Лабораторный практикум для работы с MS SQL Server.

**Лаборат****орная работа №1**

**«Установка соединения с сервером Microsoft SQL Server,**

**создание, копирование и удаление базы данных»**

Время выполнения: 60 минут.

Цель: Познакомиться с основными принципами создания базы данных в MS SQL Server. Получить навыки для создания, удаления, резервного копирования и восстановления базы данных. Изучить SQL-операторы для создания, подключения и удаления базы данных.

Методические указания.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL. Для выполнения работ из практикума рекомендуется использовать версию 2012 или 2019.

Для начала давайте вспомним, что такое база данных, и из каких объектов она состоит. Как правило БД – это не только таблицы и связи между ними, но также представления, процедуры и функции. С ними мы будем работать в следующих лабораторных работах. Такая система хранится в двух и более файлах.

Давайте рассмотрим из каких файлов состоит БД.

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.

Каждый файл может использоваться лишь одной базой данных.

Также существует такое понятие как 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.

В SQL Server используется два типа баз данных: системные и пользовательские. Системные базы данных необходимы серверу SQL для корректной работы. А пользовательские базы данных создаются пользователями сервера и могут хранить любую произвольную информацию. Их можно изменять и удалять, создавать заново. Собственно, это те базы данных, которые мы будем создавать и с которыми мы будем работать.

К системным базам данных относятся: master, model, msdb, tempbd. База данных master является самой главной базой данных сервера, в случае ее отсутствия или повреждения сервер не сможет работать. Она хранит все используемые логины пользователей сервера, их роли, различные конфигурационные настройки, имена и информацию о базах данных и другую информацию. Также, если не было дополнительных настроек, она является базой данных по умолчанию, то есть при подключении к серверу, вы автоматически начинаете работу в БД master.

*Самостоятельно прочитайте про назначение и размещение системных БД.*

Теперь можем приступать к работе с сервером. Для начала убедитесь, что он запущен.

Для этого запустите SQL Server Configuration Manager. В диспетчере конфигурации найдите «Службы SQL Server» и убедитесь, что SQL Server находится в состоянии «Работает». Если нет, нажмите правой кнопкой мыши на состояние и выберите «Запустить».

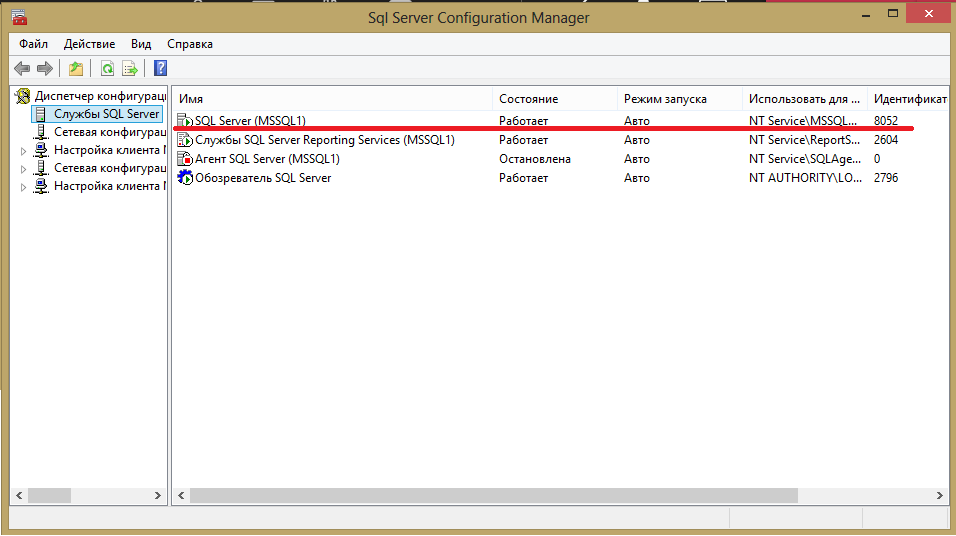


Рис.1. Правильное состояние SQL Server

После того, как вы убедились, что сервер запущен, можете запускать Microsoft SQL Management Studio.

У вас появится окно «Соединение с сервером». Давайте его рассмотрим подробнее.

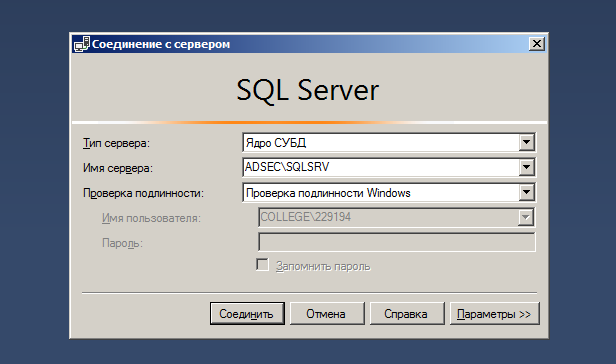


Рис.2 Окно «Соединение с сервером»

Выберите в качестве типа сервера «Компонент Database Engine» и введите имя сервера (если оно вам неизвестно, спросите у преподавателя).

Обратите внимание, что в MS SQL Server поддерживается два варианта проверки подлинности: Windows и SQL Server.

После ввода данных нажмите кнопку «Соединить». Вам откроется следующий интерфейс.

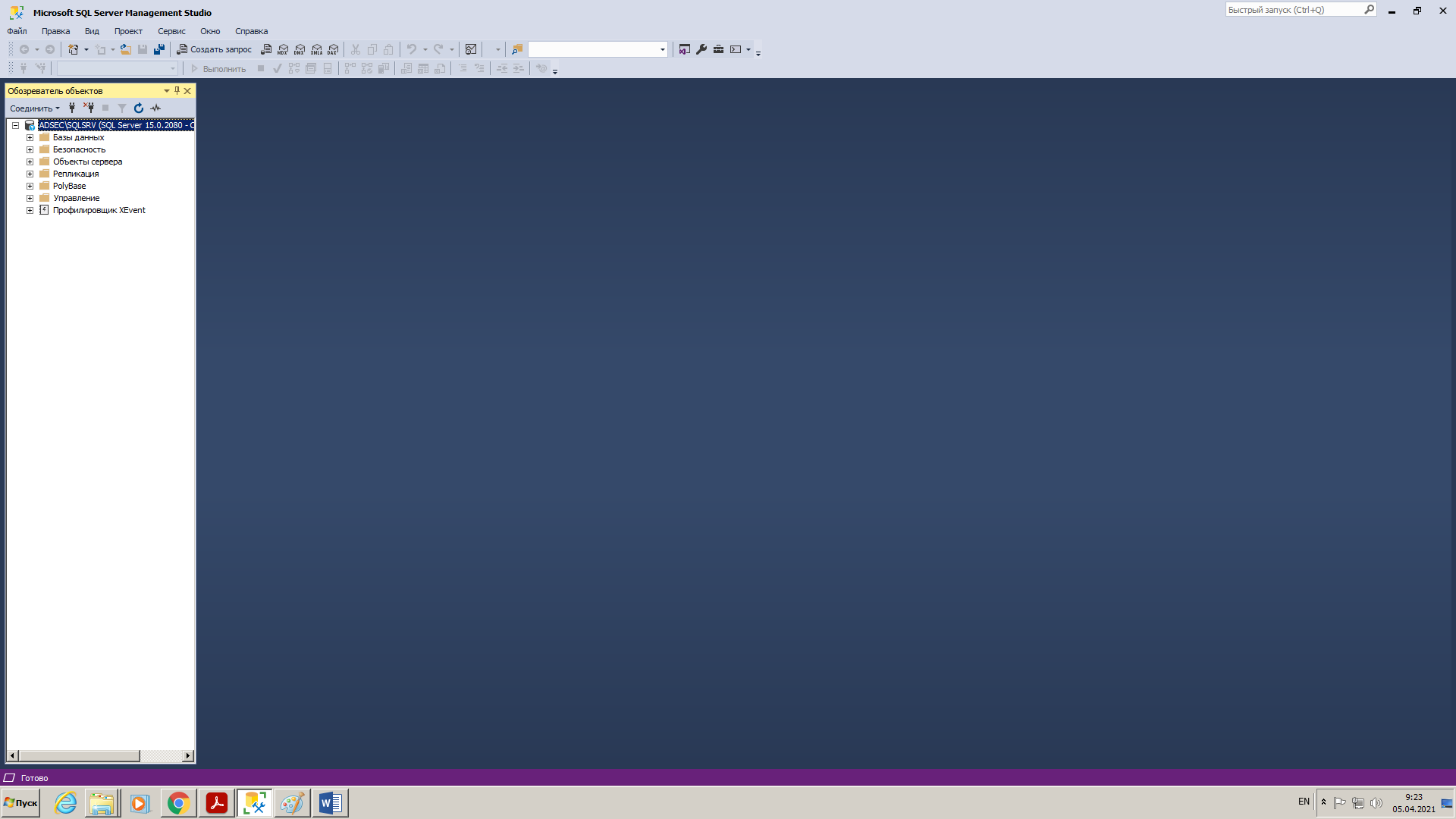


Рис. 3 Окно работы с MS SQL Server Management Studio

Создать базу данных можно двумя способами. Давайте рассмотрим каждый из них.

Первый способ – через Обозреватель объектов. На вашем сервере нажмите на папку «Базы данных» и в выпадающем списке выберите «Создать базу данных…» (см. Рис. 4).

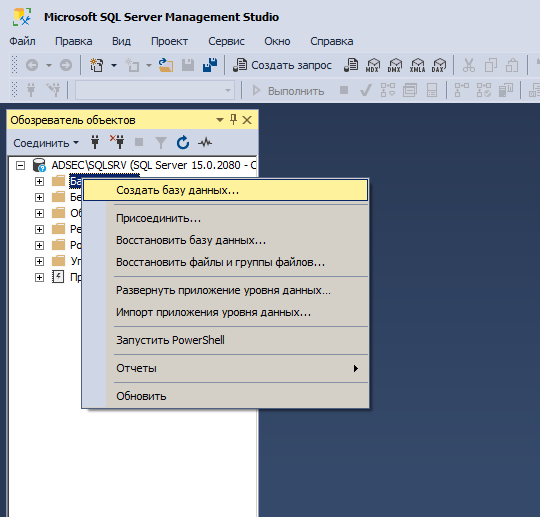


Рис. 4 «Создать базу данных…»

Вам откроется окно (см. Рис. 5). Для начала работы вам достаточно ввести имя базы данных и нажать кнопку «ОК». Но давайте рассмотрим, какие еще опции нам доступны при создании.

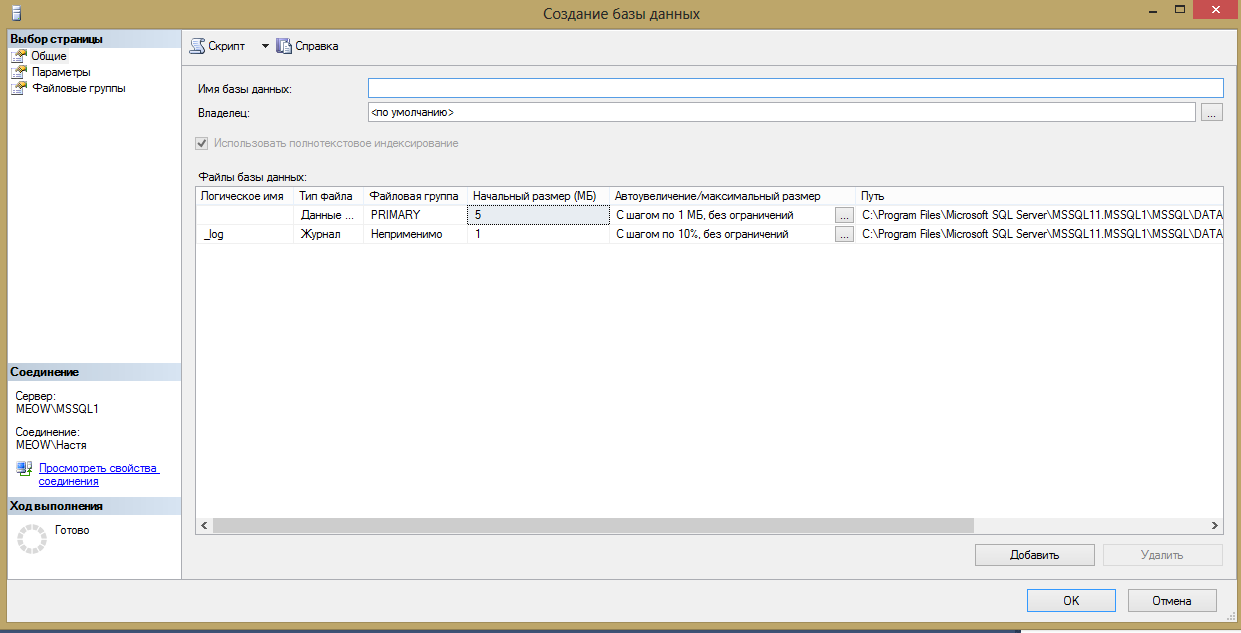


Рис. 5. Окно «Создание базы данных»

На первой вкладке мы можем задать начальный размер нашей БД, максимальный размер, а также шаг автоувеличения. По умолчанию они соответственно равны 5МБ и 1МБ.

На вкладке «Параметры» можно установить особые настройки. Такие как язык БД, уровень совместимости, параметры восстановления и другие.

После введения всех необходимых данных нажмите “ok”. Слева в обозревателе объектов появится ваша новая БД. Помимо БД создается sql-скрипт для ее создания. Выберите вашу БД, нажмите правой кнопкой мыши. Далее *Создать скрипт для БД -> Используя CREATE -> Новое окно редактора запросов*.

Второй способ – с помощью SQL-скрипта.

Для этого на панели инструментов выберите команду «Создать запрос» (либо используйте сочетание клавиш Ctrl+N).

Команда для создания БД – CREATE DATABASE.

Синтаксис команды выглядит так:

CREATE DATABASE имя\_бд

ON

( NAME = логическое\_имя\_файла,

FILENAME = 'имя\_файа\_ос',

SIZE = размер, --пример 10

MAXSIZE = максимальный\_размер, --пример 30 или UNLIMITED

FILEGROWTH = шаг\_автоувеличения)

LOG ON --далее данные файла журнала

(NAME = логическое\_имя\_файла,

FILENAME = 'имя\_файа\_ос',

SIZE = размер, --пример 10

MAXSIZE = максимальный\_размер, --пример 30 или UNLIMITED

FILEGROWTH = шаг\_автоувеличения)

) ;

При создании БД можно не указывать дополнительные данные, а просто использовать команду CREATE DATABASE имя\_бд. В таком случае будет создана БД со стандартным размером и именами файлов.

После выполнения кода, ваша БД будет создана.

Как уже говорилось выше после подключения к серверу, если не было дополнительных установок, вы начинаете работу с системной БД master. Чтобы подключиться к другой БД, достаточно сделать двойной щелчок мыши по нужной БД. Если нет ограничений доступа, после этих действий вам будут доступны ее элементы в Обозревателе объектов.

Удалить базу данных можно тремя способами.

1. Через контекстное меню «Удалить».
2. Через sql-скрипт c помощью команды DROP DATABASE. Полный синтаксис выглядит так: DROP DATABASE имя\_БД.
3. Просто удалив файл БД.

Любая современная СУБД имеет систему резервного копирования и восстановления.

Резервное копирование (backup) базы данных и восстановление из резервной копии (restore) – два важнейших и наиболее частых процесса, осуществляемых администраторами баз данных.

Резервное копирование базы данных – единственный надежный способ предохранить данные от потери в результате поломки диска, сбоев электропитания, действий злоумышленников и ошибок в программах. В процессе резервного копирования создается независимый от платформы "снимок" базы данных, с помощью которого можно перенести данные на другую операционную систему или даже другую платформу.

Для создания резервной копии базы данных с помощью программы "SQL Server Management Studio" необходимо подключиться к базе данных, выбрать из контекстного меню базы данных *Задачи-> Создать резервную копию*. В открывшемся диалоговом окне "Мастер резервного копирования" задать несколько параметров и нажать кнопку *Выполнить*.

После выбора пути и файла для резервной копии в окне Back Up Database нажатием на OK запускаем процесс создания резервной копии. В случае успешной работы появится сообщение.

В результате будет создан файл с резервной копией. Стандартным расширением таких файлов для "SQL Server Management Studio" является "\*.bak". Файл с резервной копией базы данных обычно на порядок меньше оригинала.

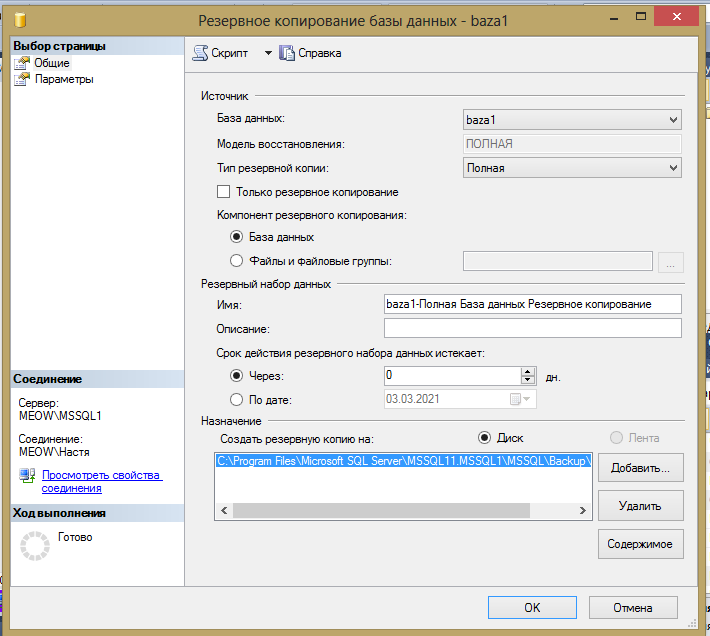


Рис. 6. Окно "Мастер резервного копирования"

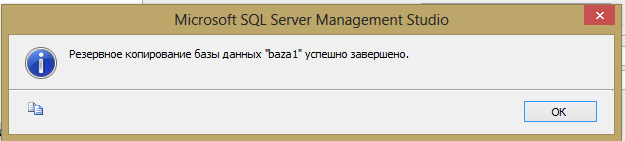


Рис. 7. Сообщение об успешном резервном копировании

Для восстановления базы данных из резервной копии используется команда *База данных -> Восстановление базы данных*. В результате откроется диалоговое окно "Мастер восстановления баз данных", в котором надо выбрать имя БД куда будет восстанавливаться база данных, в которую будет помещен результат, способ восстановления, файл, из которого будет восстанавливаться база данных, отмечаем выбранную резервную копию, и нажать кнопку *Восстановить*. Запускаем процесс восстановления. В случае успешного выполнения получим сообщение.

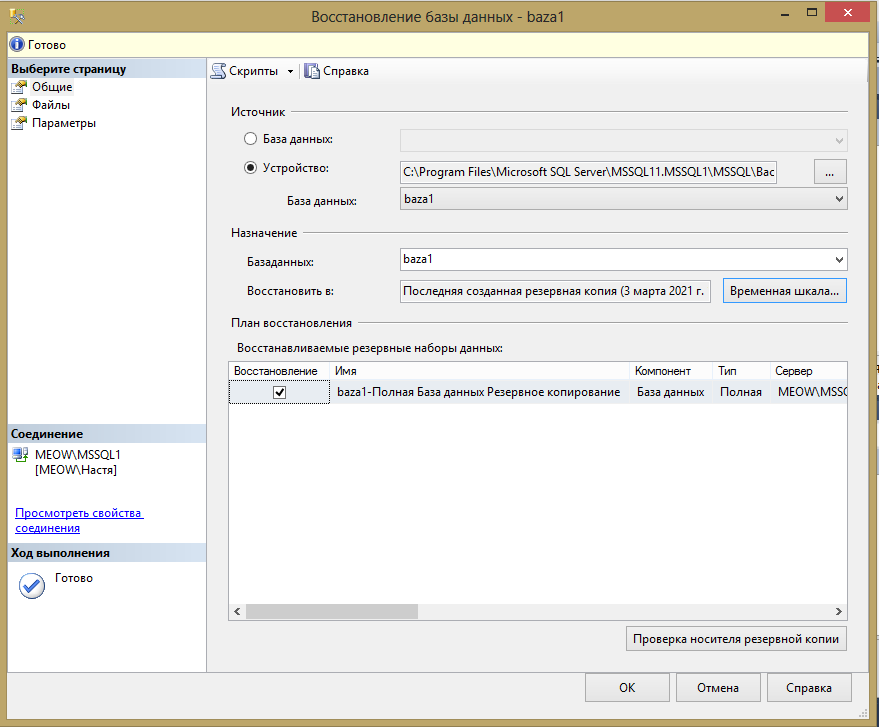


Рис. 8. Окно "Мастер восстановления баз данных"

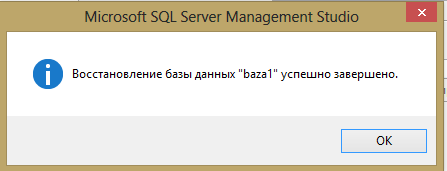


Рис. 9. Сообщение об успешном восстановлении

Задания на лабораторную работу.

1. Откройте SQL Server Management Studio.
2. Подключитесь к серверу.
3. Создайте базу данных с помощью контекстного меню с именем *Ваша\_Фамилия\_Номер\_группы\_номер\_в\_журнале.*
4. Извлеките команду для создания БД.
5. Удалите БД с помощью контекстного меню.
6. Создайте БД с помощью скрипта с именем *ВашаФамилия\_ПредметнаяОбластьИзЛР1.*
7. Создайте резервную копию этой БД.
8. Удалите БД с помощью SQL-кода.
9. Восстановите БД из файла с резервной копией.
10. Сохраните файл сценария.

Отчет должен содержать:

1. Титульный лист с номером и названием работы, фамилией студента и номером группы.
2. Цель работы.
3. Подписанные скриншоты выполненных заданий.
4. Коды всех выполненных команд.
5. Код из пункта 4.

**Лабораторная работа № 2**

**«Импорт данных. Работа с csv и txt файлами»**

Время выполнения: 40 минут.

Цель: Научиться правильно импортировать данные разных форматов, а также экспортировать данные.

Методические указания.

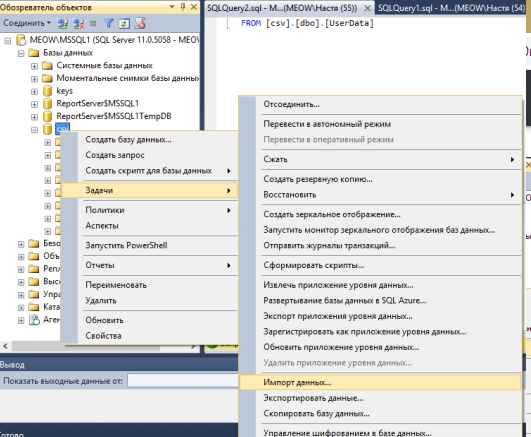
Данные в csv и txt файлах, хранятся в виде строк, где атрибуты разделены между собой запятыми. В MS SQL Server работать с такими файлами довольно просто, благодаря Мастеру импорта и экспорта. Однако, надо работать внимательно. Чтобы импортировать данные из csv или txt файла, зайдите в Мастера импорта и экспорта: нажмите правой кнопкой мыши на вашу БД в обозревателе объектов, затем выберете пункт «Задачи», а далее «Импорт данных». 

Рис. 10. Импорт данных

В качестве источника данных выберите «Неструктурированный файл». В обзоре файлов выберите необходимый файл.

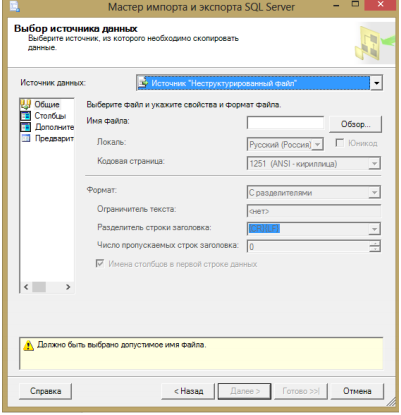


Рис. 11. Выбор источника данных

Если его нет в предложенных убедитесь, что указан нужный формат.



Рис. 12. Убедитесь, что видны все файлы, а не только txt

Теперь давайте посмотрим на доступные функции «Неструктурированного файла».

Локаль используется для интерпретации данных, зависящих от локаля, например, дат, а кодовая страница используется для конвертации строковых данных в формат Юникод.

Текстовый файл может иметь разделители, фиксированную ширину полей или смешанный формат.

• В формате с разделителями для выделения столбцов и строк используются разделители столбцов и строк.

• В формате с фиксированной шириной для определения столбцов и строк используются параметры ширины. Этот формат также включает символ для заполнения поля до его максимальной ширины.

• В формате без выравнивания справа для выделения всех столбцов, кроме последнего, используются параметры ширины; для последнего столбца используется разделитель строк.

Ограничитель текста необязательная характеристика. Например, можно указать, что текстовые поля должны быть заключены в кавычки.

Ниже приведены объяснения возможных разделителей строки заголовка:

{CR}{LF} В качестве разделителей для строки заголовка используются сочетания символов возврата каретки и перевода строки.

{CR} В качестве разделителей для строки заголовка используются символы возврата каретки.

{LF} В качестве разделителей для строки заголовка используются символы перевода строки.

Точка с запятой {;} В качестве разделителя для строки заголовка используется точка с запятой.

Двоеточие {:} В качестве разделителя для строки заголовка используется двоеточие.

Запятая {,} В качестве разделителя для строки заголовка используется запятая.

Табуляция {t} В качестве разделителя для строки заголовка используется символ табуляции.

Вертикальная черта {|} В качестве разделителя для строки заголовка используется вертикальная черта.

Количество строк заголовка или строк начальных данных – это данные которые следует пропускать при необходимости. Не забудьте указать ожидать или задать имена столбцов в первой строке данных. Далее укажите разделители строк и столбцов. Вам будет доступен предварительный просмотр. Выберите его и посмотрите, как будет выглядеть ваше отношение после импорта.

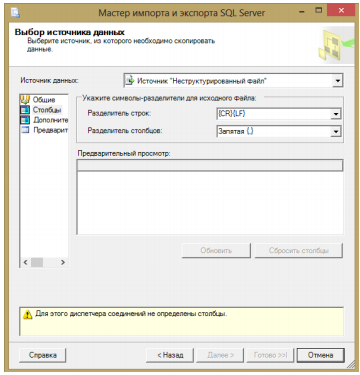


Рис. 12. Предварительный просмотр

Как правило, при импорте все данные будут иметь тип varchar. Это не очень удобно, например, если вы храните *Даты, Цены* или *Числа.* Конечно, это можно изменить уже при импорте.

Нажимайте далее, пока не дойдете до окна «Выбор исходных таблиц и представлений».

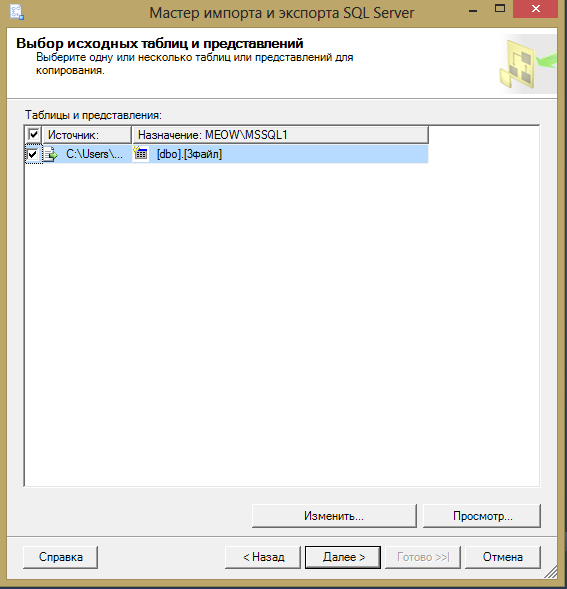


Рис. 13. «Выбор исходных таблиц и представлений»

Внизу окна есть кнопка «Изменить», после нажатия вам будет доступно окно настроек отношений. Вы сможете разрешить или изменить значение NULL, изменить тип или длину атрибута.

После завершения работы, выберите «Готово».

Задания на лабораторную работу.

1. Спросите у преподавателя, где находятся файлы для импорта.
2. Импортируйте каждый и них в вашу БД.
3. Следуя указаниям «мастера импорта и экспорта» экспортируйте любую таблицу в csv или txt файл.
4. Вставьте в отчет выбранные вами свойства и результат экспорта.

**Лабораторная работа №3.**

**«Создание и заполнение таблиц»**

Время выполнения: 80 минут.

Цель: Научиться создавать отношения и заполнять их. Получить навык создания БД с помощью даталогической модели.

Методические указания.

Невозможно представить себе базу данных без отношений. Создать таблицу можно двумя способами: с помощью конструктора или скрипта. Первый способ мы рассмотрим с вами позже, а в этой работе будем использовать скрипты.

Перед тем, как мы рассмотрим непосредственно само создание, давайте вспомним основные типы данных в MS SQL Server Management. Конечно, их намного больше, полный список вы можете изучить самостоятельно.

**Числовые типы данных**

* **BIT**: хранит значение 0 или 1. Фактически является аналогом булевого типа в языках программирования. Занимает 1 байт.
* **INT**: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта. Наиболее используемый тип для хранения чисел.
* **DECIMAL**: хранит числа c фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой.
* **MONEY**: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. Представляет денежные величины и занимает 8 байт. Эквивалентен типу DECIMAL(19,4).
* **FLOAT**: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части.
* **REAL**: хранит числа от –340E+38 to 3.40E+38. Занимает 4 байта. Эквивалентен типу FLOAT(24).

**Типы данных, представляющие дату и время**

* **DATE**: хранит даты от 0001-01-01 (1 января 0001 года) до 9999-12-31 (31 декабря 9999 года). Занимает 3 байта.
* **TIME**: хранит время в диапазоне от 00:00:00.0000000 до 23:59:59.9999999. Занимает от 3 до 5 байт.

Может иметь форму TIME(n), где n представляет количество цифр от 0 до 7 в дробной части секунд.

* **DATETIME**: хранит даты и время от 01/01/1753 до 31/12/9999. Занимает 8 байт. Есть несколько способов записать дату, например: ‘гггг-мм-дд’ или ‘дд/мм/гггг ‘.

Время можно записывать, например так:’чч:мм’, ‘чч:мм:сс’.

**Строковые типы данных**

* **CHAR**: хранит строку длиной от 1 до 8 000 символов. На каждый символ выделяет по 1 байту. Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

Количество символов, которое может хранить столбец, передается в скобках. Например, для столбца с типом CHAR(10) будет выделено 10 байт. И если мы сохраним в столбце строку менее 10 символов, то она будет дополнена пробелами.

* **VARCHAR**: хранит строку. На каждый символ выделяется 1 байт. Можно указать конкретную длину для столбца - от 1 до 8 000 символов, например, VARCHAR(10). Если строка должна иметь больше 8000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб: VARCHAR(MAX).

Не подходит для многих языков, так как хранит символы не в кодировке Unicode.

В отличие от типа CHAR, если в столбец с типом VARCHAR(10) будет сохранена строка в 5 символов, то в столбце будет сохранено именно пять символов.

* **NCHAR**: хранит строку в кодировке Unicode длиной от 1 до 4 000 символов. На каждый символ выделяется 2 байта. Например, NCHAR(15).
* **NVARCHAR**: хранит строку в кодировке Unicode. На каждый символ выделяется 2 байта. Можно задать конкретный размер от 1 до 4 000 символов. Если строка должна иметь больше 4000 символов, то задается размер MAX, а на хранение строки может выделяться до 2 Гб.

**Бинарные типы данных**

* **BINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт.
* **VARBINARY**: хранит бинарные данные в виде последовательности от 1 до 8 000 байт, либо до 2^31–1 байт при использовании значения MAX (VARBINARY(MAX)).

Теперь давайте вспомним об одном ограничении, которое можно накладывать на любой атрибут, а именно возможность оставлять его значения пустым. Как правило, запрет накладывается на атрибуты, которые являются первичными или внешними ключами. По умолчанию все атрибуты НЕОБЯЗАТЕЛЬНЫ к заполнению. А это свойство регулируется с помощью скрипта null/not null.

Вспомнив необходимые параметры, можем создать отношения.

В общем виде он выглядит так:

CREATE TABLE table\_name(

column1 datatype <NOT NULL>,

column2 datatype <NOT NULL>,

column3 datatype <NOT NULL>,

.....

columnN datatype <NOT NULL>)

Например, чтобы создать таблицу с названием Студенты, атрибутами: id, Имя, Фамилия, Адрес, причем id обязателен к заполнению, код будет следующим:

CREATE TABLE Студенты(

id int NOT NULL,

Имя nvarchar(15),

Фамилия nvarchar(25),

Адрес nvarchar(50))

Для заполнения таблиц воспользуемся построчным заполнением. Код для построчного заполнения выглядит так:

INSERT INTO TABLE\_NAME [(column1, column2, column3,...columnN)]

VALUES (value1, value2, value3,...,valueN)

Если вы хотите внести данные для всех атрибутов, указывать их список после названия таблицы. Это особенно актуально, если вы вносите только некоторые данные. Например, только id и Фамилию. Давайте рассмотрим пример, для заполнения двух строк из предыдущего примера.

INSERT INTO Студенты

VALUES (1, ‘Карина’, ‘Петрова’,’пр. Художников 20’,

(2, ‘Игнат’, ‘Петров’,’пр. Художников 20’)

Если бы нам было необходимо внести только фамилию и id (без него никак, так как мы уже указали в настройках, что этот атрибут обязательный к заполнению), то код будет следующим:

INSERT INTO Студенты (id, Фамилия)

VALUES (3, ‘Бровина’)

Задания на лабораторную работу.

На основании даталогической модели из лабораторной работы создайте в вашей БД таблицы, а также заполните их минимум пятью строками.

**Лабораторная работа № 4.**

**«Разработка ограничений»**

Время выполнения: 80 минут.

Цель: Изучить используемые в SQL Server типы ограничений. Изучить SQL-операторы для работы с ограничениями.

Методические указания.

Ограничение – это очень важный объект БД. Это один из элементов, с помощью которого мы можем обеспечить целостность для нашей БД. Это механизм, обеспечивающий автоматический контроль соответствия данных установленным условиям, или ограничениям целостности.

Имеется пять ограничений целостности, различающихся по функциональности и области применения:

**NULL** - действует на уровне столбца и пользовательского типа данных и либо разрешает (NULL), либо запрещает (NOT NULL) хранение значений NULL.

**CHECK** - действует на уровне столбца и ограничивает диапазон значений, которые могут быть сохранены в столбце, путем проверки логического условия для вводимых данных. При вводе или изменении данных вводимое значение подставляется в условие. Если полученный результат TRUE, то изменения данных принимаются, иначе - отвергаются и генерируется сообщение об ошибке. Для одного столбца можно задать несколько ограничений типа CHECK.

**UNIQUE** - действует на уровне столбца и гарантирует уникальность в столбце вводимых значений. В отличии от ограничения PRIMARY KEY, это ограничение допускает хранение значений NULL.

**PRIMARY KEY** - действует на уровне столбца или таблицы и гарантирует уникальность в пределах таблицы первичного ключа, состоящего из одного или нескольких столбцов. Ни для одного из столбцов ключа не должно быть установлено свойство NULL. Когда используется один столбец, то для него необходимо также задать и свойство UNIQUE. В таблице создается только один первичный ключ.

**FOREIGN KEY** - действует на уровне таблицы и связывается с одним из кандидатов на первичный ключ в другой таблице. Таблица, в которой определен внешний ключ с помощью этого ограничения, называется зависимой, а таблица с кандидатом на первичный ключ - главной. В зависимую таблицу нельзя вставить строку, если внешний ключ не имеет соответствующего значения в главной таблице. Из главной таблицы нельзя удалить строку, если с ней связана хотя бы одна строка в зависимой таблице.

Формат задания ограничения таков: FOREIGN KEY REFERENCES имя главной таблицы (кандидат на первичный ключ или ее ключ).

В конце используется

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

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

Эти строки указывают, какое действие совершается над строками в изменяемой таблице, когда эти строки имеют ссылочную связь и строка родительской таблицы, на которую указывает ссылка, обновляется. Параметр по умолчанию — NO ACTION.

**NO ACTION** Компонент Компонент Database Engine возвращает ошибку, а обновление строки родительской таблицы откатывается.

**CASCADE** Соответствующие строки обновляются в ссылающейся таблице, если эта строка обновляется в родительской таблице.

**SET NULL** Всем значениям, составляющим внешний ключ, присваивается значение NULL, когда обновляется соответствующая строка в родительской таблице. Для выполнения этого ограничения внешние ключевые столбцы должны допускать значения NULL.

**SET DEFAULT** Всем значениям, составляющим внешний ключ, присваивается их значение по умолчанию, когда обновляется соответствующая строка в родительской таблице.

Ограничения можно добавлять во время создания отношения или добавить к уже созданным. Рассмотрим общий синтаксис и примеры для каждого из этих случаев.

CREATE TABLE table\_name(

column1 datatype <not null> constraint type,

column2 datatype <not null> constraint type,

column3 datatype <not null> constraint type,

.....

columnN datatype <not null> constraint type)

Например, чтобы создать таблицу с названием Студенты, атрибутами: id, Имя, Фамилия, Адрес, причем id обязателен к заполнению, а также является первичным ключом, код будет следующим:

CREATE TABLE Студенты ( id int NOT NULL primary key,

Имя nvarchar(15),

Фамилия nvarchar(25),

Адрес nvarchar(50))

Чтобы создать таблицу Успеваемость с атрибутами id и средний балл, где id – внешний ключ, связывающий эту таблицу с таблицей Студенты, следует использовать следующий код.

CREATE TABLE Успеваемость ( id int NOT NULL foreign key references Студенты(id) on delete cascade on update cascade,

[Средний балл] float)

Теперь рассмотрим пример создания таблицы Пациенты с атрибутами id, номер медкнижки, ФИО и номером телефона. Причем id – первичный ключ, номер медкнижки – уникальные значения, а номер телефона имеет ограничение ввода данных типа +7(000)000-00-00, где 0 – любая цифра.

CREATE TABLE Пациенты( id int NOT NULL primary key,

[номер медкнижки] int unique,

ФИО nvarchar(50),

[номер телефона] char(16) check ([номер телефона]

like ‘+7([0-9][0-9][0-9])[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]’))

Далее давайте рассмотрим примеры, когда нам необходимо добавить ограничения к уже созданным таблицам. Любой такой запрос будет начинаться со слов alter table tablename. Позже мы будем отдельно рассматривать, как изменять те или иные объекты в БД. Далее мы пишем add constraint и тип ограничения.

Приведу код на добавление таких же ограничений, как и в предыдущих примерах, но, если бы мы их добавляли к уже существующим таблицам.

Добавление первичного ключа в таблице Студенты:

ALTER TABLE Студенты

ADD CONSTRAINT C1 –-добавляем ограничение с именем с1

PRIMARY KEY (id)—-тип ограничение пк, накладываем на атрибут id

Добавление внешнего ключа в таблице Успеваемость:

ALTER TABLE Успеваемость

ADD CONSTRAINT C2 –-добавляем ограничение с именем с2

FOREIGN KEY (id)—-тип ограничение вк, накладываем на атрибут id

REFERENCES Студенты (id) –-соединяем с атрибутом id таблицы

--Студенты

Добавление ограничения уникальности и ввода данных в таблице Пациенты

ALTER TABLE Пациенты

ADD CONSTRAINT C3 –-добавляем ограничение с именем с3

UNIQUE ([номер медкнижки])—-накладываем на атрибут номер

--медкнижки

ALTER TABLE Пациенты

ADD CONSTRAINT C4 –-добавляем ограничение с именем с4

CHECK ([номер телефона]

like ‘+7([0-9][0-9][0-9])[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9]’)

Задания на лабораторную работу.

* 1. На основании даталогической и инфологической моделей, создайте в вашей БД связи между отношениями и другие ограничения.
  2. Создайте таблицу Client, в которой будут храниться данные о Клиентах компании. В этом отношении спроектируйте следующие атрибуты и ограничения для них. Id – автоинкримент и первичный ключ, Имя, Фамилия и Отчество имеют только ограничения по длине, Возраст - от 14 до 99 лет, номер телефона - начинается с ‘+7’ и содержит еще 10 цифр после этого.

**Лабораторная работа № 5**

**«Работа с внешними ключами»**

Время выполнения: 70 минут.

Цель: Изучить синтаксис Foreign key и понятие ссылочной целостности. Изучить разницу между разными параметрами ключевых слов on delete on update.

Методические указания.

Ссылочная целостность—это состояние реляционной базы данных в которой записи не могут ссылаться на несуществующие записи в этой базе данных.

FOREIGN KEY—особый вид ограничения, которое позволяет предотвратить нарушение ссылочной целостности при удалении/изменении информации в таблицах предках.

Пример нарушения ссылочной целостности:

Пусть существуют две таблицы. Catalogs, являющаяся таблицей-предком, содержащие в себе упоминания о категориях товаров в интернет магазине, и таблица products являющаяся таблицей-потомком, со всеми товарами этого магазина.

|  |  |
| --- | --- |
| id\_catalog | name |
| 1 | Процессоры |
| 2 | Материнские платы |
| 3 | Видеоадаптеры |
| 4 | Жёсткие диски |
| 5 | Оперативная память |

Рис. 14. Таб. Catalogs

|  |  |  |
| --- | --- | --- |
| Id\_product | Name | Id\_catalog |
| 1 | Celeron 1.8 | 1 |
| 2 | Celeron 2.0GHz | 1 |
| 3 | Celeron 2.4GHz | 1 |
| 4 | Celeron D 320 2.4GHz | 1 |
| 5 | Celeron D 325 2.53GHz | 1 |
| 6 | Celeron D 315 2.26GHz | 1 |
| 7 | Intel Pentium 4 3.2GHz | 1 |
| 8 | Intel Pentium 4 3.0GHz | 1 |
| 9 | Intel Pentium 4 3.0GHz | 1 |
| 10 | Gigabyte GA-8I848P-RS | 2 |
| 11 | Gigabyte GA-8IG1000 | 2 |
| 12 | Gigabyte GA-8IPE1000G | 2 |
| 13 | Asustek P4C800-E Delux | 2 |
| 14 | Asustek P4P800-VM\L i865G | 2 |
| 15 | Epox EP-4PDA3I | 2 |
| 16 | ASUSTEK A9600XT/TD | 3 |
| 17 | ASUSTEK V9520X | 3 |
| 18 | SAPPHIRE 256MB RADEON 9550 | 3 |
| 19 | GIGABYTE AGP GV-N59X128D | 3 |
| 20 | Maxtor 6Y120P0 | 4 |
| 21 | Maxtor 6B200P0 | 4 |
| 22 | Samsung SP0812C | 4 |
| 23 | Seagate Barracuda ST3160023A | 4 |
| 24 | Seagate ST3120026A | 4 |
| 25 | DDR-400 256MB Kingston | 5 |
| 26 | DDR-400 256MB Hynix Original | 5 |
| 27 | DDR-400 256MB PQI | 5 |
| 28 | DDR-400 512MB Kingston | 5 |
| 29 | DDR-400 512MB PQI | 5 |
| 30 | DDR-400 512MB Hynix | 5 |

Рис. 15. Таб. Products

При удалении категории из таблицы catalogs, в таблице products останутся товары, которые не привязаны ни к одной из категорий, что может повлечь массу проблем для магазина. В результате следующего запроса получим отношения на рис. 16

DELETE FROM catalogs WHERE name = 'Процессоры'

SELECT \* FROM catalogs

SELECT \* FROM products WHERE id\_catalog = 1

|  |  |
| --- | --- |
| id\_catalog | name |
| 2 | Материнские платы |
| 3 | Видеоадаптеры |
| 4 | Жёсткие диски |
| 5 | Оперативная память |

|  |  |  |
| --- | --- | --- |
| Id\_product | Name | Id\_catalog |
| 1 | Celeron 1.8 | 1 |
| 2 | Celeron 2.0GHz | 1 |
| 3 | Celeron 2.4GHz | 1 |
| 4 | Celeron D 320 2.4GHz | 1 |
| 5 | Celeron D 325 2.53GHz | 1 |
| 6 | Celeron D 315 2.26GHz | 1 |
| 7 | Intel Pentium 4 3.2GHz | 1 |
| 8 | Intel Pentium 4 3.0GHz | 1 |
| 9 | Intel Pentium 4 3.0GHz | 1 |

Рис. 16. Результаты запроса

Это явление называется нарушением ссылочной целостности.

На ссылочную целостность базы данных как правило оказывают четыре типа изменений:

* Добавление новой записи в таблице-потомке. Например, добавление новой товарной позиции в таблицу products. Следует заметить, что важную роль играет изменение именно таблицы-потомка, т.к. изменение таблицы-предка (catalogs) не приведет к нарушению ссылочной целостности, т.к. наличие пустой категории товаров допустимо
* Обновление внешнего ключа в таблице-потомке. Эта ситуация похожа на первую и может произойти при изменении у товара ссылки на несуществующий раздел каталога, например, товар с id\_catalog равным 50.
* Удаление записи из таблицы-предка. Эта ситуация рассмотрена выше.
* Изменение записи в таблице-предке. Эта ситуация отличается от рассмотренной выше тем что категория каталога не удаляется, а принимает новый id.

Для того что бы контролировать ссылочную целостность в базе данных необходимо, чтобы таблицы были связаны при помощи конструкции FOREIGN KEY, которая имеет вид:

FOREIGN KEY [index\_name] (index\_col\_name, …)

REFERENCES tbl\_name (index\_col\_name,…)

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

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

FOREIGN KEY — используется при создании/изменении таблиц-потомков таблицах. В данном примере FOREIGN KEY, следует использовать в таблице products. Данная конструкция позволяет задать в таблице-потомке внешний ключ с именем index\_name на столбцах таблицы которые перечисляется в круглых скобках. Можно использовать один или несколько столбцов.

Ключевое слово REFERENCES задаёт таблицу-предка tbl\_name на которую будет ссылаться внешний ключ. Поля таблицы-предка задаются в круглых скобках, один или несколько.

Необязательные конструкции ON DELETE и ON UPDATE, определяют поведение MySQL при удалении/обновлении записей из таблицы-предка.

Допустимые параметры для ключевых слов ON DELETE и ON UPDATE:

* CASCADE — При удалении/обновлении записей в таблице-предке, будут так же обновлены/удалены записи из таблицы-потомка с существующим первичным ключом
* SET NULL — При удалении/обновлении записей в таблице-предке, записи из таблицы потомка с существующим первичным ключом будут обновлены на NULL
* NO ACTION — При удалении/обновлении записей в таблице-предке, записи из таблицы-потомка с существующим первичным ключом изменены не будут.

Добавление для таблицы products из примера:

ALTER TABLE products ADD CONSTRAINT fk\_catalog

FOREIGN KEY (id\_catalog) REFERENCES catalogs (id\_catalog)

ON DELETE CASCADE

ON UPDATE CASCADE

Это приведет к тому что изменения таблицы catalogs приведет к автоматическому изменению таблицы products.

Задания на лабораторную работу.

1. Создать 4 отношения и заполнить, любым удобным способом. У Вас должны получиться следующие отношения:

Таблица поставщиков (shippers)

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поставщика | Фамилия | Рейтинг | Город |
| 1 | Смит | 20 | Лондон |
| 2 | Джонс | 10 | Париж |
| 3 | Блейк | 30 | Париж |
| 4 | Кларк | 20 | Лондон |
| 5 | Адамс | 30 | Афины |

Таблица деталей (details)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер детали | Название | Цвет | Вес | Город |
| 1 | Гайка | Красный | 12 | Лондон |
| 2 | Болт | Зеленый | 17 | Париж |
| 3 | Винт | Голубой | 17 | Рим |
| 4 | Винт | Красный | 14 | Лондон |
| 5 | Кулачок | Голубой | 12 | Париж |
| 6 | Блюм | Красный | 19 | Лондон |

Таблица изделий (products)

|  |  |  |
| --- | --- | --- |
| Номер изделия | Название | Город |
| 1 | Жесткий диск | Париж |
| 2 | Перфоратор | Рим |
| 3 | Считыватель | Афины |
| 4 | Принтер | Афины |
| 5 | Флоппи-диск | Лондон |
| 6 | Терминал | Осло |
| 7 | Лента | Лондон |

Таблица поставок (supplies)

|  |  |  |  |
| --- | --- | --- | --- |
| Номер поставщика | Номер детали | Номер изделия | Количество |
| 1 | 1 | 1 | 200 |
| 1 | 1 | 4 | 700 |
| 2 | 3 | 1 | 400 |
| 2 | 3 | 2 | 200 |
| 2 | 3 | 3 | 200 |
| 2 | 3 | 4 | 500 |
| 2 | 3 | 5 | 600 |
| 2 | 3 | 6 | 400 |
| 2 | 3 | 7 | 800 |
| 2 | 5 | 2 | 100 |
| 3 | 3 | 1 | 200 |
| 3 | 4 | 2 | 500 |
| 4 | 6 | 3 | 300 |
| 4 | 6 | 7 | 300 |
| 5 | 2 | 2 | 200 |
| 5 | 2 | 4 | 100 |
| 5 | 5 | 5 | 500 |
| 5 | 5 | 7 | 100 |
| 5 | 6 | 2 | 200 |
| 5 | 1 | 4 | 100 |
| 5 | 3 | 4 | 200 |
| 5 | 4 | 4 | 800 |
| 5 | 5 | 4 | 400 |
| 5 | 6 | 4 | 500 |

2. Связать таблицы между собой.

3. Установить внешний ключ в последней таблице на соответствующие атрибуты. Но для первой таблицы создать каскадное удаление и обновление, для второй – установку значений null в удаленных строках, а для третьей установить отсутствие действия.

4. Удалить из таблиц shippers, details, products строки с 1 номером, вывести таблицу supplies на экран.

5. Проанализируйте изменения.

6. Сделайте вывод, который будет содержать ответ на вопрос: «Как вам кажется, какой вариант параметров on delete on update целесообразнее было бы использовать для всех таблиц в этой работе?»

Контрольные вопросы

1. Какие типы ограничений вы знаете? Какие у них назначения?
2. Зачем нужны ограничения?
3. Можно ли обойтись без ограничений в БД?
4. Существуют ли какие-либо ограничения в таблице по умолчанию?
5. Какие условия необходимо соблюсти, чтобы связать две таблицы связью?
6. Какие типы ключей вы знаете?
7. При связи двух таблиц, накладываются ли какие-либо ограничения на дочернюю или родительскую таблицу?
8. Зачем нужны ключевые слова on update on delete?
9. Какие параметры on update on delete существуют?
10. Какой синтаксис у ограничения типа CHECK?

**Лабораторная работа № 6**

**«Выборка данных. Выбор строк по условию»**

Время выполнения: 80 минут.

Цель: Изучить запросы на выборку к одной таблице, научиться использовать запросы, сочетание нескольких функций, операторов, кодовых слов.

Методические указания.

В SQL имеется единственный оператор, который предназначен для выборки данных из базы данных – SELECT. Он относится к подмножеству DML.

Ниже приведен почти полный синтаксис оператора SELECT.

SELECT [DISTINCT | ALL]

FROM Имя\_таблицы

[WHERE <условие поиска>]

[GROUP BY Колонка [, Колонка ...]]

[HAVING <условие поиска>]

[ORDER BY <список сортировки>];

После слова SELECT следует указать атрибут или список атрибутов, которые необходимо вывести на экран, а далее после слова FROM название таблицы или нескольких таблиц из которой мы выбираем атрибуты.

Пример. Пусть у нас есть таблица Students с атрибутами id, Name, Surname, Age. Чтобы вывести на экран только имена и фамилии студентов, следует использовать следующий код.

SELECT Name, Surname FROM Students;

Чтобы вывести все столбцы вашего отношения вместо перечисления всех атрибутов можно использовать знак \*.

Запрос SELECT \* FROM Students; выведет все строки и все столбцы отношения Students на экран.

В этой работе мы будем использовать только условия. За условие отбора строк отвечает ключевое слово WHERE. Ниже приведем основные группы операторов, ключевых слов и функций, с помощью которых мы можем фильтровать информацию.

**Операторы сравнения**

Особенностью операторов сравнения является то, что независимо от типов операндов их результатом являются логические значения. К ним можно отнести «=», «>», «<». Предположим у нас есть отношение Сотрудники с атрибутами код, ФИО, должность, стаж. Если мы хотим вывести только строки с сотрудниками, которые занимают должность продавца, то следует использовать следующий код:

SELECT \* FROM Сотрудники

WHERE должность= ‘Продавец’;

А если бы мы хотели вывести только строки с сотрудниками, чей стаж больше 7 лет, то можно использовать следующий код:

SELECT \* FROM Сотрудники

WHERE стаж>7;

**Логические операторы**

Операндами и результатом логических операторов являются логические значения. Стандартными логическими операторами являются AND, OR и NOT. 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.

Чтобы вывести на экран всех продавцов со стажем больше 2 лет из предыдущего примера, следует использовать следующий код.

SELECT \* FROM Сотрудники

WHERE стаж>2 AND должность= ‘Продавец’;

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

Например, если мы хотим вывести на экран всех сотрудников, кроме продавцов необходимо написать следующий код:

SELECT \* FROM Сотрудники

WHERE NOT должность= ‘Продавец’;

**Выражения над столбцами**

В выражениях фразы WHERE в качестве значения аргументов имена столбцов могут быть не только имена столбцов, а также выражения над столбцами.

Например, пусть у нас есть таблица ЗП, в которой есть следующие атрибуты ID\_сотрудника, зп, премия.

Чтобы вывести все записи с сотрудниками, у которых ЗП и премия больше 40000, используем следующий запрос:

SELECT \* FROM ЗП

WHERE ЗП+премия>40000;

**Проверка на принадлежность множеству**

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

Имя\_Атрибута IN (значение1, …, значениеN)

Например, выведем все строки из таблицы Сотрудники, со стажем 3,4 или 5 лет.

SELECT \* FROM Сотрудники

WHERE стаж IN (3,4,5);

Также оператор IN можно сочетать с отрицанием NOT (ставится перед IN) или с выражениями над множествами.

**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 следующий:

Имя\_Атрибута BETWEEN минимум AND максимум.

Например, выведем всех сотрудников, со стажем от 2 до 10 лет.

SELECT \* FROM Сотрудники

WHERE стаж BETWEEN 2 AND 10;

**Проверка на соответствие шаблону**

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'). Общий синтаксис для использования шаблона выглядит так:

Имя\_Атрибута [NOT] LIKE ‘шаблон’.

**Проверка на неопределенное значение**

Наличие значения NULL во фразе WHERE приводит к тому, что условие принимает истинностное значение UNKNOWN и соответствующая строка не включается в результат. Чтобы проверить столбец на неопределенное значение, следует применить унарный оператор IS NULL, имеющий такой синтаксис:

Имя\_Атрибута IS [NOT] NULL

**Агрегатные функции** – встроенные функции T-SQL, обрабатывающие коллекцию значений и возвращающее одно значение. К таким функциям относятся, например, нахождение минимального и максимального числа. Обрабатывая некоторый объем данных, мы получаем лишь одно значение на выходе.

Рассмотрим основные агрегатные функции и их синтаксис.

* COUNT (выражение) – возвращает количество значений в группе.

Имеет место следующая запись COUNT (\*), которая позволяет подсчитать количество записей в отношении. Также может использоваться с ключевым словом DISTINCT, которое позволяет посчитать только уникальные значения.

Пример. Пусть у нас есть таблица Guests в базе данных отеля с данными о постояльцах, когда-либо останавливающихся в отеле. Отношение содержит следующие атрибуты: ФИО гостя, даты пребывания, страну и электронную почту.

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Дата пребывания | Страна | E-mail |
| Арман Руссо | 17/11/2019-21/11/2019 | Франция | Russeau1986@gmail.com |
| Юджин Земски | 26/08/2020-3/09/2020 | США | Zemskijjj@yahoo.com |
| … | … | … | … |

Рис. 17 Таблица Guests

Для того, чтобы узнать сколько гостей посетило наш отель, мы можем использовать данный код:

SELECT COUNT(\*) FROM Guests;

Но, если мы хотим узнать из скольких разных стран гости приезжали в наш отель, то можно использовать следующий код:

SELECT COUNT DISTINCT (Страна) FROM Guests;

* MAX, MIN (выражение) – возвращает максимальное и минимальное значение соответственно.

Синтаксис:

Select Max(имя\_атрибута) from Имя\_таблицы

* AVG (выражение) – возвращает среднее арифметическое значение в группе.

Синтаксис:

Select Avg(имя\_атрибута) from Имя\_таблицы

* SUM (выражение) – возвращает сумму всех значений в группе.

Синтаксис:

Select Sum(имя\_атрибута) from Имя\_таблицы

Все агрегатные функции могут использоваться в сочетании с оператором where, если вам нужна фильтрация.

Задания на лабораторную работу

1. Импортируйте данные из файла, в соответствии с вариантом.

2. Для своего отношения создайте следующие запросы:

* 3 запроса с использованием оператора сравнения;
* 3 запроса с использованием логических операторов and, or, not;
* 1 запрос с комбинацией логических операторов;
* 2 запроса на использование выражений над столбцами;
* 3 запроса с использованием агрегатных функций;
* 1 запрос с проверкой на принадлежность множеству;
* 2 запроса с проверкой на принадлежность диапазону значений;
* 2 запроса с проверкой на соответствие шаблону (разные);
* 1 запрос с проверкой на неопределенное значение.

Отчет должен содержать код запроса и результат выборки.

**Лабораторная работа № 7**

**«Выборка данных.**

**Многотабличные запросы, запросы на соединения»**

Время выполнения: 70 минут.

Цель: Изучить многотабличные запросы на выборку, а также получить навыки работы с оператором SELECT.

Методические указания.

В информационной системе сущности могут быть взаимосвязанными. Связь между таблицами устанавливается за счет размещения специального столбца первичного ключа одной таблицы, которая называется родительской, в другой таблице, которая называется дочерней. 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.

**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 с помощью следующего синтаксиса:

SELECT \* FROM имя\_таблицы1, …, имя\_таблицыN

Вместо знака \* мы можем использовать название интересующих нас атрибутов.

Пример. Пусть у нас есть две таблицы: Студенты и Курсы. В таблице Студенты хранится информация о потенциальных посетителях курсов, у нее следующие атрибуты: код, ФИО, телефон. В таблице Курсы список возможных для посещения курсов, у нее следующие атрибуты: Код, Название, количество часов.

Первые несколько строк каждой из таблиц см на рис. 18 и рис. 19.

|  |  |  |
| --- | --- | --- |
| Код | ФИО | Телефон |
| 1 | Иванов Иван Иванович | +79001112233 |
| 2 | Федоров Алексей Иванович | +79013334235 |
| … | … | … |

Рис. 18 Таб. Студенты

|  |  |  |
| --- | --- | --- |
| Код | Название | Количество часов |
| 1 | Программирование для начинающих | 70 |
| 2 | Английский язык | 115 |
| … | … | … |

Рис. 19 Таб. Курсы

Тогда запрос SELECT \* FROM Студенты, Курсы;

даст нам следующее отношение:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Код | ФИО | Телефон | Код | Название | Количество часов |
| 1 | Иванов Иван Иванович | +79001112233 | 1 | Программирование для начинающих | 70 |
| 1 | Иванов Иван Иванович | +79001112233 | 2 | Английский язык | 115 |
| 2 | Федоров Алексей Иванович | +79013334235 | 1 | Программирование для начинающих | 70 |
| 2 | Федоров Алексей Иванович | +79013334235 | 2 | Английский язык | 115 |
| … | … | … | … | … | … |

Как мы видим, такие атрибуты как Код теряют свой смысл, к тому же имена данных атрибутов дублируются, что усложняет работу с отношением. Вместо \* мы также можем указать конкретные столбцы. В результате запроса

SELECT Студенты.ФИО, Студенты.Телефон, Курсы.Название FROM Студенты, Курсы;

Получим:

|  |  |  |
| --- | --- | --- |
| ФИО | Телефон | Название |
| Иванов Иван Иванович | +79001112233 | Программирование для начинающих |
| Иванов Иван Иванович | +79001112233 | Английский язык |
| Федоров Алексей Иванович | +79013334235 | Программирование для начинающих |
| Федоров Алексей Иванович | +79013334235 | Английский язык |
| … | … | … |

Обратите внимание, что мы указываем путь до атрибута через таблицу разделяя их имена точкой.

**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

Соединение, в котором после WHERE указаны условия соответствия строк из отношений, называется **соединение по равенству**.

Пусть у нас есть таблицы Кафедры и Факультеты. В таблице Кафедры есть атрибут код\_факультета, который является внешним ключом и определяет к какому факультету принадлежит кафедра. Синтаксис для вывода полной информации из обеих таблиц будет следующий:

SELECT \* FROM Кафедры, Факультеты

WHERE Факультеты.Код=Кафедры.код\_факультета;

Данный пример более реалистичный и осознанный. Здесь мы выводим не все возможные комбинации, а учитываем связь между кафедрами и факультетами.

Также в WHERE мы можем указывать и другие условия для фильтра. Например, вывести только те строки, где код факультета равен 3.

SELECT \* FROM Кафедры, Факультеты

WHERE Факультеты.Код=Кафедры.код\_факультета

AND Факультеты.Код=3;

Используя FROM для условия далее следует использовать один из видов соединения, левое внешнее, правое внешнее, естественное соединение или полное внешнее, о них можно прочесть дальше.

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

1. 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.

2. 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.

3. Во фразе FROM перечислить необходимые таблицы.

4. 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.

**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.

Пример. 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 = 'Сидоров А.О.;

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.

Пример. 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**

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.

**Левое внешнее соединение**

Общий синтаксис выглядит так:

SELECT имя\_столбца1, …, имя\_столбцаN FROM

Имя\_таблицы1 LEFT JOIN Имя\_таблицы2

В результате такого запроса будет выведено отношение, в котором будут строки, соответствующие обеим таблицам, а также строки из Таблицы1, для которых не нашлось соответствия в Таблице2.

**Правое внешнее соединение**

Общий синтаксис выглядит так:

SELECT имя\_столбца1, …, имя\_столбцаN FROM

Имя\_таблицы1 RIGHT JOIN Имя\_таблицы2

В результате такого запроса будет выведено отношение, в котором будут строки, соответствующие обеим таблицам, а также строки из Таблицы2, для которых не нашлось соответствия в Таблице1.

**Полное внешнее соединение**

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.

Синтаксис будет выглядеть так:

SELECT имя\_столбца1, …, имя\_столбцаN FROM

Имя\_таблицы1 FULL OUTER JOIN Имя\_таблицы2

**Перекрестное соединение**

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.

Синтаксис для создания перекрёстного соединения выглядит так:

SELECT имя\_столбца1, …, имя\_столбцаN FROM

Имя\_таблицы1 CROSS JOIN Имя\_таблицы2

**Внутреннее (естественное) соединение**

Внутреннее соединение возвращает только те строки, которые имеет соответствие во всех таблицах, участвующих в запросе.

Синтаксис выглядит так:

SELECT имя\_столбца1, …, имя\_столбцаN FROM

Имя\_таблицы1 INNER JOIN Имя\_таблицы2

*Примечание. Для всех соединений и запросов мы можем использовать группировку, сортировку, условие для фильтра, а также сочетание нескольких условий с помощью логических операторов AND, OR, NOT.*

Задания на лабораторную работу.

* 1. Используйте данные, которые были созданы в ЛР3.
  2. Самостоятельно создать следующие запросы:
* 1 запрос с использованием декартового произведения двух таблиц;
* 2 запроса с использованием соединения двух таблиц по равенству;
* 1 запрос с использованием соединения двух таблиц по равенству и условием отбора;
* 1 запрос с использованием соединения по трем таблицам;
* создать копии ранее созданных запросов на соединение по равенству на запросы с использованием внешнего полного соединения таблиц (JOIN).
* 1 запрос с использованием левого внешнего соединения;
* 1 запрос на использование правого внешнего соединения;
* 1 запрос с использованием внутреннего соединения.
* 1 запрос на проверку какого-либо условия через самосоединение.
  1. Отчет должен содержать код или скриншот запросов и результат их выполнения. На скриншоте результатов должно быть видно имя вашей БД.

**Лабораторная работа № 8**

**«Выборка данных.**

**Группировка и сортировка данных»**

Время выполнения: 70 минут.

Цель: Изучить используемый в реляционных СУБД оператор извлечения данных из таблиц SELECT и выполнение группировки и сортировки данных. Изучить синтаксис языка модификации данных. Научится использовать встроенные функции в запросах.

Методические указания.

**Сортировка данных.**

Для того, чтобы выводить данные в том или ином порядке, существует специальный оператор ORDER BY. Оператор ORDER BY позволяет отсортировать извлекаемые значения по определенному столбцу.

Синтаксис:

SELECT столбцы FROM Имя\_таблицы

ORDER BY Имя\_столбца

По умолчанию применяется сортировка по возрастанию. С помощью дополнительного оператора DESC можно задать сортировку по убыванию, оператор ASC задает сортировку по возрастанию. если это необходимо. Также можно отсортировать сразу по нескольким столбцам, при этом необязательно сортировать все эти столбцы только по убыванию или возрастанию.

Пример. Рассмотрим нашу таблицу Guests из предыдущего примера. Нам необходимо отсортировать данные в ней по странам, по возрастанию (от А до Я), и именам, но ради примера, отсортируем их по убыванию (от Я до А). Для этого применим следующий код:

SELECT \* FROM Guests

ORDER BY Страна ASC, ФИО DESC

**Группировка.**

Для того, чтобы разделить данные в таблицы на определенные группы, используется оператор GROUP BY.

Синтаксис:

SELECT столбцы

FROM таблица

[WHERE условие\_фильтрации\_строк]

[GROUP BY столбцы\_для\_группировки]

[HAVING условие\_фильтрации\_групп]

Рассмотрим оператор HAVING. Он используется, если вам нужна фильтрация уже сгруппированных данных. Оператор WHERE с сгруппированными данными не используется.

*Примечание*. Группировка данных неразрывно связана с агрегатными функциями. Повторите их перед тем как приступать к работе.

Задания на лабораторную работу.

* 1. Импортировать данные из ресурса, указанного преподавателем, в соответствии с вариантом.
  2. Самостоятельно создать следующие запросы:

**Для 1 варианта**:

-Запрос, который покажет, сколько человек выбрало тот или иной предмет для первого экзамена, сделайте сортировку по убыванию.

-Запрос, который покажет, какие предметы в качестве первого экзамена выбрало более 50 человек.

-Запрос, который покажет сколько студентов выбрало в качестве второго экзамена физику, химию или биологию. Создайте два запроса один с помощью WHERE, другой используя HAVING.

-Запрос, который покажет, сколько студентов выбрало тот или иной предмет в качестве второго экзамена и получивших более 100 баллов за оба экзамена

-Запрос, который покажет сколько студентов выбрали Биологию в качестве первого или второго предмета.

**Для 2 варианта:**

-Запрос, который покажет, сколько человек выбрало тот или иной предмет в качестве любимого, сделайте сортировку по убыванию.

-Запрос, который покажет, какие предметы в качестве любимого выбрало более 70 человек.

-Запрос, который покажет сколько студентов выбрало в качестве любимого физику, русский или английский. Создайте два запроса один с помощью WHERE, другой используя HAVING.

-Запрос, который покажет, сколько студентов выбрало тот или иной предмет в качестве любимого и имеющих средний балл хотя бы за один год более 4.

-Запрос, который покажет сколько студентов имею средний бал за 2015 или 2016 год менее 3.

**Для 3 варианта:**

-Запрос, который покажет, сколько книг продается у того или иного автора, сделайте сортировку по убыванию.

-Запрос, который покажет, какие авторы продают более пяти книг.

-Запрос, который покажет сколько книг продает Автор1, Автор2, Автор3. Создайте два запроса один с помощью WHERE, другой используя HAVING.

-Запрос, который покажет, сколько книг продает тот или иной автор, считая книги с рейтингом более 3.

-Запрос, который покажет сколько книг с рейтингом менее 3 и ценой более 500 продается в магазине.

**Для 4 варианта:**

-Запрос, который покажет, сколько различных товаров предоставляет тот или иной поставщик, сделайте сортировку по убыванию.

-Запрос, который покажет, какие поставщики предоставляют более десяти товаров.

-Запрос, который покажет сколько товаров предоставляет Поставщик1, Поставщик2, Поставщик3. Создайте два запроса один с помощью WHERE, другой используя HAVING.

-Запрос, который покажет, сколько товаров предоставляет тот или иной поставщик, считая товары дороже 1500 рублей.

-Запрос, который покажется сколько товаров с ценой более 800 рублей и количеством менее 5000 есть на складе.

* 1. Отчет должен содержать код или скриншот запросов и результат их выполнения.

Контрольные вопросы.

1. Что такое агрегатная функция?
2. Зачем нужно самосоединение?
3. Какие типы соединений вы знаете?
4. Что такое группировка? Приведите пример использования.
5. Как вывести всю строку, содержащую максимальное значение по тому или иному атрибуту?
6. В чем разница между операторами WHERE и HAVING?
7. В чем разница между функциями COUNT и SUM?
8. Можно ли отсортировать данные таблицы по нескольким атрибутам? Если нет, то почему, если да, какой будет результат?
9. Можно ли одновременно использовать в одном запросе операторы WHERE и HAVING?
10. В какой последовательности происходит сортировка по умолчанию?
11. Как можно отсортировать уже сгруппированные записи?

**Лабораторная работа № 9**

**«Инструкция MERGE»**

Время выполнения: 80 минут.

Цель: Изучить принципы слияния таблиц, настройки и параметры инструкции MERGE.

Методические указания.

MERGE – операция в языке T-SQL, при которой происходит обновление, вставка или удаление данных в таблице на основе результатов соединения с данными другой таблицы или SQL запроса. Другими словами, с помощью MERGE можно осуществить слияние двух таблиц, т.е. синхронизировать их.

В операции MERGE происходит объединение по ключевому полю или полям основной таблицы (*в которой и будут происходить все изменения*) с соответствующими полями другой таблицы или результата запроса. В итоге если условие, по которому происходит объединение, истина (WHEN MATCHED), то мы можем выполнить операции обновления или удаления, если условие не истина, т.е. отсутствуют данные (WHEN NOT MATCHED), то мы можем выполнить операцию вставки (*INSERT добавление данных*), также если в основной таблице присутствуют данные, которые отсутствуют в таблице (*или результате запрос*а) источника (WHEN NOT MATCHED BY SOURCE), то мы можем выполнить обновление или удаление таких данных.

В дополнение к основным перечисленным выше условиям можно указывать «*Дополнительные условия поиска*», они указываются через ключевое слово **AND**.

Синтаксис выглядит так:

MERGE <Основная таблица>

USING <Таблица или запрос источника>

ON <Условия объединения>

[ WHEN MATCHED [ AND <Доп. условие> ]

THEN <UPDATE или DELETE>

[ WHEN NOT MATCHED [ AND Доп. условие> ]

THEN <INSERT> ]

[ WHEN NOT MATCHED BY SOURCE [ AND <Доп. условие> ]

THEN <UPDATE или DELETE> ] [ ...n ] ;

Обратите внимание, что в конце инструкции MERGE обязательно ставится точка с запятой.

Например,

|  |
| --- |
| --Заполнение сводной таблицы при помощи инструкций MERGE |
|  |
| --Вставка данных из Группа1 |
| --RunnersData - целевая сводная таблица |
| **MERGE** RunnersData **AS** TARGET |
| --RunnerGroup1 - источник |
| **USING** RunnerGroup1 |
| **AS** SOURCE |
| --Условие соединения |
| **ON** **(**TARGET**.**RunnerID **=** SOURCE**.**RunnerID**)** |
| --В источнике и целевой таблице есть совпадение по условию, |
| --если есть несовпадение по времени забега (NULL или не равно источнику), то обновить его в целевой таблице |
| **WHEN** MATCHED **AND** TARGET**.**RunnerTimeSeconds1 **<>** SOURCE**.**RunnerTimeSeconds **OR** TARGET**.**RunnerTimeSeconds1 **IS** **NULL** |
| **THEN** **UPDATE** **SET** TARGET**.**RunnerTimeSeconds1 **=** SOURCE**.**RunnerTimeSeconds |
| --В целевой таблице нет записи с нужным RunnerID, добавить новую запись |
| **WHEN** **NOT** MATCHED **BY** TARGET |
| **THEN** **INSERT** **(**RunnerID**,** RunnerTimeSeconds1**)** **VALUES** **(**SOURCE**.**RunnerID**,** SOURCE**.**RunnerTimeSeconds**)**  **И т.д. для других таблиц;** |

Задания на лабораторную работу.

В соревнованиях по бегу участвовало несколько спортсменов. Для соревнования выделили 3 бегущих группы. У спортсмена была возможность бежать в любой из этих групп, при этом можно было бежать 2 или 3 раза, в этом случае учитывается лучший результат.

Каждому участвующему спортсмену дается свой номер. Ниже приведены таблицы с результатами по каждой группе с номером атлета и временем, за которое он пробежал дистанцию.

|  |  |
| --- | --- |
| Номер спортсмена | Время, с |
| 371 | 56 |
| 372 | 57 |
| 373 | 58 |
| 374 | 56 |
| 375 | 51 |
| 376 | 62 |
| 377 | 61 |
| 378 | 67 |
| 379 | 48 |
| 380 | 50 |

Таб. 1. Группа 1.

|  |  |
| --- | --- |
| Номер спортсмена | Время, с |
| 372 | 61 |
| 374 | 58 |
| 376 | 57 |
| 378 | 54 |
| 380 | 53 |

Таб. 2. Группа 2.

|  |  |
| --- | --- |
| Номер спортсмена | Время, с |
| 370 | 58 |
| 371 | 57 |
| 372 | 67 |
| 373 | 62 |
| 374 | 55 |
| 375 | 52 |
| 376 | 60 |
| 377 | 58 |
| 378 | 59 |
| 379 | 53 |
| 380 | 54 |

Таб. 3. Группа 3.

1. Создайте в своей базе данных три таблицы и заполните их, аналогично таблицам 1, 2 и 3. Номер спортсмена должен заполнятся не с клавиатуры, а с помощью свойства identity(x,y).
2. Создайте сводную таблицу, в которой будут храниться все данные из первых трех. Добавьте в нее данные с помощью операции MERGE. Незаполненные ячейки должны быть пустые (NULL).

|  |  |  |  |
| --- | --- | --- | --- |
| Номер спортсмена | Время 1, с | Время 2, с | Время 3, с |
| 370 | NULL | NULL | 58 |
| 371 | 56 | NULL | 57 |
| 372 | 57 | 61 | 67 |
| 373 | 58 | NULL | 62 |
| 374 | 56 | 58 | 55 |
| 375 | 51 | NULL | 52 |
| 376 | 62 | 57 | 60 |
| 377 | 61 | NULL | 58 |
| 378 | 67 | 54 | 59 |
| 379 | 48 | NULL | 53 |
| 380 | 50 | 53 | 54 |

Таб. 4. Возможный вид сводной таблицы для п.2.

1. Создайте новую таблицу, в которой будет лучший результат каждого спортсмена. Сортировку выполните с помощью операции MERGE. Обратите внимание, что лучшее время – это минимальное время.

|  |  |
| --- | --- |
| Номер спортсмена | Лучшее время |
| 370 | 58 |
| 371 | 56 |
| 372 | 57 |
| 373 | 58 |
| 374 | 55 |
| 375 | 51 |
| 376 | 57 |
| 377 | 58 |
| 378 | 54 |
| 379 | 48 |
| 380 | 50 |

Таб. 5. Возможный вид таблицы лучших результатов для п.3.

1. Выведите на экран номер победителя и его результат любым удобным способом.

|  |  |
| --- | --- |
| Победитель | Результат |
| 379 | 48 |

Таб. 6 Возможный вид таблицы победителя и его результат для п.4.

1. Аналогично выведите на экран худший результат за все соревнование.
2. Отчет должен содержать все коды и результаты ваших действий.

**Лабораторная работа № 10**

**«Функции»**

Время выполнения: 80 минут.

Цель: Изучить разные типы функций, научиться создавать и вызывать функции разных типов.

Методические указания.

Задания на лабораторную работу.

1. Создайте функцию, выполняющую роль калькулятора, со следующими возможностями: сложение, вычитание, умножение, деление, sin, cos, tg, ctg, возведение в заданную степень.
2. Протестируйте функцию.
3. Создайте функцию, которая при вводе даты, будет возвращать название сезона (Зима, Весна, Лето, Осень).
4. Протестируйте функцию.
5. Создайте функцию, которая будет разбивать входную строку на отдельные строки. Символ пробела является разделителем строк.
6. Протестируйте функцию.
7. Импортируйте документ Doc.xsl.
8. Работа с таблицей StudList. Создайте функцию, возвращающую список студентов, обучающихся или отчисленных, в зависимости от входного параметра. Предполагается, что, если в столбце Studying стоит No, студент отчислен, в противном случае – обучается.
9. Протестируйте функцию.

Отчет должен содержать код или скриншот запросов для создания функций, результат ваших действий.

**Лабораторная работа № 11**

**«Процедуры»**

Время выполнения: 80 минут.

Цель: Изучить разные типы процедур, научиться создавать и вызывать процедуры без входных и выходных параметров, с входными параметрами.

Методические указания.

Часто операция с данными представляет набор инструкций, которые необходимо выполнить в определенной последовательности или набор действий, которые нужно выполнять с некоторой периодичностью. В этом случае более оптимально будет инкапсулировать все эти действия в один объект - **хранимую процедуру**.

То есть по сути хранимые процедуры представляют набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект. Также хранимые процедуры позволяют ограничить доступ к данным в таблицах и тем самым уменьшить вероятность преднамеренных или неосознанных нежелательных действий в отношении этих данных.

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

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

Общий синтаксис выглядит так:

CREATE PROCEDURE Имя\_процедуры AS

Тело процедуры

Чтобы вызвать процедуры используется команда **EXEC** или **EXECUTE**.

EXEC Имя\_процедуры

Процедуры могут принимать параметры. Параметры бывают входными - с их помощью в процедуру можно передать некоторые значения. И также параметры бывают выходными - они позволяют возвратить из процедуры некоторое значение. После названия процедуры идет список входных параметров, которые определяются также, как и переменные - название начинается с символа **@**, а после названия идет тип переменной. При вызове такой процедуры параметры указываются после имени процедуры.

Пример.

Пусть у нас есть таблица Students с параметрами имя, фамилия, отчество, дата рождения, номер паспорта. Нам необходимо создать процедуру, которая бы вносила данные о новом студенте. Таким образом, входными параметрами будут значения атрибутов Students. Синтаксис создания такой процедуры будет выглядеть следующим образом:

CREATE PROCEDURE New\_student

@name NVARCHAR(20),

@surname NVARCHAR(20),

@patronymic NVARCHAR(20),

@dob DATE,

@docnum CHAR(10)

AS

INSERT INTO Students

VALUES(@name, @surname, @patronymic, @dob, @docnum)

Вызов такой функции может выглядеть так:

EXEC New\_student ‘Александр’, ‘Белов’, ‘Алексеевич’, ‘2003-05-12’,’4017111222’

Изучите правила использования выходных параметров и возвращения результата здесь: <https://metanit.com/sql/sqlserver/11.3.php> и приступайте к выполнению работы.

Задания на лабораторную работу.

1. Работа с таблицей StudList из предыдущей ЛР. Создать процедуру с входным параметром и return. Процедура должна получать входной параметр – Имя студента, а на выходе выдать 1 или 0, в зависимости от того обучается он или нет.

2. Проверить процедуру.

3. Работа с таблицей Shop. Импортируйте таблицу Shop. В магазине проходит акция. Для всех, кто оставил комментарий на сайте, скидка 5%. Создайте процедуру, которая уменьшит цены в таблице Shop на 5% (атрибут Price), в тех строках, где значение столбца Comment yes.

4. Проверить процедуру.

5. Работа с таблицей Shop. Создать процедуру с входным параметром. Входной параметр – номер клиента. Процедура удаляет все строки с заказами этого клиента.

6. Работа с таблицей Shop. Создать процедуру с входным и выходным параметрами, где входной параметр – номер клиента, а выходной – самый дорогой заказ выбранного клиента.

7. Проверить процедуру.

Контрольные вопросы:

1. Какие типы процедур вы знаете?
2. Где хранятся процедуры?
3. Какой синтаксис у создания процедур?
4. Какой синтаксис у выполнения процедур?
5. В чем разница между функцией и процедурой?
6. Зачем нужны процедуры?

**Лабораторная работа № 12**

**«Создание, программирование и управление триггерами»**

Время выполнения: 80 минут.

Цель: Изучить назначения и типы триггеров, условия их активации, синтаксис команд языка T – SQL для их создания, модификации, переименования, программирования и удаления.

Методические указания.

Триггер – это специальный тип хранимых процедур, которые запускаются сервером автоматически при выполнении тех или иных действий с данными таблицы. Триггеры реагируют на попытку вставки ( insert), изменения( update) или удаления( delete) данных.

У триггеров есть 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, однако можно создавать для представлений.

Синтаксис команды создания триггера следующий:

CREATE TRIGGER Имя\_триггера

ON Имя\_таблицы | Имя\_представления

FOR | AFTER | INSTEAD OF

INSERT | UPDATE | DELETE

AS команда\_SQL

Синтаксис команды удаления триггера:

DROP TRIGGER Имя\_триггера

Синтаксис команды редактирования триггера:

ALTER TRIGGER Имя\_триггера

ON Имя\_таблицы | Имя\_представления

FOR | AFTER | INSTEAD OF

INSERT | UPDATE | DELETE

AS команда\_SQL

Также допускается вызов хранимых процедур, включая системные, но такие задания мы будем выполнять в следующей работе.

**Пример триггера, который будет выводить удаленные значения, после удаления строк**

Пусть у нас существует следующее отношение BigTab.

|  |  |  |  |
| --- | --- | --- | --- |
| Серийный N | Имя владельца | Дата покупки | Гарантийное обслуживание |
| 111111111 | Перевалов Константин | 10.08.2018 | есть |
| 111111112 | Первакова Марина | 23.11.2013 | нет |
| … | … | … | … |

Чтобы создать триггер, который бы выводил удаленные значения нам потребуется следующий код:

CREATE TRIGGER FirstTrigger

ON BigTab

AFTER DELETE

AS Select \* from deleted

Задания на лабораторную работу.

* 1. Импортируйте данные из файла AUTH.xls
  2. Создайте триггер Trigger\_1, который после изменения данных будет выводить строки, со старыми и новыми значениями.
  3. Протестируйте ваш триггер.
  4. Создайте триггер Trigger\_2, который вместо удаления выведет следующую информацию: «попытка удаления информации об au\_id»
  5. Протестируйте ваш триггер.
  6. Создайте триггер Trigger\_3, который будет считать и выводить количество удаленных строк
  7. Протестируйте ваш триггер.

Контрольные вопросы:

1. Какие типы триггеров вы знаете?

2. Можем ли мы обращаться к журналу транзакций при создании триггеров?

3. Как можно протестировать ваш триггер?

4. Сколько триггеров может иметь одно отношение?

5. Можно ли создать один триггер на несколько операций?

6. Как можно увидеть созданные триггеры?

7. Как можно изменить тело триггера?

**Лабораторная работа № 13**

**«Создание и управление транзакциями»**

Время выполнения: 80 минут.

Цель: Изучение способов обеспечения надежной работы SQL Server с помощью механизма транзакций и контрольных точек, приобретение навыков управления транзакциями, а также ознакомление с физической и логической архитектурой журнала транзакций.

Методические указания.

Одним из способов повышения надежности работы системы MS SQL Server является применение встроенного в систему механизма транзакций и контрольных точек и умелое его управление. Транзакция – это одна или несколько последовательных команд языка T – SQL, образующих логически завершенный пакет и выполняемых как единое целое. Если по какой-либо причине хотя бы одна из команд пакета не выполняется, то происходит откат системы к состоянию, в котором она была до начала транзакции, и транзакция считается не выполненной.

По умолчанию каждая команда выполняется как самостоятельная транзакция. При необходимости в пакете можно явно указать начало и конец транзакции. Обработка транзакций в любой системе управления базами данных должна производиться с соблюдением следующих правил – правил ASID (Atomicity, Consistency, Isolation и Durability): Atomicity – атомарность: выполняемые в транзакции изменения либо выполняются все, либо не выполняются вовсе; Consistency – согласованность: все данные после выполнения транзакции должны находиться в согласованном состоянии с соблюдением всех правил и ограничений целостности; Isolation – изолированность: изменения данных, выполняемых Durability – долговечность: после завершения транзакции ничто не может вернуть систему в состояние, в котором она была до начала транзакции (происходит фиксация транзакции).

Для управления явными транзакциями применяют команды:

BEGIN TRANSACTION Имя\_транзакции – начало транзакции;

COMMIT TRANSACTION Имя\_транзакции – фиксирование изменений и конец транзакции;

ROLLBACK TRANSACTION Имя\_транзакции – откат транзакции;

Если в команде ROLLBACK не задано имя транзакции, то откатываются все вложенные транзакции и транзакция самого высокого уровня. Если же имеется необходимость откатить лишь часть транзакций, то предварительно надо создать точку сохранения с помощью команды SAVE TRANSACTION, которую следует указывать при откате.

Функция @@TRANCOUNT предназначена для определения количества активных транзакций, начатых в активном соединении. Во всех транзакциях нельзя использовать следующие команды: ALTER DATABASE, BACKUP LOG, CREATE DATABASE, DISK INIT, DROP DATABASE, DUMP TRANSACTION, LOAD DATABASE, LOAD TRANSACTION, RECONFIGURE, RESTORE DATABASE, RESTORE LOG, UPDATE STATISTICS, а также системной хранимой процедуры sp\_droption и любой другой хранимой процедуры, изменяющей значения в системной базе master. Контрольные точки минимизируют данные в журнале транзакций.

Задания на лабораторную работу.

1. Проверьте работу транзакции следующим образом. Выберите любое отношение в вашей базе данных. Создайте транзакцию, в которой будет содержаться один запрос вставки. Запустите ее дважды. В первом случае завершите транзакцию командой COMMIT, второй раз ROLLBACK. Проанализируйте результаты. Вставьте скриншот результата в отчет.
2. Работа с вложенными транзакциями.

Создать вложенные транзакции, выполнив следующие команды:

CREATE TABLE table\_name (a1 int) -- 0-й уровень

BEGIN TRANSACTION tr1-- 1-й уровень

INSERT INTO table\_name VALUES (11)

BEGIN TRANSACTION tr2-- 2-й уровень

INSERT INTO table\_name VALUES (22)

BEGIN TRANSACTION tr3 -- 3-й уровень

INSERT INTO table\_name VALUES (33)

SELECT \* FROM table\_name

SELECT 'Вложенность транзкций', @@TRANCOUNT

ROLLBACK TRANSACTION

SELECT \* FROM table\_name -- откат на 0-й уровень

SELECT 'Вложенность транзакций', @@TRANCOUNT

Проанализируйте результаты.

1. Работа с данными из файла groups. Импортируйте все листы. В первом листе находятся данные студентов, поступивших в 2018 году. В других листах содержаться списки групп. Задача импортировать данные из списков групп в сводную таблицу всех поступивших. Создайте явную транзакцию, в которой данные о студентах будут вноситься, а, если эти данные уже были внесены раннее, то транзакция будет откатываться. Результаты продемонстрируйте в отчете.
2. Изучите самостоятельно команду SET IMPLICIT\_TRANSACTIONS ON/OFF.

Продемонстрируйте разницу между этими режимами на примере любой таблицы в вашей базе данных.

Контрольные вопросы:

1. Зачем нужны транзакции?
2. С какими режимами транзакции вы работали?
3. Какие команды являются признаком начала неявной транзакции?
4. Зачем нужны контрольные точки транзакций? Продемонстрируйте их работу.
5. Можно ли узнать количество активных транзакций?

**Лабораторная работа № 14**

**«Создание и управление курсорами»**

Время выполнения: 80 минут.

Цель: Изучение назначения и типов курсоров, синтаксиса языка T – SQL для создания и открытия курсоров, выборки данных из курсора и изменения строк таблиц с помощью курсоров, удаления данных, закрытия и освобождения курсоров, а также приобретения навыков их применения и управления с помощью команд и системных хранимых процедур SQL Server.

Методические указания

Операции в реляционной базе данных выполняются над множеством строк. Например, набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE инструкции. Такой полный набор строк, возвращаемых инструкцией, называется результирующим набором. Можно в результирующий набор не включать те или иные столбцы, применяя вертикальную фильтрацию. Результирующие наборы могут содержать сотни тысяч строк, и клиентские приложения не всегда справляются с таким объемом данных.

Приложения, особенно интерактивные, не всегда эффективно работают с результирующим набором как с единым целым. Им нужен механизм, позволяющий обрабатывать одну строку или небольшое их число за один раз. Для решения этой проблемы используются курсоры, которые представляют собой окна, налагаемые на результирующий набор данных и выделяющие требуемую часть данных. Курсоры являются расширением результирующих наборов, которые предоставляют такой механизм. Перемещая созданный курсор, можно получить доступ ко всем результирующим данным. Таким образом, курсоры SQL Server представляют собой механизм обмена данными между сервером и клиентом, который минимизирует ресурсы клиентского приложения.

Курсоры позволяют усовершенствовать обработку результатов:

* позиционируясь на отдельные строки результирующего набора;
* получая одну или несколько строк от текущей позиции в результирующем наборе;
* поддерживая изменение данных в строках в текущей позиции результирующего набора;
* поддерживая разные уровни видимости изменений, сделанных другими пользователями для данных, представленных в результирующем наборе;
* предоставляя инструкциям Transact-SQL в скриптах, хранимых процедурах и триггерах доступ к данным результирующего набора.

Однако всегда, когда это возможно, следует избегать использования курсоров и применять команды SELECT, UPDATE, DELETE и INSERT.

Один курсор может базироваться на нескольких таблицах, расположенных в разных базах данных. Операция считывания определенных в курсоре данных называется выборкой (fetch). Если за одну операцию курсор позволяет выбрать несколько строк таблицы, то такой курсор называется блочным.

По способу просмотра данных курсоры бывают последовательные (forward only), которые обеспечивают просмотр строк только в одном направлении – от начала к концу, и прокручиваемые, которые допускают просмотр в обоих направлениях и переход к произвольной строке.

По представляемым возможностям курсоры делятся на четыре типа: статические, динамические, последовательные ключевые. Тип курсора определяется на стадии его создания и не может быть изменен.

Статический курсор (static cursor) называют также курсорами моментального снимка (snapshot cursor). При открытии такого курсора сервер выбирает все данные, соответствующие заданным критериям, и сохраняет результирующий набор строк в системной базе данных tempdb без изменения, если даже исходные строки и изменяются. Поэтому статический курсор всегда открывается в режиме “только для чтения”.

Динамические курсоры (dynamic cursor) противоположны статическим. При их использовании не создается полная копия исходных данных, а выполняется динамическая выборка данным из исходных таблиц только при обращении пользователя к тем или иным данным, при этом на время выборки соответствующие строки блокируются сервером. После выборки строк исходные строки могут изменяться пользователями, но эти изменения уже не отражаются в выбранных строках. С другой стороны, изменения в выбранных строках не будут видны другим пользователям, пока они не будут подтверждены (committed).

Последовательный курсор выбирает данные только от начала к его концу. Он не хранит результирующий набор. Строки считываются из базы данных, как только они выбираются в курсоре. Это позволяет отображать все изменения в базе данных. В курсоре видно самое последнее состояние данных.

Курсоры, зависящие от набора ключей (keyset-driven cursor), или ключевые курсоры, построены на основе уникальных идентификаторов. Множество всех уникальных идентификаторов (ключей) строк таблиц базы данных называется набором ключей. Сервер блокирует строки исходных таблиц только на время составления таблицы ключей. Ключевой курсор представляет набор ключей, идентифицирующих строки полного результирующего набора курсора. Набор ключей строится в системной базе данных tempdb.

При работе с курсорами можно выделить пять основных операций: создание курсора, открытие курсора, выборка из курсора и изменение строк данных с помощью курсора, закрытие курсора и освобождение курсора.

Синтаксис команды для создания курсора следующий:

DECLARE Имя курсора CURSOR [LOCAL/GLOBAL] [FORWARD\_ONLY\SCROLL] [STATIC\KEYSET\DINAMIC\FAST\_FORWARD] [READ\_ONLY\SCROLL\_LOCKS\OPTIMISTIC] [TYPE\_WARNING]

FOR select\_statement [FOR UPDATE [OF column\_name [,…n]]].

Для открытия курсора используется команда:

OPEN {{[GLOBAL] Имя курсора}\Имя переменной}.

Для выборки данных необходимо применять команду:

FETCH [[NEXT\PRIOR\FIRST\LAST\ABSOLUTE {n\@nvar}\ RELATIVE {n\@nvar} ] FROM ] {{[GLOBAL] Имя курсора}\Имя переменной} []INTO @ Имя переменной [,…n]]

Команды UPDATE, DELETE, CLOSE и DEALLOCATE позволят соответственно произвести изменение данных, удаление данных, закрытие и освобождение курсора.

Задания на лабораторную работу.

1. Создайте таблицу Clients со следующими атрибутами: Id, Name, Address.
2. С помощью курсора создайте запрос, который будет выводить информацию о всех клиентах c нечетным Id.
3. Создайте таблицу OlympMath с результатами Олимпиады по математике за 2019, 2020 и 2021 года. Таблица OlympMath состоит из следующих атрибутов: id, Name, res19, res20, res21.
4. С помощью курсора создайте запрос, который будет находить и выводить лучший результат для каждого участника. То есть максимальный в строке.
5. Создайте две таблицы со следующей структурой (папки и файлы):

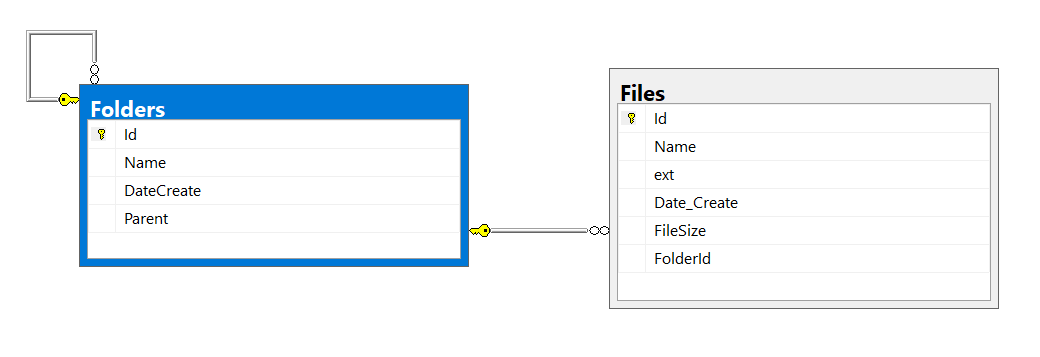


Рис.14 Таблицы Папки и файлы

Реализуйте функцию, которая будет принимать на входе идентификатор каталога и выводить все файлы и каталоги, которые находятся в заданном каталоге и во всех подкаталогах.

**Лабораторная работа № 15**

**«Подключение к БД,**

**отображение данных в форме с помощью DataGridView»**

Время выполнения: 160 минут.

Цель: Изучение синтаксиса для подключения к базе данных через приложение Windows Forms. Простейший способ отображения данных.

Методические указания

Перед тем, как приступить к синтаксису элементов библиотеки System.Data.SQLClient, давайте вспомним элементы формы, которые вам пригодятся для выполнения данной работы.

Начнем с самой формы. Рассмотрим следующие методы и свойства:

* Text – позволяет изменить текст в шапке форме. Не путать с Name. Name – это имя объекта класса.
* BackColor – позволяет изменить цвет фона.
* FormBorderStyle – не только меняет стиль формы, но и позволяет сделать ее размер фиксированным. Для этого нужно выбрать любую опцию со словом Fixed.
* Hide – скрывает форму.
* Close – закрывает форму.
* Show – показывает форму.

Также вам потребуются элемент Button. У кнопки тоже есть свойство Text, которое отвечает за текст на самой кнопке.

У элемента Panel важным для нас свойством является Visible. Оно может принимать значения true и false. Что отвечает за видимость панели с ее элементами или нет соответственно.

Наконец, самый важный элемент в этой работе DataGridView. После добавления этого элемента на форму, вам сразу предлагается добавить источник данных. Добавьте его следуя инструкции Мастера (Рис.15).

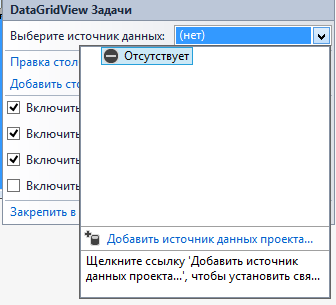


Рис.15 Добавление источника данных.

После успешного подключения в коде у вас появится строка, с помощью которой происходит подгрузка данных из БД. Вы можете скопировать и вставлять ее в те участки кода, которые отвечают за обновления данных.

Теперь давайте рассмотрим строку подключения к БД. Общий ее синтаксис выглядит так:

Data Source= Имя сервера; Initial catalog=имя БД; Integrated Security=True;

Удобнее всего занести эту строку в переменную и при необходимости обращаться к переменной. Например, так:

**string** connectionString = @" Data Source= Имя сервера; Initial catalog=имя БД; Integrated Security=True";

Теперь нам необходимо передать эту строку в специальный объект класса SqlConnection. У объектов этого класса есть методы Open и Close, которые осуществляют подключение к БД и отключение от нее соответственно. Все запросы, обращенные к БД, необходимо писать между Open и Close. Подробнее мы разберем это в других лабораторных работах.

Задания на лабораторную работу.

1. Создайте отношения в базе данных, согласно схеме ниже (Рис. 16).

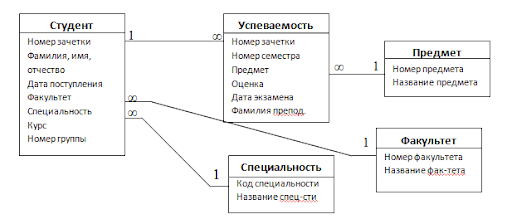


Рис.16 Схема данных.

1. Заполните каждую из таблиц данными (минимум три строки в каждой таблице).
2. Создайте интерфейс с помощью Windows Forms со следующими требованиями:

* Все таблицы должны отображаться с помощью элемента DataGridView.
* Подключения к БД производится через строку подключения.
* Таблицы Студент, Успеваемость, Специальность отображаются на отдельных формах.
* Таблицы Предмет и Факультет отображаются на одной форме, но на разных элементах panel.
* Размер формы должен быть фиксированный. Изменения размера недоступно для пользователя.
* У каждой формы должно быть имя.
* На формах должны быть кнопки для переключения между таблицами.
* На формах должны быть кнопки для обновления данных.
* Все элементы должны быть понятны пользователю, а текст должен легко читаться.
* Цвет фона формы выберите в соответствии с вариантом:

|  |  |
| --- | --- |
| Номер варианта | Цвет |
| 1 | Aqua |
| 2 | AntiqueWhite |
| 3 | Blue |
| 4 | BlueViolet |
| 5 | Brown |
| 6 | CornflowerBlue |
| 7 | Chocolate |
| 8 | Crimson |
| 9 | DarkGreen |
| 10 | DarkMagenta |
| 11 | Gray |
| 12 | Lavender |
| 13 | LightPink |
| 14 | LightSeaGreen |
| 15 | MediumOrchid |
| 16 | MistyRose |
| 17 | Olive |
| 18 | Orange |
| 19 | OrangeRed |
| 20 | PapayaWhip |
| 21 | Plum |
| 22 | Red |
| 23 | Salmon |
| 24 | SandyBrown |
| 25 | SteelBlue |
| 26 | Tan |
| 27 | Teal |
| 28 | Tomato |
| 29 | Sienna |
| 30 | Yellow |

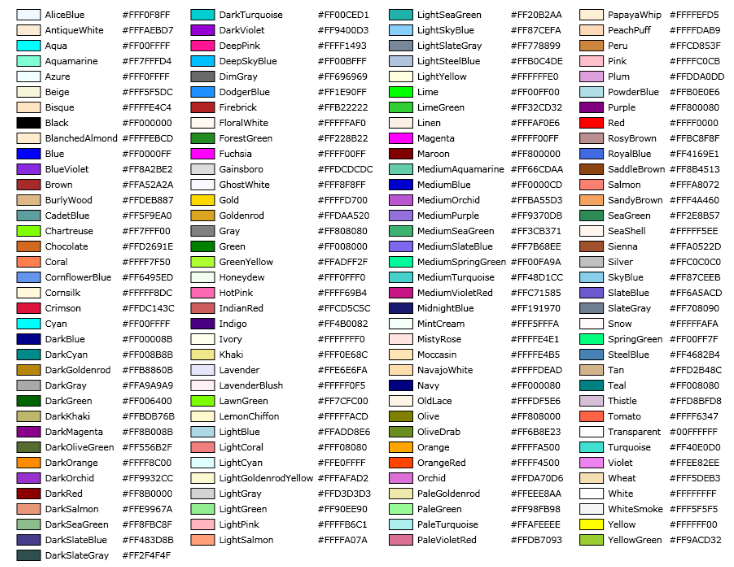
1. Отчет должен содержать скриншоты отношений из БД, листинг кода для создания форм, а также листинг SQL запросов, скриншоты форм. 

Рис.17 Цветовая палитра.

**Лабораторная работа № 16**

**«Удаление, добавление и изменение данных с помощью запросов»**

Время выполнения: 80 минут.

Цель: Изучение синтаксиса для управления данными с помощью элементов формы.

Методические указания

Для того, чтобы передать некоторый запрос от формы к БД используется класс SqlCommand. В нее необходимо передать текст нашей команды, а также объект класса SqlConnection. Из этого можно сделать вывод, что одно приложение может связываться с разными базами данных.

Все SQL запросы в самом общем виде можно разделить на те, что возвращают некоторые значения или выборку (например, select запросы), а также те, которые совершают некоторое действие (например, удаляют или добавляют данные). Как работать с первыми подробнее рассмотрим в следующих работах, а в этой разберем, с помощью какого метода и в каком порядке необходимо передать запросы манипуляции данными.

У класса SqlCommand есть три метода ExecuteNonQuery, ExecuteScalar, ExecuteReader. В этой работе мы будем использовать первый. ExecuteNonQuery – метод, который используется для выполнения запросов изменяющих данные (Update, Insert, Delete). Он возвращает количество измененных строк. Например, если ваш запрос удалит 5 строк, метод вернет число 5.

Приведем пример части кода, в котором выполняется подключение к БД, а также передается запрос на удаление строк из таблицы Товары (код, наименование, цена), где код равен 2.

**string** connectionString = @" Data Source= MyServer; Initial catalog=MyDataBase; Integrated Security=True";

**SqlConnection** MyConnection **= new SqlConnection (**connectionString);

**string** ComDel = " Delete from Товары where код=2";

**SqlCommand** cmd1 **= new SqlCommand (**ComDel,MyConnection **);**

MyConnection.Open();

cmd1.ExecuteNonQuery();

MyConnection.Close();

Теперь рассмотрим вариант, когда нам надо считать некоторые данные с формы.

Пусть мы вводим код товара в текст бокс TextBox1, а далее удаляем только необходимые записи.

Здесь есть два варианта решения этой задачи. Первый самый простой, просто считать текст с формы, а далее добавить его в текст запроса. Предыдущий код можем преобразовать следующим образом.

**string** connectionString = @" Data Source= MyServer; Initial catalog=MyDataBase; Integrated Security=True";

**SqlConnection** MyConnection **= new SqlConnection (**connectionString);

**string** ComDel = " Delete from Товары where код="+TextBox1.text;

**SqlCommand** cmd1 **= new SqlCommand (**ComDel,MyConnection **);**

MyConnection.Open();

cmd1.ExecuteNonQuery();

MyConnection.Close();

Но у этого способа есть нежелательные возможные последствия в виде SQL инъекций. Поэтому лучше использовать параметры. Это объекты класса SqlParameter. На вход мы передаем строковую переменную, в которой содержится имя переменной и источник данных. После этого необходимо добавить конкретный параметр к вашей команде с помощью метода Add.

Изменим наш код, добавив в него параметры.

**string** connectionString = @" Data Source= MyServer; Initial catalog=MyDataBase; Integrated Security=True";

**SqlConnection** MyConnection **= new SqlConnection (**connectionString);

**string** ComDel = " Delete from Товары where код=@code”;

**SqlCommand** cmd1 **= new SqlCommand (**ComDel,MyConnection **);**

**SqlParameter pr1 = new SqlParameter (“@code”,** TextBox1.text **);**

**Cmd1.Parameter.Add(pr1);**

MyConnection.Open();

cmd1.ExecuteNonQuery();

MyConnection.Close();

Задания на лабораторную работу.

1. Импортируйте документ Bibl.xsl в вашу БД.
2. Создайте интерфейс, в котором будет DataGridView, текстовые поля для добавления и изменения данных, а также интерфейс для удаления данных.

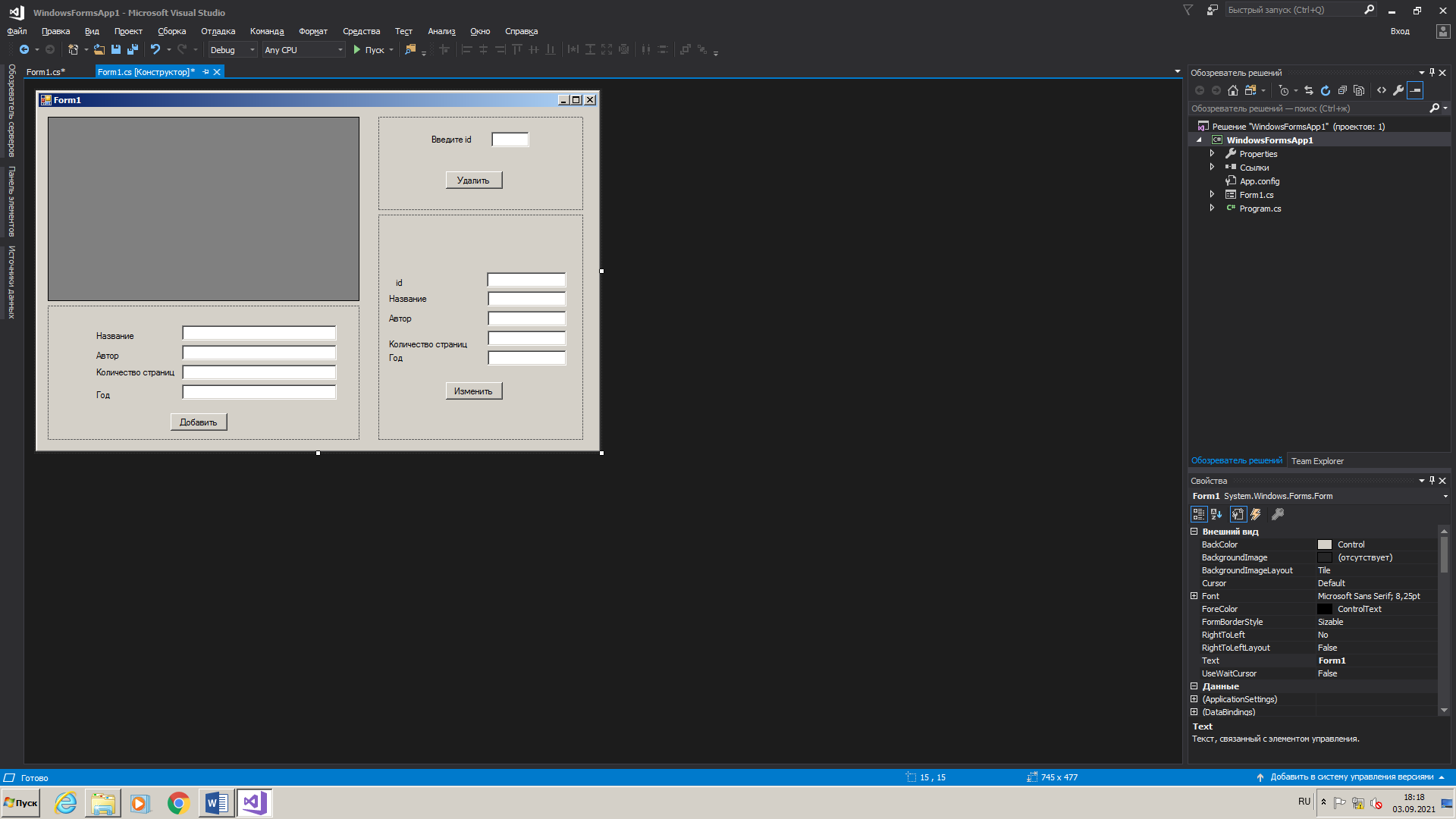


Рис. 18 – Возможный вид вашей формы.

1. Реализуйте функции добавления, изменения и добавления данных с помощью SQL запросов. Обратите внимание, что результат ваших манипуляций должен сразу отображаться на DataGridView.
2. Проверьте правильность работы вашего приложения. Отчёт должен содержать листинг кода и скриншоты результатов.

**Лабораторная работа № 17**

**«Различные способы отображения данных»**

Время выполнения: 80 минут.

Цель: Изучение различных элементов формы, которые можно использовать для отображения данных, а также их синтаксис.

Методические указания

В прошлой работе мы с вами изучили метод ExecuteNonQuery(). В этой изучим ExecuteScalar(), ExecuteReader(), а также вспомним какие элементы, кроме DataGridView можно использовать для демонстрации данных из БД.

SQL запросы, которые возвращают данные можно также выделить в две группы. Первая возвращает набор данных, вторая – одно значение. Для второй группы отлично подойдет метод ExecuteScalar(). Он возвращает первую строку результата запроса типа object, то есть его целесообразно использовать, когда вы возвращаете всего одну строку. Например, в случае запросов с агрегатными функциями или с фильтрацией по первичному ключу.

Пусть у нас есть таблица Товары (код, наименование, цена). Напишем код, который выведет значение максимальной цены в listBox1.

**string** connectionString = @" Data Source= MyServer; Initial catalog=MyDataBase; Integrated Security=True";

**SqlConnection** MyConnection **= new SqlConnection (**connectionString);

**string** ComDel = " Select Max(Цена) from Товары";

**SqlCommand** cmd1 **= new SqlCommand (**ComDel,MyConnection **);**

MyConnection.Open();

String b=cmd1.ExecuteScalar().ToString();

listBox1.Items.Add(b);

MyConnection.Close();

Если же нам необходимо вывести несколько строк, то надо работать с методом ExecuteReader(). Он передает текст команды через подключение и возвращает объект класса SqlDataReader, он представляет собой способ чтения потока строк из вашей БД. Значит и обращаться к нему нужно соответствующим образом. Здесь нам может понадобиться метод Read(), который перемещает SqlDataReader к следующей записи.

Приведем в пример код, который будет передавать на listBox1 наименование всех товаров из, уже известной нам таблицы, Товары.

**string** connectionString = @" Data Source= MyServer; Initial catalog=MyDataBase; Integrated Security=True";

**SqlConnection** MyConnection **= new SqlConnection (**connectionString);

**SqlDataReader** Reader1;

**string** ComDel = " Select Наименование from Товары";

**SqlCommand** cmd1 **= new SqlCommand (**ComDel,MyConnection **);**

MyConnection.Open();

Reader1=cmd1.ExecuteReader();

While (Reader1.Read())

{

listBox1.Items.Add(Reader1[0].ToString());

}

MyConnection.Close();

Задания на лабораторную работу.

1. Создайте таблицу Users с атрибутами Name, login, role. Причем значение login должно быть уникальным, а role может быть user или admin.
2. Занесите в таблицу Users 5 строк.
3. Создайте форму. Разместите на ней объекты label, button и textbox.
4. Выполните подключение к БД.
5. Осуществите следующий принцип работы формы: в текстовое поле необходимо ввести логин. По нажатию кнопки в лейбле появляется текст «Привет, *Имя\_пользователя*». Причем Имя\_пользователя должно соответствовать логину.
6. Разместите на форме объект listBox и любой элемент, который будет отвечать за выбор роли (textbox, radiobutton, checkbox, combobox и т.д.), а также кнопку.
7. Реализуйте следующий механизм. Пользователь выбирает роль, нажимает кнопку в лист боксе появляются имена пользователей, соответствующие этой роли.

**Лабораторная работа № 18**

**«Создание окна входа и регистрации пользователя»**

Время выполнения: 160 минут.

Цель: Изучение синтаксиса для подключения к базе данных через приложение Windows Forms. Простейший способ отображения данных.

Методические указания

Вы уже обладаете всеми знаниями для создания окна входа и регистрации. Воспользуйтесь методическими указаниями из предыдущих лабораторных работ, а также знаниями, полученными на занятиях по разработке программных модулей. Здесь опишем основной функционал окна входа.

В самом простом случае для формы входа и регистрации достаточно иметь два текстбокса для ввода логина и пароля, две кнопки – «вход» и «регистрация» и подписи для создания интуитивно понятного интерфейса. Окно для ввода пароля должно скрывать вводимые знаки пользователя.

Вы можете усовершенствовать ваш интерфейс, например, при нажатии на кнопку регистрации раскрыть новые текстовые поля, которые будут отвечать за внесение дополнительных данных (ФИО, даты рождения, номера телефона и т.д.).

Здесь мне бы хотелось описать, что происходит на уровне связи формы и БД. Данные о пользователе хранятся в БД. Конечно, их надо шифровать и не хранить пароли в открытом виде. Но это мы будем изучать немного позже. В этой работе будем хранить пароли в открытом виде, но больше так не делайте!

Когда пользователь вводит данные для входа, на базу данных в таблицу с пользователями отправляется запрос, которые проверяет есть ли пользователи с такими данными. Пусть мы ввели логин в textBox1, а пароль в textBox2, а данные хранятся в таблице Users (id, login, password). Тогда запрос, проверяющий пользователей может выглядеть следующим образом.

select count(\*) from Users where login=textBox1.Text and password= textBox2.text

В результате запроса мы получаем количество строк с соответствующими логином и паролем. Если оно равняется 0, значит таких пользователей в базе нет. В этом случае форма должна как-то сообщить пользователю, что произошла ошибка ввода данных. Подобными запросами можно проверить правильность отдельно введенных данных. Например, для случаев, когда пользователь существует, но пароль введен неверно.

В случае регистрации по нажатию кнопки все данные введенные в поля сохранятся в базе данных. В качестве запроса здесь можно использовать INSERT, а также параметры.

Задания на лабораторную работу.

1. Работайте с таблицей Users из предыдущей ЛР.
2. Создайте форму для входа и регистрации пользователя по описанию из методических указаний, учитывая следующие требования:

* При регистрации по умолчанию пользователю назначается роль User, а данные заносятся в таблицу Users.
* Если пользователь выбрал уже существующий логин, появляется сообщение «Такой логин уже занят, выберите другое имя пользователя»
* Зарегистрироваться с пустыми текстовыми полями невозможно.
* Если пользователь ввел верные данные, появляется Сообщение «Данные введены верно. Добро пожаловать!»
* Если пользователь ввел верный логин, но неверный пароль, то появляется сообщение «Вы ввели неверный пароль».
* Если пользователь ввел неверные данные, то появляется сообщение «Вы ввели неверные данные».
* Текстовое поле с паролем должно скрывать вводимые символы.

**Лабораторная работа № 19**

**«Создание приложения с опросом»**

Время выполнения: 160 минут.

Цель: Вспомнить способы обработки данных с различных элементов формы. Научиться выбирать удобный способ для сбора и обработки информации, а также осуществлять его.

Задания на лабораторную работу.

1. Создайте приложение Windows Forms.
2. В нем создайте приветственную форму. Именно на этом этапе наш пользователь будет выбирать хочет он пройти опрос или нет. В этой форме должны быть **две кнопки**, по нажатию одной их них, мы переходим в новую форму с опросом, по нажатию второй – форма закрывается. **Размер формы** должен быть **фиксированным**, чтобы пользователь не мог менять его сам. Стиль оформления свободный (Рис).



Рис.19 Пример окна формы

1. Создайте вторую форму, на которую перейдет наш пользователь, чтобы пройти опрос. В этой форме будет вопрос о возрасте, оформленный в виде **выпадающего списка**. Выбор ответа должен быть **строго из представленных в списке**, **невозможно ввести свой вариант**. Создайте кнопку «Далее», после нажатия которой будет открываться не новая форма, а в этой же форме появится новый вопрос (Рис.20).



Рис.20 Выпадающий список

1. Следующий вопрос «Любите ли Вы гулять?» должен быть реализован с помощью выбора из двух вариантов «Да/Нет». Также должна присутствовать кнопка «Далее», после нажатия которой мы все еще остаемся в старой форме, но при этом появится новый вопрос (Рис.21).



Рис.21 Новый вопрос

1. Если человек ответил да, то его отправляют на новый вопрос, если нет, то выскакивает окно с сообщением «Спасибо, опрос окончен» (Рис.22).



Рис.22

1. Новый и последний опрос совершается с помощью множественного выбора любимых мест для прогулки. Также должна быть кнопка «Отправить!», после нажатия которой форма закрывается, а все данные отправляются в БД (Рис.23).



Рис.23

1. Создайте БД, в которую будут отправляться результаты опроса.
2. Подумайте, как лучше и целесообразнее хранить данные (В одной таблице или нескольких, записать ответ или подсчитывать количество того или иного выбора и т.д.)
3. Создайте следующие представления:

* Сколько человек отказалось от прохождения опроса.
* Какой процент всех опрашиваемых не любит гулять.
* Сколько человек выбрало то или иного место для прогулки, в процентах (Например, в парке 20%, на улице 10% и т.д.).
* Сколько людей предпочитает загородный отдых, а сколько – городской, в процентах. Будем считать категории «В парке», «На улице», «По дому» городскими, а остальные загородными.

Примечание: вы можете сделать опрос со своими вопросами, но ваша форма должна содержать те же элементы формы, что и в примерах выше.

**Лабораторная работа № 20**

**«Создание интерфейса администратора»**

Время выполнения: 80 минут.

Цель: Усовершенствовать работу номер 18, добавив к ней личный кабинет администратора. Научиться создавать инструменты работы с БД для администратора.

Методические указания

Перед тем как давать методические указания к этой лабораторной работе, давайте определимся с тем, какой функционал есть у администратора. Как правило администратор, который работает с базой данных, имеет доступ ко всем таблицам, а также может редактировать все данные, удалять и добавлять их.

В работе номер 15 мы отображали все таблицы вашей БД в формах, а в работе номер 18 делали окно входа. Теперь давайте рассмотрим способ, с помощью которого можно выполнять некоторые действия с таблицей проще и быстрее, чем передавая одни и те же запросы на БД.

Для этого мы будем использовать объекты класса SqlDataAdapter.

Задания на лабораторную работу.

1. Возьмите за основу вашу БД из ЛР 15 и форму входу из ЛР 18.
2. Реализуйте следующее действие: пользователь, чья роль при входе идентифицируется, как администратор, может иметь доступ ко всем таблицам вашей БД.
3. Самостоятельно выберите способ отображения таблиц. Все на одной форме, на разных формах, панелях или с помощью TabControl.
4. Помимо просмотра, он может добавлять изменять и удалять любые строки из БД.
5. Проверьте каждое из этих действий. Убедитесь, что изменения происходят не только в вашей форме, но и в базе данных.
6. Для обычного пользователя реализуйте просмотр всех данных.

**Лабораторная работа № 21**

**«Реализация функции сортировки, поиска и фильтрации**

**в интерфейсе вывода данных»**

Время выполнения: 80 минут.

Цель: Усовершенствовать работу номер 18, добавив к ней личный кабинет администратора. Научиться создавать инструменты работы с БД для администратора.

Методические указания

Задания на лабораторную работу.

1. За основу возьмите ЛР 21. Для обычных пользователей выполните следующие действия.
2. Реализуйте функции фильтра данных по следующим атрибутам: ФИО студента, номер зачетки, ФИ преподавателя, название предмета.
3. Реализуйте функцию поиска по уникальному атрибуту в каждом отношении.
4. Отредактируйте отображение таблиц, скройте таблицы и данные, которые не нужны для работы обычного пользователя. Например, такой столбец, как номер факультета в таблице студент не несет никакой информации. Они полезны лишь для администраторов баз данных. Отобразите вместо номера название факультета. И т.д.

**Лабораторная работа № 22**

**«Создание приложения для формирования отчетов»**

Время выполнения: 80 минут.

Цель: Усовершенствовать работу номер 18, добавив к ней личный кабинет администратора. Научиться создавать инструменты работы с БД для администратора.

Методические указания

Задания на лабораторную работу.

**Лабораторная работа № 23**

**«Шифрование данных»**

Время выполнения: 80 минут.

Цель: Изучить различные способы шифрования данных в MS SQL Server. Научиться их применять для защиты данных в таблицах.

Методические указания

В MS SQL можно шифровать данные с помощью сертификата, алгоритмов с ассиметричным ключом, алгоритмов с симметричным ключом, шифрования с помощью пароля. Рассмотрим каждый из этих способов.

Последовательно рассмотрим все 4 пункта.

**Шифрование с помощью сертификата**

Сертификат представляет собой объект безопасности с цифровой подписью, который содержит открытый ключ для SQL Server. Можно использовать сертификаты, сформированные внешними средствами, или, созданные в SQL Server. Также для сертификатов можно установить дату окончания действия. Приведем пример создания такого сертификата. Выполните эти действия в вашей БД.

Создайте таблицу:

CREATE TABLE TableCertEnc1(Name nvarchar(max));

Заведите сертификат:

CREATE CERTIFICATE OurCert1

ENCRYPTION BY PASSWORD = 'MyPassword'

WITH SUBJECT = 'example1';--определяет цель выдачи сертификата (его значением заполняется соответствующее поле сертификата в соответствии со стандартом X.509v1)

Используйте функцию EncryptByCert() для вставки зашифрованной информации в таблицу. EncryptByCert() в качестве аргумента требует id сертификата, который мы получаем с помощью Cert\_ID(). И само значение, которые вы хотите внести в таблицу:

INSERT INTO TableCertEnc1

values(EncryptByCert(Cert\_ID('OurCert1'), 'Secret Name'));

Выведите информацию и посмотрите результат:

Select \* From TableCertEnc1

Для дешифровки используется функция DecryptByCert со следующими параметрами:

* 1ый параметр - идентификатор сертификата
* 2ой параметр - строка, переменная или столбец для расшифровки
* 3ий параметр - пароль, указанный при создании сертификата

SELECT (Convert(Nvarchar(max), DecryptByCert(Cert\_ID('OurCert1'), Name, N'MyPassword'))) FROM TableCertEnc1;

**Шифрование с помощью ассиметричного ключа**

Асимметричные ключи используются для защиты симметричных ключей. Их можно также использовать для ограниченного шифрования данных и цифровой подписи объектов БД. Ассиметричный ключ состоит из закрытого ключа и соответствующего открытого ключа. Приведем пример.

Создайте отдельную таблицу:

CREATE TABLE TableAsKey(Name nvarchar(100));

Создайте ассиметричный ключ. В данной команде устанавливается алгоритм шифрования и пароль. Всего доступны пять алгоритмов (RSA\_4096, RSA\_3072, RSA\_2048, RSA\_1024, RSA\_512, последние два являются устаревшими и использовать их не рекомендуется).

CREATE ASYMMETRIC KEY AsKey

WITH ALGORITHM = RSA\_2048

ENCRYPTION BY PASSWORD = 'MyPassword';

Зашифровка информации и вставка в таблицу происходит следующим образом:

INSERT INTO TableAsKey

values(EncryptByAsymKey(AsymKey\_ID('AsKey'), 'Secret Name'));

Выведите зашифрованную информацию (секретные данные не должны быть видны):

Select \* from TableAsKey

Дешифровка и вывод информации:

SELECT (Convert(Nvarchar(100), DecryptByAsymKey(AsymKey\_ID('AsKey'),Name, N'MyPassword'))) FROM TableAsKey;

**Шифрование с помощью симметричного ключа**

Создаем отдельную таблицу:

CREATE TABLE TableSymKey(Name nvarchar(100));

Создаем ассиметричный ключ, в данной команде устанавливается алгоритм шифрования и пароль:

CREATE SYMMETRIC KEY SymKey

WITH ALGORITHM = AES\_128

ENCRYPTION BY PASSWORD = 'MyPassword';

Открытие симметричного ключа и вставка зашифрованной информации:

OPEN SYMMETRIC KEY SymKey DECRYPTION BY PASSWORD = 'MyPassword';

INSERT INTO TableSymKey

values(EncryptByKey(Key\_GUID('SymKey'), N'Secret Name'));

select \* from TableSymKey

Чтение и дешифровка информации из таблицы:

SELECT (Convert(Nvarchar(100), DecryptByKey(Name))) FROM TableSymKey;

CLOSE SYMMETRIC KEY SymKey

Стоит отметить, что шифрование/дешифровка с помощью симметричного ключа происходит быстрее нежели шифрование/дешифровка с помощью асимметричного ключа (однако шифрование с помощью асимметричного ключа считается более надежным).

**Шифрование с помощью пароля**

CREATE TABLE TablePassword(Name nvarchar(100));

INSERT INTO TablePassword

values(EncryptByPassphrase('MyPassword', N'Secret Name'));

Select \* from TablePassword

SELECT (Convert(Nvarchar(100), DecryptByPassphrase('MyPassword', Name))) FROM TablePassword;

**Защита симметричного ключа пользовательским сертификатом**

Шаг 1. Создаем главный ключ БД

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'MyPassword;

--DROP MASTER KEY

Шаг 2. Создаем таблицу в которой будем шифровать зарплату

CREATE TABLE Programmers

(Id int identity,

Name nvarchar(100),

Lang nvarchar(100),

Salary varbinary(128))

--Drop table Programmers

Шаг 3. Создаем сертификат для защиты симметричного ключа

CREATE CERTIFICATE Cert1

WITH SUBJECT = 'Secret info - Salary';

GO

CREATE SYMMETRIC KEY SymKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY CERTIFICATE Cert1;

GO

--DROP SYMMETRIC KEY SymKey

Шаг 4. Открываем симметричный ключ (обратите внимание на использование сертификата)

OPEN SYMMETRIC KEY SymKey

DECRYPTION BY CERTIFICATE Cert1;

Шаг 5. Вставка нескольких строк в таблицу Programmers с зашифрованной зарплатой

INSERT INTO Programmers VALUES

('Peter', 'C++', EncryptByKey(Key\_GUID('SymKey'), '3000')),

('Mary', 'C++', EncryptByKey(Key\_GUID('SymKey'), '2500')),

('Konstantin', 'Java', EncryptByKey(Key\_GUID('SymKey'), '23415')),

('Vladimir', 'Java', EncryptByKey(Key\_GUID('SymKey'), '56432')),

('Kate', 'C#', EncryptByKey(Key\_GUID('SymKey'), '23456'))

--DELETE FROM TABLE3

Шаг 6. Вывод информации

SELECT \*, CONVERT(varchar, DecryptByKey(Salary)) AS [Decrypted Salary]

FROM Programmers

SELECT \* FROM Programmers

Шаг 7. Закрываем симметричный ключ

CLOSE SYMMETRIC KEY SymKey

Задания на лабораторную работу.

1. Изучите все способы шифрования в методическом указании и выполните действия, описанные в нем.
2. Создайте таблицу Сотрудники со следующими атрибутами (Имя, Фамилия, Зарплата)
3. Зашифруйте и дешифруйте имена и фамилии сотрудников всеми способами, описанными в методическом указании.

**Лабораторная работа № 24**

**«Создание окна входа пользователя**

**с использованием функций хеширования»**

Время выполнения: 80 минут.

Цель: Усовершенствовать работу номер 18, добавив к ней личный кабинет администратора. Научиться создавать инструменты работы с БД для администратора.

Методические указания

Задания на лабораторную работу.

1. Возьмите за основу ЛР 18.
2. Измените способ хранения паролей в вашей БД с открытого на хешированный для всех пользователей.
3. Измените код приложения так, чтобы мы могли сравнивать введенный пароль с его хешированным вариантом в БД.
4. Осуществите вход и проверьте правильность работы программы.

**Лабораторная работа № 25**

**«Определение прав доступа пользователя к данным»**

Время выполнения: 60 минут.

Цель: Научиться определять права доступа для защиты данных в базе.

Методические указания

SQL обычно применяется в многопользовательских средах, которые требуют разграничения прав пользователей с точки зрения доступа к данным и прав на выполнение с ними тех или иных манипуляций. Для этих целей в SQL реализованы средства, позволяющие устанавливать и контролировать привилегии пользователей бд.

Каждый пользователь SQL имеет специальное имя (идентификатор), с помощью которого осуществляется идентификация пользователя с целью установки и определения его прав с точки зрения доступа к данным.

Каждая посланная к СУБД команда запроса ассоциируется СУБД с идентификатором доступа к данным конкретного пользователя. Пользователь определяется следующей командой:

CREATE USER <имя пользователя> IDENTIFIED <пароль>. Пользователь становится известен БД, но не может выполнять никаких операций.

DROP USER<имя пользователя> - удаление пользователя.

Назначаемые привилегии - это разрешение на выполнение указанным пользователям данной команды над определённым объектом БД.

Имеется несколько типов привилегий, соответствующих нескольким типам операций. GRANT – Установка, REVOKE – отмена привилегий.

Привилегии над объектом означают, что пользователь имеет право(привилегию) на выполнение данной команды только на конкретном объекте БД. Привилегии объекта связаны одновременно и с пользователями и таблицами БД, то есть привилегия даётся конкретному пользователю в конкретной таблице (это может быть, как базовая таблица, так и представление).

Пользователь, создавший таблицу любого вида – ее владелец, то есть имеет все привилегии, относящиеся к таблице и может передавать привилегии другим пользователям (SELECT, INSERT, DELETE, UPDATE и др.).

Могут быть и нестандартные привилегии объекта (INDEX, SYNONIM, ALTER, EXECUTE).

Назначение этих привилегий осуществляется командой GRANT.

Пользователь, являющийся владельцем, может передать другому пользователю привилегию выборки с помощью следующей команды:

CREATE SELECT ON STUDENT (таблица)

TO IVANOV (имя пользователя);

Без наличия других привилегий IVANOV может только выбирать значения.

Для команды GRANT сначала проверяется допустимость этой команды для давшего её пользователя. IVANOV дать этой команды самостоятельно не может. Он также не может предоставить право SELECT другому пользователю, так как таблица принадлежит не ему.

Например, команда:

GRANT INSERT, SELECT ON EXAM\_MARKS TO IVANOV, PETROV; позволяет предоставить привилегии вставки и выборки сразу двум пользователям IVANOV, PETROV. Однако, в стандарте невозможно предоставить привилегии **для нескольких таблиц** одной командой GRANT.

Для того, чтобы предоставить все привилегии используют команду ALL.

Пример: GRANT ALL PRIVILEGES ON EXAM\_MARKS TO IVANOV; Аргумент public используется для передачи указанных привилегий всем пользователям:

GRANT SELECT ON EXAM\_MARKS TO PUBLIC;

Предоставление выбора информации из таблицы EXAM\_MARKS любому пользователю.

Привилегии, которые не определяются в терминах специальных объектов данных, называются привилегиями системы или правами БД. Они включают в себя права создавать объекты данных, отличающиеся от базовых таблиц и представлений.

Привилегии системы должны дополнять, а не заменять привилегии объекта, которые стандарт требует от создателя представления. В любой системе имеются типы пользователей, которые обладают большинством или всеми привилегиями (администратор БД).

CONNECT – подключение, права создать представления и синонимы, RESOURCE – создать базовые таблицы.

DBA – права администратора.

В большинстве реализаций SQL нового пользователя создаёт DBA (то есть администратор БД) автоматически предоставляющий новому пользователю привилегию CONNECT. Например,

GRANT CONNECT TO PETROV IDENTIFIED BY “petrovpassword”;

Задания на лабораторную работу.

1. Передайте пользователю PETROV право на изменение в базе данных значений оценок для записей о студентах;
2. Передайте пользователю SIDOROV право передавать другим пользователям права на осуществление запросов к таблице EXAM\_MARKS;
3. Отмените привилегию INSERT по отношению к таблице STUDENT у пользователя IVANOV и у всех других пользователей, которым она была им предоставлена.