

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ**

**СРЕДНЕПРОФЕССИОНАЛЬНЫЙ КОЛЛЕДЖ**

З.С. Шовкарова

# **МДК.07.01.УПРАВЛЕНИЕ И АВТОМАТИЗАЦИЯ БАЗ ДАННЫХ**

**ПРАКТИКУМ**

для обучающихся IV курса специальности  
09.02.07 Информационные системы и программирование

Черкесск, 2023

УДК 004.65  
ББК 16.35  
Ш78

Рассмотрено на заседании ЦК «Информационные и естественнонаучные дисциплины».

Протокол № 1 от 31. 08. 2022 г.

Рекомендовано к изданию редакционно-издательским советом СКГА

Протокол № 24 от 26.09.2022 г.

**Рецензенты:** Черных Л.А. –председатель ЦК «Информационные и естественнонаучные дисциплины»

**Ш78 Шовкарова, З.С..** МДК.07.01. Управление и автоматизация баз данных: практикум для обучающихся IV курса специальности 09.02.07 Информационные системы и программирование / З.С. Шовкарова. – Черкесск: БИЦ СКГА, 2023. – 84с.

Практикум содержит теоретический и практический материал, изучение которого позволит студенту освоить основные понятия и получить практические знания при работе с СУБД SQL Server.

**УДК 004.65**  
**ББК 16.35**

## СОДЕРЖАНИЕ

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 1. Построение схемы данных	4
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 2 Составление словаря данных	9
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 3 Разработка технических требований к серверу баз данных	23
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 4 Разработка требований к корпоративной сети	27
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 5 Конфигурирование сети	27
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 6 Сравнение технических характеристик серверов	32
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 7 Формирование аппаратных требований и схемы банка данных	38
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 8 Установка и настройка сервера MySQL. Установка и настройка сервера.	43
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 9 Выполнение запросов к базе данных	49
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 10 Выполнение изменений в базе данных, создание триггеров	63
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 11 Создание запросов и процедура изменение структуры базы данных	67
ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 12 Работа с журналом аудита и базы данных. Мониторинг нагрузки сервера	80

## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 1. Построение схемы базы данных

**Цель:** получить навыки построения схемы данных.

**Ход работы:**

Создание диаграммы базы данных.

При работе БД должна обеспечиваться целостность данных. Под целостностью данных понимают обеспечения целостности связей между записями в таблицах при удалении записей из первичных таблиц. То есть, при удалении записей из первичных таблиц автоматически должны удаляться связанные с ними записи из вторичных таблиц.

В случае несоблюдения целостности данных со временем в БД накопится большое количество записей во вторичных таблицах, связанных с несуществующими записями в первичных таблицах, что приведёт к сбоям в работе БД и её засорению неиспользуемыми данными.

Внешний ключ – это столбец (или комбинация столбцов), совпадающий с первичным ключом не которой таблицы. Если значение внешнего ключа соответствует значению первичного ключа, становится понятно, что между объектами базы данных представленными совпадающими строками, существует логическое взаимоотношение.

Одним из главных ограничений отношения является ссылочная целостность, которая определяет, что каждое значение внешнего ключа, не являющееся NULL-значением, должно ссылаться (REFERENCES) на некоторое существующее значение первичного ключа.

Для обеспечения целостности данных в SQL Server используют диаграммы и триггеры. Диаграммы – это компоненты БД, которые блокируют удаление записей из первичных таблиц если существуют связанные с ними записи во вторичных таблицах. Следовательно, диаграммы предотвращают нарушение целостности данных.

В БД Microsoft SQL Server 20XX все диаграммы находятся в папке Диаграммы базы данных обозревателя объектов.

Создадим диаграмму, обеспечивающую целостность данных нашей БД «publishing».

Для создания новой диаграммы в БД «publishing» щёлкните правой кнопкой мыши по папке Диаграммы базы данных и в появившемся меню выберем пункт Новая диаграмма базы данных. Сначала появится окно с вопросом о добавлении нового объекта Диаграмма. В этом окне нужно нажать кнопку Да. Затем появится окно «Добавление таблицы» предназначенное для добавления таблиц в новую диаграмму (рисунок 1).

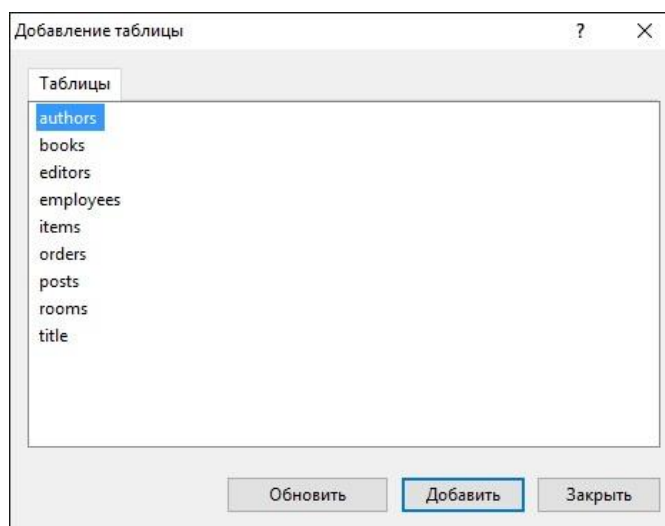


Рисунок 1– Окно Добавление таблицы

В окне добавления таблиц выделите все таблицы нашей БД и нажмите кнопку «Добавить». Закройте окно «Добавление таблицы» нажатием на кнопку «Закреть». Появится окно диаграммы, где будут отображены отобранные таблицы. Обратите внимание, что на данном этапе таблицы являются не связанными, т.е. отсутствует ограничение целостности по внешнему ключу.

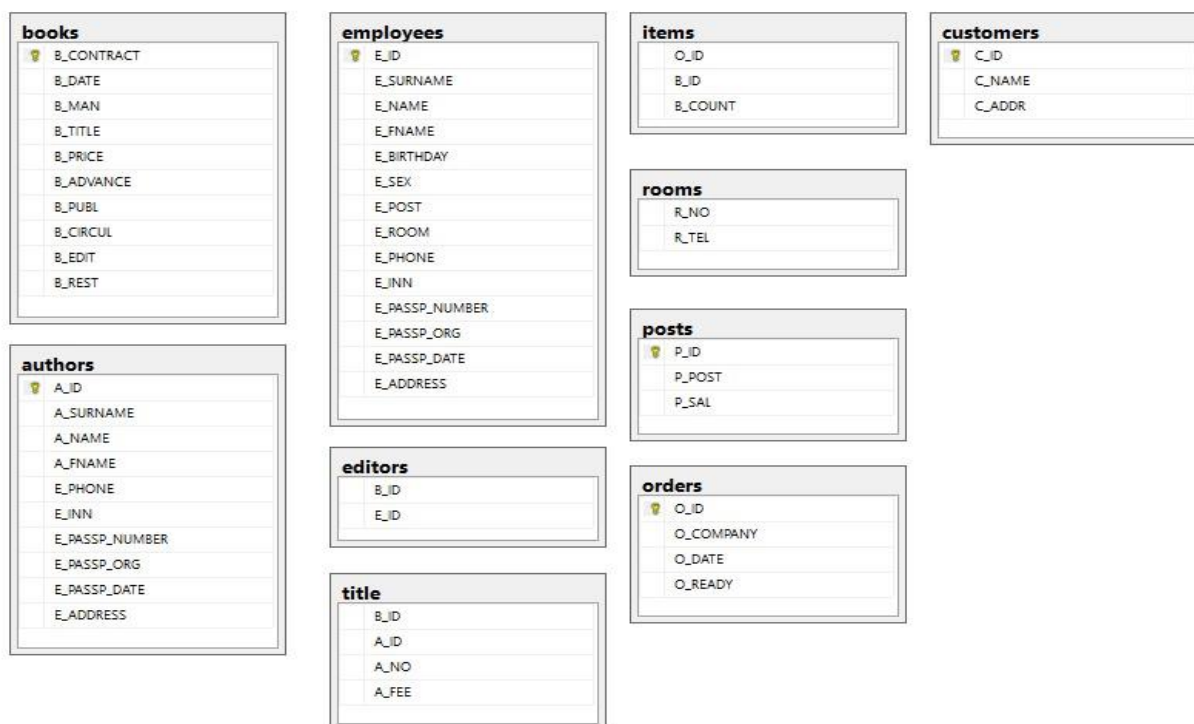


Рисунок 2 – Сущности на диаграмме

Теперь необходимо определить связи между таблицами. Давайте организуем связь между сотрудником и должностью. Перед этим уточним, что в случае если мы захотим удалить из БД должность, то необходимо

обнулить должность у сотрудника. Для этого необходимо перетащить поле P\_ID из таблицы Posts на такое же поле в таблице Employees. Появится окно создания связи между таблицами «Таблицы и столбцы» (рисунок 3).

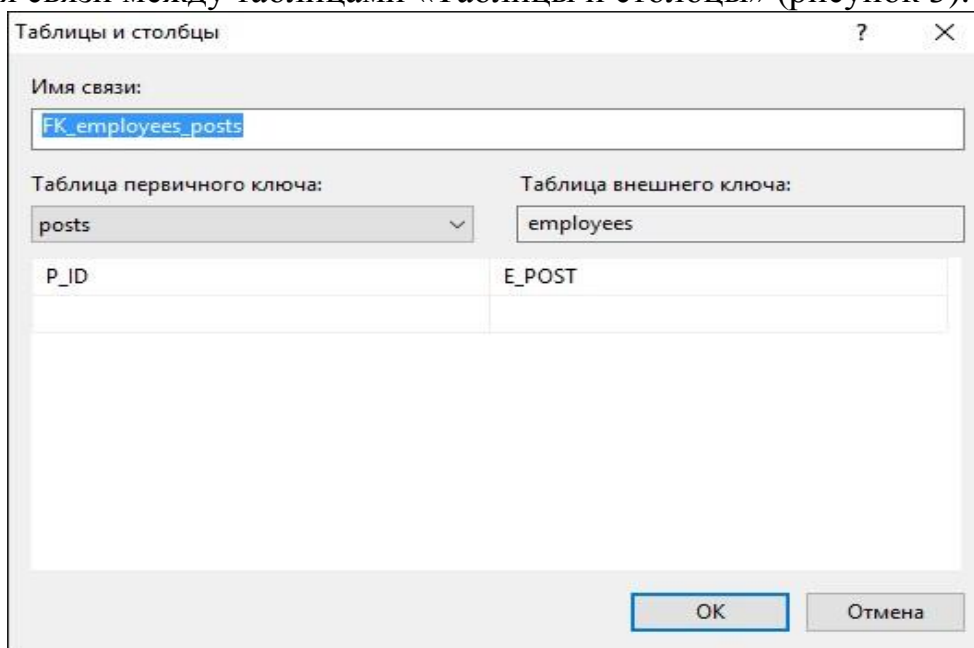


Рисунок 3 – Окно Таблицы и столбцы

В окне создания связи нажмите кнопку ОК. Появится окно настройки свойств связи **Связь по внешнему ключу** (рисунок 4).

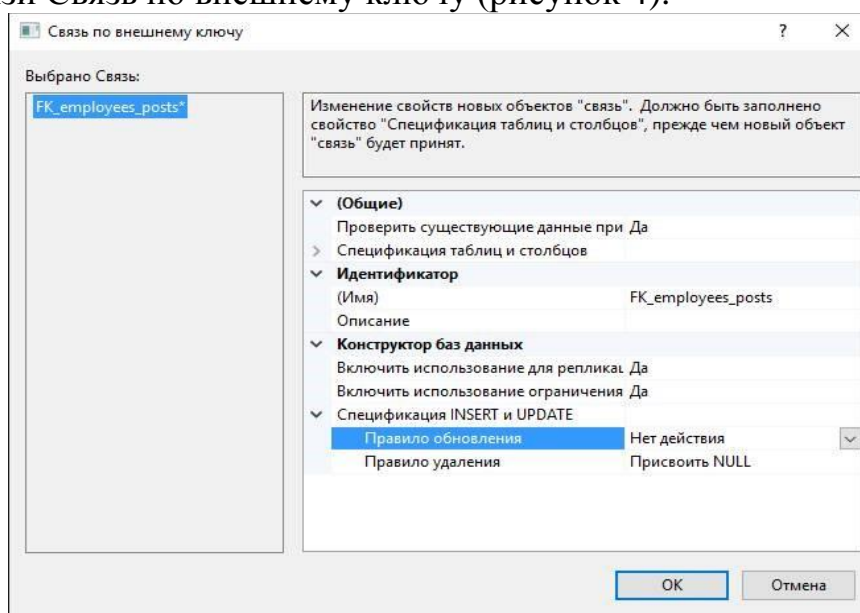


Рисунок 4 – Окно Связь по внешнему ключу

Спецификация INSERT и UPDATE определяет, что происходит, если значение первичного ключа, на которое ссылается внешний ключ, удаляется или обновляется:

– установить NULL (SET NULL) – значение внешнего ключа заменяется NULL-значением. Это невозможно, когда внешний ключ является частью первичного ключа своей таблицы.

– установить значение по умолчанию (SET DEFAULT) – значение внешнего ключа заменяется значением столбца, используемым по умолчанию.

– каскадно (CASCADE) – удаляются или обновляются все строки внешнего ключа.

– нет действия (NO ACTION) – после обновления нельзя модифицировать значения.

В диаграмме между таблицами employees и posts появится связь в виде линии (рисунок 5).

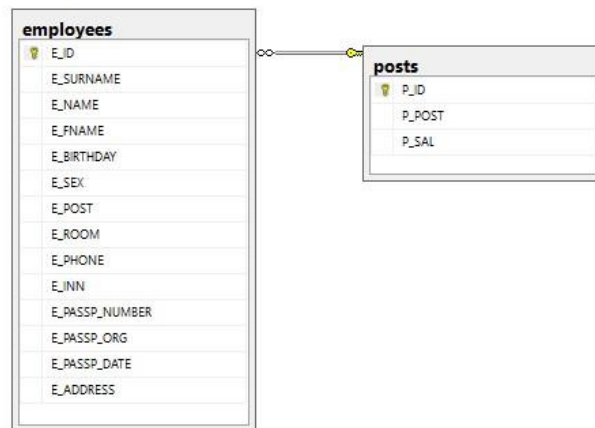


Рисунок 5 – Связь между таблицами employees и posts

Связывание таблиц с помощью конструкции FOREIGN KEY.

Для указания внешнего ключа таблицы нужно добавить конструкцию FOREIGN KEY в конструкцию CREATE TABLE:

– FOREIGN KEY <имя внешнего ключа> REFERENCES <таблица первичного ключа>

– ON UPDATE <операция обновления>

– ON DELETE <операция удаления>

Имена столбцов внешних ключей указываются после ключевых слов FOREIGN KEY. В конструкции REFERENCES содержится имя таблицы того первичного ключа, на который производится ссылка. Если столбцы первичного ключа заданы в конструкции PRIMARY KEY своей таблицы, нет необходимости перечислять их имена. Если же имена столбцов не являются частью конструкции PRIMARY KEY, необходимо перечислить столбцы первичного ключа в конструкции REFERENCES.

Тогда создание таблицы Employees с помощью запроса будет выглядеть следующим образом:

```
CREATE TABLE Employees
(
    [E_ID] [int] IDENTITY(1,1) NOT NULL, [E_SURNAME] [nvarchar](20)
NOT NULL,
    [E_NAME] [nvarchar](20) NOT NULL,
    [E_FNAME] [nvarchar](30) NULL,
```

[E\_BIRTHDAY] [date] NULL,  
 [E\_SEX] [bit] NULL,  
 [E\_POST] [int] NULL,  
 [E\_ROOM] [int] NULL,  
 [E\_PHONE] [nvarchar](10) NULL,  
 [E\_INN] [nvarchar](12) NOT NULL,  
 [E\_PASSP\_NUMBER] [nvarchar](12) NOT NULL,  
 [E\_PASSP\_ORG] [nvarchar](50) NOT NULL,  
 [E\_PASSP\_DATE] [date] NOT NULL,  
 [E\_ADDRESS] [nvarchar](80) NULL,  
 PRIMARY KEY (E\_ID),  
 FOREIGN KEY (P\_ID) REFERENCES Posts  
 ON UPDATE NO ACTION  
 ON DELETE SET NULL

);

Аналогично для всех остальных таблиц необходимо проделать те же самые действия. В итоге схема отношений будет выглядеть следующим образом.

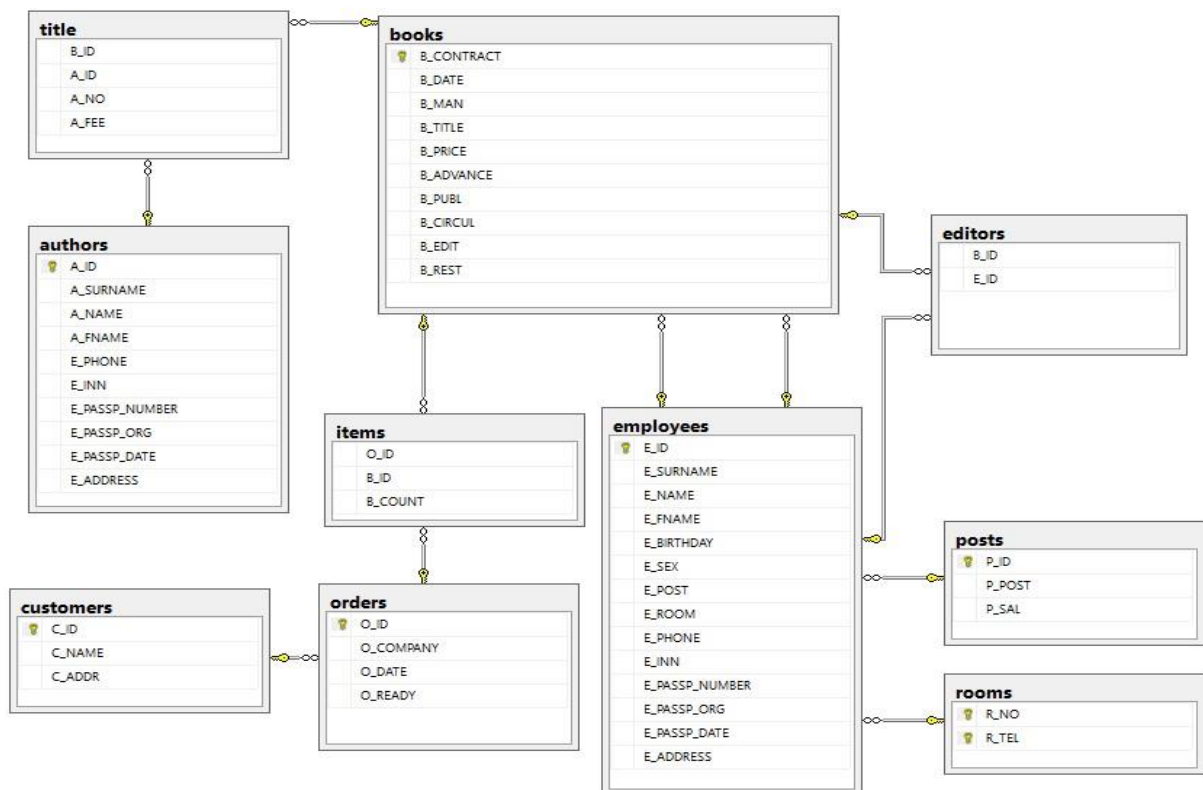


Рисунок 6 – Схема базы данных рассматриваемого примера

**Задание:** Разработать на основе индивидуальной базы данных схему данных.



## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 2. Составление словаря данных

**Цель:** приобрести навыки создания словаря данных по имеющейся информационной модели.

### **Ход работы:**

Создать словарь данных в формате Ms Excel по предлагаемой модели предметной области.

Определить предварительный объём дискового пространства сервера для информационной базы.

### *Краткие теоретические сведения.*

Словарь данных – это набор реляционных таблиц и представлений, которые содержат информацию о таблицах или метаданные (данные о данных).

Метаданные – описание типов сущностей и связей между ними.

Первичный ключ – это поле или набор полей со значениями, которые являются уникальными для всей таблицы. Значения ключа могут использоваться для обозначения всех записей, при этом каждая запись имеет отдельное значение ключа. Каждая таблица может содержать только один первичный ключ.

Внешний ключ (FK) – это столбец или сочетание столбцов, которое применяется для принудительного установления связи между данными в двух таблицах с целью контроля данных, которые могут храниться в таблице внешнего ключа.

Потенциальный ключ – подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и не сократимости (минимальности).

### Пример.

Дан фрагмент схемы базы данных:

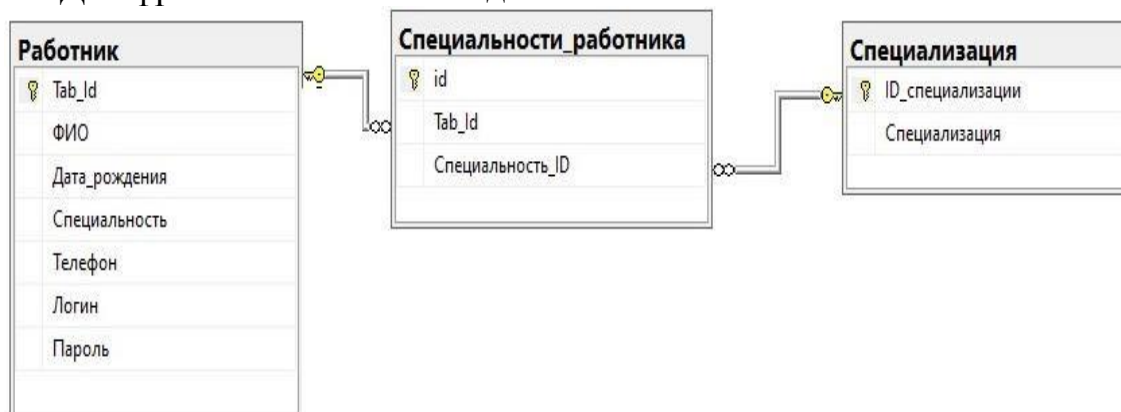


Рисунок 7- Фрагмент схемы базы

Даны SQL-запросы на создание этого фрагмента базы данных:

```
CREATE TABLE [Работник]
```

```
(
```

```

Tab_Id int IDENTITY(1,1) PRIMARY KEY NOT NULL,
ФИО varchar(70) NOT NULL,
Дата_рождения date NOT NULL,
Телефон varchar(15) UNIQUE NOT NULL, Логин varchar(16) UNIQUE
NOT NULL,
Пароль varchar(12) UNIQUE NOT NULL
)
CREATE TABLE [Специализация]
(
ID_специализации int IDENTITY(1,1) PRIMARY KEY NOT NULL,
Специализация varchar(40) NOT NULL
)
CREATE TABLE [Специальности_работника]
(
id int IDENTITY(1,1) PRIMARY KEY NOT NULL, Tab_Id int NOT
NULL,
Специальность_ID int NOT NULL
CONSTRAINT FK1 FOREIGN KEY (Tab_Id) REFERENCES [Работник]
(Tab_Id),
CONSTRAINT FK2 FOREIGN KEY (Специальность_ID) REFERENCES
[Специализация] (ID_специализации)
)

```

Задание: создать словарь данных по предлагаемому формату описания таблиц. Описание таблиц выполнить в файле формата Ms Excel по предлагаемому формату описания таблиц, каждая таблица на отдельном листе, файл сохранить под именем Словарь данных.xls в родной папке.

Таблица 1- Таблица «Работник»

Имя поля.	Тип поля	Длина поля (байт)	Описание поля	Ключ (первичный, потенциальный, внешний)	Пример данных поля
Tab_Id	int	4	Табельный номер работника	Первичный	49
ФИО	varchar(70)	72	ФИО работника		Хохловский Дмитрий Евгеньевич
Дата_рождения	date	3	Дата рождения работника		2003-06-18
Телефон	varchar(15)	17	Контактный телефон работника	Потенциальный	8-961-003-33-27

Логин	varchar(16)	18	Логин работника в системе	Потенциальны	hohlovsky2017
Пароль	varchar(12)	14	Пароль работника в системе	Потенциальн	qwerty2003
Длина одной записи:		128	Байт		
Предполагаемое число работников:		8			
Занятость дискового пространства:		1024	Байт		

Таблица 2- Таблица «Специализация»

Имя поля.	Тип поля	Длина поля (байт)	Описание поля	Ключ (первичный, потенциальный, внешний)	Пример данных поля
ID_специализации	int	4	Код специализации	Первичный	1
Специализация	varchar(40)	42	Названия специализации		Парикмахер
Длина одной записи:		46	Байт		
Предполагаемое число специализаций:		6			
Занятость дискового пространства:		276	Байт		

Таблица 3- Таблица «Специальности работника»

Имя поля.	Тип поля	Длина поля (байт)	Описание поля	Ключ (первичный, потенциальный, внешний)	Пример данных поля
id	int	4	Номер записи	Первичный	3
Tab_Id	int	4	Табельный номер работника	Внешний	49
Специальность_ID	int	4	Код специализации	Внешний	1

			и		
Длина одной записи:		8	Байт		
Максимальное число записей в таблице:		48			
Занятость дискового пространства:		384	Байт		

Для приблизительного определения занятости дискового пространства добавить дополнительный лист Сводная информация:

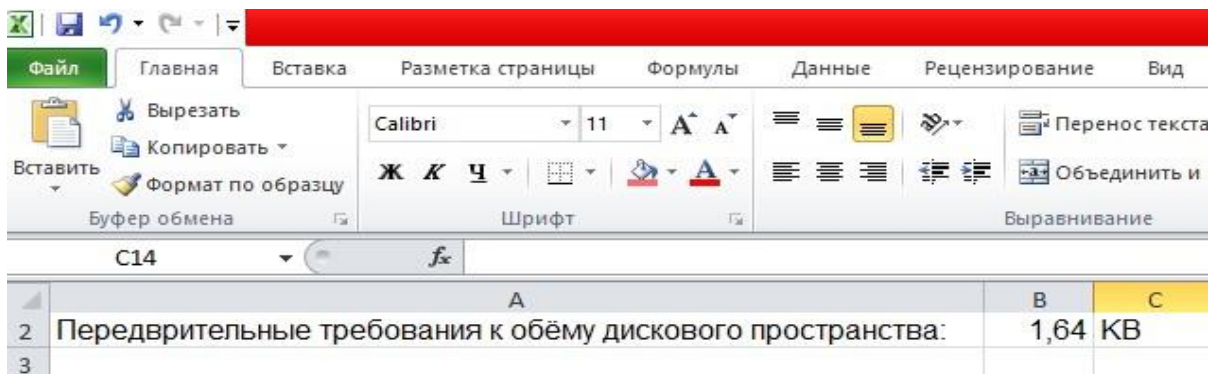


Рисунок 8- Лист Сводная информация

Вариант 1. Файл работы сохранить в родной папке под именем Вариант1.xlsx

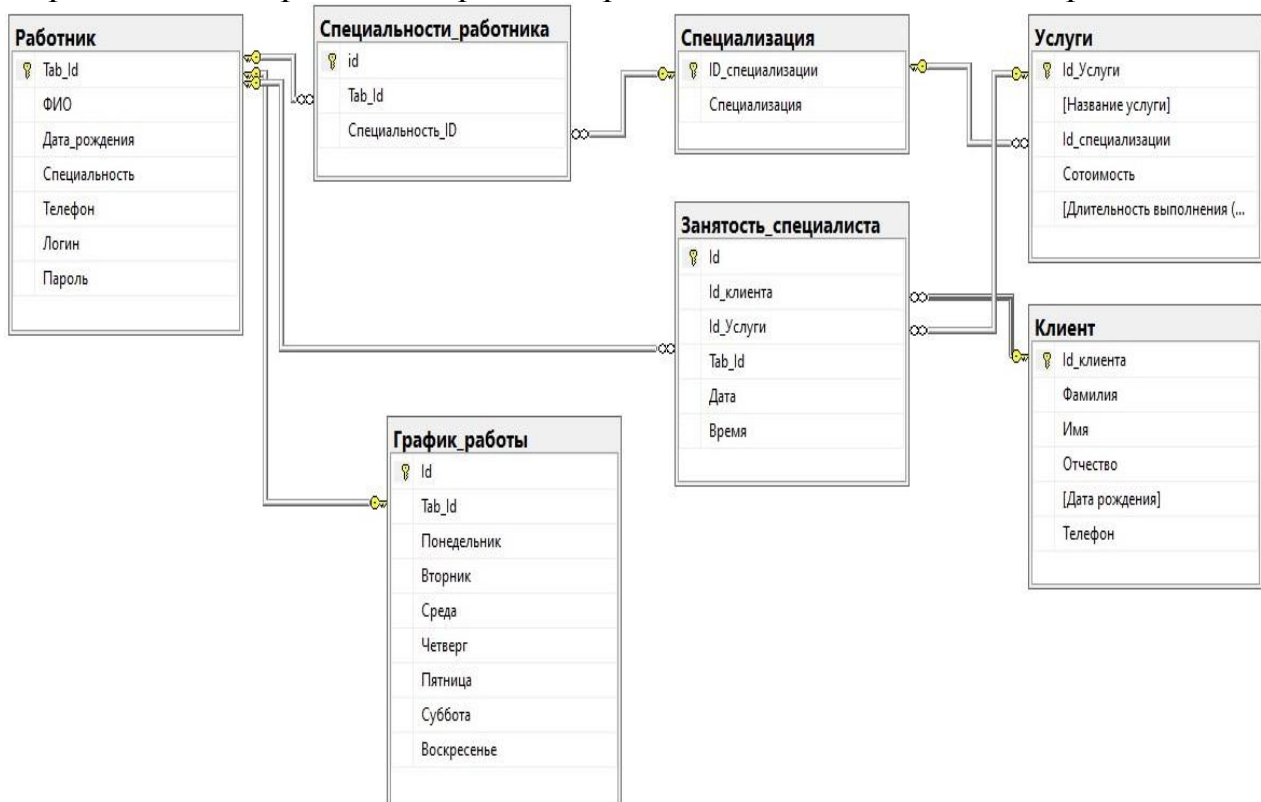


Рисунок 9 – Схема данных

```

USE [Salon_krasot]
CREATE TABLE [Работник]
(
Tab_Id int IDENTITY(1,1) PRIMARY KEY NOT NULL,
ФИО varchar(70) NOT NULL,
Дата_рождения date NOT NULL,
Телефон varchar(15) UNIQUE NOT NULL,
Логин varchar(16) UNIQUE NOT NULL,
Пароль varchar(12) UNIQUE NOT NULL
)

CREATE TABLE [Специализация]
(
ID_специализации int IDENTITY(1,1) PRIMARY KEY NOT NULL,
Специализация varchar(40) NOT NULL
)

CREATE TABLE [Специальности_работника]
(
id int IDENTITY(1,1) PRIMARY KEY NOT NULL,
Tab_Id int NOT NULL,
Специальность_ID int NOT NULL
CONSTRAINT FK1 FOREIGN KEY (Tab_Id) REFERENCES [Работник]
(Tab_Id),
CONSTRAINT FK2 FOREIGN KEY (Специальность_ID) REFERENCES
[Специализация] (ID_специализации)
)
CREATE TABLE [График_работы]
(
Id int IDENTITY(1,1) PRIMARY KEY NOT NULL,
Tab_Id int UNIQUE NOT NULL,
Понедельник varchar(11),
Вторник varchar(11),
Среда varchar(11),
Четверг varchar(11),
Пятница varchar(11), Суббота varchar(11), Воскресенье varchar(11)
CONSTRAINT FK3 FOREIGN KEY (Tab_Id) REFERENCES [Работник]
(Tab_Id)
)
CREATE TABLE [Услуги]
(
Id_Услуги int IDENTITY(1,1) PRIMARY KEY NOT NULL,
[Название услуги] varchar(50) NOT NULL,
Id_специализации int NOT NULL,

```

```

Сотоимость decimal(19,2) NOT NULL,
[Длительность выполнения (Мин.)] int NOT NULL
CONSTRAINT FK4 FOREIGN KEY (Id_специализации) REFERENCES
[Специализация] (ID_специализации)
)
CREATE TABLE [Клиент]
(
Id_клиента int IDENTITY(1,1) PRIMARY KEY NOT NULL,
Фамилия varchar(25) NOT NULL,
Имя varchar(25) NOT NULL,
Отчество varchar(25), [Дата рождения] date,
Телефон varchar(15) NOT NULL
)
CREATE TABLE [Занятость_специалиста]
(
Id int IDENTITY(1,1) PRIMARY KEY NOT NULL,
Id_клиента int NOT NULL,
Id_Услуги int NOT NULL,
Tab_Id int NOT NULL, Дата date NOT NULL,
Время time NOT NULL
CONSTRAINT FK5 FOREIGN KEY (Id_клиента) REFERENCES
[Клиент] (Id_клиента),
CONSTRAINT FK6 FOREIGN KEY (Tab_Id) REFERENCES [Работник]
(Tab_Id),
CONSTRAINT FK7 FOREIGN KEY (Id_Услуги) REFERENCES
[Услуги] (Id_Услуги)
)

```

Вариант 2.Файл работы сохранить в родной папке под именем  
Вариант2.xlsx



Рисунок 10 – Схема данных

```

CREATE TABLE [dbo].[Table_отделение](
    [код_отделе] [varchar](30) NOT NULL,    [фио зав отделением]
[varchar](30) NOT NULL,
    [пороль] [varchar](10) NOT NULL PRIMARY KEY ([код_отделе]);

```

```

CREATE TABLE [dbo].[Table_спицальность](
    [код_спицальности] [varchar](10) NOT NULL,
    [название спицальности] [varchar](100) NOT NULL,
    [код_отделение] [varchar](30) NOT NULL, PRIMARY KEY
([код_спицальности]);

```

```

CREATE TABLE [dbo].[Table_группа](
    [код_группы] [varchar](10) NOT NULL PRIMARY KEY,
    [код_спицальности] [varchar](10) NOT NULL,
);

```

```

CREATE TABLE [dbo].[Table_студенты](
    [№студенческого] [varchar](10) NOT NULL,
    [фио] [varchar](50) NOT NULL,
    [группа] [varchar](10) NOT NULL,
    [дата рождения] [date] NOT NULL,
    [телефон] [varchar](12) NOT NULL,
    [домашний адрес] [varchar](70) NOT NULL, PRIMARY KEY
([№студенческого]);

```

```

CREATE TABLE [dbo].[родственники](
    [код записи] [int] NOT NULL,
    [№студенческого] [varchar](10) NOT NULL,
    [код родства] [varchar](50) NOT NULL,
    [контакты] [nchar](12) NOT NULL, PRIMARY KEY ([код записи]);
alter table [Table_спицальность] add constraint FK1 foreign key
([код_отделение]) references
[Table_отделение] ([код_отделе]);
alter table [родственники] add constraint FK2 foreign key
([№студенческого]) references
[Table_студенты]
([№студенческого]);
alter table [Table_студенты] add constraint FK3 foreign key
([группа]) references
[Table_группа]
([код_группы]);
alter table [Table_группа] add constraint FK4 foreign key
([код_спицальности]) references
[Table_спицальность]
([код_спицальности]);

```

Вариант 3. Файл работы сохранить в родной папке под именем  
Вариант3.xlsx

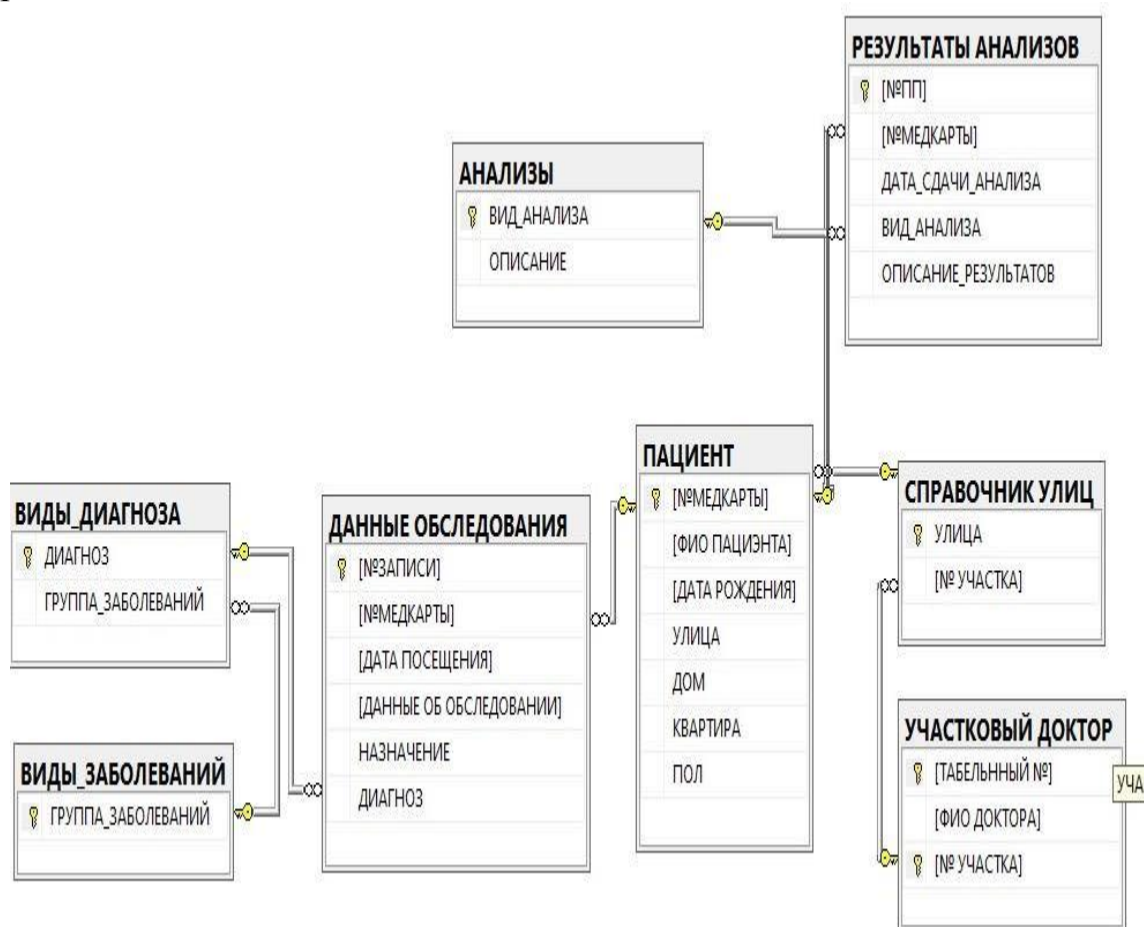


Рисунок 11 – Схема данных

```

USE [поликлиника]
CREATE TABLE [dbo].[УЧАСТКОВЫЙ ДОКТОР](
    [ТАБЕЛЬНЫЙ №] [int] NOT NULL,
    [ФИО ДОКТОРА] [varchar](50) NOT NULL,
    [№ УЧАСТКА] [int] NOT NULL,
    UNIQUE ([№ УЧАСТКА]),
    PRIMARY KEY ([ТАБЕЛЬНЫЙ №],[№ УЧАСТКА]));
USE [поликлиника]
CREATE TABLE [dbo].[СПРАВОЧНИК УЛИЦ](
    [УЛИЦА] [varchar](50) NOT NULL,
    [№ УЧАСТКА] [int] NOT NULL,
    PRIMARY KEY ([УЛИЦА]));
USE [поликлиника] alter table [СПРАВОЧНИК УЛИЦ] add
CONSTRAINT FK1 foreign key ([№ УЧАСТКА]) references [УЧАСТКОВЫЙ
ДОКТОР]([№ УЧАСТКА]);
USE [поликлиника]
CREATE TABLE [ПАЦИЕНТ]( [№МЕДКАРТЫ] [int] NOT NULL,
    [ФИО ПАЦИЕНТА] [varchar](50) NOT NULL,

```



```

[ДАТА РОЖДЕНИЯ] [date] NOT NULL,
[УЛИЦА] [varchar](50) NOT NULL,
[ДОМ] [int] NOT NULL,
[КВАРТИРА] [int] NOT NULL,    [ПОЛ] [bit] NOT NULL,
PRIMARY KEY ([№МЕДКАРТЫ]);
alter table [ПАЦИЕНТ] add CONSTRAINT FK2 foreign key ([УЛИЦА])
references [СПРАВОЧНИК УЛИЦ]([УЛИЦА]);
USE [поликлиника]
CREATE TABLE [АНАЛИЗЫ](
[ВИД_АНАЛИЗА] [varchar](50) NOT NULL,
[ОПИСАНИЕ] [varchar](max) NOT NULL,
PRIMARY KEY ([ВИД_АНАЛИЗА]);
USE [поликлиника]
CREATE TABLE [ВИДЫ_ДИАГНОЗА](
[ДИАГНОЗ] [varchar](50) NOT NULL,
[ГРУППА_ЗАБОЛЕВАНИЙ] [varchar](50) NOT NULL,
PRIMARY KEY ([ДИАГНОЗ]);
USE [поликлиника]
CREATE TABLE [ВИДЫ_ЗАБОЛЕВАНИЙ](
[ГРУППА_ЗАБОЛЕВАНИЙ] [varchar](50) NOT NULL,
PRIMARY KEY ([ГРУППА_ЗАБОЛЕВАНИЙ]);
alter table [ВИДЫ_ДИАГНОЗА] add CONSTRAINT FK3 foreign key
([ГРУППА_ЗАБОЛЕВАНИЙ]) references [ВИДЫ_ЗАБОЛЕВАНИЙ]
([ГРУППА_ЗАБОЛЕВАНИЙ]);
USE [поликлиника]
CREATE TABLE [ДАННЫЕ ОБСЛЕДОВАНИЯ](
[№ЗАПИСИ] [int] NOT NULL,
[№МЕДКАРТЫ] [int] NOT NULL,
[ДАТА ПОСЕЩЕНИЯ] [date] NOT NULL,
[ДАННЫЕ ОБ ОБСЛЕДОВАНИИ] [varchar](max) NOT NULL,
[НАЗНАЧЕНИЕ] [varchar](max) NOT NULL,
[ДИАГНОЗ] [varchar](50) NOT NULL,
PRIMARY KEY ([№ЗАПИСИ]);
alter table [ДАННЫЕ ОБСЛЕДОВАНИЯ] add CONSTRAINT FK4
foreign key ([№МЕДКАРТЫ]) references [ПАЦИЕНТ]([№МЕДКАРТЫ]); alter
table [ДАННЫЕ ОБСЛЕДОВАНИЯ] add CONSTRAINT FK5 foreign key
([ДИАГНОЗ]) references [ВИДЫ_ДИАГНОЗА]([ДИАГНОЗ]);
USE [поликлиника]
CREATE TABLE [РЕЗУЛЬТАТЫ АНАЛИЗОВ](
[№ПП] [int] NOT NULL,
[№МЕДКАРТЫ] [int] NOT NULL,
[ДАТА_СДАЧИ_АНАЛИЗА] [date] NOT NULL,
[ВИД_АНАЛИЗА] [varchar](50) NOT NULL,
[ОПИСАНИЕ_РЕЗУЛЬТАТОВ] [varchar](max) NOT NULL,

```

PRIMARY KEY ([№ПП]);

USE [поликлиника]

```
alter table [РЕЗУЛЬТАТЫ АНАЛИЗОВ] add CONSTRAINT FK6 foreign  
key ([№МЕДКАРТЫ]) references [ПАЦИЕНТ]([№МЕДКАРТЫ]); alter table  
[РЕЗУЛЬТАТЫ АНАЛИЗОВ] add CONSTRAINT FK7 foreign key  
([ВИД_АНАЛИЗА]) references [АНАЛИЗЫ]([ВИД_АНАЛИЗА]);
```

Вариант 4. Файл работы сохранить в родной папке под именем  
Вариант4.xlsx

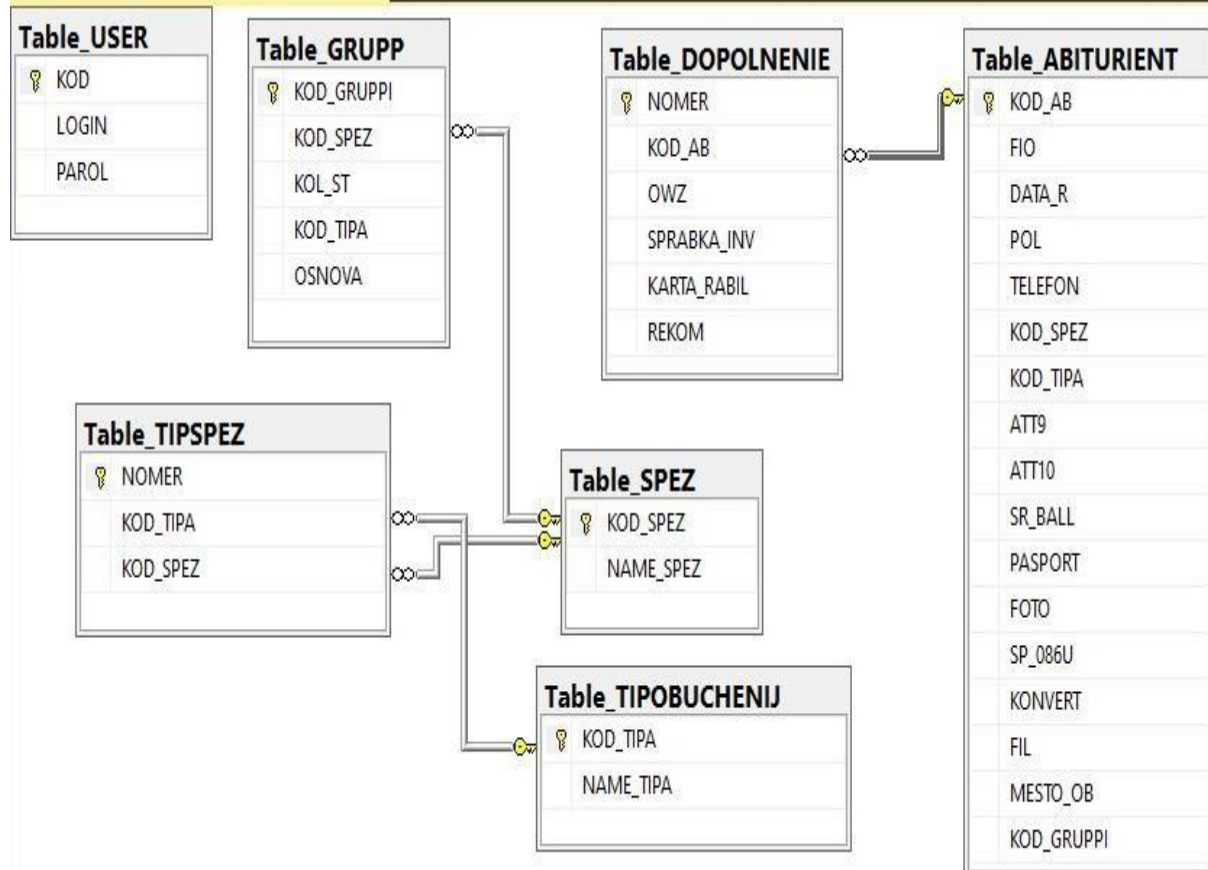


Рисунок 12 – Схема данных

USE [PR\_KOM]

```
CREATE TABLE [dbo].[Table_ABITURIENT](  
    [KOD_AB] [int] IDENTITY(1,1) NOT NULL,  
    [FIO] [varchar](50) NOT NULL,  
    [DATA_R] [date] NOT NULL,  
    [POL] [varchar](7) NOT NULL,  
    [TELEFON] [varchar](15) NULL,  
    [KOD_SPEZ] [varchar](10) NOT NULL,  
    [KOD_TIPA] [int] NOT NULL,  
    [ATT9] [varchar](8) NULL,  
    [ATT10] [varchar](8) NULL,  
    [SR_BALL] [decimal](3, 2) NOT NULL,  
    [PASPORT] [bit] NOT NULL,
```

```

[FOTO] [int] NOT NULL,
[SP_086U] [bit] NOT NULL,
[KONVERT] [int] NOT NULL,
[FIL] [bit] NOT NULL,
[MESTO_OB] [bit] NOT NULL,
[KOD_GRUPPI] [varchar](10)
PRIMARY KEY ([KOD_AB]));
CREATE TABLE [dbo].[Table_DOPOLNENIE](
[NOMER] [int] IDENTITY(1,1) NOT NULL,
[KOD_AB] [int] NOT NULL,
[OWZ] [bit] NULL,
[SPRABKA_INV] [bit] NOT NULL,
[KARTA_RABIL] [bit] NOT NULL,
[REKOM] [bit] NOT NULL,
PRIMARY KEY ([NOMER]));
CREATE TABLE [dbo].[Table_GRUPP](
[KOD_GRUPPI] [varchar](10) NOT NULL,
[KOD_SPEZ] [varchar](10) NOT NULL,
[KOL_ST] [int] NULL,
[KOD_TIPA] [int] NOT NULL,
[OSNOVA] [varchar](10) NOT NULL,
PRIMARY KEY ([KOD_GRUPPI]);
CREATE TABLE [dbo].[Table_SPEZ](
[KOD_SPEZ] [varchar](10) NOT NULL,
[NAME_SPEZ] [varchar](100) NULL,
PRIMARY KEY ([KOD_SPEZ]);
CREATE TABLE [dbo].[Table_TIPOBUCHENIJ](
[KOD_TIPA] [int] IDENTITY(1,1) NOT NULL,
[NAME_TIPA] [varchar](100) NOT NULL,
PRIMARY KEY ([KOD_TIPA]);
CREATE TABLE [dbo].[Table_USER](
[KOD] [int] IDENTITY(1,1) NOT NULL,
[LOGIN] [varchar](50) NULL,
[PAROL] [varchar](6) NULL,
PRIMARY KEY ([KOD]);
CREATE TABLE [dbo].[Table_TIPSPEZ](
[NOMER] [int] IDENTITY(1,1) NOT NULL,
[KOD_TIPA] [int] NOT NULL,
[KOD_SPEZ] [varchar](10) NOT NULL,
PRIMARY KEY ([NOMER]);
alter table [Table_ABITURIENT] add constraint FK1 foreign key
([KOD_SPEZ]) references [Table_SPEZ] ([KOD_SPEZ]); alter table
[Table_GRUPP] add constraint FK2 foreign key ([KOD_SPEZ]) references
[Table_SPEZ] ([KOD_SPEZ]); alter table [Table_TIPSPEZ] add constraint FK3

```

foreign key ([KOD\_SPEZ]) references [Table\_SPEZ] ([KOD\_SPEZ]); alter table [Table\_TIPSPEZ] add constraint FK5 foreign key ([KOD\_TIPA]) references [Table\_TIPOBUCHENIJ] ([KOD\_TIPA]); alter table [Table\_DOPOLNENIE] add constraint FK4 foreign key ([KOD\_AB]) references [Table\_ABITURIENT] ([KOD\_AB]);

Вариант 5. Файл работы сохранить в родной папке под именем Вариант5.xlsx

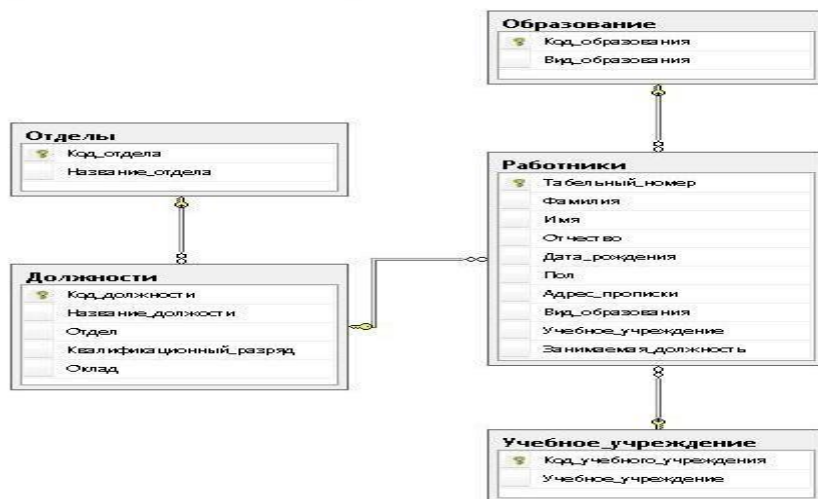


Рисунок 13 – Схема данных

```

USE [КАДРЫ]
CREATE TABLE [dbo].[Table_РАБОТНИК](
    [TAB_NOMER] [int] NOT NULL,
    [FIO_sotrudnika] [varchar](50) NOT NULL,
    [VID_OBRAZOV] [varchar](50) NOT NULL,
    [OBRAZOV_UCH] [varchar](50) NOT NULL,
    [DOLJ] [varchar](50) NOT NULL,
    [OTDEL] [varchar](50) NOT NULL,
    [DATA_ROJDENIJ] [date] NOT NULL,
    [SNILS] [varchar](10) NOT NULL,
    PRIMARY KEY ([TAB_NOMER])
)
USE [КАДРЫ]
CREATE TABLE [dbo].[Table_УЧ_УЧРЕЖДЕНИЕ](
    [UCH_UCHREGDENIE] [varchar](50) NOT NULL,
    PRIMARY KEY ([UCH_UCHREGDENIE])
)
USE [КАДРЫ]
CREATE TABLE [dbo].[Table_ОБРАЗОВАНИЕ](
    [VID_OBRAZOV] [varchar](50) NOT NULL,
    PRIMARY KEY CLUSTERED ([VID_OBRAZOV])
)
USE [КАДРЫ]
CREATE TABLE [dbo].[Table_ДОЛЖНОСТИ](
    [DOLJ] [varchar](50) NOT NULL,
    [OKLAD] [REAL] NOT NULL,

```

```

[OTDEL] [varchar](50) NOT NULL,
PRIMARY KEY ([DOLJ]))
USE [КАДРЫ]
CREATE TABLE [dbo].[Table_ОТДЕЛ](
[OTDEL] [varchar](50) NOT NULL, PRIMARY KEY ([OTDEL]))
alter table [Table_РАБОТНИК] add CONSTRAINT FK1 foreign key
([VID_OBRAZOV]) references [Table_ОБРАЗОВАНИЕ]([VID_OBRAZOV]);
alter table [Table_РАБОТНИК] add CONSTRAINT FK2 foreign key
([OBRAZOV_UCH]) references
[Table_УЧ_УЧРЕЖДЕНИЕ]([UCH_UCHREGDENIE]);
alter table [Table_ДОЛЖНОСТИ] add CONSTRAINT FK4 foreign key
([OTDEL]) references [Table_ОТДЕЛ]([OTDEL]); alter table
[Table_РАБОТНИК] add CONSTRAINT FK5 foreign key ([DOLJ]) references
[Table_ДОЛЖНОСТИ]([DOLJ]);

```

Вариант 6. Файл работы сохранить в родной папке под именем  
Вариант6.xlsx

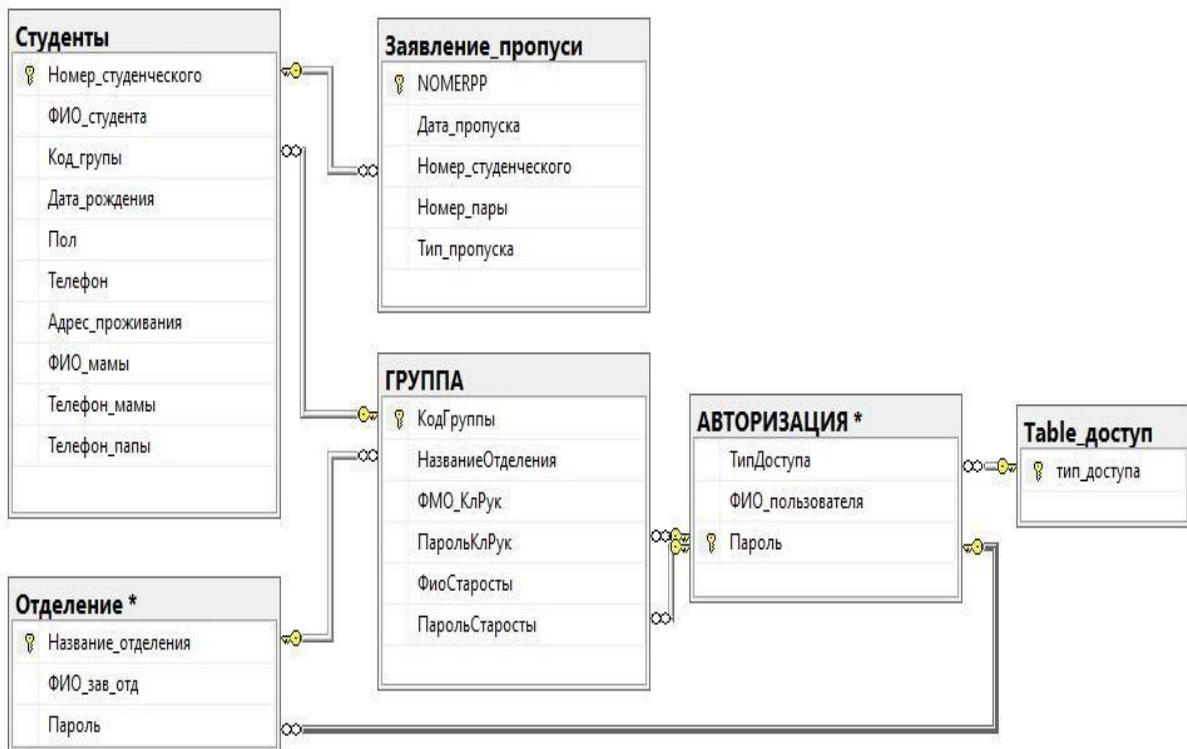


Рисунок 14 – Схема данных

```

USE [BPPKPROPUSKI123]
CREATE TABLE [dbo].[Table_доступ](
[тип_доступа] [varchar](50) PRIMARY KEY NOT NULL)
CREATE TABLE [dbo].[Студенты](
[Номер_студенческого] [varchar](255) PRIMARY KEY NOT
NULL,

```

```

[ФИО_студента] [varchar](255) NOT NULL,
[Код_группы] [varchar](255) NOT NULL,
[Дата_рождения] [varchar](255) NOT NULL,
[Пол] [varchar](255) NOT NULL,
[Телефон] [varchar](255) NOT NULL,
[Адрес_проживания] [varchar](255) NOT NULL,
[ФИО_мамы] [varchar](255) NOT NULL,
[Телефон_мамы] [varchar](255) NOT NULL,
[Телефон_папы] [varchar](255) NOT NULL) CREATE TABLE
[dbo].[Заявление_пропуса](
[NOMERPP] [int] PRIMARY KEY NOT NULL,
[Дата_пропуска] [date] NOT NULL,
[Номер_студенческого] [varchar](255) NOT NULL,
[Номер_пары] [int] NOT NULL,
[Тип_пропуска] [varchar](255) NOT NULL)
CREATE TABLE [dbo].[Отделение](
[Название_отделения] [varchar](255) PRIMARY KEY NOT NULL,
[ФИО_зав_отд] [varchar](255) NOT NULL,
[Пароль] [varchar](10) NOT NULL)
CREATE TABLE [dbo].[ГРУППА](
[КодГруппы] [varchar](255) PRIMARY KEY NOT NULL,
[НазваниеОтделения] [varchar](255) NOT NULL,
[ФМО_КлРук] [varchar](255) NOT NULL,
[ПарольКлРук] [varchar](10) NOT NULL,
[ФиюСтаросты] [varchar](255) NULL,
[ПарольСтаросты] [varchar](10) NULL)
CREATE TABLE [dbo].[АВТОРИЗАЦИЯ](
[ТипДоступа] [varchar](50) NOT NULL,
[ФИО_пользователя] [varchar](255) NOT NULL,
[Пароль] [varchar](10) PRIMARY KEY NOT NULL)
alter table Студенты add constraint FK1 foreign key ([Код_группы])
references ГРУППА (КодГруппы);
alter table [Заявление_пропуса] add constraint FK2 foreign key
([Номер_студенческого]) references Студенты (Номер_студенческого); alter
table ГРУППА add constraint FK3 foreign key (НазваниеОтделения) references
Отделение (Название_отделения); alter table АВТОРИЗАЦИЯ add constraint
FK4 foreign key (ТипДоступа) references Table_доступ (тип_доступа); alter
table ГРУППА add constraint FK5 foreign key (ПарольКлРук) references
АВТОРИЗАЦИЯ (Пароль); alter table ГРУППА add constraint FK6 foreign key
(ПарольСтаросты) references АВТОРИЗАЦИЯ (Пароль); alter table Отделение
add constraint FK7 foreign key (Пароль) references АВТОРИЗАЦИЯ (Пароль).

```

**Задание:** Разработать схему данных для индивидуальной БД.

### ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 3. Разработка технических требований к серверу баз данных

**Цель:** освоить технологии оценки требований к серверу баз данных.

**Ход работы:**

Повторить теоретический материал.

Создать документ в формате Ms Excel, сохранить документ в родной папке ТЗ\_Сервер\_№варианта.xlsx.

	А	В	С
1	Исходная нагрузка(количество пользователей)		150
2	От 100 пользователей рекомендуется разделять физически сервер базы данных и сервер приложений.		
3	Аппаратное обеспечение		
4	Требования к серверу баз данных		
5	Минимальные требования		Рекомендуемые требования
6	Процессор (количество ядер)	2	Процессор: 2х ядерный процессор;
7	Процессор (тактыая частота в Гц)	3,2 и выше	Тактовая частота: 3,2 Гц или выше;
8	Оперативная память (Гб)	3	Оперативная память: 5 Гб или выше;
9	Дисковое пространство RAID10 (Тб)	0,32	Дисковое пространство: рабочие - RAID10 не менее 4 дисков (SCSI или SAS) суммарным объемом не менее 1 Тб
10	Дисковое пространство для хранения архивных копий (Тб)	0,64	Для хранения архивных копий - RAID1 2 диска (SCSI или SAS) суммарным объемом не менее 2 Тб
11			Диск для горячей замены (Hot Spare)
12	<b>Требования к серверу приложений (web-сервер) аналогичные</b>		
13	<b>Каналы связи:</b>	В случае развертывания системы в пределах локальной сети необходимо обеспечить пропускную способность сети в 100 Мбит/сек. Пропускная способность внешнего канала связи не менее 20 Мбит/сек с возможностью расширения.	

Рисунок 15 – документ в формате Ms Excel

Расчёт минимальных требований к серверу базы данных осуществить по формулам в ячейках В6, В8, В9, В10 исходя из нагрузки-ячейка С1.

Выбрать рекомендуемые требования, исходя из нагрузки.

*Кратки теоретические сведения.*

Для поддержания бесперебойной работы крупных проектов используют производительные сервера или целые кластеры серверных машин, где стоит, как правило, СУБД — комплекс программ для создания и манипулирования данными. Главное назначение выделенного сервера БД состоит в размещении, обработке и хранении информации силами достаточно производительной конфигурации, при этом все это происходит посредством одной из предустановленных СУБД. Непосредственно сама система управления базами предоставляет доступ к ним клиентам и приложениям и обеспечивает оперативную обработку запросов. Описанный формат взаимодействия также называют архитектурой типа «клиентсервер».

Любое обращение к реляционной БД происходит в большинстве случаев на самом распространенном языке запросов SQL. В свою очередь платформа, на которой запущена СУБД, «понимающая» этот язык, и называется SQL-сервером.

При небольших нагрузках допустимо (а иногда и оправданно) разместить базу данных на основной вычислительной машине. Более крупные проекты, где число ежедневных запросов к базе превышает 500, разумнее реализовывать уже на отдельном SQL-сервере. Это позволяет



оборудованию не расплываться на сторонние задачи, а сосредоточиться на выполнении типовых процессов, под которые заранее рассчитаны ресурсы и мощность оборудования.

Требования к обеспечению сервера баз данных

Требования для сервера БД и WEB-сервера идентичны. Рекомендованы физически разные машины.

Формулы расчета требований, исходя из количества зарегистрированных пользователей при условии, что одновременно работать будут максимум 50% пользователей, следующие:

Процессор: количество одновременно работающих пользователей /100, ядер (3,2 ГГц и выше).

Оперативная память: количество одновременно работающих пользователей /50.

Дисковое пространство RAID10: не менее 4 дисков (SCSI или SAS) суммарным объемом: 9мб \* 100 создаваемых записей в день одним пользователем \* количество зарегистрированных пользователей \* 2,5. Для хранения архивных копий: 9мб \* 100 создаваемых записей в день одним пользователем \* количество зарегистрированных пользователей \* 5.

Аппаратное обеспечение	
Рекомендуемые требования к аппаратному обеспечению	
До 100 зарегистрированных пользователей	
Требования к серверу базы данных:	
Процессор: 1 ядерный процессор; Тактовая частота: 3,2 ГГц или выше; Оперативная память: 1 Гб или выше; Дисковое пространство: рабочие - RAID10 не менее 4 дисков (SCSI или SAS) суммарным объемом не менее 200 Гб Для хранения архивных копий - RAID1 2 диска (SCSI или SAS) суммарным объемом не менее 500 Гб в расчете на 1 год работы в системе Диск для горячей замены (Hot Spare)	В количестве 1 шт.
Требования к серверу приложений (web-сервер) аналогичные	

От 100 пользователей рекомендуется разделять физически сервер базы данных и сервер приложений.
До 500 зарегистрированных пользователей:
Требования к серверу базы данных:



<p>Оперативная память: 5 ГБ или выше;  Дисковое пространство: рабочие - RAID10 не менее 4 дисков (SCSI или SAS) суммарным объемом не менее 1 Тб  Для хранения архивных копий - RAID1 2 диска (SCSI или SAS) суммарным объемом не менее 2 Тб  Диск для горячей замены (Hot Spare)</p>	1 шт.
Требования к серверу приложений (web-сервер) аналогичные	
До 1000 зарегистрированных пользователей:	
Требования к серверу базы данных:	
<p>Процессор: 5ти ядерный процессор;  Тактовая частота: 3,2 ГГц или выше;  Оперативная память: 12 ГБ или выше;  Дисковое пространство: рабочие - RAID10 не менее 4 дисков (SCSI или SAS) суммарным объемом не менее 2 Тб.  Для хранения архивных копий - RAID1 2 диска (SCSI или SAS) суммарным объемом не менее 4,5 Тб  Диск для горячей замены (Hot Spare)</p>	В количестве 1шт.
Требования к серверу приложений (web-сервер) аналогичные	
До 5000 зарегистрированных пользователей:	
Требования к серверу базы данных:	
<p>Процессор: 25ти ядерный процессор;  Тактовая частота: 3,2 ГГц или выше;  Оперативная память: 50 ГБ или выше;  Дисковое пространство: рабочие - RAID10 не менее 4 дисков (SCSI или SAS) суммарным объемом не менее 11 Тб  Для хранения архивных копий - RAID1 2 диска (SCSI или SAS) суммарным объемом не менее 21 Тб  Диск для горячей замены (Hot Spare)</p>	В количестве 1 шт.
Требования к серверу приложений (web-сервер) аналогичные	
До 10000 зарегистрированных пользователей:	
Требования к серверу базы данных:	
<p>Процессор: 50ти ядерный процессор;  Тактовая частота: 3,2 ГГц или выше;  Оперативная память: 100 ГБ или выше;  Дисковое пространство: рабочие - RAID10 не менее 4 дисков (SCSI или SAS) суммарным объемом не менее 22 Тб  Для хранения архивных копий - RAID1 2 диска (SCSI или SAS) суммарным объемом не менее 44 Тб  Диск для горячей замены (Hot Spare)</p>	В количестве 1 шт.
Требования к серверу приложений (web-сервер) аналогичные	

***Задание:***

Вариант 1. 150 зарегистрированных пользователей. Разработать технические требования к серверам базы данных: минимальные и рекомендуемые.

Вариант 2. 2700 зарегистрированных пользователей. Разработать технические требования к серверам базы данных: минимальные и рекомендуемые.

Вариант 3. 430 зарегистрированных пользователей. Разработать технические требования к серверам базы данных: минимальные и рекомендуемые.

Вариант 4. 8000 зарегистрированных пользователей. Разработать технические требования к серверам базы данных: минимальные и рекомендуемые.

Вариант 5. 1500 зарегистрированных пользователей. Разработать технические требования к серверам базы данных: минимальные и рекомендуемые.

## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 4. Разработка требований к корпоративной сети

## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 5. Конфигурирование сети

Несмотря на то что мастер настройки сети автоматически создает все необходимые сетевые параметры, свойства сетевых протоколов могут не соответствовать текущей конфигурации локальной сети. Иными словами, мастер не всегда на «отлично» справляется со своей работой. Если, открыв папку Сетевое окружение, вы не увидите в ней значков, подключенных к локальной сети компьютеров, придется изменять настройки сетевых протоколов вручную. Для этого понадобятся следующие параметры:

1. IP-адрес вашего компьютера;
2. маска подсети;
3. IP-адрес основного шлюза доступа к Интернету.

Откройте папку Сетевое окружение и щелкните на ссылке отобразить сетевые подключения в левой панели Сетевые задачи. Откроется окно содержащее значки всех настроенных в системе сетевых подключений Сетевые подключения. Дважды щелкните на значке соответствующего сетевого подключения, чтобы открыть окно с данными о состоянии подключения локальной сети.

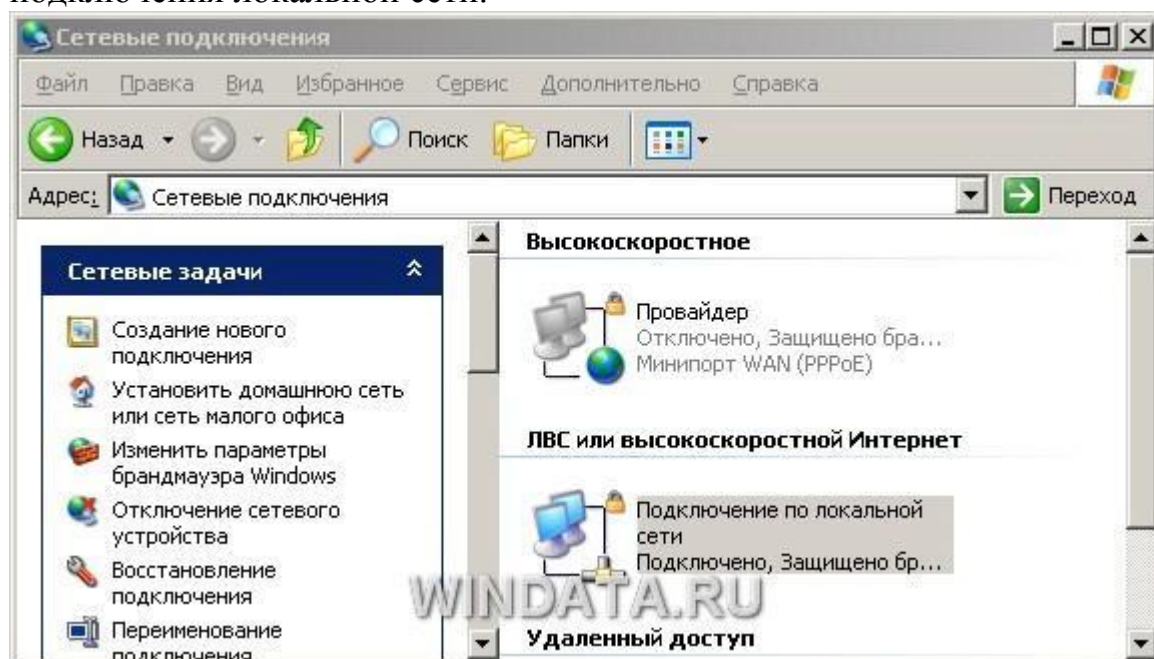


Рисунок 16 – Сетевые подключения

В частности, в окне указана длительность активного сетевого соединения, скорость соединения, активность (сколько байтов информации отправлено и принято). Все параметры сетевого соединения представлены в этом же окне на вкладке Поддержка. Там можно узнать тип IP-адреса (введен вручную или назначен DHCP), IP-адрес компьютера, маску подсети и IP-адрес основного шлюза.

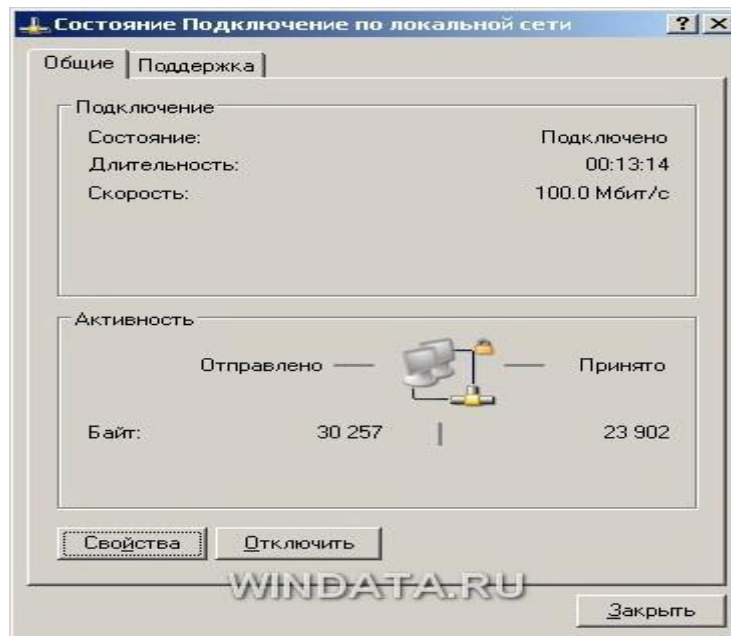


Рисунок 17 – Сетевые подключения  
Вкладка Поддержка.

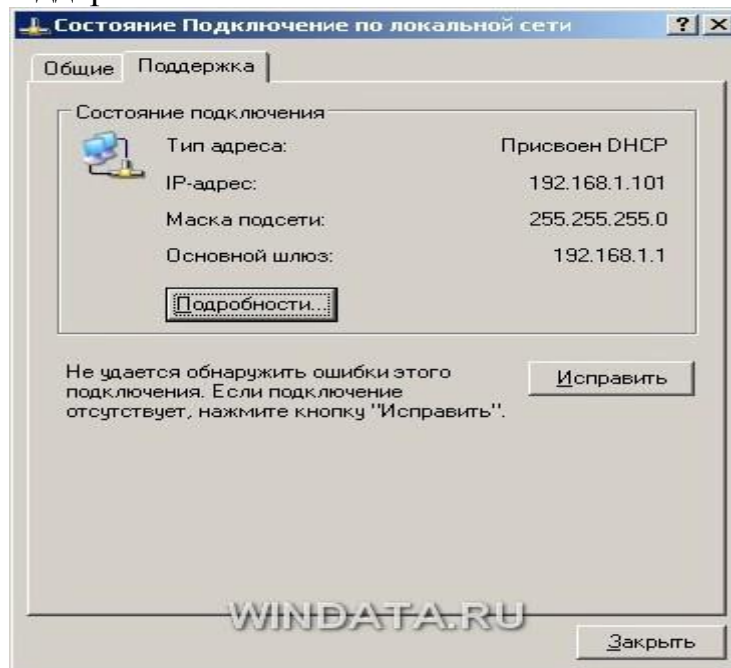


Рисунок 18 – Сетевые подключения

Кроме того, если щелкнуть на кнопке **Подробнее**, можно получить дополнительные сведения, такие как физический MAC-адрес сетевого адаптера. В окне также расположена кнопка **Исправить**, позволяющая исправить некоторые проблемы, связанные с подключением.

Щелкните на кнопке **Подробнее**, чтобы открыть это окно.

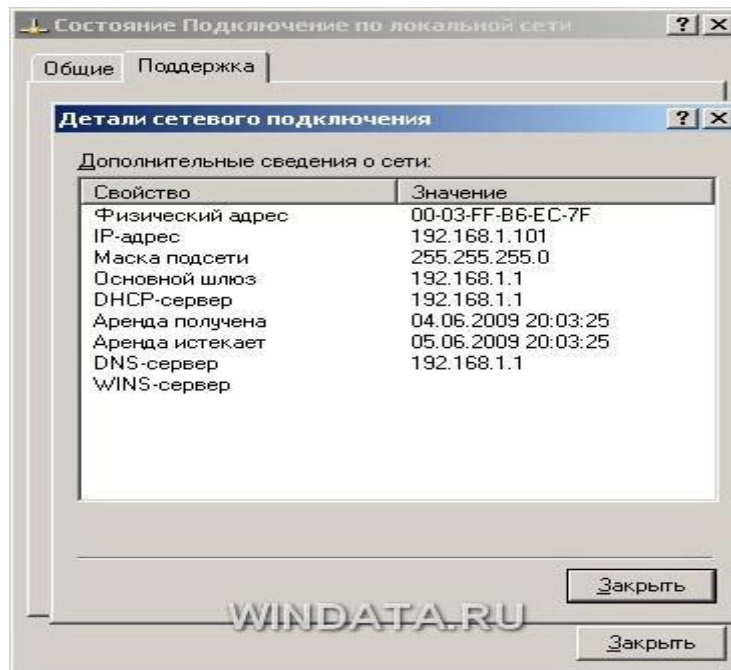


Рисунок 19 – Сетевые подключения

Щелкните на кнопке Исправить, чтобы исправить проблемы с сетевым подключением. Порой действительно помогает :)

Чтобы внести какие-либо изменения в конфигурацию локальной сети, щелкните на кнопке Свойства. Откроется окно со свойствами сетевого подключения.

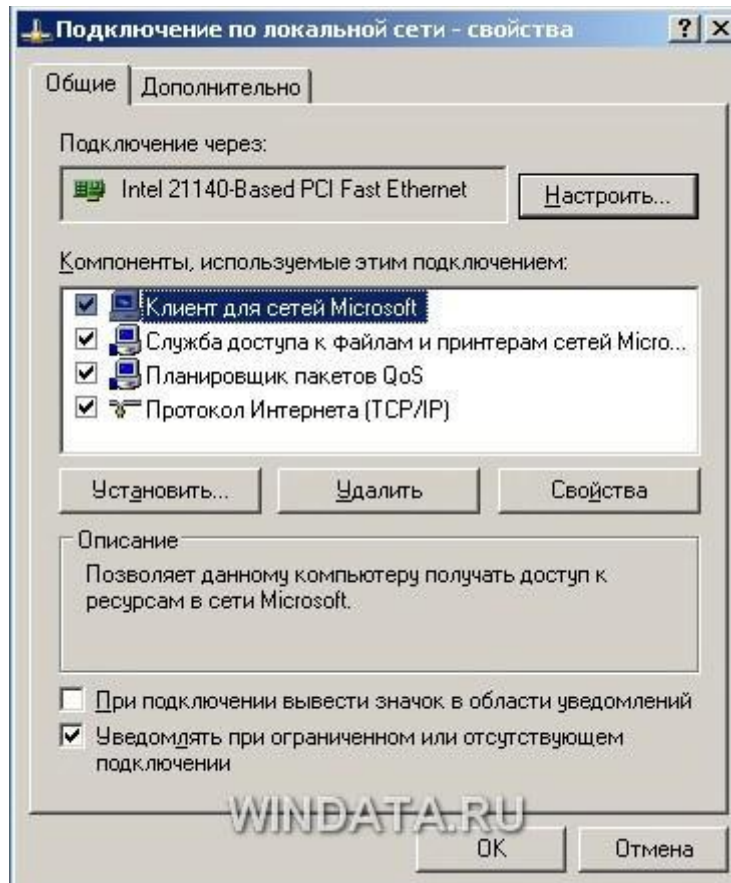


Рисунок 20 – Сетевые подключения

Чтобы изменить аппаратные настройки сетевой платы, щелкните на кнопке Настроить. Кроме того, установите флажок При подключении вывести значок в области уведомлений, чтобы при подключении к локальной сети в области уведомления Windows XP отображался специальный значок.

Настройка параметров TCP/IP – основной шаг, позволяющий добиться работоспособности локальной сети. В окне Подключение по локальной сети выберите пункт Протокол Интернета (TCP/IP) и щелкните на кнопке Свойства. Откроется окно Свойства: Протокол Интернета (TCP/IP).

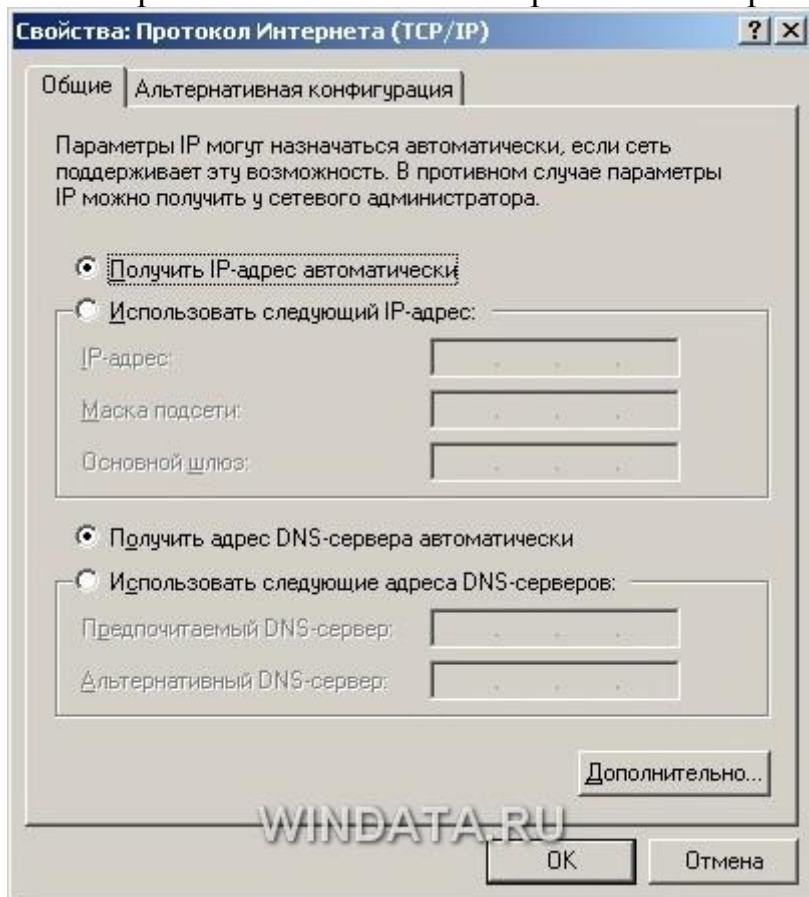


Рисунок 21 – Сетевые подключения

Для стандартной домашней сети можно рекомендовать такие параметры. Адреса компьютеров указывайте в диапазоне 192.168.0.2-192.168.0.50, т.е. первый компьютер получает адрес 192.168.0.2, второй – 192.168.0.3 и т.д. Адрес 192.168.0.1, как правило, присваивается основному шлюзу сети. Маску подсети укажите как 255.255.255.0. Во многих случаях такая конфигурация подойдет для организации работы локальной сети.

Если мастер настройки сети выполнил свою работу, то IP-адрес компьютеру назначается автоматически. В противном случае адрес придется указать вручную. Для этого выберите переключатель Использовать следующий IP-адрес и введите IP-адрес компьютера в поле Использовать следующий IP-адрес, а в поле Маска подсети – маску подсети. Если в сети используется определенный шлюз, такой как маршрутизатор, укажите его IP-

адрес в поле Основной шлюз. Вводить IP-адреса первичного и вторичного DNS-серверов, как правило, не обязательно (хотя порой и требуется).

Если компьютер используется в нескольких сетях, щелкните на вкладке Альтернативная конфигурация. В ней можно, выбрав переключатель ввести параметры альтернативной конфигурации IP, включая IP-адрес, маску подсети и основной шлюз, а также предпочитаемые и альтернативные DNS-серверы.

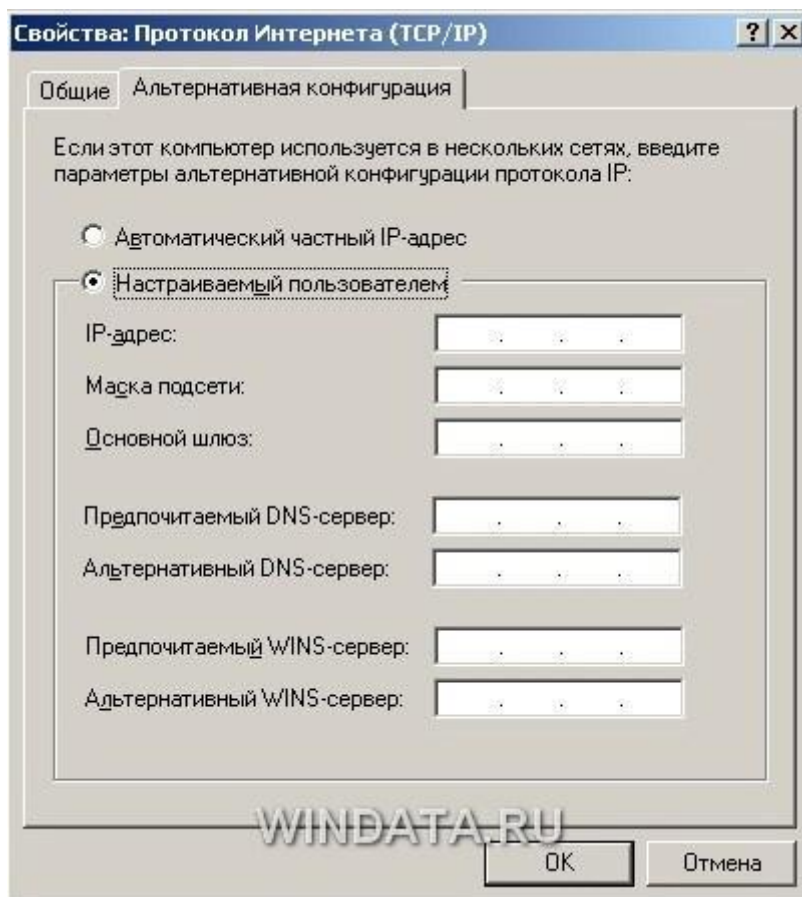


Рисунок 21 – Сетевые подключения

Щелкните на кнопке ОК, чтобы сохранить произведенные изменения. Для того чтобы изменения вступили в силу, потребуются перезагрузка компьютера. Если все параметры были указаны верно, после перезагрузки локальная сеть будет активизирована, и компьютеры смогут обмениваться данными.



## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 6. Сравнение технических характеристик серверов

**Цель:** изучить способы организации локальных сетей и их настройку для работы с базой данных.

### *Теоретическая часть*

Многие коммерческие предприятия имеют данные, доступ к которым предлагается пользователям вне компании - другим предприятиям, покупателям или производителям. Например, предприятие может предлагать потенциальным покупателям доступ к дополнительной информации о продуктах предприятия в надежде увеличить сбыт. Должны быть учтены и нужды служащих предприятия. Например, доступной может быть служебная информация о графиках работы и отпусков, обучении персонала, политике компании и т.п. Подобная база данных может быть создана и сделана легко доступной для пользователей средствами SQL и Internet.

Прикладная часть клиент-серверного приложения (back-end application) состоит из сервера базы данных, источников данных и соответствующего промежуточного программного обеспечения, используемого для подключения приложения к Web или удаленной базе данных по локальной сети.

Еще раз напомним, что к наиболее распространенным серверам баз данных относят Oracle, Informix, Sybase, Microsoft SQL Server и Borland InterBase. Именно с сервера начинается разворачивание приложения баз данных либо на все предприятие в рамках локальной сети (LAN) или сети intranet предприятия, либо в Internet. Разворачивание (porting) представляет собой процесс внедрения приложения в среду, доступную пользователям. Сервер базы данных должен быть установлен на рабочем месте администратора базы данных, который, в свою очередь, должен понимать и производственные потребности предприятия, и требования самого приложения.

Промежуточное программное обеспечение приложения состоит из сервера Web и средств, позволяющих подключить сервер Web к серверу базы данных. Главной целью в данном случае является наличие в Web приложения, предоставляющего доступ к корпоративным данным.

### *Интерфейсная часть*

Интерфейсная часть приложения (front-end application) - это та часть приложения, с которой имеет дело пользователь. Интерфейсная часть приложения может быть коммерческим продуктом какой-нибудь компании, производящей программное обеспечение на продажу, либо продуктом, разработанным внутри предприятия с применением различных программных средств.

До того, как на рынке сложилось имеющееся сегодня разнообразие приложений, предлагающих интерфейс пользователя базы данных, пользователю необходимо было уметь программировать на языках типа C



HTML или любом другом из множества процедурных языков программирования, с помощью которых разрабатывались приложения для Web. Языки типа ANSI C, COBOL, FORTRAN или Pascal использовались для разработки интерфейсной части внутри предприятия, и соответствующий интерфейс пользователя был, как правило, текстовым. Сегодня большинство новых приложений интерфейсной части предлагают графический пользовательский интерфейс (GUI).

Интерфейсная часть приложения призвана обеспечить пользователю простоту доступа к базе данных и работы с ней. Внутренние процессы программный код и происходящие в ней события должны быть незаметными для пользователя. Интерфейсная часть приложения должна быть разработана для того чтобы по возможности избавить пользователя от необходимости теряться в догадках и интуитивно чувствовать систему в целом. Новые технологии позволяют сделать приложения более понятными и простыми в применении, что дает пользователю возможность сосредоточиться на решении своих конкретных задач, повышая в конечном итоге эффективность своего труда.

Имеющиеся на сегодня средства разработки приложений достаточно просты в применении и объектно-ориентированны, что достигается использованием в них пиктограмм, возможностей перетаскивания объектов с помощью мыши, а также различных мастеров, автоматически генерирующих объекты с заданными свойствами. Среди наиболее популярных средств разработки Web-приложений следует упомянуть C++Builder, IntraBuilder фирмы Borland и Visual J++, C++ фирмы Microsoft. Для разработки программ, предназначенных для работы в рамках локальной сети предприятия, используют PowerBuilder фирмы Powersoft, Developer/2000 фирмы Oracle Corporation, Visual Basic фирмы Microsoft и Delphi фирмы Borland.

Прикладная часть располагается на сервере, там же размещается и сама база данных. Пользователями прикладной части являются разработчики базы данных, программисты, администраторы базы данных, системные администраторы и системные аналитики. Интерфейсная часть приложения размещается на машинах-клиентах, которыми обычно являются персональные компьютеры конечных пользователей. Интерфейсная часть приложения рассчитана на самую широкую аудиторию пользователей, включающую операторов ввода данных, бухгалтеров и т. д. Пользователь должен иметь возможность доступа к базе данных по сети-и такая сеть может быть как локальной (LAN), так и глобальной (WAN). Для предоставления пользователю такой возможности используется промежуточное программное обеспечение (например, драйвер ODBC).

Удаленный доступ к базе данных

База данных может быть локальной, и тогда вы имеете возможность подключиться к ней непосредственно. Но, как правило, пользователю требуется доступ к базе данных, которая находится на некотором удалении от его системы. Удаленная база данных - это некоторая база данных, не

являющаяся локальной, т. е. расположенной на том сервере, к которому вы подключены в данный момент, и предполагающая для доступа к ней использование сети и определенных сетевых протоколов.

Доступ к удаленной базе данных можно осуществить несколькими способами. Говоря в общем, доступ к удаленной базе данных осуществляется с помощью Подключения к сети или Internet посредством использования промежуточного программного обеспечения (например, стандартных средств ODBC).

Локальный сервер базы данных и главный локальный сервер часто оказываются одним и тем же объектом, поскольку база данных, как правило, размещается на главном локальном сервере. Но подключиться к удаленной базе данных с главного локального сервера можно и без подключения к локальной базе данных. Для конечного пользователя чаще всего предлагается подключение к удаленной базе данных средствами интерфейсного приложения. Во всех этих случаях используется передача запросов к базе данных по сети.

Технология ODBC (Open Database Connectivity - открытый интерфейс доступа к базам данных) обеспечивает возможность доступа к удаленным базам данных с помощью подходящего драйвера. Драйвер ODBC используется интерфейсным приложением для получения доступа к прикладной части базы данных для взаимодействия с данными нижнего уровня. Для доступа к удаленной базе данных, кроме того, может понадобиться и сетевой драйвер. Приложение вызывает функции ODBC, а соответствующий драйвер обеспечивает загрузку драйвера ODBC. Драйвер ODBC обрабатывает вызов функции, пересылает запрос к базе данных и возвращает результат этого запроса. На сегодня ODBC является стандартом, используемым целым рядом продуктов, в частности, PowerBuilder, FoxPro, Visual C++, Visual Basic, Delphi, Microsoft Access и многими другими.

Как часть ODBC, любой производитель систем управления базами данных предлагает со своими базами данных программный интерфейс приложения (API). Для примера из таких предлагаемых продуктов можно отметить Open Call Interface (OCI) фирмы Oracle и SQLGateway или SQLRouter фирмы Centura.

#### Другие интерфейсы доступа к данным

В дополнение к драйверу ODBC многие производители систем управления базами данных предлагают свое программное обеспечение, предназначенное для организации доступа к удаленным базам данных. Каждый из таких продуктов оказывается специфическим для системы управления базами данных конкретного производителя и, вообще говоря, не предполагает переносимости на другие типы серверов баз данных.

Oracle Corporation для организации доступа к удаленным базам данных предлагает свой продукт под именем Net8. Net8 можно использовать практически с любыми сетевыми протоколами, в частности, TCP/IP, OSI, SPX/IPX и многими другими. Кроме того, Net8 может работать под

управлением почти любой из наиболее распространенных операционных систем.

Sybase Incorporated предлагает свой продукт под именем Open Client/C Developers Kit, поддерживающий продукты других производителей, в частности Net8 фирмы Oracle.

Доступ к удаленным базам данных с помощью интерфейса Web

Доступ к удаленным базам данных посредством интерфейса Web подобен доступу к базам данных по локальной сети. Главное отличие состоит в том, что в Web все запросы к базе данных направляются через Web-сервер.

Конечный пользователь инициирует доступ к удаленной базе данных с помощью браузера Web. Браузер Web используется для связывания с заданным URL или адресом IP в Internet, соответствующим нужному серверу Web. Сервер Web, проверив имя и пароль пользователя, пересылает пользовательский запрос базе данных, которая, в свою очередь, тоже может потребовать проверки имени и пароля. Затем сервер базы данных вернет результаты запроса серверу Web, а последний отобразит эти результаты в окне пользовательского браузера Web. Несанкционированный доступ к конкретному серверу может пресекаться с помощью брандмауэра (firewall).

Брандмауэр (firewall) - это аппаратно-программная система межсетевой защиты от несанкционированного доступа к серверу. Для защиты от несанкционированного доступа к серверу может использоваться как одна, так и несколько таких систем. Точно также одна или несколько систем защиты могут использоваться для управления доступом к серверу базы данных и к самой базе данных.

При пересылке информации по Web следует принять все возможные меры безопасности на всех уровнях. К таким уровням можно отнести сервер Web, главный локальный сервер и удаленную базу данных. Частные данные, такие как идентификационные номера служащих, всегда должны быть защищены от доступа к ним случайных лиц и не должны распространяться в Web. SQL и Internet

SQL можно использовать в рамках приложений, создаваемых средствами C или COBOL. Точно так же SQL можно использовать и в Internet-приложениях, создаваемых средствами таких языков программирования, как Java. Текст HTML тоже можно транслировать в запрос SQL, чтобы потом с помощью интерфейсного приложения Web переслать этот запрос удаленной базе данных. Возвращенный базой данных результат затем транслируется обратно в текст HTML и отображается браузером Web на экране пользователя, пославшего запрос. В следующих разделах использование SQL в Internet обсуждается подробнее.

С изобретением и распространением Internet по всему миру данные стали доступными покупателям и производителям в любой стране. Такие данные обычно бывают доступными только для чтения с помощью соответствующего интерфейсного приложения.

Предназначенные для покупателей данные могут состоять из общей информации для покупателей, информации о конкретных продуктах, бланков заказов, информации о текущих и выполненных ранее заказах и т.д. Частная информация, например, информация о корпоративной стратегии и о служащих компании доступной быть не должна.

Наличие информационной странички в Web стало почти обязательным для компаний, стремящихся успешно конкурировать с другими в своем бизнесе. Страничка Web оказывается весьма эффективным средством информирования большого числа потенциальных покупателей об услугах, продуктах и других аспектах деятельности компании, не требуя при этом чрезмерных затрат.

Предоставление доступа к данным служащим и привилегированным клиентам

С помощью Internet или сети intranet компании базу данных можно сделать доступной для служащих этой компании или ее клиентов. Использование технологий Internet оказывается весьма удобным для информирования служащих о политике компании, преимуществах работы в ней, обучающих программах и т. п.

Интерфейсные приложения Web, использующие SQL

Имеется целый ряд приложений, обеспечивающих доступ к базам данных. Многие из таких приложений предлагают графический интерфейс пользователя, так что пользователю даже нет необходимости понимать SQL, чтобы составить запрос к базе данных. В таких приложениях пользователю предлагается указывать и щелкать мышью на объектах, представляющих таблицы, манипулировать данными этих объектов, задавать критерии отбора возвращаемых данных и т. д. Такие приложения часто разрабатываются и настраиваются в полном соответствии с конкретными требованиями конкретной компании. SQL и intranet

IBM изначально создавала SQL для доступа к базам данных, размещенным на мэйнфреймах, с клиентских машин пользователей. Пользователи при этом связывались с мэйнфреймами по локальной сети. Позже SQL стал стандартным языком коммуникации пользователей с базами данных. Intranet, по сути, является миниатюрным аналогом Internet. Основным различием между ними является то, что intranet предназначена для использования внутри некоторой организации, а Internet открыта для доступа всем. Пользовательский (клиентский) интерфейс в intranet остается тем же, что и в модели клиент/сервер. Запросы SQL направляются базе данных сервером Web с использованием соответствующего языка (например, HTML).

Безопасность в рамках базы данных значительно выше, чем в Internet. Поэтому всегда используйте средства безопасности, предлагаемые вашим сервером базы данных.

### **Ход работы:**

Задание 1. Войдите в Internet и ознакомьтесь с информационными страницами нескольких из представленных там компаний. Если ваша компания тоже имеет информационную страницу в Web, сравните ее с информационными страницами конкурентов. Ответьте для себя на следующие вопросы в отношении просмотренных страниц.

а. Открывается ли страница быстро или ее открытие тормозится наличием слишком большого числа графических изображений?

б. Интересно ли читать представленную на странице информацию?

в. Получили ли вы в результате чтения имеющейся на странице информации представление о предлагаемых компанией услугах и продуктах и о компании в целом?

г. Если на странице предлагается доступ к некоторой базе данных, то достаточно ли быстро осуществляется такой доступ?

д. Можно ли сделать вывод об использовании на данной странице Web каких-либо средств безопасности?

Задание 2. Если в вашей компании используется intranet, войдите в сеть и посмотрите, какая информация о компании там представлена. Доступна ли там какая-нибудь база данных? Если да, то кто является производителем соответствующей системы управления базами данных? Какого типа интерфейсные приложения предлагаются при этом конечному пользователю?

## **ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 7. Формирование аппаратных требований и схемы банка данных**

### *Теоретическая часть*

#### *Основные понятия банков данных и знаний.*

Сегодня трудно себе представить сколько-нибудь значимую информационную систему, которая не имела бы в качестве основы или важной составляющей базу данных. Концепции и технологии баз данных складывались постепенно и всегда были тесно связаны с развитием систем автоматизированной обработки информации.

Использование баз данных и информационных систем становится неотъемлемой составляющей деятельности современного человека и организаций, шагающих в ногу со временем. В связи с этим большую актуальность приобретает построение и эффективное применение соответствующих технологий и программных продуктов.

Перерабатывать большой объем информации в заданные сроки без специальных средств практически невозможно. К сожалению, большая часть информации еще находится вне ЭВМ, что объясняется отсутствием достаточного количества и номенклатуры технических средств обработки. Но если учесть, что стоимость ЭВМ снижается, то можно предположить, что в перспективе машинная обработка информации будет основной повсеместно. В ЭВМ могут храниться и обрабатываться не только печатные тексты, но и чертежи, фотографии, запись голосов и т. д.

Методы организации процессов обработки информации, реализуемые в концепции банков данных и знаний, позволили по-новому подойти к их реализации в автоматизированных системах. Рассматривая данные как один из ресурсов автоматизированных систем (АС), можно сказать, что банк данных (БнД) централизованно управляет этим ресурсом в интересах всей системы. Наличие централизованного управления данными — главная отличительная черта банка данных.

Таким образом, банк данных — это информационная система, реализующая централизованное управление данными в интересах всех пользователей АС, в состав которой она входит.

Предметная область — это область применения конкретного БнД. Различают банки данных, применяемые в сфере управления предприятиями и организациями, транспортом, в медицине, научных исследованиях и т. д.

Банки данных возникли в связи с потребностью в интеграции данных. Дальнейшее продвижение в этом направлении потребовало решения проблемы интеграции и процессов обработки, что привело к появлению банков знаний.

В банках знаний решаемые задачи интегрируются как по данным, так и по их обработке, возрастает интеллектуализация этих систем, цель которой — максимальное удовлетворение потребностей пользователей. Использование формальных методов преобразования и интерпретации данных позволяет ав-

томатизировать процессы обработки накопленных в системе знаний, их получение и синтез. Одной из отличительных особенностей банков знаний является наличие в них так называемого интеллектуального интерфейса, в состав которого входят база знаний, процессор общения и программа-планировщик. Интеллектуальный интерфейс решает проблему перевода текста, написанного на естественном языке и содержащего условие задачи, в рабочую программу решения этой задачи на ЭВМ. От пользователя требуется ввести в систему корректную постановку задачи, которая его интересует, на том профессиональном языке, на котором он работает в своей предметной области, а интеллектуальный интерфейс должен выполнить всю работу, которую ранее выполнял программист.

Банки данных и знаний являются одним из основных компонентов автоматизированных систем различных уровней и типов. Их создают для многих отраслей и сфер народного хозяйства: планирования, учета, управления предприятиями, статистики, здравоохранения и др.

Концепция банков данных стала определяющим фактором при создании систем автоматизированной обработки информации. Рассматривая общие вопросы, связанные с функционированием баз данных, обсуждаются основные компоненты банков данных и получивший наибольшее распространение трехуровневый подход к построению банков данных, включающий внешний, концептуальный и внутренний уровни представления данных.

Банк данных является современной формой организации хранения и доступа к информации. Существует много определений банка данных. Мы будем использовать следующее определение: банк данных — это система специальным образом организованных данных (баз данных), программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.

Массив данных, хранимый в вычислительной системе, называют базой данных. База данных вместе с системой управления ею является составной частью банка данных.

База данных (БД) – именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области.

Банк знаний (БнЗ) — это автоматизированная система, содержащая различные виды знаний (например, концептуальные, понятийные знания) о предметной области. Хранящиеся в БнЗ знания используются для вывода новых знаний на основании специальных механизмов, имеющихся в БнЗ.

С БнД и БнЗ в процессе их создания и эксплуатации взаимодействуют пользователи различных категорий, основными из которых являются конечные пользователи. Ими являются специалисты предметных областей, для удовлетворения информационных потребностей которых и создаются БнД и БнЗ. Конечные пользователи различаются сферой интересов, информационными потребностями, квалификацией и т. п. Конечными пользователями могут быть как физические лица, так и различные вычислительные процессы,

задачи, а иногда и целые системы, взаимодействующие с БНД и БНЗ. Во всех случаях результатом взаимодействия является информация, данные, знания. Банк данных – основной элемент информационной системы, в нем хранится информация по определенной проблеме в виде, допускающем компьютерную обработку. При этом сами данные образуют базу данных, а банк, наряду с базой, содержит программные средства обработки данных и реализации запросов, т. е. систему управления базой данных (СУБД). Как правило, банки данных являются системами коллективного пользования и информация, хранящаяся в них, доступна по телекоммуникационным сетям. В современном мире существует огромное число банков данных. В них содержатся сведения коммерческого характера, данные по библиотечным фондам, системам здравоохранения, транспорта и т. д.

Иногда в составе банка данных выделяют архивы. Основанием для этого является особый режим использования данных, когда только часть данных находится под оперативным управлением СУБД. Все остальные обычно располагаются на носителях, не управляемых СУБД. Одни и те же данные в разные моменты времени могут входить как в базы данных, так и в архивы. Банки данных могут не иметь архивов, но если они есть, то состав банка данных может входить и система управления архивами.

Эффективное управление внешней памятью являются основной функцией СУБД. Эти специализированные средства настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет решать некоторые задачи уже потому, что их выполнение будет занимать слишком много времени. При этом ни одна из таких специализированных функций (построение индексов, буферизация данных, организация доступа и оптимизация запросов) не является видимой для пользователя и обеспечивает независимость между логическим и физическим уровнями системы: прикладной программист не должен писать программы индексирования, распределять память на диске и т. д.

Основными требованиями, предъявляемыми к БНД, являются: адекватность отображения предметной области (полнота, целостность и непротиворечивость данных, актуальность информации (т. е. ее соответствие состоянию объекта на данный момент времени));

возможность взаимодействия пользователей разных категорий и в разных режимах, обеспечение высокой эффективности доступа для разных приложений;

дружелюбность интерфейсов и быстрое освоение системы, особенно для конечных пользователей;

обеспечение секретности и конфиденциальности для некоторой части данных;

определение групп пользователей и их полномочий; обеспечение взаимной независимости программ и данных; обеспечение надежности функционирования БНД, защита данных от случайного и преднамеренного разрушения;



возможность быстрого и полного восстановления данных в случае их разрушения;

технологичность обработки данных, приемлемые характеристики функционирования БД (стоимость обработки, время реакции системы на запросы, требуемые машинные ресурсы и др.).

Компоненты банка данных

Банк данных является сложной человеко-машинной системой, включающей в свой состав различные взаимосвязанные и взаимозависимые компоненты.

Информационная компонента. Ядром БД является база данных (БД). База данных — это поименованная совокупность взаимосвязанных данных, находящихся под управлением системы управления базой данных.

Существует множество определений базы данных. Некоторые из них имеют право на существование. Другие устарели и не соответствуют современным представлениям о БД. Так, в ранних определениях базы данных указывалось на их не избыточность, отсутствие дублирования данных в них. На самом деле это не так. В базах данных может наблюдаться избыточность информации. Она может быть вызвана спецификой используемой модели данных, не позволяющей полностью устранить дублирование, или технологическими причинами (обеспечение большей надежности, сокращение времени реакции системы и др.). Но это должна быть управляемая избыточность, причины и цели возникновения которой известны администратору базы данных и управляются как им, так и системой управления базами данных (СУБД).

Системой управления базой данных называется совокупность языковых и программных средств, облегчающих для пользователей выполнение всех операций, связанных с организацией хранения данных, их корректировкой и доступом к ним.

Программные средства БД представляют собой сложный комплекс, обеспечивающий взаимодействие всех частей информационной системы при ее функционировании.

Основу программных средств БД представляет СУБД. В ней можно выделить ядро СУБД, обеспечивающее организацию ввода, обработки и хранения данных, а также другие компоненты, обеспечивающие настройку системы, средства тестирования, утилиты, обеспечивающие выполнение вспомогательных функций (восстановление баз данных, сбор статистики о функционировании БД и др.). Важной компонентой СУБД являются трансляторы или компиляторы для используемых ею языковых средств.

Управляет базой данных администратор базы данных. Необходимо отметить, что при рассмотрении всего контура управления базой данных, следует учитывать и операционную систему (ОС), поскольку программы управления базой данных выполняются непосредственно под управлением ОС. Подавляющее большинство СУБД работает в среде универсальных операционных систем и взаимодействует с ОС при обработке обращений к БД.

Для обработки запросов к БД пишутся соответствующие программы, которые представляют прикладное программное обеспечение БД.

Языковые средства СУБД являются важнейшей компонентой банков данных, так как в конечном счете они обеспечивают интерфейс пользователей разных категорий с банком данных. Языковые средства большинства СУБД относятся к языкам четвертого поколения (к первому поколению языков относят машинные языки, ко второму — символические языки ассемблера, к третьему — алгоритмические языки типа PL, COBOL и т. п., которые в 60-е годы XX в. назывались языками высокого уровня, но уровень которых гораздо ниже, чем у языков четвертого поколения).

В качестве технических средств для БНД используется ЭВМ. В технической документации некоторых СУБД, а также в некоторых литературных источниках в состав БД включаются не только собственно хранимые данные о предметной области, но и описания БД. Более правильно описания баз данных считать самостоятельными компонентами БНД, даже если они и хранятся вместе с самими данными.

Организационно-методические средства представляют собой различные инструкции, методические и регламентирующие материалы, предназначенные для пользователей разных категорий, взаимодействующих с БНД.

Функционирование БНД невозможно без администраторов БНД — специалистов, обеспечивающих создание, функционирование и развитие БНД.

Структура банка данных. БНД — совокупность специальным образом организованных (структурированных) данных и связей между ними. Иными словами, БД — это так называемое датологическое (от англ. data — данные) представление информации о предметной области.

Если в состав БНД входит одна БД, банк принято называть локальным, если БД несколько — интегрированным.

ВС — вычислительная система, включающая технические средства и общее программное обеспечение, базы данных, систему управления базами данных, администратора баз данных (АБД), а также обслуживающий персонал и словарь данных.

В состав любой СУБД входят языки двух типов:

язык описания данных (с его помощью описываются типы данных, их структура и связи);

язык манипулирования данными (его часто называют языком запросов к БД), предназначенный для организации работы с данными в интересах всех типов пользователей.

Словарь данных предназначен для хранения единообразной и централизованной информации обо всех ресурсах данных конкретного банка: об объектах, их свойствах и отношениях для данной ПО;

данных, хранимых в БД (наименование, смысловое описание, структура, связи и т. п.);

возможных значениях и форматах представления данных;

источниках возникновения данных;

кодах защиты и разграничении доступа пользователей к данным и т. п.

## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 8. Установка и настройка сервера MySQL. Установка и настройка сервера

**Цель:** Познакомиться с основными принципами создания базы данных в MS SQL Server. Изучить операции, проводимые с базами данных в целом. Получить навыки использования программы "SQL Server Management Studio" для создания, удаления, регистрации, подключения, извлечения метаданных, резервного копирования и восстановления базы данных.

*Исходные данные.*

Студент получает индивидуальный вариант исходных данных с кратким описанием предметной области, который используется при выполнении всех описанных в данном пособии практических работ. При этом каждая очередная практическая работа является продолжением выполненной ранее и поэтому они должны обязательно выполняться последовательно.

Используемые программы:

1. Работающий на компьютере сервер "MS SQL Server 2008 R2". 2. Установленная платформа .NET Framework 2.0, 3.0, 3.5 или 4.0.
2. Операционная система Microsoft Windows 2000/XP/2003/Vista/Windows 7/Windows 8.
3. Приложение "SQL Server Management Studio 2008 rus", установленное на локальном компьютере.

*Теоретические сведения.*

На сегодняшний день известно более двух десятков серверных СУБД, из которых наиболее популярными являются Oracle, Microsoft SQL Server, Informix, DB2, Sybase, InterBase, MySQL.



Рисунок 22 – серверные СУБД

Для выполнения практических работ будет использоваться сервер "Microsoft SQL Server 2008".

Microsoft® SQL Server™ – это система анализа и управления реляционными базами данных в решениях электронной коммерции, производственных отраслей и хранилищ данных.

Microsoft SQL Server – система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов – SQL, создан совместно Microsoft и Sybase. SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

В SQL Server 2008 имеется большой набор интегрированных служб, расширяющих возможности использования данных: вы можете составлять запросы, выполнять поиск, проводить синхронизацию, делать отчеты, анализировать данные. Все данные хранятся на основных серверах, входящих в состав центра обработки данных. К ним осуществляется доступ с настольных компьютеров и мобильных устройств. Таким образом, вы полностью контролируете данные независимо от того, где вы их сохранили.

Система SQL Server 2008 позволяет обращаться к данным из любого приложения, разработанного с применением технологий Microsoft .NET и Visual Studio, а также в пределах сервисно-ориентированной архитектуры и бизнес-процессов — через Microsoft BizTalk Server. Сотрудники, отвечающие за сбор и анализ информации, могут работать с данными, не покидая привычных приложений, которыми они пользуются каждый день, например приложений выпуска 2007 системы Microsoft Office.

В Microsoft SQL базы данных хранятся в виде обычных файлов на диске. Как минимум на одну БД приходится таких файлов 2: \*.mdf и \*.ldf. В первом хранятся сами данные, таблицы, индексы и пр., а во втором находится т.н. transaction log, в котором находится информация необходимая для восстановления БД.

Файл с базой данных представляет собой набор страниц одинакового размера. Размер страницы задается при создании базы данных и может быть изменен только при ее восстановлении из резервной копии. Чтение и запись данных в базе данных осуществляется постранично.

Все операции с базой данных должны производиться только посредством команд к SQL-серверу. Для клиентских приложений эти файлы абсолютно бесполезны и при правильной организации доступа пользователей к файлам в сети, вообще не должны быть доступны.

Сервер СУБД не имеет интерфейса пользователя и для выполнения операций с базой данных ему необходимо посылать команды либо с помощью командной строки или с помощью какой-либо прикладной программы.

Для выполнения операций с базой данных при проведении практических работ предлагается использовать программу " SQL Server Management Studio 2008 Rus"(рис. 23), представляющую собой наиболее распространенное и удобное средство администрирования баз данных под управлением MS SQL Server (Среда Management Studio Express доступна для свободной загрузки из центра загрузки Майкрософт - [http://download.microsoft.com/download/5/C/0/5C0C5CE4-10EB-4623-A63E8D850D55D8EF/SQLEXPRESS\\_x86\\_RUS.exe](http://download.microsoft.com/download/5/C/0/5C0C5CE4-10EB-4623-A63E8D850D55D8EF/SQLEXPRESS_x86_RUS.exe) ).

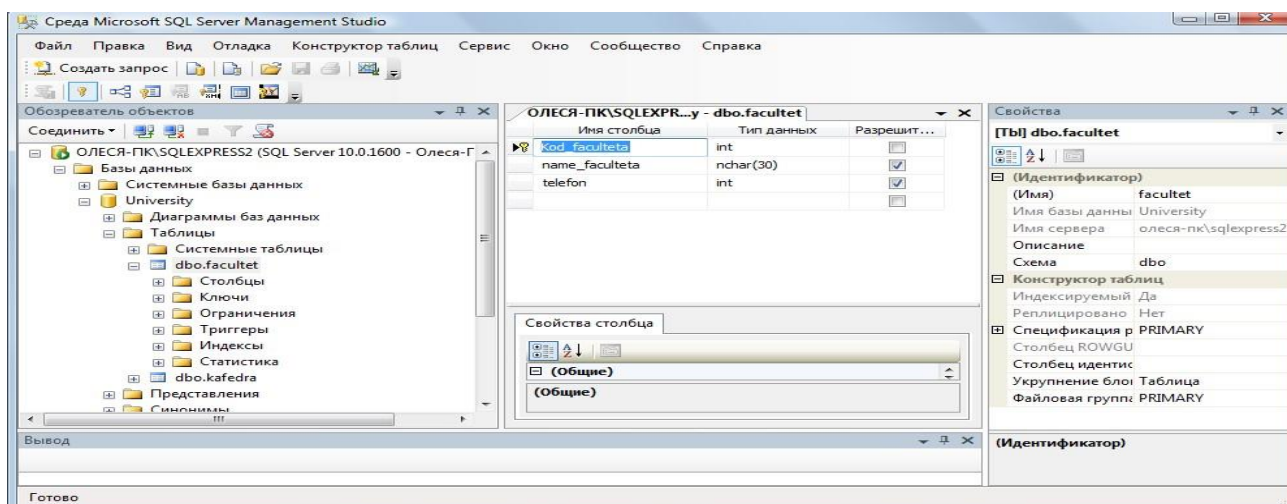


Рисунок 23 – Программа SQL Server Management Studio

Среда SQL Server Management Studio — это интегрированная среда для доступа, настройки, управления, администрирования и разработки всех компонентов SQL Server. Среда SQL Server Management Studio объединяет большое число графических средств с набором полнофункциональных редакторов сценариев для доступа к SQL Server разработчиков и администраторов с любым опытом работы.

Среда SQL Server Management Studio обеспечивает следующие основные возможности:

- поддерживает большинство административных задач для SQL Server; § единая интегрированная среда для управления SQL Server Database Engine и разработки;

- новые управляющие диалоговые окна для управления объектами в компоненте SQL Server Database Engine, службах Analysis Services, Reporting Services, Notification Services и выпуске SQL Server Compact 3.5 с пакетом обновления 1 (SP1), позволяющие выполнять действия немедленно, направлять их в редактор кода или включать эти действия в сценарий для последующего выполнения;

- экспорт и импорт регистрации сервера среды SQL Server Management Studio из одной среды Management Studio в другую;

- сохранение и печать XML-файлов плана выполнения и взаимоблокировок, созданных приложением SQL Server Profiler, просмотр их в любое время и отправка для анализа администратору;

- новые окна сообщений об ошибках и информационных сообщений, предоставляющие гораздо больше сведений и позволяющие отправлять в Майкрософт комментарии о сообщениях, копировать сообщения в буфер обмена и отправлять их по электронной почте в службу поддержки;

- встроенный веб-обозреватель для быстрого обращения к библиотеке MSDN или получения интерактивной справки;

- встроенная справка от сообществ в Интернете и т.д.

Большинство действий с базой данной MS SQL Server в среде Среда SQL Server Management Studio может быть осуществлено двумя способами: либо выполнением операторов языка SQL в окнах "Script Execute" (подключение к базе данных не обязательно) и "SQL Editor" (требуется

подключение к базе данных), либо с использованием меню и диалоговых окон. В последнем случае операторы SQL, которые требуются для выполнения данного действия, будут сгенерированы и выполнены средой SQL Server Management Studio автоматически.

**Задание:** Практическую работу следует выполнять в следующем порядке:

Создать на сервере pi\_srv (или на локальном компьютере, если нет сервера) рабочую папку для хранения файлов, получаемых при выполнении практической работы. Эта папка должна располагаться в папке \Базы данных\Группа\Студент и соответствовать номеру выполняемой практической работы.

На основании индивидуального задания выбрать имя файла создаваемой базы данных. Для имени лучше всего выбрать одно или несколько английских слов, соответствующих наименованию предметной области. Использование для имени русских слов, записанных латинскими буквами, не допускается.

Открыть приложение "Среда SQL Server Management Studio". Для этого можно либо воспользоваться меню Пуск (Пуск/Программы/ Microsoft SQL Server 2008 / Среда SQL Server Management Studio).

Создать соединение с локальным или удаленным сервером.

Создать базу данных для своей предметной области с помощью диалога, выбрав сервер "pi\_srv" или локальный сервер "Имя\_компьютера\SQLEXPRESS".

Создать базу данных и указать в качестве имени файла "\Базы данных\Группа\ФИО\_студента\Название\_БД".

Извлечь метаданные для автоматической генерации команды создания базы данных.

Удалить базу данных, выполнив команду "Database/Drop Database" (База данных/Удалить базу данных).

Создать базу данных вторым способом, выполнив в окне "Script Executive" операторы, полученные при извлечении метаданных перед предыдущим удалением.

Создать резервную копию базы данных.

Удалить базу данных.

Восстановить базу данных из резервной копии.

Сохранить файл сценария на сервере в папке "Студент", дав ему имя «лаб.№1» и стандартное расширение "\*.sql".

### **Ход работы:**

Создание соединения с сервером.

Выполните следующие инструкции:

Работа с приложением SQL Server Management Studio начинается с создания соединения с установленным сервером. Убедитесь вначале, что сервер Microsoft SQL

Server (2008) на локальной машине или на сервере компьютерного класса установлен и работает.



Откройте приложение " SQL Server Management Studio ". Для этого можно либо воспользоваться меню Пуск (Пуск/Программы/ Microsoft SQL Server 2008 / Среда SQL Server Management Studio).

В диалогом окне Соединение с сервером подтвердите заданные по умолчанию параметры и нажмите кнопку Соединить, см. рис.24.

Для соединения необходимо, чтобы поле Имя сервера содержало имя компьютера, на котором установлен SQL Server.

Если компонент Database Engine является именованным экземпляром, то поле Имя сервера должно также содержать имя экземпляра в формате <имя\_компьютера>\<имя\_экземпляра>.

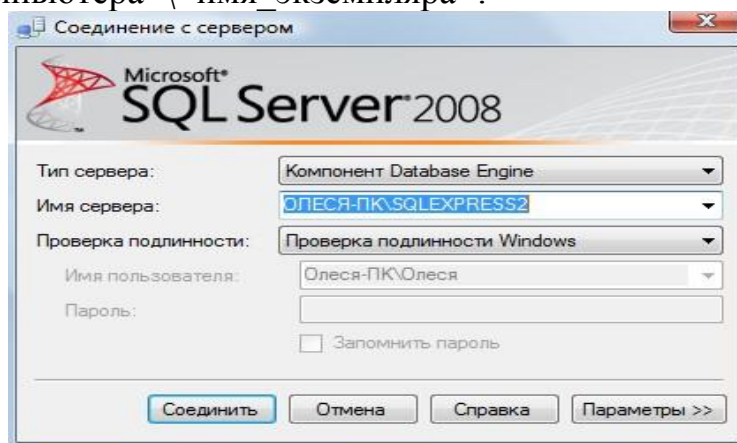


Рисунок 24 – Создание соединения с сервером

В параметрах указываем:

Тип сервера – Компонент Database Engine.

Имя сервера. Подключение может быть локальным или удаленным. Представляет собой название компьютера в сети, на котором установлен сервер СУБД. Если сервер установлен на том же компьютере, где сейчас работает пользователь, то в качестве имени используется имя компьютера и идентификатор сервера; проверка подлинности – Windows (по умолчанию), имя пользователя – имя пользователя по умолчанию, зарегистрированного на сервере MS SQL Server (задается при установке сервера), пароль – пусто или пароль для пользователя, заданного для сервера MS SQL Server;

Нажмите кнопку Соединить. Если соединение будет совершенно успешно, то на экране появятся данные сервера.

Среда Management Studio представляет данные в виде окон, выделенных для отдельных типов данных. Сведения о базе данных отображаются в обозревателе объектов и окнах документов.

Обозреватель объектов является представлением в виде дерева, в котором отображаются все объекты базы данных на сервере. Он может содержать базы данных компонента SQL Server Database Engine, служб Analysis Services, служб Reporting Services, служб Integration Services и SQL Server Compact 3.5 с пакетом обновления 1 (SP1).

Обозреватель объектов включает сведения по всем серверам, к которым он подключен. При открытии среды Management Studio пользователю предлагается применить при подключении обозревателя объектов параметры, которые использовались в прошлый раз. Чтобы подключиться к любому из серверов, следует дважды щелкнуть его в

компоненте «Зарегистрированные серверы», однако регистрировать его не обязательно.

Окно документов представляет собой наиболее крупную часть среды Management Studio. В окнах документов могут размещаться редакторы запросов и окна обзора. По умолчанию отображается страница «Сводка», подключенная к экземпляру компонента Database Engine на текущем компьютере.

Общие сведения о базах данных MS SQL Server.

Кроме четырех системных баз, SQL Server может обрабатывать до 32 734 баз данных, определяемых пользователем.

База данных представляет собой:

- набор взаимосвязанных таблиц;
- связанный набор страниц, выделенных для хранения данных MS SQL Server;
- совокупность данных при архивации;
- два и более файла;
- важную совокупность данных для целей защиты и управления.

Файлы базы данных.

База данных состоит из двух и более файлов, каждый из которых может использоваться лишь одной базой.

У файлов существуют два имени: логическое и физическое. Логическое имя подчиняется стандартным правилам выбора имен объектов SQL Server. Физическое имя представляет собой полное имя любого локального или сетевого файла. Максимальное число файлов в базе данных — 32 768. Файлы делятся на три типа:

Первичные файлы. Используются для хранения данных и информации, определяющих начальные действия с базой. База данных содержит лишь один первичный файл. Стандартное расширение — .mdf.

Вторичные файлы. Одна или несколько вспомогательных областей для хранения данных. Могут использоваться для распределения операций чтения/записи по нескольким дискам. Стандартное расширение — .ndf.

Файлы журналов. Содержат журналы транзакций базы данных. База данных содержит по крайней мере один файл журнала. Стандартное расширение — .ldf. Перед непосредственной записью транзакций в файл данных все вносимые изменения записываются в журнал.

Группы файлов.

Группы файлов предназначены для объединения нескольких файлов. Каждый файл может входить не более чем в одну группу. Файлы журналов не могут принадлежать никаким группам. Группы файлов используются для распределения операций чтения/записи по нескольким дискам. Если группа содержит более одного файла, операции записи распределяются между файлами группы. Базы данных могут содержать до 32 768 групп файлов.

У каждой базы данных имеется первичная группа файлов. Она содержит первичный файл данных и все файлы, которые не были явно назначены в другую группу файлов. Имя первичной группы файлов — PRIMARY.

**Задание:** настроить сервер БД на своем компьютере.\



## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 9. Выполнение запросов к базе данных

**Цель:** Изучить используемый в реляционных СУБД оператор извлечения данных из таблиц. Получить навыки работы с оператором SELECT в программе 'SQL Server Managment Studio'.

### Ход работы:

*Теоретические сведения.*

В SQL имеется единственный оператор, который предназначен для выборки данных из базы данных. Оператор относится к подмножеству DML. Ниже приведен почти полный синтаксис оператора SELECT.

```
SELECT [DISTINCT | ALL]
{* | <величина> [, <величина> ...]}
[INTO :Переменная [, :Переменная ...]]
FROM <tableref> [, <tableref> ...]
[WHERE <условие поиска>]
[GROUP BY Колонка [, Колонка ...]]
[HAVING <условие поиска>]
[UNION [ALL] <select_expr>]
[ORDER BY <список сортировки>];
<величина> = {Колонка | :Переменная | <константа>
| <выражение> | <функция>
| udf ([<величина> [, <величина> ...]])
| NULL | USER} [AS Псевдоним]
<константа> = Число | 'Строка'
<выражение> = SQL выражение, возвращающее единичное значение
<функция> =
COUNT (* | [ALL] <величина> | DISTINCT <величина>)
| SUM ([ALL] <величина> | DISTINCT <величина>)
| AVG ([ALL] <величина> | DISTINCT <величина>)
| MAX ([ALL] <величина> | DISTINCT <величина>)
| MIN ([ALL] <величина> | DISTINCT <величина>)
| CAST(<величина> AS <тип данных>) | UPPER (<величина>)
| GEN_ID (Имя_Генератора, <величина>)
<tableref> = {<joined_table> | table | view
| procedure[(<величина> [, <величина> ...])]}
[Псевдоним]
<joined_table> = <tableref> <join_type> JOIN <tableref> ON <условие
поиска> | (<joined_table>)
<join_type> = [INNER] | {LEFT | RIGHT | FULL } [OUTER]
<условие поиска> =
<величина> <оператор сравнения>
{<величина> | (<select_one>)}
| <величина> [NOT] BETWEEN <величина> AND <величина>
| <величина> [NOT] LIKE <величина>
```

| <величина> [NOT] IN  
 (<величина> [, <величина> ...] | <select\_list>)  
 | <величина> IS [NOT] NULL  
 | <величина> {>= | <=} <величина>  
 | <величина> [NOT] {= | < | >} <величина>  
 | {ALL | SOME | ANY} (<select\_list>)  
 | EXISTS (<select\_expr>)  
 | SINGULAR (<select\_expr>)  
 | <величина> [NOT] CONTAINING <величина>  
 | <величина> [NOT] STARTING [WITH] <величина>  
 | (<условие поиска>)  
 | NOT <условие поиска>  
 | <условие поиска>OR <условие поиска> | <условие поиска>AND  
 <условие поиска>  
 <оператор сравнения> =  
 {= | < | > | <= | >= | != | <> | !=}  
 <select\_one> = оператор SELECT, выбирающий одну колонку и  
 возвращающий ровно одно значение  
 <select\_list> = оператор SELECT, выбирающий одну колонку,  
 возвращающий ноль или много значений  
 <select\_expr> = оператор SELECT, выбирающий несколько величин и  
 возвращающий ноль или много значений  
 <список сортировки> =  
 {Колонка | Номер}  
 [ASC | DESC]  
 [, <список сортировки> ...]  
**Некоторые параметры, входящие в этот оператор, описаны в  
 таблица 4.**

**Таблица 4 Описание параметров оператора.**

Параметр	Описание
DISTINCT   ALL	DISTINCT – предотвращает дублирование данных, которые будут извлечены. ALL (по умолчанию) – приведет к извлечению всех
{*   <величина> [, <величина> ...]}	Звездочка (*) означает, что надо извлекать все колонки из указанных таблиц. <величина> [, <величина> ...] – извлекает список указанных
INTO :Переменная [, :Переменная ...]	колонок, переменных или выражений Используется только в триггерах и хранимых процедурах для операторов SELECT, возвращающих не более одной строки. Указывается список переменных, в которые извлекаются

FROM <tableref> [, <tableref> ...]	величины Указывает список таблиц, просмотров и хранимых процедур, из которых извлекаются данные. Список может включать соединения и соединения могут быть
table	Имя существующей в базе данных таблицы
view	Имя существующего базе данных просмотра
procedure	Имя существующей хранимой процедуры, предназначенной для использования в операторе SELECT
Псевдоним	Короткое альтернативное имя для таблицы, просмотра или колонки. После описания в <tableref>, псевдоним может использоваться для ссылок на таблицу или просмотр
join_type	Задаёт тип соединения, которое может быть внутренним или внешним
WHERE <условие поиска>	Указывает условие, которое ограничивает количество извлекаемых строк
GROUP BY Колонка [, Колонка ...]	Разбивает результат запроса на группы, содержащие все строки с идентичными значениями указанными в списке
HAVING <условие поиска>	Колонок Использует совместно с GROUP BY. Задаёт условие, которое ограничивает количество возвращаемых групп
UNION [ALL]	Объединяет результаты нескольких запросов. Все запросы должны извлекать одинаковое количество столбцов, тип данных каждого столба первого запроса должен совпадать с типом данных других запросов, имена столбцов в разных запросах могут отличаться. Необязательный параметр ALL
ORDER BY <список сортировки>	указывает, что надо выводить одинаковые строки Указывает колонки, по которым будет производиться сортировка извлекаемых строк. Можно указывать либо имена колонок, либо их порядковые номера в списке извлекаемых колонок. Если указать ASC, то строки будут выдаваться в

Таблица 4 Описание параметров оператора SELECT порядке возрастания значений сортируемых полей, если DESC

Как видно из синтаксиса оператора SELECT, обязательными являются только предложение SELECT с перечнем выдаваемых колонок и предложение FROM.

Пример простейшего оператора SELECT:

Выдать перечень всех служащих:

```
SELECT * FROM Employee;
```

Ниже приведено несколько упрощенных вариантов синтаксиса оператора SELECT, помогающих научиться составлять простые запросы.

Упрощенный синтаксис внутреннего соединения (стандарт SQL-92):

```
SELECT Колонка [, Колонка ...] | *  
FROM <tableref_left> [INNER] JOIN <tableref_right>  
[ON <условие поиска>]  
[WHERE <условие поиска>];
```

Упрощенный синтаксис внешнего соединения:

```
SELECT Колонка [, Колонка ...] | *  
FROM <tableref_left>  
{LEFT | RIGHT | FULL} [OUTER] JOIN  
<tableref_right>  
[ON <условие поиска>]  
[WHERE <условие поиска>];
```

Упрощенный синтаксис использования подзапроса:

```
SELECT [DISTINCT] Колонка [, Колонка ...]  
FROM <tableref> [, <tableref> ...]  
WHERE  
{expression {[NOT] IN | <оператор сравнения>}  
| [NOT] EXISTS  
}  
(SELECT [DISTINCT] Колонка [, Колонка ...]  
FROM <tableref> [, <tableref> ...]  
WHERE <условие поиска>  
);
```

5.5. Задание Практическую работу следует выполнять в следующем порядке:

Изучить синтаксис оператора SELECT и примеры запросов к учебной базе данных 'University.mdf'.

Выполнить в окне 'SQL Editor' 27 запроса к базе данных, согласно приведенным в практической работе образцам выполнения запросов и сохранять каждый под именами 'Lab5-k.sql', где k – номер запроса по порядку, в своей рабочей папке. Каждый запрос должен иметь комментарии с описанием, а файл в целом должен иметь комментарии со сведениями об авторе и дате создания.

Примечание. У вас должны быть перед выполнением этой практической работы созданы все таблицы базы данных университета, созданы ключи, а также заполнены данными.


*Выполнение sql-запросов.*

Для выполнения запросов SELECT в программе 'SQL Server Managment Studio' необходимо выполнить следующие действия:

Подключиться к базе данных и выполнить команду ‘Создать запрос’. В результате откроется окно ‘Конструктора запросов’ (рис. 1).



Рисунок 25 – Окно выполнения запросов

Ввести текст запроса согласно рис.1. 3. Нажать на панели инструментов кнопку  [Выполнить] .

Если запрос правильный, то в результате произойдет его выполнение и результат будет отображен на вкладке ‘Результаты’ (рис. 2).

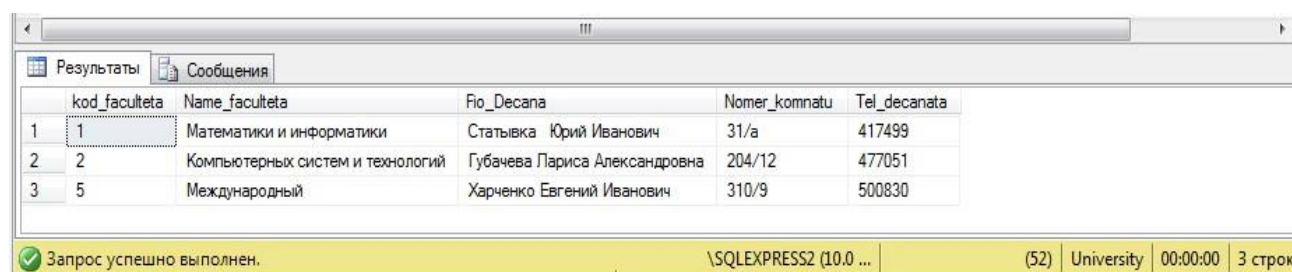


Рисунок 26-Окно с результатом выполнения запроса

Количество извлеченных в результате выполнения запроса строк отображается над сеткой с данными справа. На рис там содержится строка ‘3 строк’. В данном ТРРТ примере извлечено столько строк, сколько требуется, чтобы заполнить сетку (в ней помещается только 3 строки) \* .

Чтобы узнать, сколько всего строк соответствуют выполненному оператору, надо перейти в конец отображаемого набора данных.

Чтобы выполнить другой запрос, надо вернуться на вкладку ‘Редактора’, создать новый запрос и повторить те же действия.

К тексту ранее выполнявшихся правильных запросов можно вернуться, если перейти на вкладку ‘История’.

*Примеры создания запросов с отбором строк по условию.*

SQL дает возможность определить критерии отбора необходимых строк во фразе WHERE предложения SELECT. В этом случае строки исходных таблиц будут включены в результирующую только если строка соответствует указанным критериям. Условие - это выражение, которое может быть истинным или ложным (логическое выражение или предикат), то есть принимать логические значения TRUE или FALSE соответственно. В результирующую таблицу включаются только те строки, для которых указанное во фразе WHERE условие равно TRUE (иными словами, которые удовлетворяют заданному условию).

В случае одной таблицы механизм работы предложения SELECT с фразой WHERE следующий.

Из таблицы, указанной во фразе FROM, выбирается очередная строка. Она проверяется на соответствие условию во фразе WHERE.

Если результат равен TRUE, строка включается в результирующую таблицу и форматируется в соответствии с фразой SELECT, а если он равен FALSE, строка пропускается.

Далее будут рассмотрены основные выражения, допустимые для условия во фразе WHERE.

*Использование простейших условий.*

Простейшими считаются условия, в которых используются операторы сравнения и логические операторы.

Хотя такие условия являются простейшими в смысле семантики конструкций, они могут иметь довольно сложную структуру из многих операторов сравнений и вложенных друг в друга логических связок.


*Операторы сравнения.*

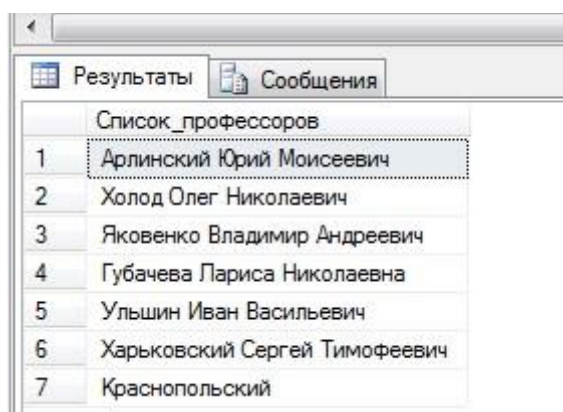
Особенностью операторов сравнения является то, что независимо от типов операндов их результатом являются логические значения. Предположим, вы хотите получить список всех профессоров.

Создайте новый запрос, введите sql-запрос, выполните его, сохраните его в рабочую папку ЛАБ6\_SQL под именем 1.sql.

Запрос 1. Вывести фамилии профессоров.

```
SELECT NAME_TEACHER AS 'Список профессоров'  
FROM TEACHER  
WHERE DOLGNOST = 'профессор';
```

Чтобы выполнить sql-команду нажмите на панели редактора кнопку . В результате выполнения данного кода будут выданы все профессора. Например,



Список_профессоров	
1	Арлинский Юрий Моисеевич
2	Холод Олег Николаевич
3	Яковенко Владимир Андреевич
4	Губачева Лариса Николаевна
5	Ульшин Иван Васильевич
6	Харьковский Сергей Тимофеевич
7	Краснопольский

Рисунок 27 – Схема данных

Слово 'профессор' в запросе является строковой константой, поэтому ее следует заключить в кавычки. Обратите внимание, что мы указали фразу SELECT без ключевого слова DISTINCT, так как тогда от нас была бы скрыта информация о существовании среди профессоров однофамильцев. Чтобы при выводе результирующий столбец имел содержательный заголовок, мы поименовали его как Список профессоров.

Это первый пример использования предиката над строковым типом данных. Здесь столбец строкового типа сравнивается со строковой константой. Запрос выполнен правильно, однако нужно всегда помнить о том, что предикаты над строками являются чувствительными к регистру букв. Например, предикат 'ИВАНОВ' = 'Иванов' будет ложным. Поэтому, если для некоторого профессора его должность была введена в таблицу TEACHER как 'Профессор', он не будет найден по условию WHERE DOLGNOST = 'профессор'.

Чтобы на предикаты над строками не влиял регистр букв, нужно использовать обычно имеющиеся в СУБД функции преобразования букв в прописные и строчные. В стандарте SQL, например, указаны функции UPPER и LOWER, выполняющие такие преобразования. Следовательно, для предыдущего запроса правильной будет записать условие фразы WHERE одним из следующих способов:

```
WHERE LOWER(DOLGNOST) = 'профессор'
```

```
WHERE UPPER(DOLGNOST) = 'ПРОФЕССОР'
```

Самостоятельно измените в исходном запросе строку условия с использованием функции изменения регистра.

Чтобы сохранить запрос нажмите правую кнопку и из контекстного меню выберите Сохранить в файл. Присвойте имя 1.sql и сохраните в папку ЛАБ5\_SQL.

Запрос 2. Найти всех студентов с стипендией, превышающим 300 грн.

В sql-редакторе создайте новый запрос и введите следующий код:

```
SELECT SUTNAME, SUTFNAME
```

```
FROM STUDENT WHERE STIPEND > 300; Выполните его.
```

Приведем несколько примеров использования операторов сравнения для столбцов строкового и временного типа. Обратите внимание, что сравнивать можно не только значение столбца с константой, но и значения столбцов между собой.

Запрос 3. Вывести фамилии и должности преподавателей, принятых на работу после 01.01.2002.

```
SELECT NAME_TEACHER AS 'Фамилия', DOLGNOST AS 'Должность'
```

```
FROM TEACHER
```

```
WHERE DATA_HIRE > '1/01/2002';
```

Запрос 4. Вывести фамилии и должности преподавателей, фамилии которых в алфавитном порядке располагаются после фамилии Сычева.

```
SELECT NAME_TEACHER, DOLGNOST
```

```
FROM TEACHER
```

```
WHERE UPPER(NAME_TEACHER) > 'Сычева';
```

Самостоятельно создайте запрос 5. Вывести фамилии преподавателей, у которых надбавка меньше ставки в 2,5 и более раз.

Логические операторы

Операндами и результатом логических операторов являются логические значения.

Стандартными логическими операторами являются AND, OR и NOT. Их действие показано в так называемых истинностных таблицах. Использование операторов сравнения вместе с логическими операторами предоставляет возможность формулировать составные условия для отбора строк таблиц.

Использование логического оператора AND

Логический оператор AND во многих случаях действует как связка 'и' в русском языке. Рассмотрим несколько примеров с использованием этого оператора.

Запрос 6. Вывести фамилии студентов, проживающих в городе Макарово и имеющих стипендию больше 100 грн.

```
SELECT SUTFNAME
FROM STUDENT
WHERE CITY = 'Макарово' AND STIPEND >100;
```

Самостоятельно создайте запрос 7. Вывести фамилии преподавателей, которые являются профессорами и ставка которых превышает 4500.

Самостоятельно создайте запрос 8. Вывести фамилии студентов учащихся на кафедре под порядковым номером 2 (Прикладная математика) с стипендией в диапазоне 100-500 грн.

Использование логического оператора OR

Логический оператор OR во многих случаях действует как связка 'или' в русском языке. Рассмотрим несколько примеров.

Запрос 9. Вывести названия кафедр, расположенных либо в 1 либо в 8 корпусе.

```
SELECT NAME_KAFEDRU, NUM_KORPUSA
FROM KAFEDRA
WHERE NUM_KORPUSA =1 OR NUM_KORPUSA =8;
```

Использование логического оператора NOT

Логический оператор NOT в русском языке передается словами 'не' и 'кроме'.

Запрос 10. Вывести названия всех факультетов, кроме факультета математики и информатики.

```
SELECT NAME_FACULTETA
FROM FACULTET
WHERE NOT LOWER(NAME_FACULTETA) = 'математики и информатики';
```

Обратите внимание, что оператор NOT должен предшествовать выражению сравнения, а не ставиться перед оператором сравнения. То есть запись LOWER(NAME\_FACULTETA) NOT ="математики и информатики" будет неверной. Учитывая, что отрицанием оператора = является оператор <>, вместо указанного условия можно было бы записать LOWER(NAME\_FACULTETA) <> "математики и информатики". Это относится ко всем операторам сравнения, так как каждый из них имеет оператор, являющийся его отрицанием.



Комбинирование логических операторов.

Логические операторы можно объединять, формируя составные условия. Возможность комбинирования обеспечивается тем, что любой логический оператор возвращает истинностное значение, а значит, его результат может использоваться в другом логическом операторе. Рассмотрим несколько примеров.

Запрос 11. Вывести фамилии, должность, ставку и надбавку ассистентов, у которых либо ставка меньше 550, либо надбавка больше 60.

```
SELECT NAME_TEACHER, DOLGNOST, Salary, Rise
FROM TEACHER
WHERE LOWER(DOLGNOST) ='ассистент' AND
(Salary < 550 OR Rise > 60);
```

Использование выражений над столбцами

До сих пор в выражениях фразы WHERE мы использовали в качестве значения одного или обоих аргументов имена столбцов. Однако аргументами могут быть и выражения над столбцами.

Запрос 12. Показать фамилии преподавателей, чья зарплата (ставка плюс надбавка) превышает 3500.

```
SELECT NAME_TEACHER AS 'Фамилия преподавателя',
Salary + Rise AS 'Его зарплата'
FROM TEACHER
WHERE Salary + Rise > 3500;
```

Запрос 13. Показать фамилии преподавателей, половина зарплата которых превышает пятикратную надбавку.

```
SELECT NAME_TEACHER
FROM TEACHER
WHERE (Salary + Rise) / 2 > 5 * Rise;
```

Использование специальных операторов

В SQL имеются операторы сравнения, позволяющие проверять значения столбцов и выражений над ними на соответствие некоторым специальным условиям:

принадлежность множеству; принадлежность диапазону; соответствие шаблону;

соответствие регулярному выражению; неопределенное значение.

В этом разделе вы узнаете, как их использовать и как с их помощью создавать составные условия.

Проверка на принадлежность множеству.

Оператор IN позволяет проверить, входит ли значение в указанное множество значений. В простейшем случае этот оператор имеет следующий синтаксис: имя\_столбца [NOT] IN (список\_значений)

Здесь список значений представляет собой перечень разделенных запятыми констант, тип которых должен соответствовать типу столбца, чье имя приведено слева. Семантика этого предиката такова: он принимает

значение TRUE, если значение столбца соответствует одной из констант списка. Приведем пример.

Запрос 14. Вывести названия и номер корпуса кафедр, расположенных в корпусах

1, 3, 12.

```
SELECT Name_Kafedru, NUM_KORPUSA  
FROM KAFEDRA  
WHERE NUM_KORPUSA IN ('1', '3', '12');
```

Использование отрицания

Так как предикат IN возвращает истинностное значение, к нему можно применить логическое отрицание. Для этого следует воспользоваться нотацией NOT IN. В этом случае предикат будет истинным, если значение столбца не входит в указанный список.

Запрос 15. Вывести названия и номер корпуса кафедр, расположенных в любых корпусах, кроме 1, 3, или 12.

```
SELECT Name_Kafedru AS 'Название кафедры',  
       NUM_KORPUSA AS 'Корпус'  
FROM KAFEDRA  
WHERE NUM_KORPUSA NOT IN ('1', '3', '12');
```

Использование выражений над столбцами

В левой части оператора IN вместо имени столбца можно использовать любое допустимое над столбцами таблицы выражение языка.

Запрос 16. Вывести фамилии преподавателей, зарплата которых (ставка + надбавка) равна 800, 900, 1000, 1100 или 1200.

```
SELECT NAME_TEACHER AS 'Фамилия преподавателя',  
       Salary + Rise AS 'Зарплата преподавателя'  
FROM TEACHER  
WHERE Salary + Rise IN (1150, 2400, 3150, 4300);
```

Более того, элементами списка в правой части оператора IN тоже могут быть выражения над столбцами, как это показано в следующем примере:

Запрос 17.

```
SELECT NAME_TEACHER, Salary, Salary + Rise  
FROM TEACHER  
WHERE Salary + Rise IN (Salary + 100, Salary + 200, Salary + 300, Salary  
+ 400, Salary + 500);
```

Проверка на принадлежность диапазону значений

Еще одной формой проверки вхождения элемента во множество является проверка на его принадлежность диапазону значений. Для этого применяется предикат BETWEEN, который определяет нахождение значения столбца между указанными минимальным и максимальным значениями. Синтаксис предиката следующий:

имя\_столбца [NOT] BETWEEN минимум AND максимум

Проверять можно значения числовых, строковых и временных типов (для строк символов предполагается алфавитное упорядочение). Оператор

BETWEEN является включающим - это означает, что крайние значения диапазона включаются в допустимые. Запрос 18. Вывести фамилии преподавателей со ставкой в диапазоне 10002000.

```
SELECT NAME_TEACHER
FROM TEACHER
WHERE Salary BETWEEN 1000 AND 2000;
```

Использование строковых значений

Использование в операторе BETWEEN в качестве границ диапазона строковых значений имеет особенности, связанные с упорядочением (это же относится и к другим операторам сравнения).

Запрос 19. Вывести фамилии преподавателей, начинающиеся на буквы от 'З' до 'Л'.

```
SELECT NAME_TEACHER
FROM TEACHER
WHERE UPPER(NAME_TEACHER) BETWEEN 'З' AND 'Л';
```

Среди строк результата нет фамилий, начинающихся на букву 'Л'. Дело в том, что при сравнении строк символов разной длины SQL предварительно дополняет более короткую строку символами пробела, а он в упорядочениях символов предшествует всем остальным. Поэтому строка, состоящая из буквы 'Л' (дополненная пробелами), всегда будет меньше любой другой строки, в которой за начальной буквой 'Л' следуют отличающиеся от пробела символы.

Чтобы это учесть, в качестве верхнего значения диапазона лучше всего указывать следующую по алфавиту букву (в данном случае — 'М').

Использование отрицания

Так как предикат BETWEEN возвращает истинностное значение, к нему можно применить логическое отрицание. Для этого следует воспользоваться нотацией NOT BETWEEN, в которой предикат будет истинным, только если значение столбца не входит в указанный диапазон. Представление отрицания нотацией NOT BETWEEN введено в язык для большей наглядности, так как с предикатом BETWEEN можно стандартным образом использовать логический оператор NOT (то есть ставить отрицание ко всему выражению, а не к предикату):

```
NOT (имя_столбца BETWEEN минимум AND максимум)
```

Круглые скобки в данном случае можно и опустить, так как они не меняют порядка исполнения операторов. В нотации NOT BETWEEN крайние значения в диапазон не включаются.

Запрос 20. Вывести названия и номер корпуса кафедр, которые не расположены в корпусах 1 и 3.

```
SELECT Name_Kafedru, NUM_KORPUSA
FROM KAFEDRA
WHERE NUM_KORPUSA NOT BETWEEN '1' AND '3';
SELECT Name_Kafedru, NUM_KORPUSA
FROM KAFEDRA
```

WHERE NOT (NUM\_KORPUSA BETWEEN '1' AND '3');

Использование выражений над столбцами

Как и в предикате IN, вместо имени столбца и границ диапазона можно использовать любое допустимое в языке выражение над столбцами таблицы, включая и функции.

Запрос 21. Показать фамилии преподавателей, принятых на работу между

01.01.2000 и 12.12.2001.

```
SELECT NAME_TEACHER, DATA_HIRE
```

```
FROM TEACHER
```

```
WHERE DATA_HIRE BETWEEN '01/01/2000' AND '12/12/2001';
```

Запрос 22. Вывести данные преподавателей, зарплата которых (ставка + надбавка) находится в диапазоне от удвоенной величины надбавки до утроенной надбавки плюс 50.

```
SELECT NAME_TEACHER, Salary + Rise, 2 * Rise, 3 * Rise + 50
```

```
FROM TEACHER
```

```
WHERE Salary + Rise BETWEEN 2 * Rise AND 3 * Rise + 50;
```

Проверка на соответствие шаблону

Когда необходимо отобрать строки таблицы, в которых значение некоторого столбца совпадает с заданной строкой символов, следует использовать обычное сравнение, как это показано выше. Однако во многих случаях можно не знать точное представление в базе данных интересующего значения. Название одной и той же кафедры, например, может храниться в одном из следующих вариантов: 'базы данных', 'организация баз данных', 'информационные системы и базы данных', 'базы данных и знаний'.

Такая же ситуация возникает, когда не известно точное написание фамилии преподавателя, название дисциплины, факультета и т. п. Специально для таких случаев предназначен оператор сравнения LIKE, позволяющий отобрать из таблицы строки на основе частичного соответствия. Упрощенный синтаксис оператора следующий:

```
имя_столбца [NOT] LIKE шаблон [ESCAPE символ_пропуска]
```

Его можно использовать только с символьными значениями.

Использование шаблона

Оператор LIKE сравнивает значение столбца с множеством значений, определяемых шаблоном. Он представляет собой строку, в которой помимо обычных символов, составляющих основу поискового выражения, можно использовать так называемые подстановочные символы (иногда они называются групповыми символами). Имеется всего два подстановочных символа, различающихся тем, что именно на их месте может стоять:

% – любая последовательность символов, включая их отсутствие; \_ — один любой символ. Подстановочные символы могут находиться в любом месте шаблона в любом наборе.

Например, шаблону '%Иван%' соответствуют строки 'Иван', 'Иванов', 'Иванченко', 'Петр Иванович', а шаблону 'л\_с\_' - 'лист', 'леса', 'лоск' (ноне 'лес', 'листок', 'плес').

Оператор LIKE, как и все другие, работающие с символьными строками, чувствителен к регистру букв, поэтому при его использовании мы рекомендуем использовать уже известные вам функции UPPER() и LOWER().

Запрос 23. Найти фамилии преподавателей на букву 'М'.

```
SELECT NAME_TEACHER
FROM TEACHER
WHERE UPPER(NAME_TEACHER) LIKE 'М%';
```

Имейте в виду, что если вы запишете условие фразы WHERE как UPPER(NAME\_TEACHER) = 'М%' или даже как 'М%' LIKE UPPER(NAME\_TEACHER), фамилии преподавателей будут сравниваться со строкой ' М%'. Во втором случае выражение является синтаксически правильным оператором LIKE, однако в нем строка 'М%' не выступает в качестве шаблона, так как расположена перед ключевым словом LIKE.

Запрос 24. Указать преподавателей, в фамилиях которых первой буквой является 'М', а четвертой – 'ы'.

```
SELECT NAME_TEACHER
FROM TEACHER
WHERE NAME_TEACHER LIKE 'М__ы%';
```

Запрос 25. Вывести названия кафедр, в которых присутствует словосочетание 'анализ' (в различных грамматических формах). SELECT Name\_Kafedru

```
FROM KAFEDRA
WHERE LOWER(Name_Kafedru) LIKE '%анализ%';
```

В левой части оператора LIKE может находиться не только имя столбца, но и любое допустимое над столбцами выражение, как это показано в следующем примере.

Запрос 26. Указать преподавателей, в фамилию и название должности которых входит в сумме не меньше пяти букв 'о'.

```
SELECT NAME_TEACHER, DOLGNOST
FROM TEACHER
WHERE LOWER(NAME_TEACHER + DOLGNOST) LIKE
'%о%о%о%о%о%';
```

Проверка на неопределенное значение

Как мы уже отмечали, наличие значения NULL во фразе WHERE приводит к тому, что условие принимает истинностное значение UNKNOWN и соответствующая строка не включается в результат. Детальное описание работы с неопределенным значением вы можете найти в уроке 10, а здесь мы покажем, как обрабатывать значение NULL во фразе WHERE.

Чтобы проверить столбец на неопределенное значение, следует применить унарный оператор IS NULL, имеющий такой синтаксис:

```
имя_столбца IS [NOT] NULL
```

Этот оператор принимает истинностное значение TRUE, если столбец имеет неопределенное значение, и FALSE — в противном случае. В нотации IS NOT NULL его действие обратное.

Запрос 27. Вывести фамилии преподавателей, у которых не задан номер телефона или идентификационный код.

```
SELECT NAME_TEACHER, INDEF_KOD, TEL_TEACHER  
FROM TEACHER  
WHERE INDEF_KOD IS NULL OR TEL_TEACHER IS NULL;
```

Задание для практической работы

Для созданной базы данных, согласно номеру варианта, самостоятельно создать на языке SQL 15 запросов с отбором строк по условию:

- 3 простейших запроса с использованием операторов сравнения;
- 3 запроса с использованием логических операторов AND, OR и NOT;
- 1 запрос на использование комбинации логических операторов;
- 1 запрос на использование выражений над столбцами;
- 2 запроса с проверкой на принадлежность множеству;
- 2 запроса с проверкой на принадлежность диапазону значений; - 2 запроса с проверкой на соответствие шаблону; - 1 запрос с проверкой на неопределенное значение.

Все программные инструкции команд SQL сохранять в файлах с расширением \*.sql в папке ФИО\_студента

## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 10. Выполнение изменений в базе данных, создание триггеров

**Цель:** Изучение назначения и типов триггеров, условий их активации, синтаксиса и семантики команд языка SQL для их создания, модификации, переименования, программирования и удаления, а также приобретение навыков их проектирования, кодирования и отладки с применением хранимых процедур для получения информации о триггерах базы данных.

### Ход работы:

Триггер SQL Server 2000 – это специальный тип хранимых процедуры, которые запускаются сервером автоматически при выполнении тех или иных действий с данными таблицы. Триггеры различаются по типу команд, на которые они реагируют:

INSERT TRIGGER – запускаются при попытке вставить данные с помощью команды INSERT;

UPDATE TRIGGER – запускаются при попытке изменения данных с помощью команды UPDATE; authsmall TRIGGER – запускаются при попытке удаления данных с помощью команды DELETE.

Параметры FOR, AFTER и INSTEAD OF, указываемые при создании триггера, определяют его поведение следующим образом:

FOR – запуск триггера при выполнении заданной в этом списке команды; AFTER – запуск триггера после успешного выполнения команд списка; INSTEAD OF – триггеры вызывается вместо выполнения команд списка.

Можно определить несколько AFTER – триггеров для каждой операции INSERT, UPDATE и DELETE. По умолчанию все триггеры являются AFTER – триггерами. Триггеры нельзя создавать для временных или системных таблиц. Команда создания триггера должна быть первой в пакете и применяться только к одной таблице. Ее формат следующий:

```
CREATE TRIGGER Имя триггера.  
ON {Имя таблицы\Имя представления}  
[WITH ENCRYPTION] -- шифрование кода триггера;  
{ {FOR\AFTER\INSTEAD OF}  
{[DELETE] [,] [INSERT] [,] [UPDATE]}  
[WITE APPEND] -- только для версий 6.5 и ниже;  
[NOT FOR REPLICATION] -- не для репликации; AS sql_statement [...n]  
-- тело триггера;  
}  
|{ {FOR\AFTER\INSTEAD OF}  
{[INSERT] [,] [UPDATE]}  
[WITE APPEND] -- только для версий 6.5 и ниже;  
[NOT FOR REPLICATION] -- не для репликации; AS {IF UPDATE  
(column) -- при изменении столбца;  
[{AND\OR} UPDATE (column) [...n]]} -- тоже;
```

```

|
IF (COLUMNS_UPDATED() {bitwise_operator}
Update_bitmask)
{comparison_operator}column_bitmask [...n]
}
sql_statement [...n] -- тело
}
|
{{FOR\AFTER\INSTEAD OF}
{[INSERT] [,] [UPDATE]}
[WITE APPEND] -- только для версий 6.5 и ниже;
[NOT FOR REPLICATION] -- не для репликации; AS {IF UPDATE
(column) -- при изменении столбца;
[{{AND\OR} UPDATE (column) [...n]}}-- тоже;
|
IF (COLUMNS_UPDATED() {bitwise_operator}
Update_bitmask)
{comparison_operator}column_bitmask [...n]
}
sql_statement [...n] -- тело триггера.
}
}

```

Вторая альтернатива команды {IF UPDATE...} используется для детального анализа изменений содержимого колонок с помощью специальных функций, битовых масок, операторов побитовой обработки, оператор сравнения и логических операторов.

Команда ALTER TRIGGER позволяет изменить параметры и тело триггера. С помощью команды DROP TRIGGER можно удалить любой триггер базы данных.

Переименовать триггер можно системной хранимой процедурой sp\_rename, а получить информацию о триггере можно при помощи системных хранимых процедур sp\_helptext и sp\_helptrigger.

Внутри триггера допускается использование любых команд языка Transact – SQL с некоторыми ограничениями. Также допускается и вызов хранимых процедур, включая системные.

### **Задание:**

Задание 1. Создать таблицу authsmall из таблицы authors базы данных Pubs и для новой таблицы запрограммировать триггер auth\_del, который будет выводить информацию о попытках удаления и количестве удаляемых строк, выполнив действия:

Создание таблицы authsmall с колонками au\_id, au\_fname, au\_lname, phone и копирование в нее данных из таблицы authors:

```

SELECT au_id, au_fname, au_lname, phone
INTO authsmall

```



```
FROM authors
PRINT 'Содержимое таблицы authsmall:' SELECT * FROM authsmall
```

Создание и программирование триггера:

```
CREATE TRIGGER auth_del
ON authsmall
FOR DELETE
AS
PRINT 'Попытка удаления' + STR (@@ POWCOUNT)+
'строк в таблице authsmall'
PRINT 'Пользователь' + CURRENT_USER
IF CURRENT_USER <> 'dbo' BEGIN
PRINT 'Удаление запрещено'
ROLLBACK TRANSACTION
END
ELSE
PRINT 'Удаление разрешено'
```

3. Тестирование триггера :

```
DELETE FROM authsmall WHERE au_fname = 'Johnson'
```

```
DELETE FROM authsmall WHERE 2*2=5
```

Задание 2. Создать триггер auth\_upd для таблицы authsmall, построенный в первом задании, который будет разрешать изменение столбца au\_id этой таблицы всем, кроме владельца dbo, выполнив следующие действия:

1. Создание и программирование триггера:

```
CREATE TRIGGER auth_upd
ON authsmall
FOR UPDATE
AS
SET NOCOUNT ON -- не сообщать о завершении команд;
PRINT 'Попытка изменения данных в таблице authsmall'
IF (COLUMNS_UPDATED () &1) != 0 -- 1-й столбец;
PRINT 'Изменение столбца au_id'
IF (COLUMNS_UPDATED () &2) != 0 -- 2-й столбец;
PRINT 'Изменение столбца au_fname'
IF (COLUMNS_UPDATED () &4) != 0 -- 3-й столбец;
PRINT 'Изменение столбца au_lname' IF UPDATE (Phone)
PRINT 'Изменение столбца phone'
IF ((CURRENT_USER = 'dbo') AND
(COLUMNS_UPDATED()&1) != 0 -- 1-ый стлбец;
BEGIN
PRINT 'Пользователь dbo не может изменять' + 'идентификационный
номер автора'
ROLLBACK TRANSACTION
END
```

2. Тестирование триггера:

```
UPDATED authsmall SET phone = '415 986 - 7020', au_fname = 'John'
```

```
WHERE au_lname = 'Green'
```

```
UPDATED authsmall SET phone = '913 843 - 7302', au_id = '748-126859'
```

```
WHERE au_lname = 'Smith'
```

Задание 3. Создать триггер для команд INSERT и UPDATE, запрещающий производить изменения для автора Billy Geitsi, выполнив действия:

1. Создание и программирование триггера:

```
CREATE TRIGGER auth_ins_upd ON authsmall
```

```
FOR INSERT, UPDATE
```

```
AS
```

```
IF EXISTS (SELECT * FROM authsmall -- inserted; WHERE au_lname =  
'Geitsi' -- фамилия; au_fname = 'Billy') -- имя;
```

```
BEGIN
```

```
PRINT 'Недопустимо написание книги'+
```

```
'автором Billy Geitsi'
```

```
ROLLBACK TRANSACTION END
```

2. Тестирование триггера:

```
UPDATE authsmall SET au_lname = 'Geitsi', au_fname = 'Billy' WHERE  
au_lname = 'Smith'.
```

## ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 11. Создание запросов и процедур на изменение структуры базы данных

**Цель:** Изучить используемый в реляционных СУБД оператор извлечения данных из таблиц. Получить навыки работы с оператором SELECT в программе 'SQL Server Managment Studio'.

*Теоретические сведения.*

При проектировании стремятся создавать таблицы, в каждой из которых содержалась бы информация об одном и только одном типе сущности. Это облегчает модификацию базы данных и поддержку ее целостности. Именно так мы поступили, создавая учебную базу данных. Однако сущности могут быть взаимосвязанными. Кафедры связаны с факультетами по признаку вхождения в их состав, преподаватели работают на кафедрах, студенты учатся на кафедрах и т. д.

Связь между таблицами устанавливается за счет размещения специального столбца первичного ключа одной таблицы, которая называется родительской, в другой таблице, которая называется дочерней. Столбец (или совокупность столбцов) дочерней таблицы, определенный для связи с родительской таблицей, называется внешним ключом.

Наличие внешних ключей является основной для инициирования поиска по многим таблицам.

Одна из наиболее важных особенностей предложения SELECT — это способность использования связей между различными таблицами, а также вывода содержащейся в них информации. Операция, которая приводит к соединению из двух таблиц всех пар строк, для которых выполняется заданное условие, называется соединением таблиц. Для того чтобы указать соединяемые таблицы, их следует перечислить через запятую во фразе FROM.

*Декартово произведение таблиц.*

Соединение таблиц - это частный случай операции декартового произведения (или просто произведения). Декартово произведение двух таблиц — это таблица, состоящая из всех возможных пар строк обеих таблиц. Это определение можно естественным образом расширить на любое количество таблиц. В SQL декартово произведение выражается указанием имен перемножаемых таблиц во фразе FROM и указанием всех их столбцов во фразе SELECT.

Так, произведение таблиц FACULTET и KAFEDRA выражается следующим образом:

```
SELECT *  
FROM FACULTET, KAFEDRA
```

Так как результирующая таблица содержит много столбцов и они не помещаются по ширине страницы, мы приведем только интересующие нас столбцы произведения этих таблиц.

Запрос. Декартово произведение таблиц.

```
SELECT FACULTET.Name_faculteta, FACULTET. Kod_faculteta,  
       KAFEDRA. Kod_faculteta, KAFEDRA.Name_Kafedru  
FROM   FACULTET, KAFEDRA;
```

Каждая строка таблицы факультетов оказалась соединенной с каждой строкой таблицы кафедр, в результате получилось 27 строк (3 факультета x 9 кафедр = 27 комбинаций).

В произведении может участвовать много таблиц. Например, произведение таблиц факультетов, кафедр и преподавателей записывается следующим образом:

```
SELECT *  
FROM   FACULTET, KAFEDRA, TEACHER
```

*Условие соединения.*

Соединение таблиц может быть указано во фразе WHERE или во фразе FROM. Сначала рассмотрим первый вариант. Большинство запросов, имеющих несколько таблиц во фразе FROM, содержат фразу WHERE, в которой указаны условия, попарно сравнивающие столбцы из различных таблиц. Такое условие называется условием соединения. В этом случае SQL предполагает сцепление только тех пар строк из разных таблиц, для которых условие соединения принимает истинное значение. Теоретически при соединении сначала выполняется декартово произведение указанных таблиц в одну, а затем из нее отбираются строки согласно условию соединения. Естественно, ни одна СУБД не работает таким образом.

Фраза WHERE помимо условия соединения может также содержать другие условия, каждое из которых ссылается на столбцы соединенной таблицы. Эти условия производят отбор строк соединенной таблицы.

Соединения можно разделить на следующие категории.

Внутренние соединения (типичные операции соединения, использующие такие операторы сравнения, как = или <>). Они включают эквивалентные соединения и естественные соединения.

Внутренние соединения используют оператор сравнения для установки соответствия строк из двух таблиц на основе значений общих столбцов в каждой таблице. Примером может быть получение всех строк, в которых идентификационный номер студента одинаковый как в таблице students, так и в таблице courses.

Внешние соединения. Внешние соединения бывают левыми, правыми и полными.

Если внешние соединения задаются в предложении FROM, они указываются с одним из следующих наборов ключевых слов.

```
LEFT JOIN или LEFT OUTER JOIN
```

Результирующий набор левого внешнего соединения включает все строки из левой таблицы, заданной в предложении LEFT OUTER, а не только те, в которых соединяемые столбцы соответствуют друг другу. Если строка в левой таблице не имеет совпадающей строки в правой таблице,

результатирующий набор строк содержит значения NULL для всех столбцов списка выбора из правой таблицы. RIGHT JOIN или RIGHT OUTER JOIN

Правое внешнее соединение является обратным для левого внешнего соединения. Возвращаются все строки правой таблицы. Для левой таблицы возвращаются значения NULL каждый раз, когда строка правой таблицы не имеет совпадающей строки в левой таблице.

#### FULL JOIN или FULL OUTER JOIN

Полное внешнее соединение возвращает все строки из правой и левой таблицы. Каждый раз, когда строка не имеет соответствия в другой таблице, столбцы списка выбора другой таблицы содержат значения NULL. Если между таблицами имеется соответствие, вся строка результирующего набора содержит значения данных из базовых таблиц.

#### *Перекрестные с соединения.*

Перекрестное соединение возвращает все строки из левой таблицы. Каждая строка из левой таблицы соединяется со всеми строками из правой таблицы. Перекрестные соединения называются также декартовым произведением.

Таблицы или представления в предложении FROM могут указываться в любом порядке с внутренним соединением или полным внешним соединением. Однако важен порядок таблиц или представлений, заданных при использовании левого или правого внешнего соединения.

#### *Соединение таблиц по равенству.*

Если таблицы соединяются по равенству значений пары столбцов (группы столбцов) из различных таблиц, такая операция называется соединением таблиц по равенству. Соединение по равенству, в отличие от декартового произведения, позволяет соединить только те пары строк, которые действительно взаимосвязаны друг с другом. Так, например, мы можем соединить таблицы факультетов и кафедр по условию FACULTET.Kod\_faculteta = KAFEDRA.Kod\_faculteta. В таком варианте мы соединяем таблицы осмысленно, так как каждая строка таблицы FACULTET соединяется только со строками соответствующих кафедр. На базе таблиц FACULTET и KAFEDRA мы получаем таблицу со столбцами из обеих таблиц, имеющую строки с понятным смыслом. Можно также сказать, что в таблицу KAFEDRA вместо столбца Kod\_faculteta мы вставляем все характеристики (столбцы) соответствующего факультета из таблицы FACULTET.

Соединение таблиц используется, когда необходимо вывести значения столбцов: – разных таблиц;

– одной таблицы, но отвечающих условию, заданному на другой таблице.

Эти два варианта, а также их комбинация, характерны для любого вида соединения, а не только по равенству. Перейдем к рассмотрению примеров.

#### *Вывод столбцов разных таблиц.*

Этот вид запросов характерен тем, что фраза WHERE содержит только условие соединения, а список фразы SELECT содержит имена столбцов из различных таблиц.

Запрос 29. Вывести названия кафедр и номера их групп.

```
SELECT Name_Kafedru, [Group]
FROM KAFEDRA, STUDENT
WHERE KAFEDRA.kod_kafedru = STUDENT.kod_kafedru;
```

или

```
SELECT Name_Kafedru, student.[GROUP]
FROM KAFEDRA, STUDENT
WHERE KAFEDRA.kod_kafedru = STUDENT.kod_kafedru;
```

Мы привели два варианта запроса. В первом имена столбцов не уточняются именами таблиц, а во втором — уточняются. В данном случае это не имеет значения, оба запроса корректны.

*Уточнение имен столбцов.*

До тех пор, пока запрос относится к одной таблице, обращение к столбцам по их именам не вызывает проблем — в таблице все имена столбцов должны быть неповторяющимися. Однако как только запрос соединяет несколько таблиц, может возникнуть неоднозначность при ссылках на столбцы с одинаковыми именами из разных таблиц. Для разрешения этой неоднозначности во фразах SELECT и WHERE (как и в некоторых других фразах) имена столбцов необходимо уточнять именами таблиц.

Запрос 30. Вывести названия факультетов и их кафедр.

```
SELECT FACULTET.NAME_FACULTETA, KAFEDRA.Name_Kafedru
FROM FACULTET, KAFEDRA
WHERE FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta;
```

В этом запросе мы уточнили имена столбцов во фразах SELECT и WHERE, хотя во втором случае это не обязательно, так как используются неповторяющиеся имена. Тем не менее, рекомендуем при соединении таблиц для наглядности уточнять имена столбцов. Обратите внимание на то, что в предыдущем примере отсутствует факультет математики

— на нем нет кафедр.

Вывод столбцов с условием отбора

Вариант, когда отбираются строки одной таблицы, а условие задается с участием другой, используется довольно часто. Приведем примеры.

Запрос. Вывести названия кафедр факультета Математики и информатики.

```
SELECT KAFEDRA.Name_Kafedru AS 'Кафедры факультета
математики и информатики'
FROM FACULTET, KAFEDRA
WHERE FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta AND
LOWER(FACULTET.NAME_FACULTETA) = 'математики и информатики';
```

Запрос . Вывести фамилии доцентов кафедры информатики.

```

SELECT  TEACHER.NAME_TEACHER  AS  'Доценты  кафедры
информатики'
FROM  KAFEDRA, TEACHER
WHERE  KAFEDRA.kod_kafedru = TEACHER. kod_kafedru AND
LOWER(KAFEDRA.Name_Kafedru) = 'информатики' AND
LOWER(TEACHER.DOLGNOST) = 'доцент';

```

В последнем запросе помимо условия соединения используется также отбор строк по условиям, заданным для разных таблиц.

Синонимы таблиц.

Синонимы таблиц часто используются для задания более лаконичного имени таблицы, по которому можно сослаться на нее в любых других местах запроса. Приведем пример.

Запрос 33. Вывести названия кафедр, на которых имеются студенты со стипендией >200 грн.

```

SELECT DISTINCT k.Name_Kafedru
FROM  KAFEDRA k, STUDENT s
WHERE k.Kod_kafedru = s. Kod_kafedru AND s.Stipend > 400;

```

*Запросы по трем и более таблицам.*

SQL позволяет формулировать запросы, которые предполагают использование трех и более таблиц. При этом следует применять ту же методику соединения, что и для двух таблиц. Рассмотрим простой пример соединения трех таблиц.

Запрос. Вывести названия тех кафедр факультета математики и информатики, на которых работают профессора.

```

SELECT DISTINCT KAFEDRA.Name_Kafedru
FROM  FACULTET, KAFEDRA, TEACHER
WHERE FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta AND
KAFEDRA.Kod_kafedru = TEACHER.Kod_kafedru AND
FACULTET.Name_faculteta = 'Математики и информатики' AND
TEACHER.DOLGNOST = 'профессор';

```

Для ответа на запрос необходимы три таблицы: на таблицах факультетов и преподавателей заданы условия отбора, а из таблицы кафедр следует вывести столбец названий. Поэтому три необходимые таблицы указываются во фразе FROM, а во фразе WHERE производится их соединение по условию равенства первичного и внешнего ключей:

FACULTET. Kod\_faculteta = KAFEDRA. Kod\_faculteta -- соединение таблиц факультетов и кафедр

KAFEDRA. Kod\_kafedru = TEACHER. Kod\_kafedru -- соединение таблиц кафедр и преподавателей

Таблица, образуемая в результате соединений, будет иметь столько же строк, сколько имеется в таблице преподавателей (если все преподаватели работают на кафедрах). Выясним, почему это так, но сначала заметим, что результат соединения таблиц не зависит от порядка соединения. Поэтому

рассмотрим случай, когда сначала мы соединяем таблицы кафедр и преподавателей, а затем результат соединяем с таблицей факультетов.

Так как между таблицами кафедр и преподавателей существует связь типа одинко-многим, их соединение фактически означает приписывание к строке каждого преподавателя данных о его кафедрах. Количество строк этого соединения будет равным количеству преподавателей. Связь между таблицами факультетов и кафедр также имеет тип один-ко-многим, поэтому второе соединение означает, что к каждой строке таблицы, полученной после первого соединения, приписываются данные о факультете кафедры. Таким образом, количество строк останется равным числу преподавателей.

Вернемся к запросу. Последние два условия фразы WHERE отбирают строки из соединенной таблицы, а во фразе SELECT указан выводимый столбец. Ключевое слово DISTINCT указано в нем потому, что названия кафедр в соединенной таблице могут повторяться.

Запрос. Вывести фамилии ассистентов факультета математики и информатики.

```
SELECT  TEACHER.NAME_TEACHER AS 'Ассистенты ф-та математики и информатики'
```

```
FROM  FACULTET, KAFEDRA, TEACHER
WHERE FACULTET.Kod_faculteta = KAFEDRA.Kod_faculteta AND
KAFEDRA.Kod_kafedru = TEACHER.Kod_kafedru AND
FACULTET.Name_faculteta = 'Математики и информатики' AND
TEACHER.DOLGNOST = 'ассистент';
```

В этом случае для ответа нужны две таблицы — факультетов и преподавателей. Однако они связаны между собой опосредованно, через таблицу кафедр. Поэтому для соединения таблиц факультетов и преподавателей следует использовать таблицу кафедр.

Сформулируем общую процедуру составления многотабличного запроса и приведем пример ее использования.

Определить множество таблиц, необходимых для ответа на запрос. В это множество должны входить таблицы, на столбцах которых сформулированы условия, а также те, столбцы которых необходимо вывести. Это так называемые базовые таблицы запроса.

В структуре взаимосвязанных таблиц найти путь, соединяющий базовые таблицы. Это так называемый путь вычисления запроса. В результате вы получите перечень таблиц, необходимых для формулировки запроса. Это так называемые таблицы запроса.

Во фразе FROM перечислить необходимые таблицы.

Во фразе WHERE соединить таблицы запроса и при необходимости задать условия отбора строк в базовых таблицах запроса. 5. Во фразе SELECT перечислить выводимые столбцы.

*Вывод всех столбцов соединяемой таблицы.*

В многотабличном запросе конструкция SELECT \* означает выбор всех столбцов соединенной таблицы. Например, результирующая таблица



следующего запроса состоит из 21 столбца: 5 столбцов таблицы факультетов, 6 столбцов таблицы кафедр и 10 столбцов таблицы преподавателей.

```
Запрос. SELECT *
FROM FACULTET f, KAFEDRA k, TEACHER t
WHERE f.Kod_faculteta = k.Kod_faculteta AND k.Kod_kafedru =
t.Kod_kafedru;
```

При наличии в запросе многих таблиц конструкция SELECT \* становится не очень практичной. В связи с этим в различных СУБД предоставляется возможность использовать во фразе SELECT многотабличных запросов выражение имя\_таблицы.\* для указания вывода всех столбцов конкретной таблицы. Например:

```
SELECT f.*, k.FIO_ZAVKAF, t.*
FROM FACULTET f, KAFEDRA k, TEACHER t
WHERE f.Kod_faculteta = k.Kod_faculteta AND k.Kod_kafedru =
t.Kod_kafedru;
```

*Другие виды соединений по равенству.*

Логическая связь между таблицами поддерживается взаимосоответствием столбцов первичного и внешнего ключей. Все рассмотренные до сих пор запросы для соединения таблиц использовали именно эту связь. Однако SQL позволяет связывать таблицы по любой паре столбцов, которые имеют сравнимые типы данных, независимо от того, имеет ли эта связь какой-либо смысл. Рассмотрим ряд примеров.

Запрос. Если фамилия заведующего кафедры совпадает с фамилией декана какого-нибудь из факультетов, вывести название этой кафедры вместе с названием соответствующего факультета.

```
SELECT k.Name_Kafedru AS 'Название кафедры',
       f.NAME_FACULTETA AS 'Название факультета'
FROM FACULTET f, KAFEDRA k
WHERE f.FIO_DECANA = k.FIO_ZAVKAF;
```

Запрос. Вывести пары названий кафедр и фамилий преподавателей, у которых совпадают первичные ключи.

```
SELECT k.Name_Kafedru AS 'Название кафедры',
       t.Name_Teacher AS 'Фамилия преподавателя'
FROM KAFEDRA k, TEACHER t
WHERE k.Kod_kafedru = t.Kod_kafedru;
```

Если первый запрос не лишен смысла, то последний абсолютно бессмысленный, так как в учебной базе данных первичные ключи лишены какого-либо содержания и используются только для идентификации строк своих таблиц.

*Самосоединение таблицы.*

Как правило, взаимосвязи существуют и в пределах одной таблицы. В одних случаях эти связи являются явными, например, когда внешний ключ ссылается на первичный ключ той же самой таблицы. В других случаях эта

связь присутствует неявно, например, кафедры могут быть связаны между собой на основании того свойства, что располагаются в одном корпусе.

Для ответа на такие запросы следует осуществлять соединение таблицы со своей копией. Такое соединение иногда называют самосоединением таблицы. Несмотря на кажущуюся искусственность идеи самосоединения таблиц, существует множество запросов, которые требуют именно такого соединения. На приводимых далее примерах вы убедитесь в этом.

Чтобы произвести соединение таблицы со своей копией, необходимо указать во фразе FROM имя одной и той же таблицы два или большее количество раз, а во фразе WHERE — условие их самосоединения.

Однако в этом случае возникает следующая проблема — как ссылаться на столбцы различных копий таблицы. До сих пор проблема ссылки на столбцы с одинаковыми именами из разных таблиц разрешалась уточнением имени столбца именем таблицы. В нашем же случае соединяемые таблицы имеют одинаковые имена. Для разрешения этой проблемы без синонимов таблиц уже не обойтись. В нашем случае различным вхождениям одной и той же таблицы приписываются различные синонимы и именно по этим синонимам производится обращение к столбцам. Приведем примеры использования самосоединения.

Запрос. Вывести фамилии преподавателей, зарплата которых больше, чем у преподавателя Сидорова.

```
SELECT needed.NAME_TEACHER
FROM TEACHER needed, TEACHER given
WHERE needed.Salary + needed.Rise > given.Salary + given.Rise AND
given.NAME_TEACHER = 'Игнатъева Олеся Владимировна';
```

Симметричное соединение и удаление избыточности

При самосоединении по равенству обычно возникают избыточные строки. Рассмотрим следующий запрос.

Запрос 40. Вывести названия кафедр, располагающихся в том же корпусе, что и кафедра информатики.

```
SELECT needed.Name_Kafedru
FROM KAFEDRA needed, KAFEDRA given
WHERE needed.NUM_KORPUSA = given.NUM_KORPUSA AND
given.Name_Kafedru = 'Информатики';
```

Обратите внимание, что в результат включена и сама кафедра информатики. Для того чтобы избавиться от ненужной результирующей строки, следует добавить условие отбора: `needed.Name_Kafedru <> 'Информатика'`

При самосоединении по равенству можно получить симметричную результирующую таблицу. Суть симметричности заключается в том, что в таблице содержатся строки:

с одинаковыми значениями всех столбцов; со всеми возможными перестановками значений столбцов.

Запрос. Вывести пары номеров групп, которые принадлежат одной кафедре.

```
SELECT g1.[Group], g2.[Group], g1.kod_kafedru  
FROM STUDENT g1, STUDENT g2  
WHERE g1.kod_kafedru = g2.kod_kafedru;
```

ПРИМЕЧАНИЕ В этом запросе мы дополнительно вывели столбец с номером (первичным ключом) кафедры для большей наглядности.

Результирующая таблица оказалась симметричной, и в связи с этим содержит избыточные строки.

Простой способ избежать этого состоит в том, чтобы наложить ограничение на выбираемые пары значений таким образом, чтобы первое выдаваемое значение было меньше другого (или предшествовало ему в алфавитном порядке). Это делает результат асимметричным, поэтому пары с одинаковыми значениями, а также пары, заданные в обратном порядке, не будут появляться. Покажем это на примере варианта предыдущего запроса.

```
SELECT g1.[Group], g2.[Group], g1.kod_kafedru  
FROM STUDENT g1, STUDENT g2  
WHERE g1.kod_kafedru = g2.kod_kafedru  
AND g1.[Group] <g2.[Group];
```

Проверка правильности данных.

Самосоединение можно использовать для проверки корректности данных. Например, мы точно знаем, что в нашем вузе нет однофамильцев, занимающих разные должности. С помощью самосоединения таблицы преподавателей мы можем убедиться, что их нет и в базе данных.

Запрос. Указать преподавателей-однофамильцев, которые занимают различные должности.

```
SELECT tch1.NAME_TEACHER AS 'Препод. с различ. должностями'  
FROM TEACHER tch1, TEACHER tch2  
WHERE tch1.NAME_TEACHER = tch2.NAME_TEACHER AND  
tch1.DOLGNOST <> tch2.DOLGNOST;
```

Внешнее соединение таблиц.

Предположим, необходимо вывести список факультетов и их кафедр. Это достигается соединением таблиц FACULTET и KAFEDRA по равенству значений первичного и внешнего ключей и выбором столбцов с названиями факультетов и кафедр. Но в таком случае, если на факультете кафедр нет, он не будет включен в результат.

Для того чтобы в списке присутствовали все факультеты, даже без кафедр, необходимо использовать внешнее соединение, которое расширяет возможности обычного соединения. Внешнее соединение возвращает строки, которые удовлетворяют условию соединения, а также те строки одной из таблиц, для которых в другой не нашлось удовлетворяющих условию соединения строк.

Внутренние соединения возвращают результат, когда в обеих таблицах есть хотя бы одна строка, соответствующая условиям соединения.

Внутренние соединения исключают строки, не соответствующие ни одной строке в другой таблице. Однако внешние соединения возвращают все строки хотя бы из одной таблицы или представления, упомянутых в предложении FROM, если они удовлетворяют условиям поиска WHERE или HAVING.

Все строки, получаемые из левой таблицы, образуют левое внешнее соединение, а строки, получаемые из правой таблицы, — правое внешнее соединение. Все строки их обеих таблиц возвращаются в полном внешнем соединении.

Для внешних соединений в предложении FROM SQL Server использует ключевые слова ISO:

LEFT OUTER JOIN или LEFT JOIN;

RIGHT OUTER JOIN или RIGHT JOIN; FULL OUTER JOIN или FULL JOIN.

Работа с левыми внешними соединениями Рассмотрим примеры.

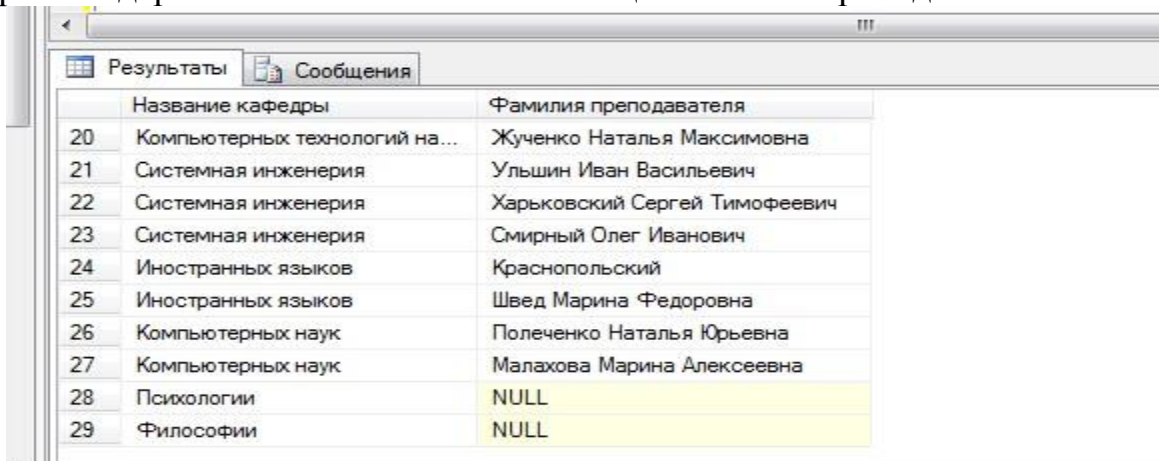
Рассмотрим соединение таблиц KAFEDRA и TEACHER по столбцам kod\_kafedru. В результате будут выведены только те кафедры, для которых были написаны преподаватели.

Чтобы включить в результаты все кафедры, независимо от того, были ли написаны их преподаватели, используйте левое внешнее соединение ISO. Пример запроса:

Запрос. Вывести фамилии всех преподавателей с указанием их кафедры, если она есть.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',  
TEACHER.NAME_TEACHER AS 'фамилия преподавателя' FROM  
KAFEDRA LEFT OUTER JOIN TEACHER  
ON KAFEDRA.kod_kafedru = TEACHER.kod_kafedru;
```

Ключевые слова LEFT OUTER JOIN включают в вывод все строки таблицы KAFEDRA независимо от того, есть ли для них соответствующие значения в столбце kod\_kafedru таблицы TEACHER. Обратите внимание на то, что в результатах, где для кафедры нет соответствующего преподавателя, строки содержат значение NULL в столбце Фамилия преподавателя.



	Название кафедры	Фамилия преподавателя
20	Компьютерных технологий на...	Жученко Наталья Максимовна
21	Системная инженерия	Ульшин Иван Васильевич
22	Системная инженерия	Харьковский Сергей Тимофеевич
23	Системная инженерия	Смирный Олег Иванович
24	Иностранных языков	Краснопольский
25	Иностранных языков	Швед Марина Федоровна
26	Компьютерных наук	Полеченко Наталья Юрьевна
27	Компьютерных наук	Малахова Марина Алексеевна
28	Психологии	NULL
29	Философии	NULL

Рисунок 28-Таблица TEACHER

Работа с правыми внешними соединениями.

Рассмотрим соединение таблиц KAFEDRA и TEACHER по столбцам kod\_kafedru. Оператор правого внешнего соединения ISO, RIGHT OUTER JOIN, включает в результаты все строки второй таблицы независимо от того, есть ли для них совпадающие данные в первой таблице.

Чтобы включить в результаты всех преподавателей независимо от того, есть ли связанные с ними кафедры, используйте правое внешнее соединение ISO. Пример запроса Transact-SQL и результаты правого внешнего соединения.

Запрос. Вывести названия всех кафедр с указанием фамилий преподавателей, если они есть.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',  
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'  
FROM KAFEDRA RIGHT OUTER JOIN TEACHER  
ON KAFEDRA.kod_kafedru = TEACHER.kod_kafedru;
```

Внешнее соединение и условие отбора.

При внешнем соединении можно применять и дополнительные условия отбора строк. Как видно из следующих двух примеров, если условие относится к столбцам таблицы, к которой не применяется оператор внешнего соединения, то внешнее соединение происходит.

Запрос. Вывести названия всех кафедр корпуса 1 с указанием их преподавателей, если они есть.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',  
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'  
FROM KAFEDRA LEFT OUTER JOIN TEACHER  
ON KAFEDRA.kod_kafedru = TEACHER.kod_kafedru  
WHERE KAFEDRA.NUM_KORPUSA = '1';
```

Запрос. Вывести названия всех кафедр с указанием их преподавателей, если они есть, ставка которых больше 3000.

```
SELECT KAFEDRA.Name_Kafedru AS 'Название кафедры',  
TEACHER.NAME_TEACHER AS 'Фамилия преподавателя'  
FROM KAFEDRA RIGHT OUTER JOIN TEACHER  
ON KAFEDRA.kod_kafedru = TEACHER.kod_kafedru WHERE  
TEACHER.salary > 3000;
```

Работа с полными внешними соединениями.

Чтобы сохранить в выводе не соответствующие друг другу строки из обеих таблиц, включив их в результаты соединения, используйте полное внешнее соединение. SQL Server предоставляет оператор полного внешнего соединения, FULL OUTER JOIN, включающий все строки из обеих таблиц вне зависимости от того, есть ли в них совпадающие значения.

Использование перекрестных соединений.

Перекрестное соединение, не имеющее предложения WHERE, выполняет декартово произведение таблиц, вовлеченных в объединение.

Размер результирующего набора декартова произведения вычисляется, как произведение количества строк в первой таблице на количество строк во второй таблице. Следующий пример показывает перекрестное соединение Transact-SQL.

```
SELECT KAFEDRA.Name_Kafedru AS 'название кафедры',  
TEACHER.NAME_TEACHER AS 'фамилия преподавателя'  
FROM KAFEDRA CROSS JOIN TEACHER  
ORDER BY KAFEDRA.kod_kafedru;
```

Результирующий набор содержит 297 строк (в KAFEDRA имеется 11 строк, а в таблице TEACHER существует 27 строк; 11, умноженное на 27, равно 297).

Внешнее соединение трех и более таблиц.

В запросе может быть использовано внешнее соединение более чем двух таблиц.

Хотя в операции соединения указываются всего две таблицы, предложение FROM может содержать несколько операций объединения. Это позволяет соединять в одном запросе несколько таблиц.

При этом следует помнить, что если к столбцу таблицы A применен оператор внешнего соединения с таблицей B, то никакой другой столбец таблицы A не может содержать оператор внешнего соединения с таблицей, отличающейся от B.

В следующем примере внешнее соединение применяется для трех таблиц — факультетов, кафедр и преподавателей.

Запрос. Вывести список всех факультетов с указанием их кафедр и преподавателей.

```
SELECT f.NAME_FACULTETA AS 'Факультет', k.Name_Kafedru AS  
'Кафедра',  
t.NAME_TEACHER AS 'Преподаватель' FROM FACULTET f JOIN  
KAFEDRA k  
ON f.kod_faculteta =k.kod_faculteta  
JOIN TEACHER t  
ON k.kod_kafedru = t.kod_kafedru;
```

Следующий запрос Transact-SQL выполняет поиск наименований всех факультетов определенной кафедры и имена преподавателей этих кафедр.

Обратите внимание, что ни один из соединяемых столбцов — ни kod\_faculteta, ни kod\_kafedru, не включается в результаты. Тем не менее, соединение возможно только при использовании Kafedra в качестве промежуточной таблицы.

Среднюю таблицу соединения, Kafedra, можно назвать таблицей преобразования, или промежуточной таблицей, так как Kafedra является промежуточной точкой объединения, которая находится между двумя другими участвующими в объединении таблицами.

При наличии в инструкции нескольких операторов соединения, применяющихся либо при соединении более двух таблиц, либо при

соединении более двух пар столбцов, выражения соединения могут быть связаны операторами AND или OR.

**Задание:**

Для созданной базы данных, согласно номеру варианта, самостоятельно создать на языке SQL 15 многотабличных запросов:

1. 1 запрос с использованием декартового произведения двух таблиц;
2. 3 запроса с использованием соединения двух таблиц по равенству;
3. 1 запрос с использованием соединения двух таблиц по равенству и условием отбора;
4. 1 запрос с использованием соединения по трем таблицам;
5. создать копии ранее созданных запросов на соединение по равенству на запросы с использованием внешнего полного соединения таблиц (JOIN).
6. 1 запрос с использованием левого внешнего соединения;
7. 1 запрос на использование правого внешнего соединения;
8. 1 запрос с использованием симметричного соединения и удаление избыточности.

Все программные инструкции команд SQL сохранять в файлах с расширением \*.sql в папке ФИО\_студента

Для каждого запроса сформулировать текстовое задание, которое должно быть выполнено к базе данных.

Создать текстовый отчет, в котором отобразить sql-команды разработанных запросов и скриншоты результатов работы из СУБД SQL Server Management Studio.

## **ПРАКТИЧЕСКИЕ ЗАНЯТИЯ 12. Работа с журналом аудита базы данных. Мониторинг нагрузки сервера**

**Цель:** изучение мониторинга работы сервера

### *Теоретические сведения*

Наблюдение за базами данных выполняется с целью оценки производительности сервера. Эффективное наблюдение подразумевает регулярное создание моментальных снимков текущей производительности для обнаружения процессов, вызывающих неполадки, и постоянный сбор данных для отслеживания тенденций роста или изменения производительности.

Постоянная оценка производительности базы данных помогает добиться оптимальной производительности путем минимизации времени ответа и максимального увеличения пропускной способности. Приблизительный сетевой трафик, дисковый ввод-вывод и загрузка ЦП — ключевые факторы, влияющие на производительность. Следует тщательно проанализировать требования приложения, понять логическую и физическую структуру данных, оценить использование базы данных и добиться компромисса между такими конфликтующими нагрузками, как оперативная обработка транзакций (OLTP) и поддержка решений.

### Мониторинг и настройка производительности баз данных

В состав Microsoft SQL Server и операционной системы Microsoft Windows входят служебные программы, позволяющие следить за текущим состоянием базы данных и измерять производительность, если это состояние меняется. Для наблюдения за Microsoft SQL Server можно использовать целый ряд средств и методик. Наблюдение за SQL Server позволяет решать следующие задачи:

Определять возможности увеличения производительности. Например, выполняя мониторинг времени ответа для часто используемых запросов, можно определить, требуется ли изменить текст запроса или индексы таблицы.

Оценивать активность пользователей. Например, выполняя мониторинг пользователей, которые подключаются к экземпляру SQL Server, можно определить, правильно ли настроены параметры безопасности, и проверить работу приложений и систем разработки. Контролируя выполнение SQL-запросов, можно определить, правильно ли они написаны, и проверить результаты, которые они возвращают.

Устранять проблемы или отлаживать компоненты приложений, например хранимые процедуры.

### Мониторинг в динамической среде

Изменение этих условий приведет к изменению производительности. По результатам оценки можно заметить изменения производительности при увеличении числа пользователей, изменении методов доступа пользователей



и методов соединения, при увеличении объема содержимого базы данных, изменении клиентского приложения и данных в приложении, а также при усложнении запросов и увеличении объема сетевого трафика. С помощью средств контроля производительности можно связывать изменения отдельных показателей производительности с изменениями условий и сложных запросов. Примеры:

Отслеживая время отклика на часто используемые запросы, можно определить, нужно ли изменять запросы или индексы опрашиваемых таблиц.

Отслеживая выполнение запросов Transact-SQL можно определить правильность их написания, а также соответствие ожидаемым результатам.

Отслеживая пользователей, пытающихся подключиться к экземпляру SQL Server, можно проверить надежность защиты и протестировать приложения или системы разработки.

Время отклика — это время ожидания возврата пользователю первой строки результирующего набора в форме визуального подтверждения обработки запроса. Пропускная способность — это общее количество запросов, которые сервер может обработать за единицу времени.

С увеличением числа пользователей растет соперничество за ресурсы сервера, что в свою очередь увеличивает время ответа и уменьшает общую пропускную способность.

Задачи наблюдения и настройки производительности

Мониторинг компонентов SQL Server Необходимые действия для мониторинга компонентов SQL Server, такие как монитор активности, расширенные события, динамические административные представления и функции и т. д.

Средства контроля и настройки производительности Список средств наблюдения и настройки, доступных в SQL Server, например статистики динамических запросов и помощник по настройке ядра СУБД.

Обновление баз данных с помощью помощника по настройке запросов Поддержание стабильной производительности рабочей нагрузки во время обновления до нового уровня совместимости базы данных.

Мониторинг производительности с использованием хранилища запросов Использование хранилища запросов для автоматической регистрации журнала запросов, планов и статистики выполнения и сохранение этих данных для просмотра.

Формирование базовых показателей производительности Инструкции по формированию базовых показателей производительности.

Локализация проблем производительности Локализация проблем производительности базы данных.

Выявление узких мест Наблюдение за производительностью сервера и отслеживание его работы для выявления узких мест.

Использование динамических административных представлений для определения статистики использования и производительности представлений

Рассматриваются методы и скрипты, используемые для получения информации о производительности запросов.

Мониторинг производительности и действий сервера Использование средств наблюдения за производительностью и активностью SQL Server и Windows.

Отслеживание использования ресурсов Использование системного монитора (также известного как perfmon) для измерения производительности SQL Server с помощью счетчиков производительности.

### **Ход работы:**

Задание. Изучить инструменты диагностики SQL Server

Администраторы баз данных проводят диагностику SQL Server с помощью встроенных инструментов и скриптов, облегчающих использование этих инструментов. Скрипты в данном случае могут быть как отдельными программами, так и программными файлами, интегрируемыми в структуру софта.

В SQL Server встроено два инструмента: Activity Monitor (монитор активности) и Data Collector (сборщик данных). С их помощью выполняются практически все задачи диагностики. Activity Monitor: функции, задачи, преимущества и недостатки использования

Activity Monitor – это утилита, позволяющая оценивать активность пользователей приложения или сети. Она показывает текущее состояние SQL Server, осуществляемые на момент проверки процессы и то, как они отражаются на производительности СУБД.

Activity Monitor выглядит как окно с несколькими вкладками. Администратор базы данных может открыть такие панели:

Overview (обзор). На этой панели демонстрируется время обработки запросов процессором, количество ожидающих запросов, количество запросов в секунду, ввод и вывод данных.

Processes (процессы). На этой панели отражаются все активные процессы и подробная информация по ним. В Processes также можно запустить скрипт, который автоматически анализирует выбранный процесс.

Resource Waits (ожидающие ресурсы). На этой панели отображается, какие ресурсы необходимы СУБД для выполнения заданных функций. В перечень ресурсов входит объем оперативной памяти и сервера, сети, компиляция и др. В этой же панели администратор базы данных может просмотреть общий и средний промежуток времени ожидания ресурсов.

Data File I/O (ввод-вывод данных). На этой панели отражаются все операции, связанные с внесением изменений в файлы БД, а также полная информация об этих файлах.

Recent Expensive Queries (последние ресурсоемкие запросы). На этой панели отражаются те запросы, которые были выполнены в течение ближайших 30 секунд, и обработка которых затребовала наибольшего числа ресурсов. В некоторых версиях SQL Server эта панель называется Activity Expensive Queries (активные ресурсоемкие запросы).

Сбор и обработка данных с помощью Activity Monitor рис.29 ведется в режиме реального времени и только при условии разворачивания панели. Если администратор базы данных сворачивает панель, обработка информации прекращается. При этом на экране можно развернуть все пять панелей, чтобы оценить активность пользователей по разным параметрам.

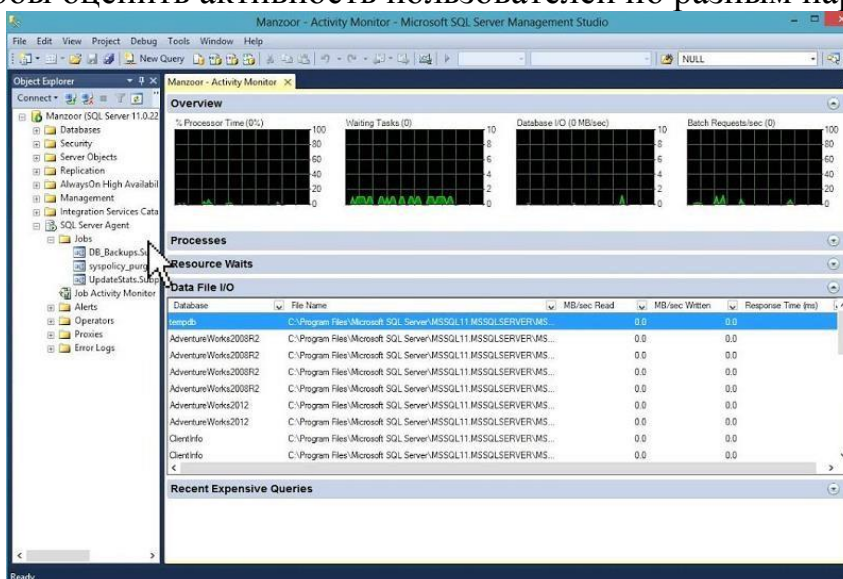


Рисунок 29- Сбор и обработка данных с помощью Activity Monitor

Для облегчения процесса использования Activity Monitor можно фильтровать, сортировать и менять местами столбцы с данными диагностики. Это делается с помощью компьютерной мышки прямо на развернутой панели.

Преимуществом использования Activity Monitor является то, что использование этого инструмента практически не отражается на производительности СУБД.

Что касается недостатков этого инструмента, в их число можно включить:

1. ограниченное количество параметров, по которым ведется диагностика;
2. отсутствие эталона, то есть шаблона с указанием допустимых значений параметров;
3. отсутствие возможности формирования отчета о результатах проверки для их последующего анализа.

Из этого можно сделать вывод, что Activity Monitor – это прекрасный инструмент для проведения быстрой диагностики и поиска запросов, затратных с точки зрения потребления ресурсов.

**ШОВКАРОВА Зарина Сейтбиевна**

# **МДК.07.01.УПРАВЛЕНИЕ И АВТОМАТИЗАЦИЯ БАЗ ДАННЫХ**

**ПРАКТИКУМ**

для обучающихся IV курса специальности  
09.02.07 Информационные системы и программирование

Корректор Чагова О.Х.  
Редактор Чагова О.Х.

Сдано в набор 22.08.2023г.  
Формат 60x84/16  
Бумага офсетная.  
Печать офсетная.  
Усл. печ. л. 4,88.  
Заказ № 4760  
Тираж 100 экз.

Оригинал-макет подготовлен  
в Библиотечно-издательском центре СКГА  
369000, г. Черкесск, ул. Ставропольская, 36