2019-05-31

В качестве необязательного второго параметра можно передавать количество месяцев, которые необходимо прибавить к дате. Тогда последний день месяца будет вычисляться для новой даты.

**DATEFROMPARTS**: по году, месяцу и дню создает дату

SELECT DATEFROMPARTS(2019, 5, 13)       -- 2019-05-13

**ISDATE**: проверяет, является ли выражение датой. Если является, то возвращает 1, иначе возвращает 0.

SELECT ISDATE('2019-11-02')     -- 1

SELECT ISDATE('2019-31-01')     -- 0

SELECT ISDATE('28-07-2019')     -- 0

SELECT ISDATE('MS SQL')            -- 0

**Задание к лабораторной работе**

1. Продемонстрируйте работу 5 любых функций для работы со строками на своих данных с учетом варианта.
2. Продемонстрируйте работу 5 любых функций для работы с числами на своих данных с учетом варианта.
3. Продемонстрируйте работу 5 любых функций для работы с датой и временем на своих данных с учетом варианта.

# **Лабораторная работа №10. Переменные, циклы и ветвления. Обработка ошибок.**

**Переменные**

Переменная представляет именованный объект, который хранит некоторое значение. Для определения переменных применяется выражение **DECLARE**, после которого указывается название и тип переменной. При этом название локальной переменной должно начинаться с символа @:

DECLARE @название\_переменной тип\_данных

Например, определим переменную title, которая будет иметь тип NVARCHAR с максимальной длиной 50 символов:

DECLARE @Title NVARCHAR(50)

Также можно определить через запятую сразу несколько переменных:

DECLARE @Title NVARCHAR(50), @Count INT, @StartDate DATETIME

С помощью выражения **SET** можно присвоить переменной некоторое значение:

DECLARE @Title NVARCHAR(50), @Count INT, @StartDate DATETIME

SET @Title='Samsung Galaxy Note 10 Plus';

SET @Count = 10;

SET @StartDate = '2019-09-09'

Выражение **PRINT** возвращает сообщение клиенту. Например:

PRINT 'Hello world!'

И с его помощью мы можем вывести значение переменной:

DECLARE @Title NVARCHAR(50), @Count INT, @StartDate DATETIME

SET @Title='Samsung Galaxy Note 10 Plus';

SET @Count = 10;

SET @StartDate = '2019-09-09'

PRINT 'Title: ' + @Title

PRINT 'Count: ' + CONVERT(CHAR, @Count)

Также можно использовать для получения значения команду **SELECT**:

DECLARE @Title NVARCHAR(50), @Count INT, @StartDate DATETIME

SET @Title='Samsung Galaxy Note 10 Plus';

SET @Count = 10;

SET @StartDate = '2019-09-09'

SELECT @Title, @Count, @StartDate

Через переменные мы можем передавать данные в запросы. И также мы можем получать данные, которые являются результатом запросов, в переменные. Например, при выборке из таблиц с помощью команды **SELECT** мы можем извлекать данные в переменную с помощью следующего синтаксиса:

SELECT @переменная\_1 = спецификация\_столбца\_1,

        @переменная\_2 = спецификация\_столбца\_2,

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

        @переменная\_N = спецификация\_столбца\_N

Кроме того, в выражении **SET** значение, присваиваемое переменной, также может быть результатом команды **SELECT**.

Для сведения данных из разных таблиц мы можем использовать стандартную команду SELECT.

Допустим, у нас есть база данных с таблицами для смартфонов, покупателей и заказов:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

CREATE TABLE Orders

(Id INT PRIMARY KEY IDENTITY,

[SmartphoneId] INT NOT NULL REFERENCES Smartphones(Id) ON DELETE CASCADE,

[CustomerId] INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,

Count INT,

Date DATETIME)

Таблицы *Smartphones* и *Customers* связаны с таблицей *Orders* связью один ко многим.

Добавим в таблицы некоторые данные:

INSERT INTO Smartphones VALUES('Samsung Galaxy S8', 'Samsung', 35000, 3)

INSERT INTO Smartphones VALUES('Samsung Galaxy S8 plus', 'Samsung', 35000, 4)

INSERT INTO Smartphones VALUES('Samsung Galaxy S9', 'Samsung', 50000, 6)

INSERT INTO Smartphones VALUES('iPhone X', 'Apple', 40000, 5)

INSERT INTO Smartphones VALUES('iPhone XR', ' Apple', 50000, 7)

INSERT INTO Smartphones VALUES('iPhone XS', ' Apple', 56000, 5)

INSERT INTO Smartphones VALUES('iPhone 11', ' Apple', 65000, 10)

INSERT INTO Customers VALUES(N'Королев Ренат Александрович', '125412-565214', '89614528745', 100)

INSERT INTO Customers VALUES(N'Орлова Мария Викторовна', '418759-325698', '89174852503', 584)

INSERT INTO Customers VALUES(N'Дмитриев Николай Николаевич', '512586-023652', '89294158754', 397)

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone X'),

(SELECT Id FROM Customers WHERE CardNumber='125412-565214'),

2, '2019-04-01')

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='iPhone XR'),

(SELECT Id FROM Customers WHERE CardNumber='418759-325698'),

3, '2019-05-25')

INSERT INTO Orders VALUES(

(SELECT Id FROM Smartphones WHERE Name='Samsung Galaxy S9'),

(SELECT Id FROM Customers WHERE CardNumber='512586-023652'),

4, '2019-08-27')

Используем переменные для извлечения данных:

DECLARE @Sum INT, @MaxPrice INT, @MinPrice INT

SET @Sum = (SELECT SUM(Price) FROM Smartphones)

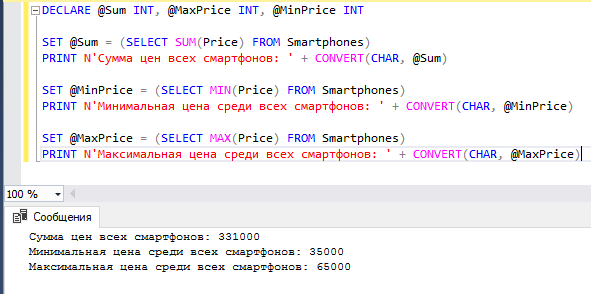
PRINT N'Сумма цен всех смартфонов: ' + CONVERT(CHAR, @Sum)

SET @MinPrice = (SELECT MIN(Price) FROM Smartphones)

PRINT N'Минимальная цена среди всех смартфонов: ' + CONVERT(CHAR,@MinPrice)

SET @MaxPrice = (SELECT MAX(Price) FROM Smartphones)

PRINT N'Максимальная цена среди всех смартфонов: ' + CONVERT(CHAR, @MaxPrice)



**Условные выражения**

Для выполнения действий по условию используется выражение **IF ... ELSE**. SQL Server вычисляет выражение после ключевого слово **IF**. И если оно истинно, то выполняются инструкции после ключевого слова **IF**. Если условие ложно, то выполняются инструкции после ключевого слова **ELSE**.

Если после **IF** или **ELSE** располагается блок инструкций, то этот блок заключается между ключевыми словами **BEGIN** и **END**:

IF условие

    {инструкция|BEGIN...END}

[ELSE

    {инструкция|BEGIN...END}]

Выражение ELSE является необязательным, и его можно опустить.

Рассмотрим пример на основе предыдущих таблиц. Допустим, в зависимости от данных в таблице *Orders* выведем соответствующее сообщение.

DECLARE @lastDate DATE

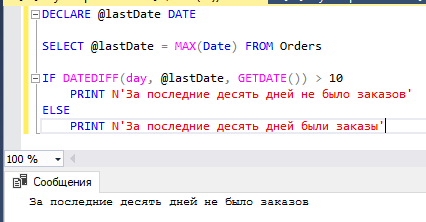
SELECT @lastDate = MAX(Date) FROM Orders

IF DATEDIFF(day, @lastDate, GETDATE()) > 10

PRINT N'За последние десять дней не было заказов'

ELSE

PRINT N'За последние десять дней были заказы'



Если после **IF** или **ELSE** идут две и более инструкций, то они заключаются в блок **BEGIN...END**:

DECLARE @lastDate DATE

SELECT @lastDate = MAX(Date) FROM Orders

IF DATEDIFF(year, @lastDate, GETDATE()) < 1

BEGIN

DECLARE @Sum INT, @MaxPrice INT, @MinPrice INT

SET @Sum = (SELECT SUM(Price) FROM Smartphones)

PRINT N'Сумма цен всех смартфонов: ' + CONVERT(CHAR, @Sum)

SET @MinPrice = (SELECT MIN(Price) FROM Smartphones)

PRINT N'Минимальная цена среди всех смартфонов: ' + CONVERT(CHAR, @MinPrice)

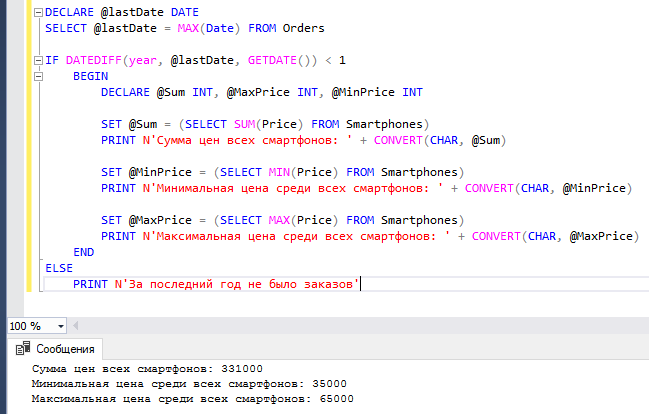
SET @MaxPrice = (SELECT MAX(Price) FROM Smartphones)

PRINT N'Максимальная цена среди всех смартфонов: ' + CONVERT(CHAR,@MaxPrice)

END

ELSE

PRINT N'За последний год не было заказов'



**Циклы**

Для выполнения повторяющихся операций в T-SQL применяются циклы. В частности, в T-SQL есть цикл **WHILE**. Этот цикл выполняет определенные действия, пока некоторое условие истинно.

WHILE условие

    {инструкция|BEGIN...END}

Если в блоке **WHILE** необходимо разместить несколько инструкций, то все они помещаются в блок **BEGIN...END**.

Например, вычислим факториал числа:

DECLARE @number INT, @factorial INT

SET @factorial = 1;

SET @number = 6;

WHILE @number > 0

BEGIN

SET @factorial = @factorial \* @number

SET @number = @number - 1

END;

PRINT @factorial

То есть в данном случае пока переменная @number не будет равна 0, будет продолжаться цикл WHILE. Так как @number равна 6, то цикл сделает шесть проходов. Каждый проход цикла называется итерацией. В каждой итерации будет переустанавливаться значение переменных @factorial и @number.

**Обработка ошибок**

Для обработки ошибок в T-SQL применяется конструкция **TRY...CATCH**. Она имеет следующий формальный синтаксис:

BEGIN TRY

    инструкции

END TRY

BEGIN CATCH

    инструкции

END CATCH

Между выражениями **BEGIN TRY** и **END TRY** помещаются инструкции, которые потенциально могут вызвать ошибку, например, какой-нибудь запрос. И если в этом блоке **TRY** возникнет ошибка, то управление передается в блок **CATCH**, где можно обработать ошибку.

В блоке **CATCH** для обработки ошибки мы можем использовать ряд функций:

**ERROR\_NUMBER():** возвращает номер ошибки

**ERROR\_MESSAGE():** возвращает сообщение об ошибке

**ERROR\_SEVERITY():** возвращает степень серьезности ошибки. Степень серьезности представляет числовое значение. И если оно равно 10 и меньше, то такая ошибка рассматривается как предупреждение и не обрабатывается конструкцией **TRY...CATCH**. Если же это значение равно 20 и выше, то такая ошибка приводит к закрытию подключения к базе данных, если она не обрабатывается конструкцией **TRY...CATCH**.

**ERROR\_STATE():** возвращает состояние ошибки

Например, добавим в таблицу данные, которые не соответствуют ограничениям столбцов:

CREATE TABLE Products(Title NVARCHAR(100) NOT NULL, [Count] INT NOT NULL)

BEGIN TRY

INSERT INTO Products VALUES(NULL, NULL)

PRINT 'Данные успешно добавлены!'

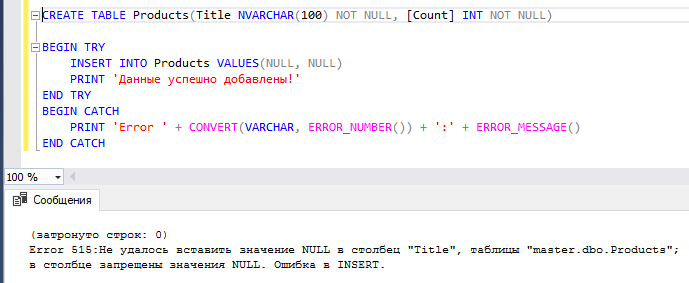
END TRY

BEGIN CATCH

PRINT 'Error '+CONVERT(VARCHAR, ERROR\_NUMBER()) + ':' + ERROR\_MESSAGE()

END CATCH

В данном случае для столбцов таблицы вставляются недопустимые данные - значения NULL, поэтому обработка программы перейдет к блоку CATCH:



**Задание к лабораторной работе**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Название** | **Условие** |
| 1 | ИС «Электронная ведомость ВУЗа» | 1. Объявить новую переменную и присвоить ей следующее значение: самую раннюю дату зачисления из таблицы *Студенты.* Вывести значение переменной на экран. 2. Найти преподавателя, который ведет дисциплину «Мат.анализ» и если это профессор, то вывести на экран «Дисциплину ведет профессор.», иначе вывести на экран «Дисциплину ведет не профессор.» 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Дисциплины.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 2 | ИС «Пассажирское автопредприятие» | 1. Объявить новую переменную и присвоить ей следующее значение: средний пробег*.* Вывести значение переменной на экран. 2. Найти пробег для машины Mazda CX7 и если это значение больше 100000, то вывести на экран «Машина не подлежит эксплуатации.», иначе вывести на экран «Машина подлежит эксплуатации.» 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Модели.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 3 | ИС «Таксопарк» | 1. Объявить новую переменную и присвоить ей следующее значение: найти самое старое транспортное средство*.* Вывести значение переменной на экран. 2. Найти цену для Эконом класса и если это значение меньше 1000, то вывести на экран «Необходимо увеличить на 200», иначе вывести на экран «Цена актуальна, не увеличивать.» 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Классы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 4 | ИС «Электронная регистратура поликлиники» | 1. Объявить новую переменную и присвоить ей следующее значение: найти самого нового сотрудника*.* Вывести значение переменной на экран. 2. Найти дату рождения для любого сотрудника и если сотруднику больше 70 лет, то вывести на экран «Сотрудник уходит на пенсию», иначе вывести на экран «Сотрудник еще не уходит на пенсию.» 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Должности.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 5 | ИС «Библиотека» | 1. Объявить новую переменную и присвоить ей следующее значение: найти самую первую изданную книгу*.* Вывести значение переменной на экран. 2. Найти дату издания для любой книги и если ей больше 60 лет, то вывести на экран «Ценный экземпляр, только читальный зал.», иначе вывести на экран «Можно забирать домой.» 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Авторы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 6 | ИС «Гостиница» | 1. Объявить новую переменную и присвоить ей следующее значение: найти сотрудника, который работает дольше всех*.* Вывести значение переменной на экран. 2. Найти дату рождения для любой горничной и если ей больше 70 лет, то вывести на экран «Сотрудник уходит на пенсию.», иначе вывести на экран «Сотрудник продолжает работать». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Номера.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 7 | ИС «Ресторан» | 1. Объявить новую переменную и присвоить ей следующее значение: найти сотрудника, который работает дольше всех*.* Вывести значение переменной на экран. 2. Найти дату рождения для любого официанта и если ей больше 70 лет, то вывести на экран «Сотрудник уходит на пенсию.», иначе вывести на экран «Сотрудник продолжает работать». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Официанты.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 8 | ИС «Кинотеатр» | 1. Объявить новую переменную и присвоить ей следующее значение: найти дату окончания показа самого нового фильма*.* Вывести значение переменной на экран. 2. Найти дату начала показа любого фильма и если разница между этой датой и текущим днем больше 30 дней, то вывести на экран «Показ фильма окончен.», иначе вывести на экран «Фильм в прокате». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Залы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 9 | ИС «Автосалон» | 1. Объявить новую переменную и присвоить ей следующее значение: найти максимальный пробег среди всех машин*.* Вывести значение переменной на экран. 2. Найти дату рождения любого покупателя и если ему больше 21 и меньше 70 лет, то вывести на экран «Можно давать в кредит.», иначе вывести на экран «Нельзя давать в кредит». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Покупатели.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 10 | ИС «Цветочный магазин» | 1. Объявить новую переменную и присвоить ей следующее значение: найти среднюю цену поставщика среди всех цветов*.* Вывести значение переменной на экран. 2. Найти страну любого поставщика и если это «Россия», то вывести на экран «Отечественный поставщик», иначе вывести на экран «Зарубежный поставщик». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Цветы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 11 | ИС «ЗАГС» | 1. Объявить новую переменную и присвоить ей следующее значение: найти дату подачи самого старого заявления*.* Вывести значение переменной на экран. 2. Найти дату подачи любого заявления и если ему оно подано более 50 лет назад, то вывести на экран «Заявление следует перевести в архив.», иначе вывести на экран «Заявление все еще актуально». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Заявители.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 12 | ИС «Отдел кадров предприятия» | 1. Объявить новую переменную и присвоить ей следующее значение: найти дату рождения самого молодого сотрудника*.* Вывести значение переменной на экран. 2. Найти дату рождения любого сотрудника и если ему больше 70 лет, то вывести на экран «Сотрудник уходит на пенсию.», иначе вывести на экран «Сотрудник еще не выходит на пенсию». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Отделы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 13 | ИС «Паспортный стол» | 1. Объявить новую переменную и присвоить ей следующее значение: найти дату подачи заявления самого старого паспорта*.* Вывести значение переменной на экран. 2. Найти дату выдачи любого паспорта и если это было больше 35 лет, то вывести на экран «Заявление можно перевести в архив.», иначе вывести на экран «Заявление еще актуально». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Паспорта.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 14 | ИС «Складской учет на предприятии» | 1. Объявить новую переменную и присвоить ей следующее значение: найти максимальную цену среди всех товаров*.* Вывести значение переменной на экран. 2. Найти сумму любой накладной и если она больше 30000, то вывести на экран «Предоставить скидку 10%.», иначе вывести на экран «Скидка не предоставляется». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Товары.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 15 | ИС «Туристическое агентство» | 1. Объявить новую переменную и присвоить ей следующее значение: найти минимальную цену среди всех путевок*.* Вывести значение переменной на экран. 2. Найти страну любой путевки и если это «Россия», то вывести на экран «Отдых на Родине.», иначе вывести на экран «Отдых за рубежом». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Туристы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 16 | ИС «Автосервис» | 1. Объявить новую переменную и присвоить ей следующее значение: найти среднюю цену за услугу среди всех транспортных средств*.* Вывести значение переменной на экран. 2. Найти цену за услугу любого транспортного средства и если она больше 10000, то вывести на экран «Предоставить скидку 10%.», иначе вывести на экран «Скидка не предоставляется». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Заказчики.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 17 | ИС «Кондитерское предприятие» | 1. Объявить новую переменную и присвоить ей следующее значение: найти среднюю цену среди всех кондитерских изделий*.* Вывести значение переменной на экран. 2. Найти цену любого кондитерского изделия и если она больше 2500, то вывести на экран «Предоставить скидку 5%.», иначе вывести на экран «Скидка не предоставляется». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Кондитерские изделия.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 18 | ИС «Ателье мод» | 1. Объявить новую переменную и присвоить ей следующее значение: найти максимальную цену среди всех изделий*.* Вывести значение переменной на экран. 2. Найти цену любого изделия и если она больше 15000, то вывести на экран «Предоставить скидку 10%.», иначе вывести на экран «Скидка не предоставляется». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Заказчики.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 19 | ИС «Электронный школьный дневник» | 1. Объявить новую переменную и присвоить ей следующее значение: найти самый высокий этаж*.* Вывести значение переменной на экран. 2. Найти этаж любого класса и если он выше 2, то вывести на экран «Только для старшеклассников.», иначе вывести на экран «Для младших классов». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Классы.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 20 | ИС «Служба доставки» | 1. Объявить новую переменную и присвоить ей следующее значение: найти среднюю цену среди всех заказов*.* Вывести значение переменной на экран. 2. Найти сумму любого заказа и если она больше 3000, то вывести на экран «Предоставить скидку 10%.», иначе вывести на экран «Скидка не предоставляется.». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Заказчики.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 21 | ИС «Сервисный центр по ремонту компьютерной техники» | 1. Объявить новую переменную и присвоить ей следующее значение: найти среднюю цену среди всех заказов*.* Вывести значение переменной на экран. 2. Найти сумму любого заказа и если она больше 15000, то вывести на экран «Предоставить скидку 10%.», иначе вывести на экран «Скидка не предоставляется.». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Виды техники.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 22 | ИС «Салон красоты» | 1. Объявить новую переменную и присвоить ей следующее значение: найти среднюю цену среди всех услуг*.* Вывести значение переменной на экран. 2. Найти цену любой услуги и если она больше 10000, то вывести на экран «Предоставить скидку 5%.», иначе вывести на экран «Скидка не предоставляется.». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Услуги.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |
| 23 | ИС «Санитарно-курортный комплекс» | 1. Объявить новую переменную и присвоить ей следующее значение: найти максимальное среднее время выполнения среди всех процедур*.* Вывести значение переменной на экран. 2. Найти среднее время любой процедуры и если оно больше 60 минут, то вывести на экран «Долгая процедура», иначе вывести на экран «Короткая процедура». 3. Используя цикл, вставить 30 новых сгенерированных записей в таблицу *Кабинеты.* 4. Продемонстрировать работу блока TRY…CATCH двумя примерами. |

# **Лабораторная работа №11. Представления и табличные объекты**

***Представления (Views)*** представляют виртуальные таблицы. Но в отличии от обычных стандартных таблиц в базе данных представления содержат запросы, которые динамически извлекают используемые данные.

Представления дают ряд преимуществ. Они упрощают комплексные SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

Для создания представления используется команда **CREATE VIEW**, которая имеет следующую форму:

CREATE VIEW название\_представления [(столбец\_1, столбец\_2, ....)]

AS выражение\_SELECT

Рассмотрим следующие таблицы:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

CREATE TABLE Orders

(Id INT PRIMARY KEY IDENTITY,

[SmartphoneId] INT NOT NULL REFERENCES Smartphones(Id) ON DELETE CASCADE,

[CustomerId] INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,

Count INT,

Date DATETIME)

Теперь добавим в эту же базу данных новое представление:

CREATE VIEW OrdersSmartphonesCustomers AS

SELECT Orders.Date AS OrderDate,

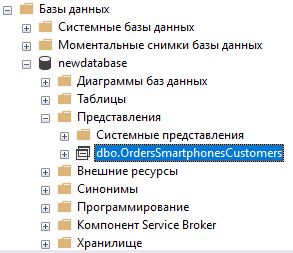
Customers.Name AS Customer,

Smartphones.Name As Product

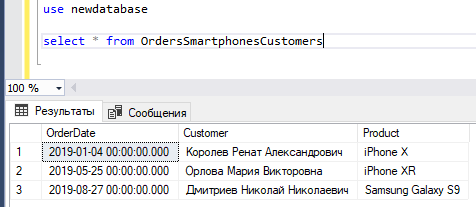
FROM Orders JOIN Smartphones ON Orders.SmartphoneId = Smartphones.Id

JOIN Customers ON Orders.CustomerId = Customers.Id

То есть данное представление фактически будет возвращать сводные данные из трех таблиц. И после его создания мы сможем его увидеть в узле Views у выбранной базы данных в SQL Server Management Studio:



Теперь используем созданное выше представление для получения данных:



При создании представлений следует учитывать, что представления, как и таблицы, должны иметь уникальные имена в рамках той же базы данных.

Представления могут иметь не более 1024 столбцов и могут обращать не более чем к 256 таблицам.

Также можно создавать представления на основе других представлений. Такие представления еще называют вложенными (nested views). Однако уровень вложенности не может быть больше 32-х.

Команда SELECT, используемая в представлении, не может включать выражения INTO или ORDER BY (за исключением тех случаев, когда также применяется выражение TOP или OFFSET). Если же необходима сортировка данных в представлении, то выражение ORDER BY применяется в команде SELECT, которая непосредственно извлекает данные из представления.

Также при создании представления можно определить набор его столбцов:

CREATE VIEW OrdersSmartphonesCustomers (OrderDate, Customer, Product) AS

SELECT Orders.Date,

Customers.Name,

Smartphones.Name

FROM Orders JOIN Smartphones ON Orders.SmartphoneId = Smartphones.Id

JOIN Customers ON Orders.CustomerId = Customers.Id

Для изменения представления используется команда **ALTER VIEW**. Эта команда имеет практически тот же самый синтаксис, что и **CREATE VIEW**:

ALTER VIEW название\_представления [(столбец\_1, столбец\_2, ....)]

AS выражение\_SELECT

Например, изменим выше созданное представление OrdersSmartphonesCustomers:

ALTER VIEW OrdersSmartphonesCustomers (OrderDate, Customer, Product, Manufacturer) AS

SELECT Orders.Date,

Customers.Name,

Smartphones.Name ,

Smartphones.Manufacturer

FROM Orders JOIN Smartphones ON Orders.SmartphoneId = Smartphones.Id

JOIN Customers ON Orders.CustomerId = Customers.Id

Для удаления представления вызывается команда **DROP VIEW**:

DROP VIEW OrdersSmartphonesCustomers

Следует отметить, что при удалении таблиц, которые используются в представлении, необходимо также удалить эти представления.

**Обновляемые представления**

Представления могут быть обновляемыми (**updatable**). В таких представлениях мы можем изменить или удалить строки или добавить в них новые строки.

При создании подобных представлений есть множество ограничений. В частности, команда SELECT в представлении не может содержать следующие команды/операторы:

* TOP,
* DISTINCT,
* UNION,
* JOIN,
* агрегатные функции типа COUNT или MAX,
* GROUP BY и HAVING,
* Подзапросы,
* Производные столбцы или столбцы, которые вычисляются на основании нескольких значений,
* Обращения одновременно к нескольким таблицам

Допустим, у нас есть следующая таблица:

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

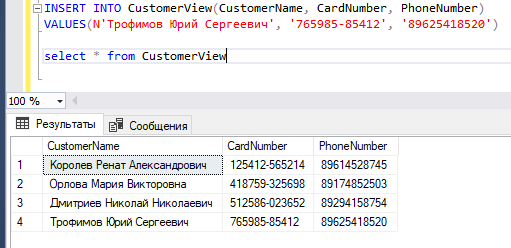
Создадим для нее обновляемое представление:

CREATE VIEW CustomerView

AS SELECT Name AS CustomerName, CardNumber, PhoneNumber

FROM Customers

Добавим в него новые данные:



Стоит отметить, что при добавлении фактически будет добавлен объект в таблицу *Customers*, которую использует представление *CustomerView*. И поэтому надо учитывать, что если в этой таблице есть какие-либо столбцы, в которые представление не добавляет данные, но которые не допускают значение NULL или не поддерживают значение по умолчанию, то добавление завершится с ошибкой.

Обновление строки представления выглядит следующим образом:

update CustomerView set CardNumber = '657493-653398'

where PhoneNumber = '89625418520'

Удаление строки представления:

delete from CustomerView where PhoneNumber = '89625418520'

**Табличные переменные**

***Табличные переменные (table variable)*** позволяют сохранить содержимое целой таблицы. Формальный синтаксис определения подобной переменной во многом похож на создание таблицы:

DECLARE @табличная\_переменная TABLE

(столбец\_1 тип\_данных [атрибуты\_столбца],

 столбец\_2 тип\_данных [атрибуты\_столбца] ....)

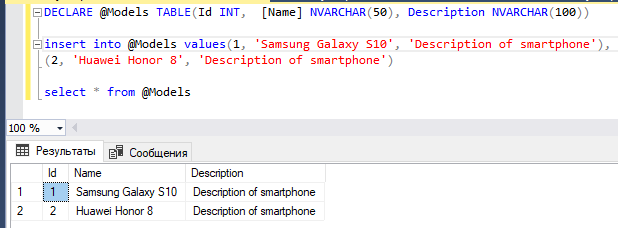
 [атрибуты\_таблицы]

Например:

DECLARE @Models TABLE(Id INT,  Name NVARCHAR(50), Description(100))

В данном случае переменная @Models будет содержать два столбца.

В дальнейшем мы сможем работать с этой переменной как с обычной таблицей, то есть добавлять в нее данные, изменять, удалять и извлекать их:



Однако следует учитывать, что такие переменные не полностью эквивалентны таблицам. Они живут в пределах одного пакета, после завершения работы которого они удаляются. То есть они носят временный характер, и физически их данные нигде не хранятся на жестком диске.

**Временные и производные таблицы**

**Временные таблицы**

В дополнение к табличным переменным можно определять временные таблицы. Такие таблицы могут быть полезны для хранения табличных данных внутри сложного комплексного скрипта.

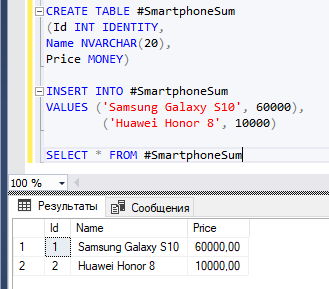
Временные таблицы существуют на протяжении сессии базы данных. Если такая таблица создается в редакторе запросов (Query Editor) в SQL Server Management Studio, то таблица будет существовать пока открыт редактор запросов. Таким образом, к временной таблице можно обращаться из разных скриптов внутри редактора запросов.

После создания все временные таблицы сохраняются в таблице **tempdb**, которая имеется по умолчанию в MS SQL Server.

Если необходимо удалить таблицу до завершения сессии базы данных, то для этой таблицы следует выполнить команду **DROP TABLE**.

Название временной таблицы начинается со знака решетки #. Если используется один знак #, то создается локальная таблица, которая доступна в течение текущей сессии. Ели используются два знака ##, то создается глобальная временная таблица. В отличие от локальной глобальная временная таблица доступна всем открытым сессиям базы данных.

Например, создадим локальную временную таблицу:



И с этой таблицей можно работать в большей степени, как и с обычной таблицей - получать данные, добавлять, изменять и удалять их. Только после закрытия редактора запросов эта таблица перестанет существовать.

Подобные таблицы удобны для каких-то временных промежуточных данных. Например, пусть у нас есть три таблицы:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

CREATE TABLE Customers

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(300),

CardNumber NVARCHAR(13),

PhoneNumber NVARCHAR(25),

Points INT)

CREATE TABLE Orders

(Id INT PRIMARY KEY IDENTITY,

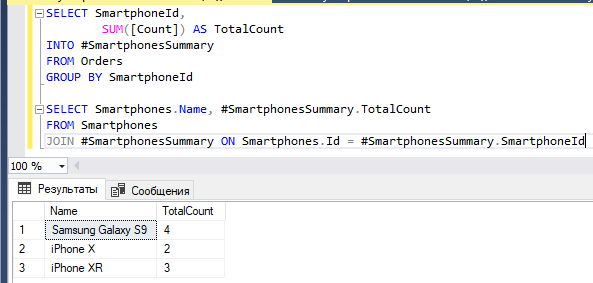
[SmartphoneId] INT NOT NULL REFERENCES Smartphones(Id) ON DELETE CASCADE,

[CustomerId] INT NOT NULL REFERENCES Customers(Id) ON DELETE CASCADE,

Count INT,

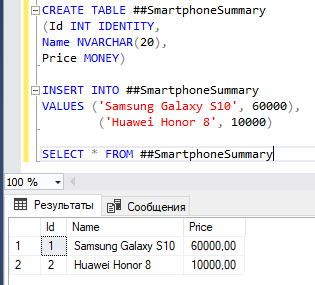
Date DATETIME)

Выведем во временную таблицу промежуточные данные из таблицы Orders:



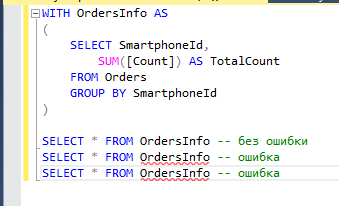
Здесь вначале извлекаются данные во временную таблицу *#SmartphonesSummary*. Причем так как данные в нее извлекаются с помощью выражения SELECT INTO, то предварительно таблицу не надо создавать. И эта таблица будет содержать id смартфона и общее количество проданного товара. Затем эта таблица может использоваться в выражениях INNER JOIN.

Аналогично определяются глобальные временные таблицы, отличие только в том, что их имя начинается с двух знаков ##:



**Производные таблицы**

Кроме временных таблиц MS SQL Server позволяет создавать производные таблицы, которые в плане производительности являются более эффективным решением, чем временные. Производная таблица задается с помощью ключевого слова **WITH**:



В отличие от временных таблиц производные хранятся в оперативной памяти и существуют только во время первого выполнения запроса, который представляет эту таблицу.

**Задание к лабораторной работе**

1. Создайте новое представление. Состав возьмите из лабораторной работы №8 задание 2 (JOIN).
2. Измените представление, добавив в него любой столбец.
3. Удалите представление.
4. Создайте обновляемое представление для одной из таблиц. Продемонстрируйте добавление, изменение и удаление данных на созданном представлении.
5. Создайте любую табличную переменную, исходя из Вашего варианта. Добавьте в нее 5 записей.
6. Создайте временную таблицу, исходя из Вашего варианта и поместите туда данные из ранее созданных таблиц.
7. Создайте производную производную таблицу, исходя из Вашего варианта.

# **Лабораторная работа №12. Хранимые процедуры и функции.**

Нередко операция с данными представляет набор инструкций, которые необходимо выполнить в определенной последовательности. Например, при добавлении покупке товара необходимо внести данные в таблицу заказов. Однако перед этим надо проверить, а есть ли покупаемый товар в наличии. Возможно, при этом понадобится проверить еще ряд дополнительных условий. То есть фактически процесс покупки товара охватывает несколько действий, которые должны выполняться в определенной последовательности. И в этом случае более оптимально будет инкапсулировать все эти действия в один объект - ***хранимую процедуру (stored procedure)***.

То есть хранимые процедуры представляет набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект. Изменится процесс покупки товара, соответственно достаточно будет изменить код процедуры. То есть процедура также упрощает управление кодом.

Также хранимые процедуры позволяют ограничить доступ к данным в таблицах и тем самым уменьшить вероятность преднамеренных или неосознанных нежелательных действий в отношении этих данных.

И еще один важный аспект - производительность. Хранимые процедуры обычно выполняются быстрее, чем обычные SQL-инструкции. Потому что код процедур компилируется один раз при первом ее запуске, а затем сохраняется в скомпилированной форме.

Для создания хранимой процедуры применяется команда **CREATE PROCEDURE** или сокращенно **CREATE PROC.**

Таким образом, хранимая процедура имеет три ключевых особенности: упрощение кода, безопасность и производительность.

Например, пусть в базе данных есть таблица, которая хранит данные о смартфонах:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

Создадим хранимую процедуру для извлечения данных из этой таблицы:

USE newdatabase

GO

CREATE PROC GetSmartphones AS

SELECT [Name], Manufacturer, Price, [COunt] FROM Smartphones

ORDER BY [Name]

Поскольку команда **CREATE PROCEDURE** должна вызываться в отдельном пакете, то после команды **USE**, которая устанавливает текущую базу данных, используется команда **GO** для определения нового пакета.

После имени процедуры должно идти ключевое слово **AS**.

Для отделения тела процедуры от остальной части скрипта код процедуры нередко помещается в блок **BEGIN...END**:

USE newdatabase

GO

CREATE PROC GetSmartphones AS

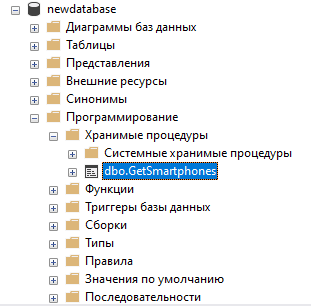
BEGIN

SELECT [Name], Manufacturer, Price, [Count] FROM Smartphones

ORDER BY [Name]

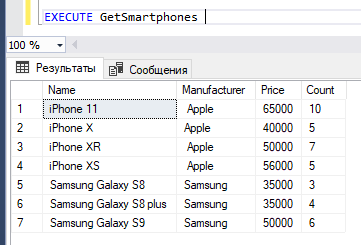
END

После добавления процедуры мы ее можем увидеть в узле базы данных в SQL Server Management Studio в разделе ***Программирование -> Хранимые процедуры (Programmability -> Stored Procedures)***:



Для выполнения хранимой процедуры вызывается команда EXEC или EXECUTE:

EXEC GetSmartphones



Для удаления процедуры применяется команда **DROP PROCEDURE**:

DROP PROCEDURE GetSmartphones

**Параметры в процедурах**

Процедуры могут принимать параметры. Параметры бывают входными - с их помощью в процедуру можно передать некоторые значения. И также параметры бывают выходными - они позволяют возвратить из процедуры некоторое значение.

Например, пусть в базе данных будет следующая таблица *Smartphones*:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

Определим процедуру, которая будет добавлять данные в эту таблицу:

USE newdatabase;

GO

CREATE PROCEDURE AddSmartphone

@name NVARCHAR(100),

@manufacturer NVARCHAR(100),

@count INT,

@price MONEY

AS

INSERT INTO Smartphones(Name, Manufacturer, [Count], Price)

VALUES(@name, @manufacturer, @count, @price)

После названия процедуры идет список входных параметров, которые определяются также как и переменные - название начинается с символа @, а после названия идет тип переменной. И с помощью команды **INSERT** значения этих параметров будут передаваться в таблицу Smartphones.

Выполним эту процедуру:

USE newdatabase;

DECLARE @Name NVARCHAR(20), @manufacturer NVARCHAR(20);

DECLARE @Count INT, @price MONEY

SET @Name = 'Galaxy A5'

SET @manufacturer = 'Samsung'

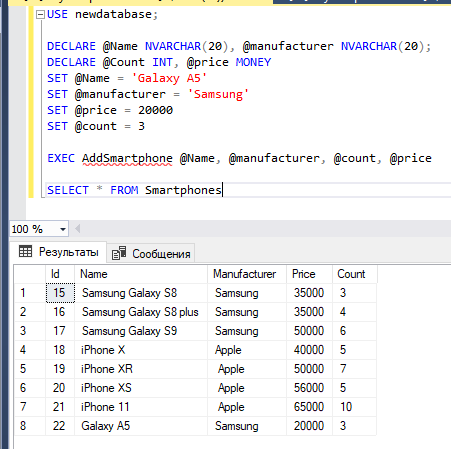
SET @price = 20000

SET @count = 3

EXEC AddSmartphone @Name, @manufacturer, @count, @price

SELECT \* FROM Smartphones

Здесь передаваемые в процедуру значения определяются через переменные. При вызове процедуры ей через запятую передаются значения. При этом значения передаются параметрам процедуры по позиции. Так как первым определен параметр @name, то ему будет передаваться первое значение - значение переменной @Name. Второму параметру - @manufacturer передается второе значение - значение переменной @manufacturer и так далее. Главное, чтобы между передаваемыми значениями и параметрами процедуры было соответствие по типу данных.



Процедуру также можно выполнить, не объявляя дополнительных переменных, а передать непосредственно значения:

EXEC AddSmartphone 'iPhone SE', 'Apple', 4, 15000

Также значения параметрам процедуры можно передавать по имени. Это удобно для того, чтобы не перепутать параметры.

USE newdatabase;

DECLARE @Name NVARCHAR(20), @manufacturer NVARCHAR(20);

SET @Name = 'Galaxy S10'

SET @manufacturer = 'Samsung'

EXEC AddSmartphone @Name = @Name,

@manufacturer = @manufacturer,

@count = 2,

@price = 60000

Параметры можно отмечать, как необязательные, присваивая им некоторое значение по умолчанию. Например, в случае выше мы можем автоматически устанавливать для количества товара значение 1, а цену = 20000 если соответствующие значения не переданы в процедуру:

USE newdatabase;

GO

CREATE PROCEDURE AddSmartphoneWithOptionalCount

@name NVARCHAR(100),

@manufacturer NVARCHAR(100),

@count INT = 1,

@price MONEY = 20000

AS

INSERT INTO Smartphones(Name, Manufacturer, [Count], Price)

VALUES(@name, @manufacturer, @count, @price)

И в этом случае для параметров @count и @price в процедуру можно не передавать значения.

**Выходные параметры**

Выходные параметры позволяют возвратить из процедуры некоторый результат. Выходные параметры определяются с помощью ключевого слова **OUTPUT**. Например, определим еще одну процедуру:

USE newdatabase;

GO

CREATE PROCEDURE GetPriceStatistics

@minPrice MONEY OUTPUT,

@maxPrice MONEY OUTPUT

AS

SELECT @minPrice = MIN(Price), @maxPrice = MAX(Price)

FROM Smartphones

При вызове процедуры для выходных параметров передаются переменные с ключевым словом OUTPUT:

USE newdatabase;

DECLARE @minPrice MONEY, @maxPrice MONEY

EXEC GetPriceStatistics @minPrice OUTPUT, @maxPrice OUTPUT

PRINT 'Минимальная цена ' + CONVERT(VARCHAR, @minPrice)

PRINT 'Максимальная цена ' + CONVERT(VARCHAR, @maxPrice)

Обратите внимание, что можно сочетать входные и выходные параметры. Например, определим процедуру, которая добавляет новую строку в таблицу и возвращает ее id:

USE newdatabase;

GO

CREATE PROCEDURE CreateSmartphone

@name NVARCHAR(20),

@manufacturer NVARCHAR(20),

@count INT,

@price MONEY,

@id INT OUTPUT

AS

INSERT INTO Smartphones([Name], Manufacturer, [Count], Price)

VALUES(@name, @manufacturer, @count, @price)

SET @id = @@IDENTITY

С помощью глобальной переменной **@@IDENTITY** можно получить идентификатор добавленной записи.

При вызове этой процедуры ей также по позиции передаются все входные и выходные параметры.

**Возращение значения**

Кроме передачи результата выполнения через выходные параметры хранимая процедура также может возвращать какое-либо значение с помощью оператора **RETURN**. Хотя данная возможность во многом нивелирована использованием выходных параметров, через которые можно возвращать результат, тем не менее, если надо возвратить из процедуры одно значение, то вполне можно использовать оператор **RETURN**.

USE newdatabase;

GO

CREATE PROCEDURE GetAvgPrice AS

DECLARE @avgPrice MONEY

SELECT @avgPrice = AVG(Price)

FROM Smartphones

RETURN @avgPrice;

После оператора **RETURN** указывается возвращаемое значение. В данном случае это значение переменной @avgPrice.

Вызовем данную процедуру:

USE newdatabase;

DECLARE @result MONEY

EXEC @result = GetAvgPrice

PRINT @result

Для получения результата процедуры ее значение сохраняется в переменную (в данном случае в переменную @result).

**Функции**

***Функции, определенные пользователем (user defined functions, UDF)*** — это конструкции, содержащие исполняемый код. Функция выполняет какие-либо действия над данными и возвращает некоторое значение/набор данных. К функциям можно обращаться из триггеров, хранимых процедура и из других программных компонентов.

Пользовательские функции по функциональности похожи на хранимые процедуры. Разница заключается в том, что возможностей у них меньше (в частности, они должны возвращать только одно значение, например, скалярное или табличное), также в функциях нельзя вставлять новые данные в таблицу или обновлять текущие данные.

Создание функции имеет следующий синтаксис:

CREATE FUNCTION [schema\_name.]function\_name

[( {@param } type [= default]) {,...}

RETURNS {scalar\_type | [@variable] TABLE}

[WITH {ENCRYPTION | SCHEMABINDING}

[AS] {block | RETURN (select\_statement)}

Параметр **schema\_name** определяет имя схемы, которая назначается владельцем создаваемой UDF, а параметр **function\_name** определяет имя этой функции. Параметр **@param** является входным параметром функции (формальным аргументом), чей тип данных определяется параметром **type**. Параметры функции — это значения, которые передаются вызывающим объектом определяемой пользователем функции для использования в ней. Параметр **default** определяет значение по умолчанию для соответствующего параметра функции. (Значением по умолчанию также может быть NULL.)

Предложение **RETURNS** определяет тип данных значения, возвращаемого функцией. Это может быть почти любой стандартный тип данных, поддерживаемый системой баз данных, включая тип данных **TABLE**. Единственным типом данных, который нельзя указывать, является тип данных **timestamp**.

Определяемые пользователем функции могут быть либо скалярными, либо табличными. Скалярные функции возвращают атомарное (скалярное) значение. Это означает, что в предложении **RETURNS** скалярной функции указывается один из стандартных типов данных. Функция является табличной, если предложение **RETURNS** возвращает набор строк.

Параметр **WITH ENCRYPTION** в системном каталоге кодирует информацию, содержащую текст инструкции CREATE FUNCTION. Таким образом, предотвращается несанкционированный просмотр текста, который был использован для создания функции. Данная опция позволяет повысить безопасность системы баз данных.

Альтернативное предложение **WITH SCHEMABINDING** привязывает функцию к объектам базы данных, к которым эта функция обращается. После этого любая попытка модифицировать объект базы данных, к которому обращается функция, претерпевает неудачу. (Привязка функции к объектам базы данных, к которым она обращается, удаляется только при изменении функции, после чего параметр **SCHEMABINDING** больше не задан.)

Для того чтобы во время создания функции использовать предложение **SCHEMABINDING**, объекты базы данных, к которым обращается функция, должны удовлетворять следующим условиям:

* все представления и другие функции, к которым обращается определяемая функция, должны быть привязаны к схеме;
* все объекты базы данных (таблицы, представления и функции) должны быть в той же самой базе данных, что и определяемая функция.

Параметр **block** определяет блок **BEGIN/END**, содержащий реализацию функции. Последней инструкцией блока должна быть инструкция **RETURN** с аргументом. (Значением аргумента является возвращаемое функцией значение.) Внутри блока **BEGIN/END** разрешаются только следующие инструкции:

* инструкции присвоения, такие как SET;
* инструкции для управления ходом выполнения, такие как WHILE и IF;
* инструкции DECLARE, объявляющие локальные переменные;
* инструкции SELECT, содержащие списки столбцов выборки с выражениями, значения которых присваиваются переменным, являющимися локальными для данной функции;
* инструкции INSERT, UPDATE и DELETE, которые изменяют переменные с типом данных TABLE, являющиеся локальными для данной функции.

По умолчанию инструкцию **CREATE FUNCTION** могут использовать только члены предопределенной роли сервера sysadmin и предопределенной роли базы данных db\_owner или db\_ddladmin. Но члены этих ролей могут присвоить это право другим пользователям с помощью инструкции **GRANT CREATE FUNCTION**. Роли пользователей и права доступа будут рассмотрены в другой лабораторной работе.

Например, создадим функцию, которая будет возвращать количество заказов, оформленных определенным клиентом (по его Id).

CREATE FUNCTION GetOrdersCountByCustomerId(@customerId INT)

RETURNS INT

BEGIN

DECLARE @sum INT

SELECT @sum = COUNT(\*) FROM Orders

where CustomerId = @customerId

RETURN @sum

END;

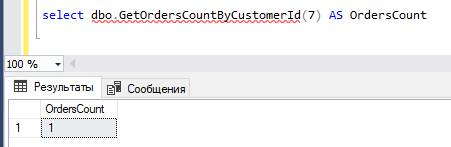
На вход функции подается Id покупателя, в теле функции происходит подсчет количества заказов, сделанных данным покупателем и с помощью команды RETURN возвращается результат.

Определенную пользователем функцию можно вызывать с помощью инструкций Transact-SQL, таких как SELECT, INSERT, UPDATE или DELETE. Вызов функции осуществляется, указывая ее имя с парой круглых скобок в конце, в которых можно задать один или несколько аргументов. Аргументы — это значения или выражения, которые передаются входным параметрам, определяемым сразу же после имени функции. При вызове функции, когда для ее параметров не определены значения по умолчанию, для всех этих параметров необходимо предоставить аргументы в том же самом порядке, в каком эти параметры определены в инструкции CREATE FUNCTION.

В примере ниже показан вызов функции GetOrdersCountByCustomerId в инструкции SELECT:

select dbo.GetOrdersCountByCustomerId(7) AS OrdersCount

В результате будет показано количество заказов, сделанных покупателем с Id = 7.



Функция может возвращать табличное значение, т.е. некоторый набор строк. В зависимости от того, каким образом определено тело функции, возвращающие табличное значение функции классифицируются как ***встраиваемые (inline)*** и ***многоинструкционные (multistatement).*** Если в предложении RETURNS ключевое слово TABLE указывается без сопровождающего списка столбцов, такая функция является встроенной. Инструкция SELECT встраиваемой функции возвращает результирующий набор в виде переменной с типом данных TABLE.

Многоинструкционная возвращающая табличное значение функция содержит имя, определяющее внутреннюю переменную с типом данных TABLE. Этот тип данных указывается ключевым словом TABLE, которое следует за именем переменной. В эту переменную вставляются выбранные строки, и она служит возвращаемым значением функции.

Рассмотрим создание возвращающей табличное значение функции, которая будет возвращать информацию о заказах, сделанных за определенный период:

CREATE FUNCTION GetOrdersByPeriod (@startDate datetime, @endDate datetime)

RETURNS TABLE

AS RETURN (SELECT Orders.[Date], Customers.[Name] AS CustomerName,

Smartphones.[Name] AS SmartphoneName, Smartphones.Manufacturer, Smartphones.Price

FROM Customers

JOIN Orders on Customers.Id = Orders.CustomerId

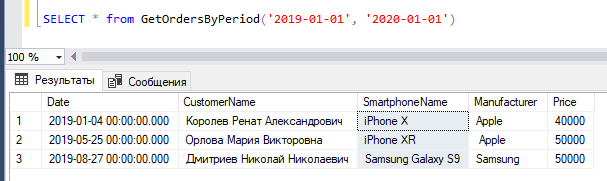
JOIN Smartphones on Orders.SmartphoneId = Smartphones.Id

WHERE Orders.[Date] BETWEEN @startDate and @endDate)

Данная функция принимает на вход два аргумента: дата начала периода, за который мы хотим получить заказы, и дату конца периода.

Для вызова данной функции можно выполнить следующий запрос:

SELECT \* from GetOrdersByPeriod('2019-01-01', '2020-01-01')



Т.к. функция возвращает табличное значение, то и обращаемся мы к ней, как к таблице, используя команду SELECT \* FROM …

**Задание к лабораторной работе**

1. Создайте две хранимые процедуры с входными параметрами и командами SELECT, INSERT/UPDATE/DELETE (выбрать как минимум 2), которые необходимы Вам в разрабатываемой информационной системе (курсовой работе). Аргументируйте свой выбор.
2. Создайте две функции (первая будет возвращать скалярное значение, а вторая - табличное), которые необходимы Вам в разрабатываемой информационной системе (курсовой работе). Аргументируйте свой выбор.

# **Лабораторная работа №13. Триггеры**

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

Формальное определение триггера:

CREATE TRIGGER имя\_триггера

ON {имя\_таблицы | имя\_представления}

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

AS выражения\_sql

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

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

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

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

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

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

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

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

Создадим триггер. Допустим, у нас есть база данных с таблицей *Smartphones*:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

Допустим, в таблице *Smartphones* хранятся данные о товарах. Но цена товара нередко содержит различные надбавки типа налога на добавленную стоимость, таможенная пошлина и так далее. Человек, добавляющий данные, может не знать все эти тонкости с налоговой базой, и он определяет чистую цену. С помощью триггера мы можем поправить цену товара на некоторую величину. Определим триггер, который будет срабатывать при добавлении и обновлении данных:

CREATE TRIGGER Smartphones\_INSERT\_UPDATE

ON Smartphones

AFTER INSERT, UPDATE

AS

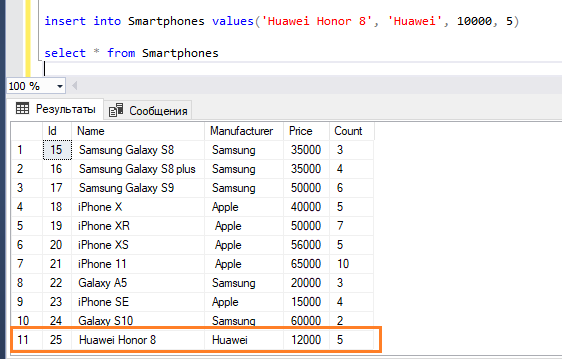
UPDATE Smartphones

SET Price = Price + Price \* 0.2

WHERE Id = (SELECT Id FROM inserted)

Таким образом, триггер будет срабатывать при любой операции INSERT или UPDATE над таблицей *Smartphones*. Сам триггер будет изменять цену смартфона, а для получения того смартфона, который был добавлен или изменен, находим этот смартфона по Id. Но какое значение должен иметь Id такой товар? Дело в том, что при добавлении или изменении данные сохраняются в промежуточную таблицу ***inserted***. Она создается автоматически. И из нее мы можем получить данные о добавленных/измененных смартфонах.

И после добавления смартфона в таблицу *Smartphones* в реальности смартфон будет иметь несколько большую цену, чем та, которая была определена при добавлении:



Для удаления триггера необходимо применить команду DROP TRIGGER:

DROP TRIGGER Smartphones\_INSERT\_UPDATE

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

DISABLE TRIGGER Smartphones\_INSERT\_UPDATE ON Smartphones

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

ENABLE TRIGGER Smartphones\_INSERT\_UPDATE ON Smartphones

**Триггеры для операций INSERT, UPDATE, DELETE**

Для рассмотрения операций с триггерами определим следующую базу данных:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

CREATE TABLE History

(Id INT IDENTITY PRIMARY KEY,

SmartphoneId INT NOT NULL,

Operation NVARCHAR(200) NOT NULL,

Date DATETIME NOT NULL DEFAULT GETDATE());

Здесь определены две таблицы: *Smartphones* - для хранения смартфонов и *History* - для хранения истории операций со смартфонами.

**Добавление**

При добавлении данных (при выполнении команды INSERT) в триггере мы можем получить добавленные данные из виртуальной таблицы INSERTED.

Определим триггер, который будет срабатывать после добавления:

CREATE TRIGGER Smartphones\_INSERT

ON Smartphones

AFTER INSERT

AS

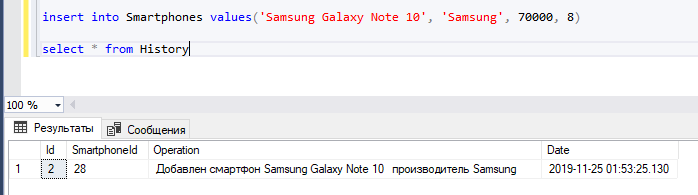
INSERT INTO History (SmartphoneId, Operation)

SELECT Id, N'Добавлен смартфон N' + [Name] + ' производитель ' + Manufacturer

FROM INSERTED

Этот триггер будет добавлять в таблицу *History* данные о добавлении смартфона, которые берутся из виртуальной таблицы INSERTED.

Выполним добавление данных в *Smartphones* и получим данные из таблицы *History*:



**Удаление**

При удалении все удаленные данные помещаются в виртуальную таблицу DELETED:

CREATE TRIGGER Smartphones\_DELETE

ON Smartphones

AFTER DELETE

AS

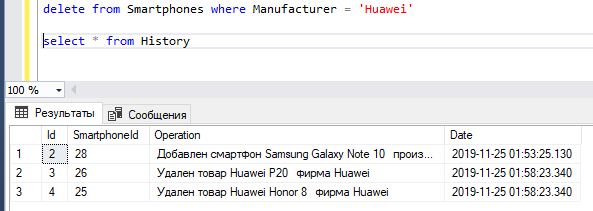
INSERT INTO History (SmartphoneId, Operation)

SELECT Id, N'Удален товар ' + [Name] + N' фирма ' + Manufacturer

FROM DELETED

Здесь, как и в случае с предыдущим триггером, помещаем информацию об удаленных смартфонах в таблицу History.

Выполним команду на удаление:



**Изменение данных**

Триггер обновления данных срабатывает при выполнении операции UPDATE. И в таком триггере мы можем использовать две виртуальных таблицы. Таблица INSERTED хранит значения строк после обновления, а таблица DELETED хранит те же строки, но до обновления.

Создадим триггер обновления:

CREATE TRIGGER Smartphones\_UPDATE

ON Smartphones

AFTER UPDATE

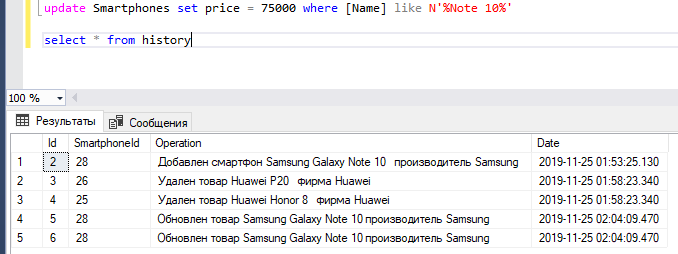
AS

INSERT INTO History (SmartphoneId, Operation)

SELECT Id, N'Обновлен товар ' + [Name] + N' производитель ' + Manufacturer

FROM INSERTED

И при обновлении данных сработает данный триггер:



**Триггер INSTEAD OF**

Триггер INSTEAD OF срабатывает вместо операции с данными. Он определяется также, как триггер AFTER, за тем исключением, что он может определяться только для одной операции - INSERT, DELETE или UPDATE. И также он может применяться как для таблиц, так и для представлений (триггер AFTER применяется только для таблиц).

Например, рассмотрим следующую таблицу:

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT,

IsDeleted bool)

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT,

IsDeleted BIT)

Здесь таблица содержит столбец *IsDeleted*, который указывает, удалена ли запись. То есть вместо жесткого удаления полностью из базы данных мы хотим выполнить мягкое удаление, при котором запись остается в базе данных.

Определим триггер для удаления записи:

CREATE TRIGGER smartphones\_delete

ON Smartphones

INSTEAD OF DELETE

AS

UPDATE Smartphones

SET IsDeleted = 1

WHERE ID =(SELECT Id FROM deleted)

Добавим некоторые данные в таблицу и выполним удаление из нее:

INSERT INTO Smartphones(Name, Manufacturer, Price)

VALUES ('iPhone X', 'Apple', 50000),

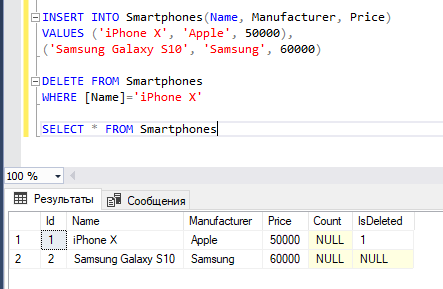
('Samsung Galaxy S10', 'Samsung', 60000)

DELETE FROM Smartphones

WHERE [Name]='iPhone X'

SELECT \* FROM Smartphones

Таким образом, удаляемые записи на самом деле не будут удаляться, просто у них будет устанавливаться значение для столбца IsDeleted:



**Задание к лабораторной работе**

1. Используя триггеры, реализуйте логирование действий пользователя (добавление, изменение, удаление) на основных таблицах в разрабатываемой Вами информационной системе (курсовой работе). Во время фиксирования изменений укажите информацию о том, какие данные были и какие стали.
2. Создайте триггер типа INSTEAD OF для любой таблицы, исходя из разрабатываемой Вами информационной системы (курсовой работы).

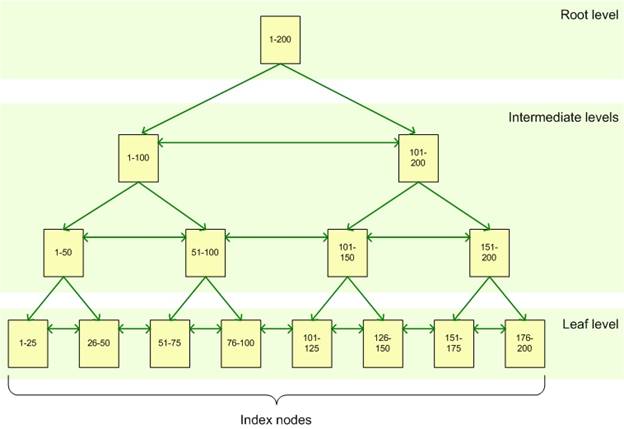
# **Лабораторная работа №14. Индексы.**

Одним из важнейших путей достижения высокой производительности SQL Server является использование индексов. Индекс ускоряет процесс запроса, предоставляя быстрый доступ к строкам данных в таблице, аналогично тому, как указатель в книге помогает вам быстро найти необходимую информацию.

Индексы создаются для столбцов таблиц и представлений. Индексы предоставляют путь для быстрого поиска данных на основе значений в этих столбцах. Например, если вы создадите индекс по первичному ключу, а затем будете искать строку с данными, используя значения первичного ключа, то SQL Server сначала найдет значение индекса, а затем использует индекс для быстрого нахождения всей строки с данными. Без индекса будет выполнен полный просмотр (сканирование) всех строк таблицы, что может оказать значительное влияние на производительность.

Вы можете создать индекс на большинстве столбцов таблицы или представления. Исключением, преимущественно, являются столбцы с типами данных для хранения больших объектов (LOB), таких как image, text или varchar(max).

Индекс состоит из набора страниц, узлов индекса, которые организованы в виде древовидной структуры — сбалансированного дерева. Эта структура является иерархической по своей природе и начинается с корневого узла на вершине иерархии и конечных узлов, листьев, в нижней части, как показано на рисунке:



Когда вы формируете запрос на индексированный столбец, подсистема запросов начинает идти сверху от корневого узла и постепенно двигается вниз через промежуточные узлы, при этом каждый слой промежуточного уровня содержит более детальную информацию о данных. Подсистема запросов продолжает двигаться по узлам индекса до тех пор, пока не достигнет нижнего уровня с листьями индекса. К примеру, если вы ищете значение 123 в индексированном столбе, то подсистема запросов сначала на корневом уровне определит страницу на первом промежуточном (intermediate) уровне. В данном случае первой страница указывает на значение от 1 до 100, а вторая от 101 до 200, таким образом подсистема запросов обратится ко второй странице этого промежуточного уровня. Далее будет выяснено, что следует обратиться к третьей странице следующего промежуточного уровня. Отсюда подсистема запросов прочитает на нижнем уровне значение самого индекса. Листья индекса могут содержать как сами данные таблицы, так и просто указатель на строки с данными в таблице, в зависимости от типа индекса: ***кластеризованный индекс*** или ***некластеризованный***.

***Кластеризованный индекс*** хранит реальные строки данных в листьях индекса. Возвращаясь к предыдущему примеру, это означает что строка данных, связанная со значение ключа, равного 123 будет храниться в самом индексе. Важной характеристикой кластеризованного индекса является то, что все значения отсортированы в определенном порядке либо возрастания, либо убывания. Таким образом, таблица или представление может иметь только один кластеризованный индекс. В дополнение следует отметить, что данные в таблице хранятся в отсортированном виде только в случае, если у этой таблицы создан кластеризованный индекс.

Кластеризованный индекс создается автоматически для таблиц, у которых есть первичный ключ (PRIMARY KEY).

Рассмотрим пример, создадим таблицу *Смартфоны:*

CREATE TABLE Smartphones

(Id INT PRIMARY KEY IDENTITY,

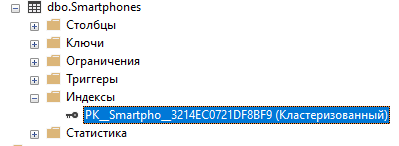
[Name] NVARCHAR(100),

Manufacturer NVARCHAR(100),

Price INT,

[Count] INT)

И в MS SQL Management Studio откроем информацию о таблице -> Индексы, можно убедиться, что кластеризованный индекс был создан автоматически. Это значит, что все данные в таблице хранятся в отсортированном виде по полю Id.



Если таблица не имеет первичного ключа, то кластеризованный индекс можно создать с помощью следующей команды:

CREATE UNIQUE CLUSTERED INDEX название\_индекса ON название\_таблицы

(

Имя\_столбца1 порядок\_сортировки (ASC/DESC),

Имя\_столбца2 порядок\_сортировки (ASC/DESC),

...

)

Представим, что в таблице *Смартфоны* нет первичного ключа. Добавим кластеризованный индекс:

CREATE UNIQUE CLUSTERED INDEX IX\_SmartphoneId ON Smartphones

(

Id ASC

)

Обратите внимание, что таблица не может иметь более одного кластеризованного индекса.

Таблица не имеющая кластеризованного индекса называется ***кучей***.

В отличие от кластеризованного индекса, листья ***некластеризованного индекса*** содержат только те столбцы (ключевые), по которым определен данный индекс, а также содержит указатель на строки с реальными данными в таблице. Это означает, что системе подзапросов необходима дополнительная операция для обнаружения и получения требуемых данных. Содержание указателя на данные зависит от способа хранения данных: кластеризованная таблица или куча. Если указатель ссылается на кластеризованную таблицу, то он ведет к кластеризованному индексу, используя который можно найти реальные данные. Если указатель ссылается на кучу, то он ведет к конкретному идентификатору строки с данными. Некластеризованные индексы не могут быть отсортированы в отличие от кластеризованных, однако вы можете создать более одного некластеризованного индекса на таблице или представлении, вплоть до 999. Это не означает, что вы должны создавать как можно больше индексов. Индексы могут как улучшить, так и ухудшить производительность системы.

В общем виде создание некластеризованного индекса выглядит следующим образом:

CREATE NONCLUSTERED INDEX название\_индекса

ON название\_таблицы(столбец1, столбец2, ...)

Например, создадим индекс для таблицы *Smartphones*, который буде включать столбец *Name.*

CREATE NONCLUSTERED INDEX IX\_SmartphoneName

ON Smartphones(Name)

Также можно создать индекс для нескольких столбцов одновременно:

CREATE NONCLUSTERED INDEX IX\_SmartphoneName

ON Smartphones(Name, Count)

В дополнение к возможности создать несколько некластеризованных индексов, вы можете также включить дополнительные столбцы (included column) в свой индекс, который не будут являться ключевыми: на листьях индекса будет храниться не только значение самих индексированных столбцов, но и значения этих не индексированных дополнительных столбцов. Этот подход позволит вам обойти некоторые ограничения, наложенные на индекс. К примеру, вы можете включить неидексируемый столбец или обойти ограничение на длину индекса (900 байт в большинстве случаев), также благодаря покрытию запроса повышается производительность, так как оптимизатор запросов может найти все значения столбцов в индексе, при этом не обращаясь к данным таблиц, что приводит к меньшему числу дисковых операций ввода-вывода. Но помните, что включение в индекс неключевых столбцов влечет за собой увеличение размера индекса, т.е. для хранения индекса потребуется больше места на диске, а также может повлечь и снижение производительности операций INSERT, UPDATE, DELETE и MERGE на базовой таблице.

В общем виде создание индекса с включенными столбцами выглядит следующим образом:

CREATE NONCLUSTERED INDEX название\_индекса

ON название\_таблицы(столбец1, столбец2, ...)

INCLUDE (столбец1, столбец2, ...)

Включим в индекс атрибут *Manufacturer.*

CREATE NONCLUSTERED INDEX IX\_SmartphoneNameCount

ON Smartphones([Name], [Count])

INCLUDE (Manufacturer)

Удаление индекса в общем виде выглядит следующим образом:

drop index название\_индекса ON название\_таблицы

Например, удалим созданный ранее индекс:

drop index IX\_SmartphoneName ON Smartphones

**Типы индексов**

В дополнение к тому, что индекс может быть либо кластеризованным, либо некластеризованным, возможно его дополнительно сконфигурировать как составной индекс, уникальный индекс или покрывающий индекс.

**Составной индекс** - может содержать более одного столбца. Вы можете включить до 16 столбцов в индекс, но их общая длина ограничена 900 байтами. Как кластеризованный, так и некластеризованный индексы могут быть составными.

**Уникальный индекс**

Такой индекс обеспечивает уникальность каждого значения в индексируемом столбце. Если индекс составной, то уникальность распространяется на все столбцы индекса, но не на каждый отдельный столбец. К примеру, если вы создадите уникальныq индекс на столбцах ИМЯ и ФАМИЛИЯ, то полное имя должно быть уникально, но отдельно возможны дубли в имени или фамилии.

Уникальный индекс автоматически создается, когда вы определяете ограничения столбца: первичный ключ или ограничение на уникальность значений:

*Первичный ключ*

Когда вы определяете ограничение первичного ключа на один или несколько столбцов, тогда SQL Server автоматически создаёт уникальный кластеризованный индекс, если кластеризованный индекс не был создан ранее (в этом случае создается уникальный некластеризованный индекс по первичному ключу).

*Уникальность значений*

Когда вы определяете ограничение на уникальность значений, тогда SQL Server автоматически создает уникальный некластеризованный индекс. Вы можете указать, чтобы был создан уникальный кластеризованный индекс, если кластеризованного индекса до сих пор не было создано на таблице.

**Покрывающий индекс** позволяет конкретному запросу сразу получить все необходимые данные с листьев индекса без дополнительных обращений к записям самой таблицы.

**Проектирование индексов**

Насколько полезны индексы могут быть, настолько аккуратно они должны быть спроектированы. Поскольку индексы могут занимать значительное дисковое пространство, вы не захотите создавать индексов больше, чем необходимо. В дополнение, индексы автоматически обновляются, когда сама строка с данными обновляется, что может привести к дополнительным накладным расходам ресурсов и падению производительности. При проектировании индексов должно приниматься во внимание несколько соображений относительно базы данных и запросов к ней.

Как было отмечено ранее индексы могут улучшить производительность системы, т.к. они обеспечивают подсистему запросов быстрым путем для нахождения данных. Однако, вы должны также принять во внимание то, как часто вы собираетесь вставлять, обновлять или удалять данные. Когда вы изменяете данные, то индексы должны также быть изменены, чтобы отразить соответствующие действия над данными, что может значительно снизить производительность системы. Рассмотрим следующие рекомендации при планировании стратегии индексирования:

* Для таблиц, которые часто обновляются используйте как можно меньше индексов.
* Если таблица содержит большое количество данных, но их изменения незначительны, тогда используйте столько индексов, сколько необходимо для улучшения производительности ваших запросов. Однако хорошо подумайте перед использованием индексов на небольших таблицах, т.к. возможно использование поиска по индексу может занять больше времени, нежели простое сканирование всех строк.
* Для кластеризованных индексов старайтесь использовать настолько короткие поля насколько это возможно. Наилучшим образом будет применение кластеризованного индекса на столбцах с уникальными значениями и не позволяющими использовать NULL. Вот почему первичный ключ часто используется как кластеризованный индекс.
* Уникальность значений в столбце влияет на производительность индекса. В общем случае, чем больше у вас дубликатов в столбце, тем хуже работает индекс. С другой стороны, чем больше уникальных значений, тем выше работоспособность индекса. Когда возможно используйте уникальный индекс.
* Для составного индекса возьмите во внимание порядок столбцов в индексе. Столбцы, которые используются в выражениях WHERE (к примеру, WHERE FirstName = 'Charlie') должны быть в индексе первыми. Последующие столбцы должны быть перечислены с учетом уникальности их значений (столбцы с самым высоким количеством уникальных значений идут первыми).
* Также можно указать индекс на вычисляемых столбцах, если они соответствуют некоторым требованиям. К примеру, выражение которые используются для получения значения столбца, должны быть детерминистическими (всегда возвращать один и тот же результат для заданного набора входных параметров).

Другое соображение, которое следует учитывать при проектировании индексов это какие запросы выполняются к базе данных. Как было указано ранее, вы должны учитывать, как часто изменяются данные. Дополнительно следует использовать следующие принципы:

* Старайтесь вставлять или модифицировать в одном запросе как можно больше строк, а не делать это в несколько одиночных запросов.
* Создайте некластеризованный индекс на столбцах, которые часто используются в ваших запросах в качестве условий поиска в WHERE и соединения в JOIN.
* Рассмотрите возможность индексирования столбцов, использующихся в запросах поиска строк на точное соответствие значений.

**Задание к лабораторной работе**

Проанализируйте Вашу базу данных. Создайте 3 индекса разных видов, аргументируйте свой выбор.

# **Лабораторная работа №15. Транзакции**

***Транзакция*** — это группа инструкций одной или нескольких баз данных, которые либо полностью фиксируются, либо полностью откатываются. Транзакции атомарны, согласованы, изолированы и устойчивы (atomic, consistent, isolated, durable — ACID). Если транзакция выполнена успешно, все инструкции в ней фиксируются. Если хотя бы одна инструкция в группе завершается ошибкой, выполняется откат всей группы.

Начало и конец транзакции зависят от параметра AUTOCOMMIT и инструкций BEGIN TRANSACTION, COMMIT и ROLLBACK. Хранилище данных SQL поддерживает следующие типы транзакций:

* Явные транзакции начинаются с инструкции BEGIN TRANSACTION и заканчиваются инструкцией COMMIT или ROLLBACK.
* Транзакции с автофиксацией автоматически запускаются в рамках сеанса и не начинаются с инструкции BEGIN TRANSACTION. Если для параметра AUTOCOMMIT установлено значение ON, каждая инструкция выполняется в транзакции, и явные инструкции COMMIT или ROLLBACK не требуются. Если для параметра AUTOCOMMIT установлено значение OFF, для определения результата транзакции требуется инструкция COMMIT или ROLLBACK. В Хранилище данных SQL транзакции с автофиксацией начинаются сразу после инструкции COMMIT или ROLLBACK или после инструкции SET AUTOCOMMIT OFF.

Транзакции имеют следующий синтаксис:

BEGIN TRANSACTION [;]

COMMIT [ TRAN | TRANSACTION | WORK ] [;]

ROLLBACK [ TRAN | TRANSACTION | WORK ] [;]

SET AUTOCOMMIT { ON | OFF } [;]

SET IMPLICIT\_TRANSACTIONS { ON | OFF } [;]

**BEGIN TRANSACTION**

Отмечает начальную точку явной транзакции.

**COMMIT [ WORK ]**

Отмечает завершение явной транзакции или транзакции с автофиксацией. Эта инструкция вызывает изменения в транзакции, чтобы всегда быть зафиксированной в базе данных. Инструкция COMMIT идентична инструкциям COMMIT WORK, COMMIT TRAN и COMMIT TRANSACTION.

**ROLLBACK [ WORK ]**

Выполняет откат транзакции на начало транзакции. Никакие изменения транзакции не фиксируются в базе данных. Инструкция ROLLBACK идентична инструкциям ROLLBACK WORK, ROLLBACK TRAN и ROLLBACK TRANSACTION.

**SET AUTOCOMMIT { ON | OFF }**

Определяет метод запуска и завершения транзакций.

**ON**

Каждая инструкция выполняется в своей транзакции, явные инструкции COMMIT или ROLLBACK не требуются. Явные транзакции разрешены, когда для параметра AUTOCOMMIT установлено значение ON.

**OFF**

Хранилище данных SQL автоматически запускает транзакцию, если транзакция уже не выполняется. Все последующие инструкции выполняются в рамках транзакции, и инструкции COMMIT или ROLLBACK необходимы для определения результата транзакции. Как только транзакция фиксируется или откатывается в этом режиме, значение OFF сохраняется, а Хранилище данных SQL запускает новую транзакцию. Явные транзакции не разрешены, если AUTOCOMMIT имеет значение OFF.

Если изменить параметр AUTOCOMMIT в активной транзакции, этот параметр не повлияет на текущую транзакцию и вступит в силу только после завершения транзакции.

Если для параметра AUTOCOMMIT установлено значение ON, выполнение другой инструкции SET AUTOCOMMIT ON не будет иметь результата. Подобным образом, если для параметра AUTOCOMMIT установлено значение OFF, выполнение другой инструкции SET AUTOCOMMIT OFF не будет иметь результата.

**SET IMPLICIT\_TRANSACTIONS { ON | OFF }**

Включает те же режимы, что и SET AUTOCOMMIT. Присвоение параметру SET IMPLICIT\_TRANSACTIONS значения ON устанавливает для соединения режим неявных транзакций. Значение OFF возвращает подключение в режим автофиксации.

Если выполнить инструкции COMMIT или ROLLBACK без активной транзакции, возникает ошибка.

Если выполнить инструкцию BEGIN TRANSACTION во время выполнения транзакции, возникает ошибка. Это может произойти, если инструкция BEGIN TRANSACTION выполняется после успешного запуска инструкции BEGIN TRANSACTION или для сеанса установлено SET AUTOCOMMIT OFF.

Если ошибка делает невозможным успешное выполнение транзакции, хранилище данных SQL автоматически выполняет ее откат и освобождает ресурсы, удерживаемые транзакцией. Это не относится к ошибкам во время выполнения инструкции. Например, если сетевое подключение клиента к экземпляру компонента Хранилище данных SQL разорвано или клиент выходит из приложения, то после того, как экземпляр получит уведомление от сети о разрыве подключения, выполняется откат всех незафиксированных транзакций для этого подключения.

Сеанс может одновременно выполнять только одну транзакцию. Точки сохранения и вложенные транзакции не поддерживаются.

Обязанностью программиста на языке SQL является вызов инструкции COMMIT только в том случае, когда все данные, относящиеся к транзакции, логически верны.

Если сеанс закрывается до завершения транзакции, транзакция откатывается.

Управление режимами транзакций выполняется на уровне сеанса. Например, если один сеанс запускает явную транзакцию или устанавливает для параметра AUTOCOMMIT значение OFF или для параметра IMPLICIT\_TRANSACTIONS значение ON, это не влияет на режимы транзакции в других сеансах.

Нельзя произвести откат транзакции после вызова инструкции COMMIT, так как измененные данные уже стали частью базы данных.

**Режим блокировки**

Хранилище данных SQL использует блокировку для гарантии целостности транзакций и поддержания согласованности баз данных, когда несколько пользователей обращаются к одним и тем же данным в одно и то же время. Блокировка используется в явных и неявных транзакциях. Каждая транзакция запрашивает блокировку разных типов ресурсов, например таблиц или баз данных, от которых эта транзакция зависит. Все блокировки Хранилище данных SQL выполняются на уровне таблиц или выше. Блокировка не дает другим транзакциям изменять ресурсы, чтобы избежать ошибок в транзакции, запросившей блокировку. Каждая транзакция снимает свои блокировки, если больше не зависит от заблокированных ресурсов. Явные транзакции сохраняют блокировки до завершения транзакции — ее фиксации или отката.

Рассмотрим пример. Допустим у нас есть база данных для банка с таблицами Clients, History.

create table Clients(

Id INT IDENTITY PRIMARY KEY,

FirstName nvarchar(100),

LastName nvarchar(200),

AccountNumber nvarchar(50),

[Sum] INT)

create table history (

Id INT IDENTITY PRIMARY KEY,

FromClientId INT References Clients(Id),

ToClientId INT References Clients(Id),

[Sum] INT,

TransactionDate datetime)

Во время денежного перевода от одного клиента другому необходимо выполнить ряд действий: вычесть сумму перевода из суммы счета первого клиента, прибавить сумму перевода к сумме счета второго клиента, сделать запись в таблицу *History* об операции. Но что если во время выполнения этих операций возникнет ошибка и, например, сумма спишется со счета первого клиента, но из-за произошедшей ошибки не зачислится на счет второго клиента? Для исключения таких ошибок следует использовать транзакции.

begin transaction

declare @sum int

set @sum = 1000

declare @senderId int

declare @receiverId int

select @senderId = id from Clients where AccountNumber = '5463 6572 9844 76912'

select @receiverId = id from Clients where AccountNumber = '1234 7655 0087 1221'

update Clients set [Sum] = [Sum] - @sum where id = @senderId

update Clients set [Sum] = [Sum] + @sum where id = @receiverId

insert into history values(@senderId, @receiverId, @sum, GETDATE())

COMMIT

В данном скрипте команда **begin transaction –** объявляет начало транзакции, затем прописываются все команды, которые необходимо выполнить и завершается все это командой **COMMIT.**

Если во время выполнения операций произойдет ошибка, то все изменения, которые уже были сделаны, будут отменены.

Если вместо операции **COMMIT** написать **ROLLBACK,** то все выполненные операции будут отменены, даже при их успешном выполнении.

**Задание к лабораторной работе**

Создайте транзакцию для разрабатываемой информационной системы.

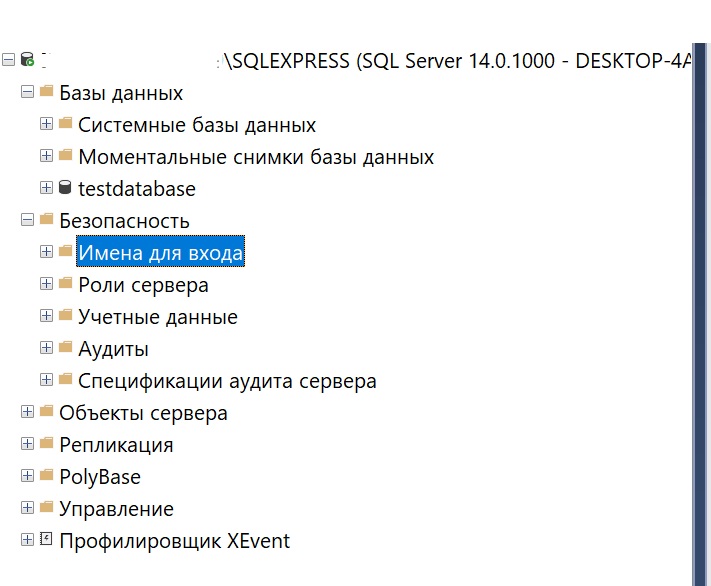
# **Лабораторная работа №16. Пользователи. Разграничение прав доступа.**

Пользователь может войти в систему баз данных, используя учетную запись пользователя Windows или регистрационное имя входа в SQL Server. Для последующего доступа и работы с определенной базой данных пользователь также должен иметь учетную запись пользователя базы данных. Для работы с каждой отдельной базой данных требуется иметь учетную запись пользователя именно для этой базы данных. Учетную запись пользователя базы данных можно сопоставить с существующей учетной записью пользователя Windows, группой Windows (в которой пользователь имеет членство), регистрационным именем или ролью.

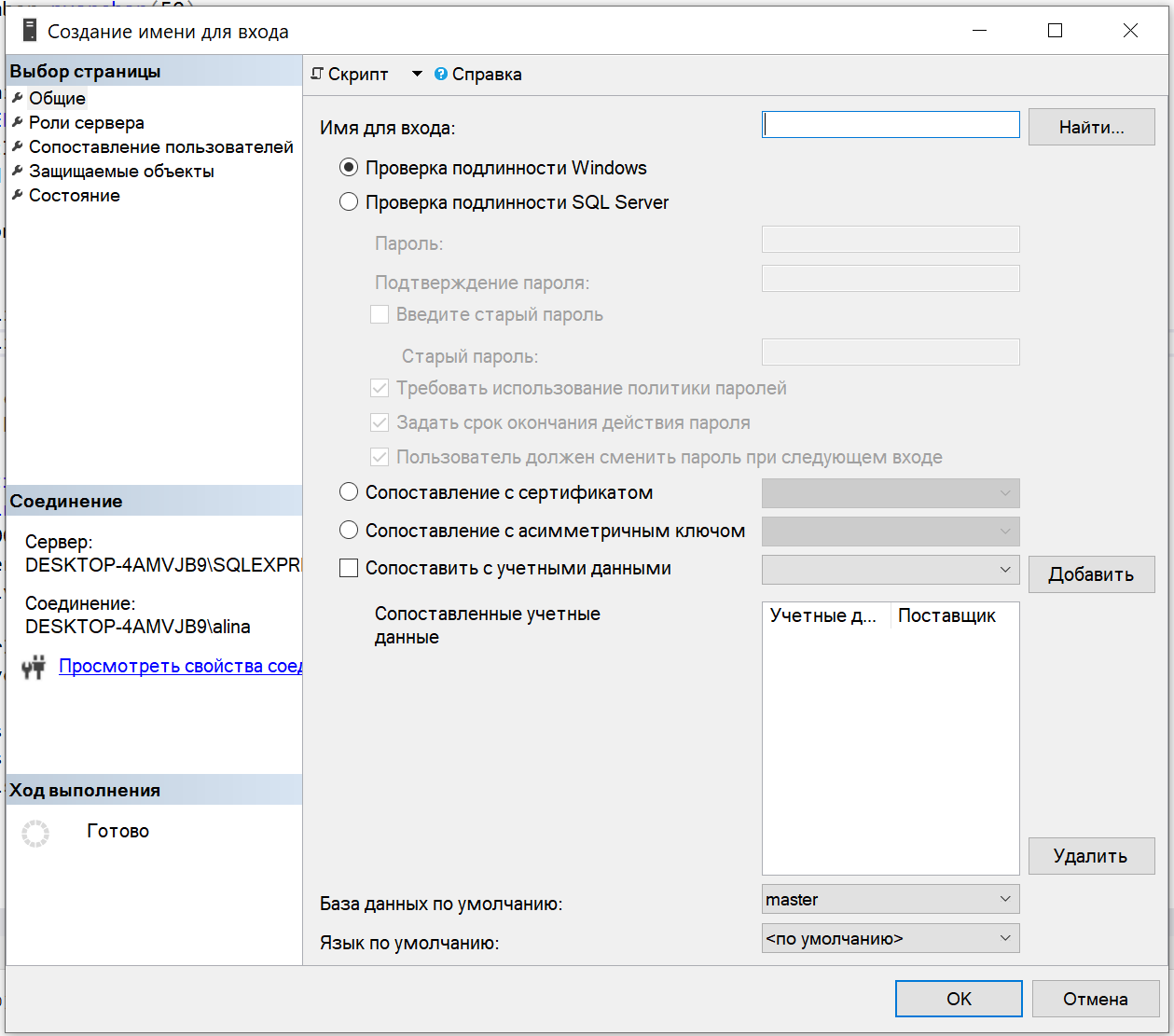
Управлять пользователями баз данных можно с помощью среды Management Studio или инструкций языка Transact-SQL.

**SQL Management Studio**

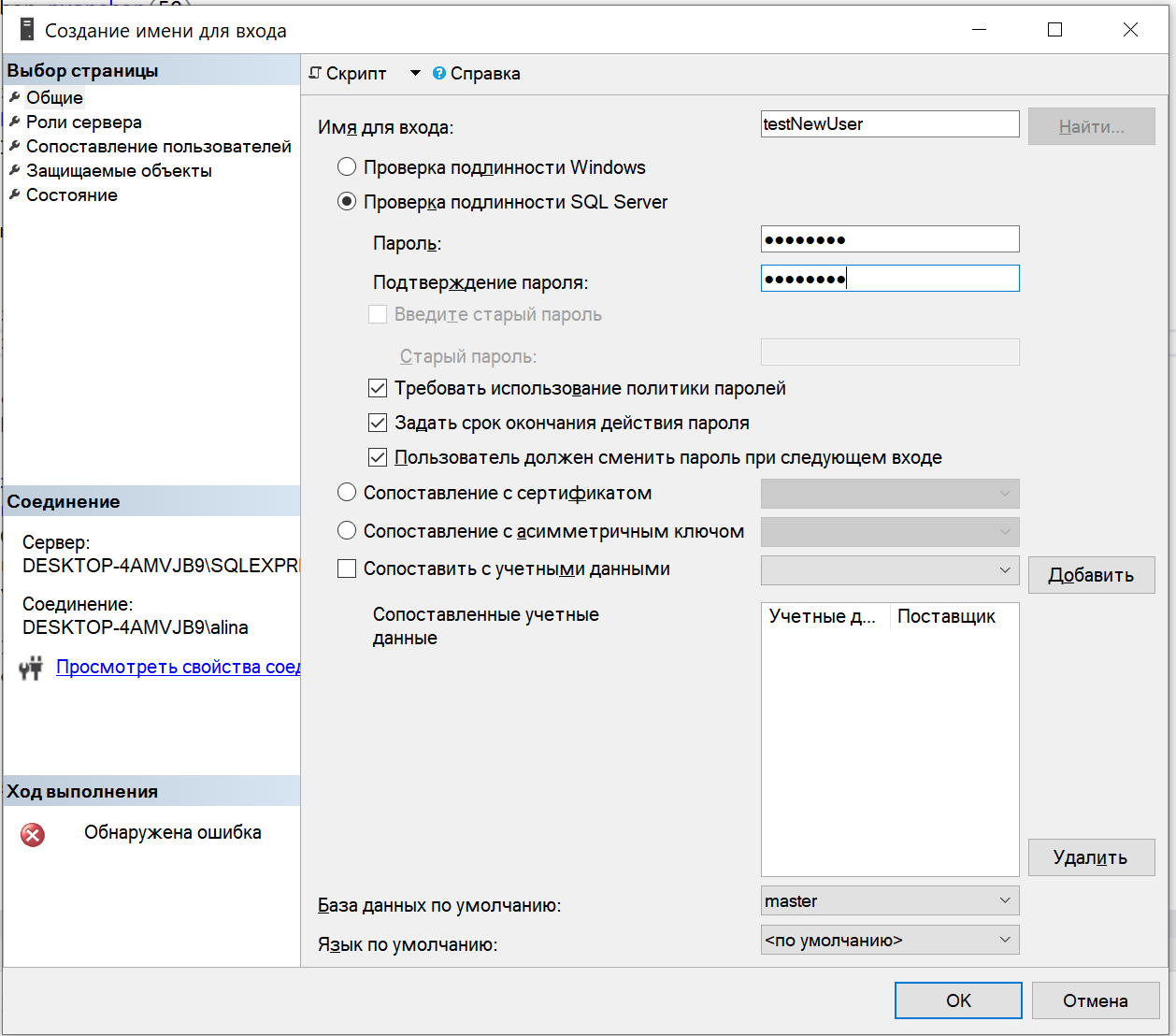
Для добавления нового пользователя необходимо сначала добавить новое имя для входа. Перейдите в **Обозреватель объектов -> Безопасность -> Имена для входа.**



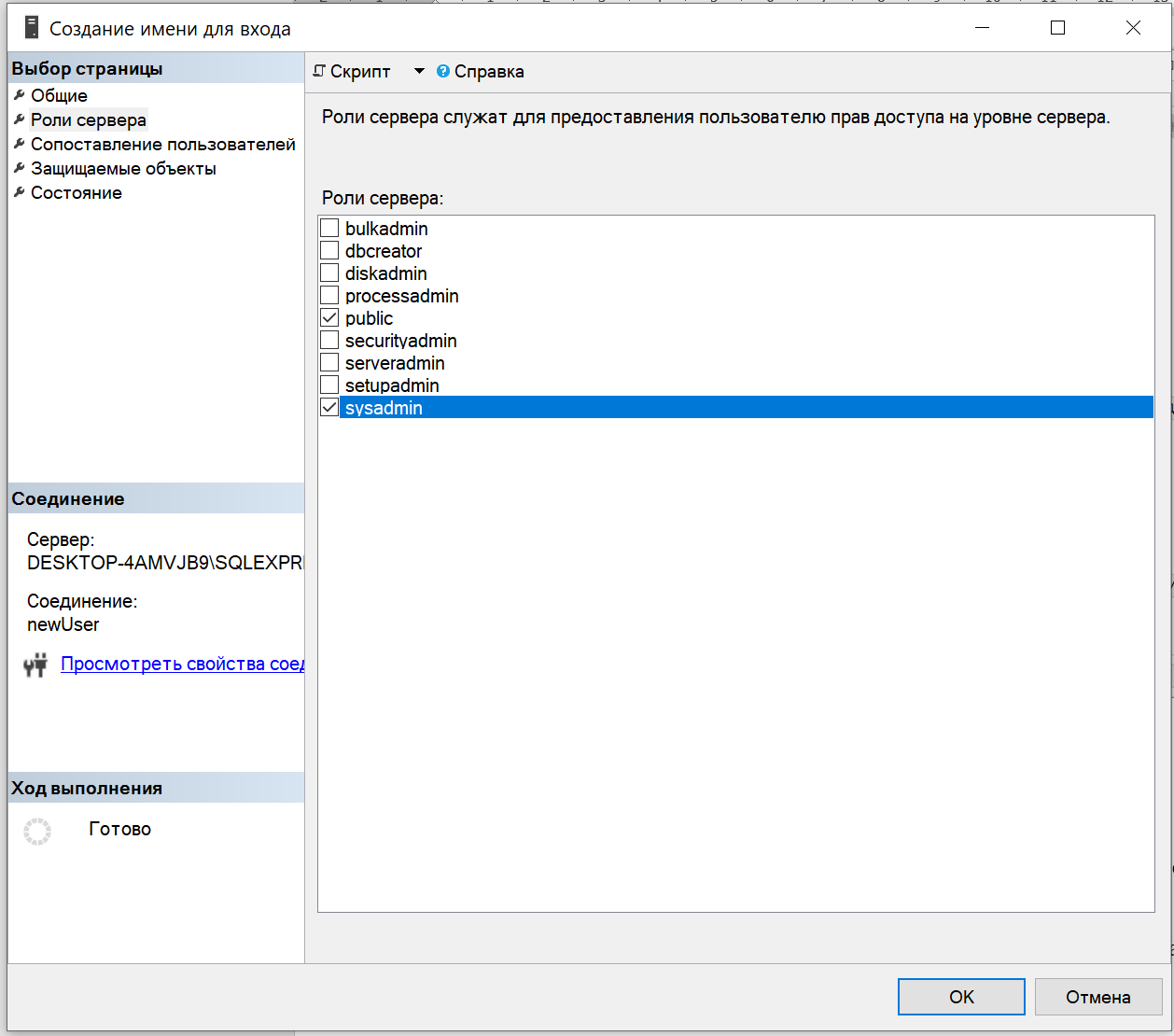
В контекстном меню выбрать **Создать имя для входа.**



Выбираем **Проверка подлинности SQL Server**, вводим имя для входа, пароль.



В разделе **Роли сервера** необходимо выбрать роли.



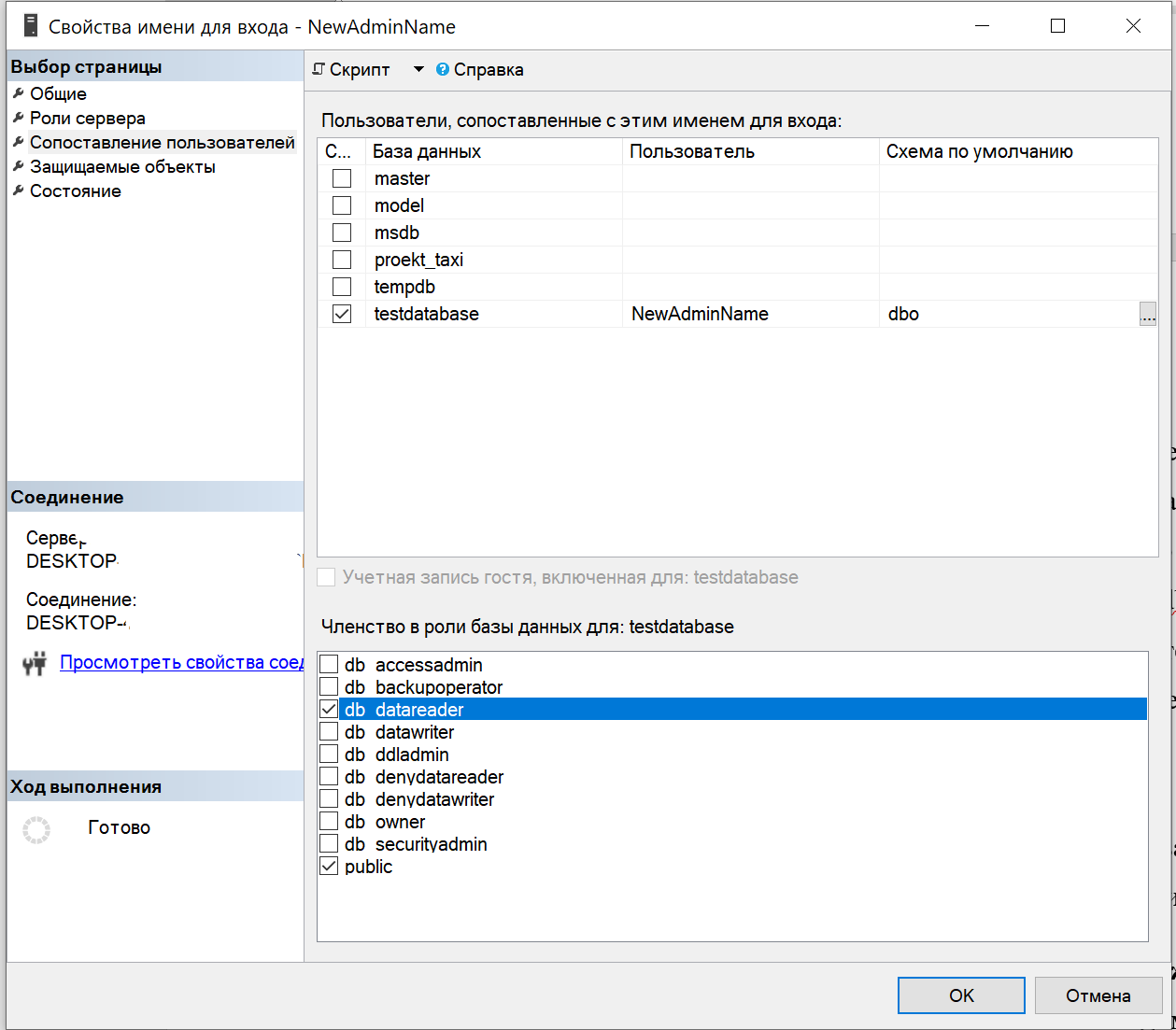
В данном разделе представлены фиксированные серверные роли.

**Фиксированные серверные роли**

Фиксированные серверные роли определяются на уровне сервера и поэтому находятся вне баз данных, принадлежащих серверу баз данных. В таблице ниже приводится список фиксированных серверных ролей и краткое описание действий, которые могут выполнять члены этих ролей:

|  |  |
| --- | --- |
| **Фиксированная серверная роль** | **Описание** |
| sysadmin | Выполняет любые действия в системе баз данных |
| serveradmin | Конфигурирует параметры сервера |
| setupadmin | Устанавливает репликацию и управляет расширенными процедурами |
| securityadmin | Управляет регистрационными именами и разрешениями для инструкции CREATE DATABASE и чтением журналов логов |
| processadmin | Управляет системными процессами |
| dbcreator | Создает и модифицирует базы данных |
| diskadmin | Управляет файлами на диске |
| bulkadmin | Элементы предопределенной роли сервера bulkadmin могут выполнять инструкцию BULK INSERT |
| public | Каждое имя входа SQL Server принадлежит к роли сервера public. Если для участника на уровне сервера не были предоставлены или запрещены конкретные разрешения на защищаемый объект, он наследует разрешения роли public на этот объект. Разрешения роли public следует назначать только тому объекту, который будет доступен всем пользователям. Нельзя изменить членство в роли public.  **Примечание**. Роль public реализуется не так, как другие роли; разрешения могут предоставляться, запрещаться или отменяться из открытых предопределенных ролей сервера. |

После выбора ролей необходимо перейти в раздел **Сопоставление пользователей.** Здесь происходит настройка того, к каким объектам данное имя входа будет иметь доступ и с какими правами. Например, выберем, что текущее имя входа будет иметь доступ только к базе данных *testdatabase* и роль db\_datareader. Таким образом, данные будут доступны только для чтения. Пользователь, использующий данное имя входа, не сможет создать, удалить или изменить данные в базе.

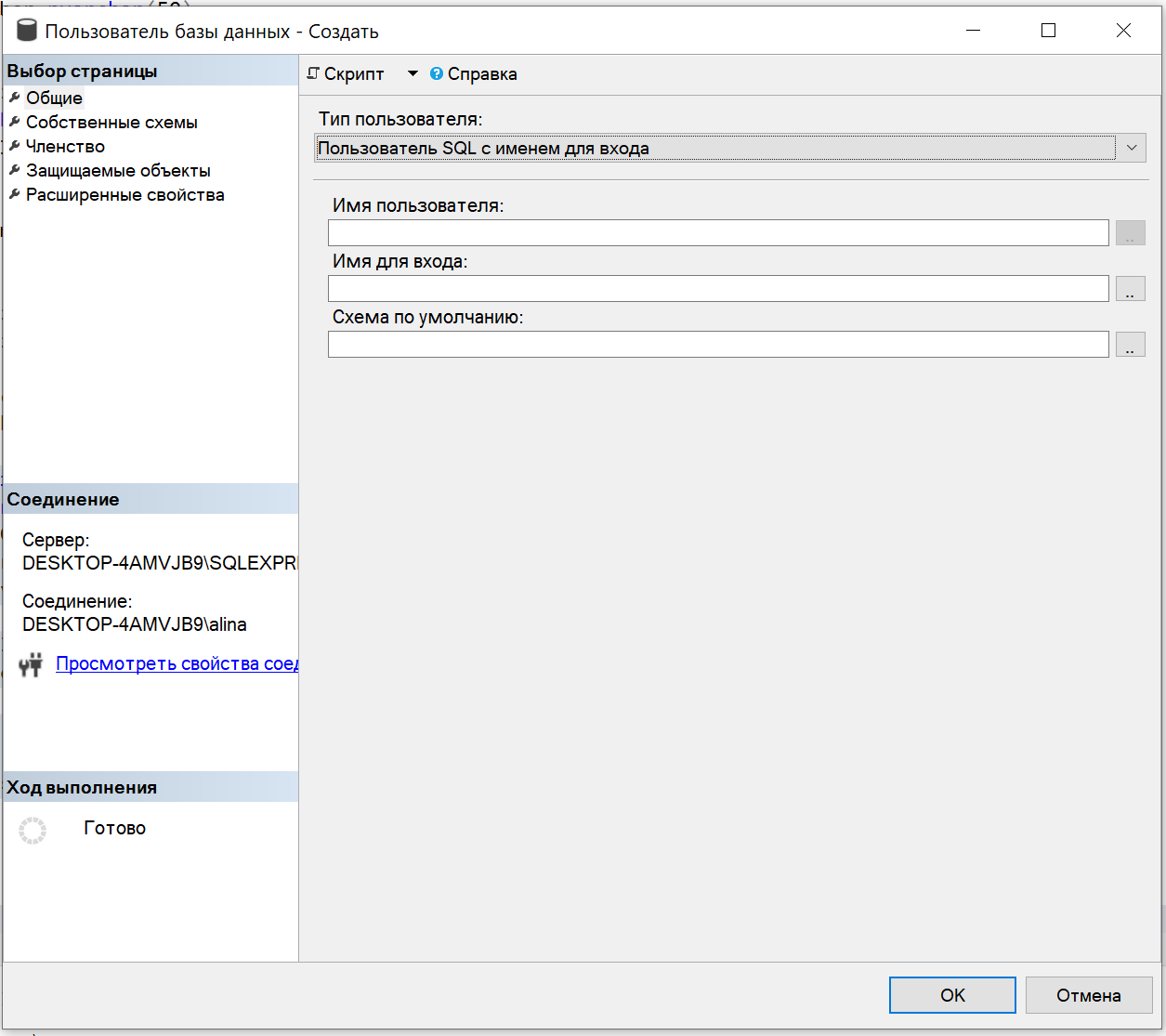


В следующей таблице представлены предопределенные роли базы данных и их возможности. Эти роли существуют во всех базах данных. За исключением открытой роли базы данных разрешения, назначенные предопределенным ролям базы данных изменять нельзя.

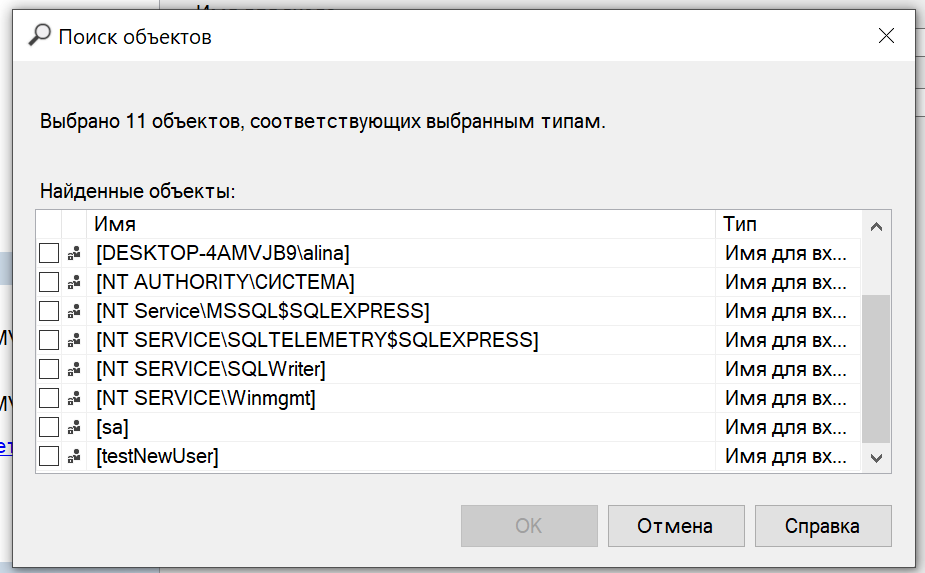
|  |  |
| --- | --- |
| **Имя предопределенной роли базы данных** | **Описание** |
| **db\_owner** | Члены предопределенной роли базы данных **db\_owner** могут выполнять все действия по настройке и обслуживанию базы данных, а также удалять базу данных в SQL Server. (В База данных SQL и Хранилище данных SQL некоторые операции по обслуживанию требуют наличие разрешений на уровне сервера и не может быть выполнены членами **db\_owner**.) |
| **db\_securityadmin** | Элементы предопределенной роли базы данных **db\_securityadmin** могут изменять членство в роли (только для настраиваемых ролей) и управлять разрешениями. Элементы этой роли потенциально могут повышать свои права доступа, поэтому необходимо отслеживать их действия. |
| **db\_accessadmin** | Члены предопределенной роли базы данных **db\_accessadmin** могут добавлять или удалять права удаленного доступа к базе данных для имен входа и групп Windows, а также имен входа SQL Server. |
| **db\_backupoperator** | Члены предопределенной роли базы данных **db\_backupoperator** могут создавать резервные копии базы данных. |
| **db\_ddladmin** | Члены предопределенной роли базы данных **db\_ddladmin** могут выполнять любые команды языка определения данных (DDL) в базе данных. |
| **db\_datawriter** | Члены предопределенной роли базы данных **db\_datawriter** могут добавлять, удалять или изменять данные во всех пользовательских таблицах. |
| **db\_datareader** | Элементы предопределенной роли базы данных **db\_datareader** могут считывать все данные из всех пользовательских таблиц. |
| **db\_denydatawriter** | Члены предопределенной роли базы данных **db\_denydatawriter** не могут добавлять, изменять или удалять данные в пользовательских таблицах базы данных. |
| **db\_denydatareader** | Члены предопределенной роли базы данных **db\_denydatareader** не могут считывать данные из пользовательских таблиц базы данных. |

После создания имена входа теперь, чтобы добавить пользователя базы данных с помощью среды Management Studio перейдите по следующему пути:

**Обозреватель объектов -> Базы данных -> [Выбираем нужную базу данных] -> Безопасность -> Пользователи.** В контекстном меню необходимо выбрать **Создать пользователя.** Откроется диалоговое окно, в котором следует ввести имя пользователя User name и выбрать соответствующее регистрационное имя (Имя для входа/Login name), которое было создано на предыдущем этапе.



Выбираем имя для входа, которое создали на предыдущем этапе.



После успешного создания пользователя можно подключиться к серверу, используя новые данные.

**Transact SQL**

Рассмотрим добавления нового имена входа и пользователя средствами Transact SQL.

В общем виде, создание имени входа выглядит следующим образом:

-- Syntax for SQL Server

CREATE LOGIN login\_name { WITH <option\_list1> | FROM <sources> }

<option\_list1> ::=

PASSWORD = { 'password' | hashed\_password HASHED } [ MUST\_CHANGE ]

[ , <option\_list2> [ ,... ] ]

<option\_list2> ::=

SID = sid

| DEFAULT\_DATABASE = database

| DEFAULT\_LANGUAGE = language

| CHECK\_EXPIRATION = { ON | OFF}

| CHECK\_POLICY = { ON | OFF}

| CREDENTIAL = credential\_name

<sources> ::=

WINDOWS [ WITH <windows\_options>[ ,... ] ]

| CERTIFICATE certname

| ASYMMETRIC KEY asym\_key\_name

<windows\_options> ::=

DEFAULT\_DATABASE = database

| DEFAULT\_LANGUAGE = language

Создадим новое имя входа:

CREATE LOGIN NewAdminName WITH PASSWORD = 'ABCD'

Для добавления пользователя средствами Transact SQL используется инструкция CREATE USER. Синтаксис этой инструкции выглядит следующим образом:

CREATE USER user\_name

[FOR {LOGIN login |CERTIFICATE cert\_name | ASYMMETRIC KEY key\_name}]

[WITH DEFAULT\_SCHEMA = schema\_name]

Параметр user\_name определяет имя, по которому пользователь идентифицируется в базе данных, а в параметре login указывается регистрационное имя, для которого создается данный пользователь. В параметрах cert\_name и key\_name указываются соответствующий сертификат и асимметричный ключ соответственно. Наконец, в параметре WITH DEFAULT\_SCHEMA указывается первая схема, с которой сервер базы данных будет начинать поиск для разрешения имен объектов для данного пользователя базы данных.

Применение инструкции CREATE USER показано в примере ниже:

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = N'NewAdminName')

BEGIN

CREATE USER [NewAdminName] FOR LOGIN [NewAdminName]

EXEC sp\_addrolemember N'db\_owner', N'NewAdminName'

END;

Операция **IF NOT EXIST** применяется для того, чтобы избежать ошибок и не пытаться повторно создать такого пользователя, если он уже был создан. Такая операция часто применяется и при создании других объектов базы данных (процедур, триггеров, индексов и т.д.)

**Регистрационное имя sa**

Регистрационное имя sa является регистрационным именем системного администратора. В версиях более ранних, чем SQL Server 2005, в которых роли отсутствовали, регистрационное имя sa предоставляло все возможные разрешения для задач системного администрирования. В более новых версиях SQL Server регистрационное имя sa включено единственно с целью обратной совместимости. Это регистрационное имя всегда является членом фиксированной серверной роли sysadmin и его нельзя удалить из этой роли.

Регистрационное имя sa следует использовать только в тех случаях, когда нет другого способа войти в систему базы данных.

**Задание к лабораторной работе**

Создайте трех пользователей с разными правами доступа в соответствии с разрабатываемой информационной системой.