Федеральное агентство по образованию

Государственное образовательное учреждение

высшего профессионального образования

Владимирский государственный университет

Кафедра управления и информатики в технических

и экономических системах

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

К ЛАБОРАТОРНЫМ РАБОТАМ

ПО ДИСЦИПЛИНЕ«БАЗЫ ДАННЫХ»

Составители:

А.Б. ГРАДУСОВ

Д.А. ГРАДУСОВ

А.А. ГАЛКИН

Зав. кафедрой А.А. Галкин

Составители А.Б. Градусов

Д.А. Градусов

А.А. Галкин

Владимир2007

2

УДК004.65

ББК32.988-5

М54

Рецензент

Кандидат технических наук, доцент кафедры

информационных систем и информационного менеджмента

Владимирского государственного университета

С.Ю. Кириллова

Печатается по решению редакционно-издательского совета

Владимирского государственного университета

Методические указания к лабораторным работам по дисцип-лине«Базы данных» / Владим. гос. ун-т; сост. : А. Б. Градусов,

Д. А. Градусов, А. А. Галкин. – Владимир: Изд-во Вадим. гос. ун-та, 2007. – 68 с.

Содержат описание четырех лабораторных работ по наиболее актуальным темам

теории и практики баз данных. В каждой работе приведены краткие теоретические све-дения, задание к работе, содержание отчета, контрольные вопросы.

Предназначены для студентов специальностей080801 – прикладная информатика в

экономике и220201 – управление и информатика в технических системах всех форм

обучения.

Ил. 49. Библиогр.: 3 назв.

УДК004.65

ББК32.988-5

М54

3

**Лабораторная работа №1**

**СОЗДАНИЕ БАЗЫ ДАННЫХ MS SQL SERVER**

***Цель работы:*** практическое освоение методов создания базы данных.

Данные в **SQL Server** организованы в нескольких различных объектах, к которым относятся:

• пользователи базы данных (**Database Users**);

• роли базы данных (**Database Roles**);

• таблицы (**Tables**);

• индексы (**Indexes**);

• ключи (**Keys**);

• представления (**SQL Server Views**);

• хранимые процедуры (**Stored Procedures**);

• триггеры **(Triggers**);

• правила (**Rules**);

• условия на значения (**Constraints**);

• значения по умолчанию (**Defaults**);

• типы данных, определенные пользователем (**User Defined Datatypes**);

• диаграммы (**Database Diagrams**).

Создание базы данных в **SQL Server** осуществляется в интерактивном режиме с помощью **SQL Server Management Studio**, который позволяет:

– создавать базу данных и журнал транзакций;

– создавать и модифицировать таблицы,

– создавать хранимые процедуры и триггеры,

– создавать представления данных;

– определять для таблиц индексы;

– устанавливать отношения между таблицами.

1. **Создание базы данных**

В дереве серверов (левая панель главного окна **Management Studio**) раскройте узел, соответствующий нужному серверу, и щелкните правой кнопкой мыши на пункте **Базы данных**. На экране появится контекстное меню, показанное на рис. 1.

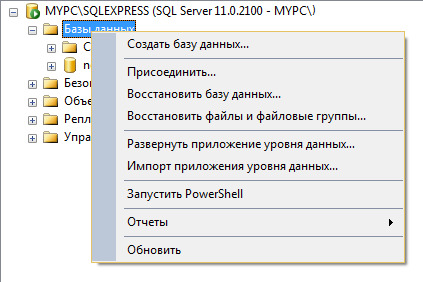


Рисунок 1

В раскрывшемся меню выберите пункт **Создать базу данных**. На экране появится диалог создания новой базы данных (рис. 2).

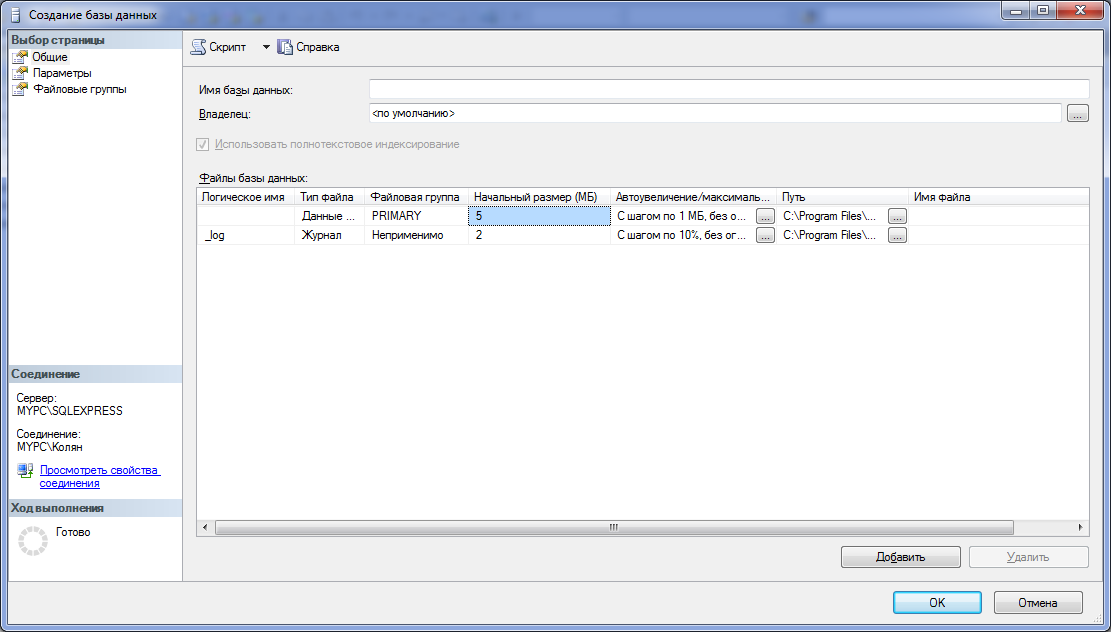


Рисунок 2

В поле ввода **Имя базы данных** введите имя создаваемой базы данных; остальные параметры можно оставить по умолчанию. После нажатия кнопки **ОК** новая БД будет создана и отобразится в списке баз данных выбранного сервера.

**2. Создание таблиц БД**

Чтобы создать таблицу, надо, по меньшей мере, сделать следующее:

• Задать имя таблицы;

• Задать имена составляющих ее столбцов;

• Определить тип данных для каждого столбца;

• Определить, использовать или не использовать заданный по умолчанию режим работы со значением **Null** для каждого столбца;

• Определить условия на значения для столбцов и прежде всего первичный и внешний ключи.

Для того чтобы начать работать с таблицами, в списке баз данных раскройте узел, соответствующий вашей БД, и щелкните правой кнопкой мыши на пункте **Таблицы**. В появившемся меню выберите пункт **Создать таблицу** (рис. 3).

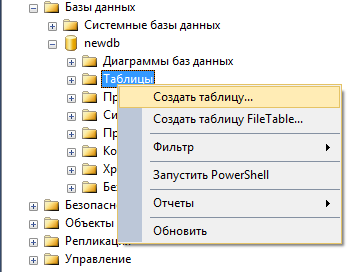


Рисунок 3

На экране появится вкладка для указания имени и типа столбцов таблицы. Также имеется возможность указать, разрешить ли значение Null для столбца. Все основные дополнительные свойства столбца можно указать ниже во вкладке **Свойства столбца**.Свойства самой таблицы, такие как имя, указываются в меню **Свойства**, располагающемся в правой части окна. В сетку в центральной части диалогового окна заносится информация о полях создаваемой таблицы:

• **Имя столбца** – имя поля;

• **Тип данных** – тип данных, хранимых в поле;

• **Разрешить значения NULL** – допускаются ли пустые значения в данном поле.

Длина поля задается с помощью вкладки **Свойства столбца**. Тип данных задается с помощью раскрывающегося списка столбца **Тип данных** диалогового окна (рис. 4).

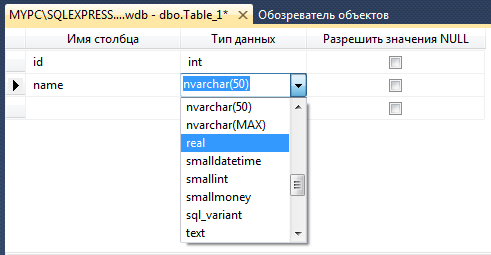


Рисунок 4

Ниже описываются некоторые основные типы данных, допустимые в **MS SQL Server:**

• **int** – целое четырехбайтное число;

• **decimal** – число с плавающей точкой;

• **char/nchar** – строка символов фиксированной длины;

• **varchar/nvarchar** – строка символов переменной длины (Length задает максимальную длину строки);

• **datetime** – дата и время.

Во вкладке **Свойства столбца** можно определить **Значение по умолчанию**, которое будет автоматически присваиваться полю, если не задано конкретное значение.

Если отпала необходимость в каком-либо столбце таблицы, его можно удалить. Для этого выделите соответствующую строку, нажмите на ней правой кнопкой мыши и в появившемся контекстном меню выберите команду **Удалить столбец** или просто нажмите кнопку **Delete**.

После того как структура таблицы будет создана, ее следует сохранить, нажав кнопку **Сохранить** на панели инструментов диалога или нажав сочетание клавиш Ctrl+S. При этом будет предложено ввести имя таблицы (рис. 5).

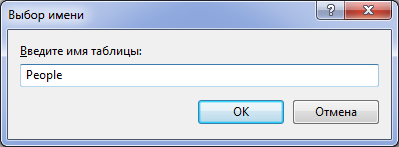


Рисунок 5

Создать таблицу можно также с помощью команды SQL, упрощенный формат которой выглядит следующим образом:

CREATE TABLE <имя\_таблицы>

(<имя\_столбца> <тип\_столбца>

[NOT NULL]

[UNIQUE | PRIMARY KEY]

[REFERENCES <имя\_родительской\_таблицы>

[<имя\_столбца>]]

, ...)

Пользователь обязан указать имя таблицы и список столбцов, для каждого столбца – его имя и тип. Также, по желанию, могут быть указаны параметры:

• **NOT NULL** – в этом случае элементы столбца всегда должны иметь определенное значение (не NULL);

• один из взаимоисключающих параметров **UNIQUE** – значение каждого элемента столбца должно быть уникальным или **PRIMARY KEY** – столбец является первичным ключом;

• **REFERNECES** <имя\_родительской\_таблицы>[<имя\_столбца>] – эта конструкция определяет, что данный столбец является внешним ключом и указывает, на ключ какой родительской таблицы он ссылается. Контроль над выполнением указанных условий осуществляет СУБД.

**3. Модификация таблицы**

Структуру таблицы можно модифицировать, т. е. изменить наименование любого поля и его тип, вставить новое поле или удалить существующее.

Чтобы модифицировать таблицу, выделите ее в обозревателе объектов правой кнопкой мыши и выберите команду **Проект** контекстного меню. В результате вы увидите уже знакомую вкладку, в которой можно достаточно просто изменять структуру таблицы и параметры столбцов.

Конечно же, эти возможности небезграничны. Рассмотрим некоторые имеющиеся ограничения:

1. Нельзя модифицировать столбец:

• задействованный в индексах;

• рассчитываемый столбец;

• ассоциированный со значением по умолчанию.

2. При изменении типов данных предыдущий тип данных должен косвенным образом конвертироваться в новый.

3. Нельзя удалить столбец, если он:

• используется в индексе;

• ассоциирован со значением по умолчанию или связан с объектом по умолчанию;

• связан с правилом.

Для модификации таблицы используются следующие команды:

* Добавить столбцы:

ALTER TABLE <имя\_таблицы> ADD

(<имя\_столбца> <тип\_столбца>

[NOT NULL]

[UNIQUE | PRIMARY KEY]

[REFERENCES

<имя\_родительской\_таблицы> [<имя\_столбца>]]

,...)

• Удалить столбцы:

ALTER TABLE <имя\_таблицы> DROP (<имя\_столбца>,...)

• Модификация типа столбцов:

ALTER TABLE <имя\_таблицы> MODIFY

(<имя\_столбца> <тип\_столбца>

[NOT NULL]

[UNIQUE | PRIMARY KEY]

[REFERENCES <имя\_родительской\_таблицы>

<имя\_столбца>])

,...)

Любые действия по созданию и изменению структуры данных можно сохранить, сгенерировав скрипт (набор команд на языке SQL, описывающий все действия сделанные вами). Это может быть полезно в определенных случаях: например, если кто-то удалил вашу базу, то вы можете без особых проблем восстановить её, причём со всеми данными, которые вы вводили с помощью команды **Insert**.

Для генерации скрипта необходимо в **Обозревателе объектов** щелкнуть правой кнопкой мыши на нужной базе данных, далее – **Задачи,** и затем – **Сформировать скрипты**. В результате появится диалоговое окно **Формирование и публикация скриптов** (рис. 6). Следуя советам мастера можно сохранить скрипт либо в отдельный файл с расширением **\*.sql**, либо опубликовать в сети. Просмотреть скрипт можно в программе **Management Studio**, для этого выберите пункт меню **Файл**, а в нем – команду **Открыть**, и затем – **Файл**.

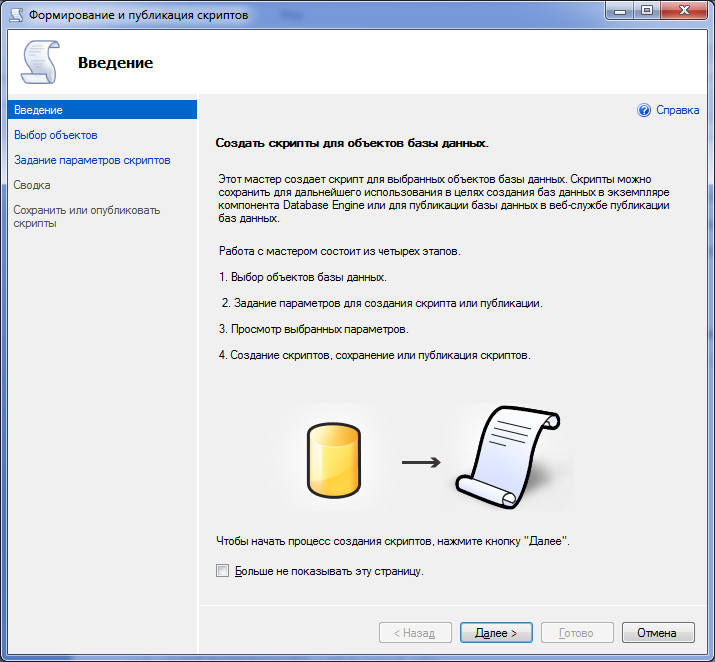


Рисунок 6

Скрипты можно писать самим. Для этого в верхнем меню выберите пункт «**Создать запрос»** или нажмите Ctrl+N. В открывшейся вкладке введите необходимые команды и нажмите кнопку «Выполнить» или «Отладка».

Например, создать базу данных **publications**:

CREATE DATABASE publications;

CREATE TABLE authors (au\_\_id INT PRIMARY KEY,

author VARCHAR(25) NOT NULL);

CREATE TABLE publishers (pub\_id INT PRIMARY KEY,

publisher VARCHAR(255) NOT NULL,

url VARCHAR(255));

CREATE TABLE titles (title\_id INT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

yearpub INT,

pub\_id INT REFERENCES publishers(pub\_id));

CREATE TABLE titleautors (au\_id INT REFERENCES authors(au\_id),

title\_id INT REFERENCES titles(title\_id));

CREATE TABLE wwwsites (site\_id INT PRIMARY KEY,

site VARCHAR(255) NOT NULL,

url VARCHAR(255));

CREATE TABLE wwwsiteauthors (au\_id INT REFERENCES authors(au\_id), site\_id INT REFERENCES wwwsites(site\_id)).

**4. Задание первичного ключа**

***Первичный ключ*** – это столбец или группа столбцов с уникальными значениями, определяющими строки таблицы базы данных. В этом смысле первичный ключ играет роль уникального индекса в поддержании целостности таблицы. Однако **SQL Server** допускает определение для таблицы только одного первичного ключа, а уникальных индексов может быть много.

***Внешние ключи*** – это столбцы в таблице, которые соответствуют первичным ключам других таблиц.

Ключи, обладая необходимой функциональностью, должны использоваться в качестве средства обеспечения целостности ссылок.

В **SQL Server** первичный и внешний ключ можно добавить к таблице с помощью **SQL Server Management Studio**:

1. Запустите **Management Studio** и выберите сервер, базу данных и таблицу, с которой вы хотите работать.

2. Щелкните правой кнопкой мыши на этой таблице и в контекстном меню выберите команду **Проект**.

3. Выберите столбец, который хотите использовать в качестве первичного ключа, и нажмите кнопку **Задать первичный ключ** на панели инструментов (рис. 7).



Рисунок 7

В результате этих действий будет создан первичный ключ.

Общепринятой практикой является создание специального служебного поля, выполняющего функции первичного ключа. Для обеспечения уникальности значений лучше всего будет использовать автоинкрементное поле. Для этого в диалоговом окне команды **Проект** добавьте в таблицу поле типа **int**. В нижней части диалога создания таблицы (рис. 8) на вкладке **Свойства столбца** укажите в поле **Спецификация идентификатора -> Идентификатор** значение **Да**. Поле **Начальное значение идентификатора** задает начальное значение автоинкрементного поля (будет присвоено при добавлении первой записи в таблицу). Поле **Шаг приращения идентификатора** содержит шаг инкремента – число, на которое будет увеличиваться значение автоинкрементного поля при добавлении каждой новой записи.

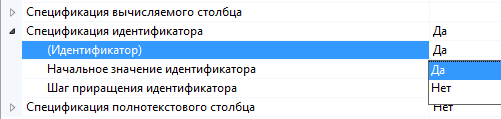


Рисунок 8

После этого выделите автоинкрементное поле в сетке и нажмите кнопку **Задать первичный ключ** на панели инструментов.

Если в первичный ключ требуется включить более одного столбца, то необходимо выполнить команду **Индексы/Ключи** (например, нажав кнопку **Управление индексами и ключами** на панели инструментов (рис. 9)).

На экране появится диалоговое окно **Индексы/Ключи** (рис. 10).



Рисунок 9

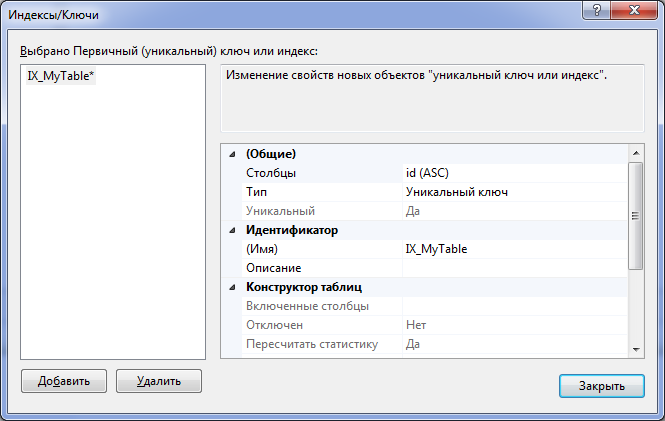


Рисунок 10

Нажмите кнопку **Добавить** и выберите в строке **Столбцы** все столбцы, которые требуется включить в первичный ключ. Затем в строке **Тип** выберите **Уникальный ключ**. Внешний ключ проще всего создать посредством диаграммы, о чем будет сказано ниже.

**5. Создание индексов**

Чтобы получить ответ на любой запрос из таблицы, не имеющей индекса, **SQL Server** вынужден сканировать таблицу, т. е. считывать в ней каждую строчку. Очевидно, что в случае больших таблиц такой процесс слишком расточителен для сервера.

Для получения оптимальной производительности системы разработчику базы данных важно правильно определить таблицы. Проектирование таблицы не определяет физический способ доступа и хранения данных. Вместо этого **SQL Server** реализует в таблице механизм индексов, который помогает оптимизировать получение ответов на запросы.

Индексы представляют собой наборы уникальных для данной таблицы значений и соответствующий им список указателей на страницы данных, где эти значения находятся в таблице физически.

Для создания новых индексов нажмите кнопку **Управление индексами и ключами** на панели инструментов (см. рис. 9). В открывшемся окне (см. рис. 10) нажмите кнопку **Добавить**. Для указания типа Индекс, необходимо задать его в поле **Тип.** После этого в поле **Имя** можно задать имя индекса. В поле **Столбцы** указываются поля, по которым будет осуществляться индексирование, а также порядок сортировки (указывается в скобках). Поле **Уникальный** позволяет запретить повторяющиеся значения в индексе.

После завершения всех операций следует нажать кнопку **Закрыть**.

**6. Создание условий на значения столбцов**

Условия на значения столбцов (constraints), которые часто называют *ограничениями*, определяются для обеспечения целостности данных в таблицах и столбцах. Рассмотрим некоторые из них.

***Условие на значение столбца* *PRIMARY KEY (Первичный ключ)***

Определение условия **PRIMARY KEY** для таблицы имеет несколько целей. Во-первых, оно гарантирует уникальность всех строк таблицы. Это делается путем запрещения ввода одинаковых значений в столбец. Кроме того, условие на значение первичного ключа запрещает ввод значений NULL в столбец, для которого оно определено.

Вторая цель заключается в том, что наличие индекса позволяет реализовать более быстрое считывание строк в таблице.

Установить это условие можно с помощью команды **Задать первичный ключ** (см. п. 4).

***Условие на значение столбца FOREIGN KEY (Внешний ключ)***

Это условие используется только вместе с предварительно определенным условием **PRIMARY KEY** в родительской таблице. Условие на значение **FOREIGN KEY** ставит в соответствие один или несколько столбцов таблицы идентичному набору столбцов, для которых определено условие на значение **PRIMARY KEY** другой таблицы. Когда обновляются значения тех столбцов таблицы, для которых определено условие на значение **PRIMARY KEY**, автоматически обновляются значения столбцов других таблиц, для которых определено условие **FOREIGN KEY**. Такое автоматическое обновление соответствующих столбцов в различных таблицах после того, как для них определены условия на значения столбцов **PRIMARY KEY** и **FOREIGN KEY**, называется декларативной ссылочной целостностью.

***Условие на значение столбца UNIQUE (Уникальный)***

Для того чтобы запретить повторение значений в любом столбце таблицы, можно применить уникальное условие на значение (**UNIQUE**) к этому столбцу. Это условие устанавливается с помощью диалогового окна **Индексы/Ключи** команды **Управление индексами и ключами**. Следует помнить, что столбец, для которого определено подобное условие на значение, не может быть первичным ключом или его частью.

***Условие на значение столбца CHECK (Проверочное ограничение)***

Проверочное условие на значение (**CHECK**) столбца устанавливает диапазон значений, которые могут быть введены в столбец таблицы базы данных.

Для установки условия на значение **CHECK** нажмите на панели инструментов кнопку **Управление проверочными ограничениями** (рис. 11).



Рисунок 11

На экране появится диалоговое окно **Проверочные ограничения** со списком уже созданных условий (рис. 12). Для создания нового ограничения нажмите кнопку **Добавить**. В поле **Имя** нужно ввести имя ограничения. В поле **Выражение** следует ввести выражение, задающее ограничение (в данном примере зададим запрет на использование имени "Вася" в качестве значения поля **name**).

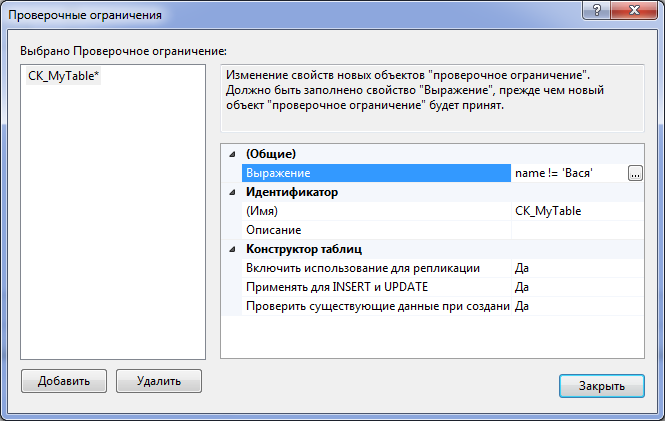


Рисунок 12

**7. Установка связей между таблицами**

Итак, мы уже имеем таблицу **People** с полями **id** и **name**, содержащую информацию о некоторых людях. Создадим таблицу **Phones** с номерами телефонов, связанную с таблицей **People**. При этом считаем, что у каждого человека может быть несколько телефонных номеров. Поля таблицы **Phones**:

• **id** – идентификатор записи типа **int**, автоинкрементный, является первичным ключом;

• **owner\_id** – идентификатор владельца данного телефонного номера типа **int**;

• **phone\_no** – телефонный номер, тип **varchar** с длиной поля 15.

Для задания связи между таблицами **Phones** и **People** в окне редактирования таблицы **Phones** нажмите кнопку **Отношения** (рис. 13).



Рисунок 13

На экране появится диалог **Связи по внешнему ключу** (рис. 14). В нем будет активна вкладка Relationships. Нажмите кнопку **Добавить** для создания новой связи. В поле Имя можно задать ее имя. В диалоговом окне **Спецификация таблиц и столбцов** в списке **Таблица первичного ключа** выберите таблицу, выступающую в отношении со стороны "1" (у нас это будет **People**). В списке **Таблица внешнего ключа** выберите таблицу, выступающую в отношении со стороны "М" (в данном примере – **Phones**). В сетке следует указать для обеих таблиц поля, по которым будет устанавливаться отношение.

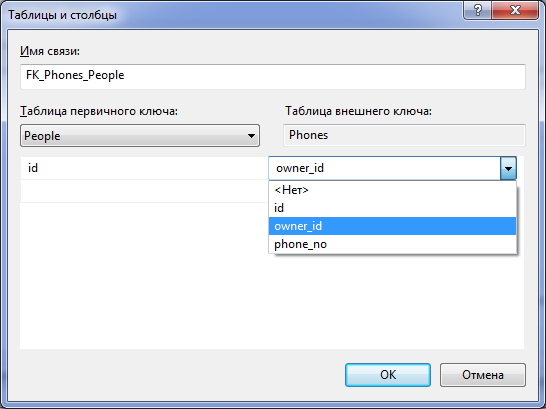


Рисунок 14

Поля **Правило обновления** и **Правило удаления** отвечают за удаление и обновление связанных записей. Например, при значении **Каскадно** будет проводиться каскадное удаление или обновление.

**8. Создание диаграммы базы данных**

В обозревателе объектов щелкните правой кнопкой мыши на элементе Диаграммы баз данных и выберите пункт Создать диаграмму базы данных (рис. 15).

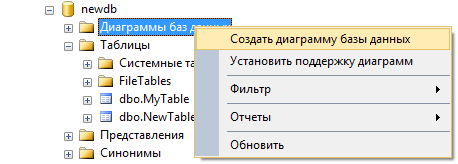


Рисунок 15

Будет запущен мастер создания диаграмм БД. В окне мастера выберите таблицы **People** и **Phones** и нажатием кнопки **Добавить** перенесите на диаграмму (рис. 16).

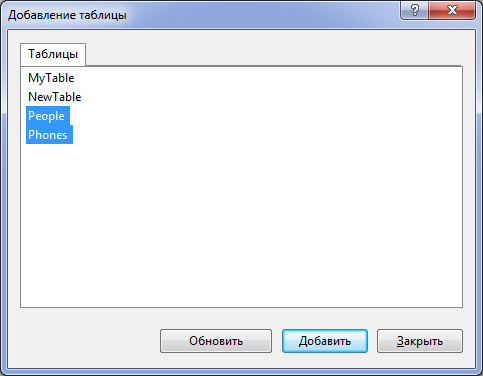


Рисунок 16

Нажмите кнопку **Закрыть**. На экране будет отображена диаграмма вашей БД (рис. 17).

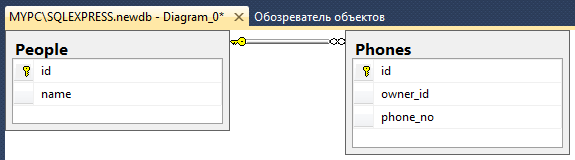


Рисунок 17

**9. Ввод данных в таблицу**

Для того чтобы ввести данные в таблицу, в списке баз данных раскройте узел, соответствующий вашей БД, и щелкните правой кнопкой мыши на пункте **Таблицы**. Теперь достаточно щелкнуть правой кнопкой мыши на интересующей таблице и в появившемся контекстном меню выбрать команду **Изменить первые 200 строк**.

Теперь, для того чтобы ввести в таблицу новое значение, перейдите к строке, отмеченной символом \* (звездочка), и вводите значения в соответствующие столбцы. **SQL Management Studio** будет автоматически проверять введенные данные и сохранять их, если они соответствуют свойствам поля.

Аналогичные действия надо выполнить при изменении данных.

Для удаления строки из таблицы достаточно выделить ее, нажать правую кнопку мыши и выбрать команду **Удалить** контекстного меню, или воспользоваться клавишей Delete.

**Порядок выполнения работы**

1. Загрузите СУБД.

2. Создайте новую базу данных с уникальным именем (см. п. 1), состоящую из четырех таблиц:

**Продавцы** (Код\_продавца, ФИО\_продавца, город\_продавца, комиссионные\_продавца, руководитель, план продаж);

**Заказчики** (Код\_заказчика, ФИО\_заказчика, город\_заказчика, рейтинг заказчика, сумма\_кредита); более высокий рейтинг указываетна большее предпочтение;

**Заказы** (номер\_заказа, сумма\_заказа, дата\_заказа, код\_продавца, код\_заказчика);

**Товар** (Код\_товара, Наименование\_товара, цена, количество).

3. Создайте с помощью **Management Studio** таблицы **Продавцы** и **Заказы** вашей базы данных (см. п. 2).

4. Задайте в них первичные ключи (см. п. 4).

5. Установите условие на значение любого столбца CHECK (см. п. 6).

6. Таблицы **Заказчики** и **Товар** создайте с помощью команды CREATE TABLE (см. п. 2, 3).

7. Создайте связи между таблицами (см. п. 7).

8. Создайте диаграмму вашей базы данных (см. п. 8).

9. Создайте скрипт вашей базы данных.

10. Введите данные во все созданные таблицы.

11. Убедитесь, что в вашей базе данных выполняются правила декларативной ссылочной целостности.

**Содержание отчета**

1. Цель работы.

2. Описание схемы базы данных

3. Описание процесса создания базы данных.

4. Описание процесса создания таблиц.

5. Описание процесса определения условий на значение.

**Контрольные вопросы и задания**

1. Из каких объектов состоит базы данных?

2. Каким образом может быть выполнено создание базы данных

пользователем?

3. Назовите способы создания таблиц в SQL Server.

4. Как создаются таблицы в SQL Server пользователем?

5. Какие типы данных используются в SQL Server?

6. Как устанавливаются связи между таблицами в базе данных?

7. Как создаются первичные и внешние ключи?

8. Каким образом выполняется добавление и удаление ключей?

9. С какой целью и как проводится индексирование?

10. Укажите последовательность действий для определения условий на значения?

11. Перечислите условия целостности данных.

**Лабораторная работа №2**

**МАНИПУЛИРОВАНИЕ ДАННЫМИ В MS SQL SERVER**

***Цель работы:*** практическое освоение команд **SELECT**, **INSERT**, **UPDATE** и **DELETE**.

**SQL Server** как универсальный сервер баз данных имеет средства для манипуляции данными средствами языка **SQL**. Для работы с **SQL**-запросами используется приложение **SQL Server Management Studio**. Набор запросов осуществляется в специальной вкладке, которая открывается после нажатия кнопки **Создать запрос,** а выполнение производится при нажатии кнопки **Выполнить** (рис. 18).

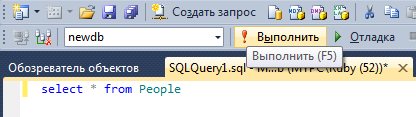


Рисунок 18

При этом выбирается база данных, к которой посылается запрос (рис. 19).

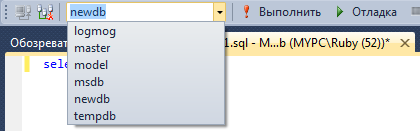


Рисунок 19

Данные могут быть отображены в текстовом виде, в виде сетки или в файле (рис. 20).



Рисунок 20

Для манипуляции и обработки данных используется подъязык SQL манипулирования данными (DML). Рассмотрим запросы на выборку данных, вставку данных, обновление и удаление.

**1. SELECT-запросы**

С целью извлечения нужных данных используются команда **SELECT**. Базовый синтаксис команды **SELECT** выглядит следующим образом:

SELECT [ALL | DISTINCT] <Список выбираемых полей | \*>

FROM [<Имя базы данных>.]<Имя таблицы> | <Имя представления>

[INTO <Получатель>]

[WHERE <Выражение>]

[GROUP BY <Выражение>]

[HAVING <Выражение>]

[ORDER BY <Выражение>]

Предложения, указанные в квадратных скобках, является необязательными.

Опция **ALL** (это значение установлено по умолчанию) выводит все записи, попавшие в результат запроса. Опция **DISTINCT** исключает из результата запроса повторяющиеся записи.

Аргумент <список выбираемых полей> содержит список полей, а также допустимых выражений, включаемых в результирующую таблицу. Элементы списка разделяются запятыми. Каждый элемент этого списка генерирует один столбец результатов запроса. В имя поля можно включать имя выбираемой таблицы или имя локального псевдонима таблица. Это бывает необходимо, если поля разных таблиц имеют одинаковые имена. Псевдоним и имя поля разделяются символом "." (точка).

Предложение **FROM** содержит список таблиц, из которых осуществляется выборка. Таблицы в этом списке разделяются запятой. Для каждой таблицы можно указать имя локального псевдонима, которое должно следовать сразу же за именем таблицы и отделяться от него одним или несколькими пробелами.

В предложении **WHERE** задаются условия отбора записей из исходных таблиц.

**INTO** указывает направление вывода результата.

Предложение **GROUP BY** группирует строки в запросе на основании значения в одном или более полях результирующей таблицы.

Предложение **HAVING** задает условие фильтра, которому должны удовлетворять группы, чтобы быть включенными в результат запроса. Предложение **HAVING** следует использовать вместе с **GROUP BY**.

Предложение **ORDER BY** сортирует результат запроса на основании одного или нескольких полей результирующей таблицы.

Обратите внимание, что каждое предложение в SQL-выражениях необходимо использовать, придерживаясь синтаксического порядка. Например, предложение **GROUP BY** должно идти до **ORDER BY**. Иначе вместо ожидаемого результата вы увидите сообщение об ошибке.

Гибкость и мощь языка **SQL** состоит в том, что он позволяет объединить все операции реляционной алгебры в одной конструкции, "вытаскивая", таким образом, любую требуемую информацию, что очень часто и происходит на практике.

1. **INSERT-запросы**

Для вставки новых данных в конец существующей таблицы используются **INSERT-**запросы. Синтаксис такого запроса имеет вид:

INSERT INTO<Имя таблицы> [(<Список полей>)] VALUES(<Список выражений>)

Так, например, чтобы ввести новую строку в таблицу

**Sklad(Nom\_Num, Name, Price, Kolvo, Postav)**, вы можете использовать следующую инструкцию:

INSERT INTO Sklad VALUES (1001, 'Транзистор КТ315А', 2.50, 100, 'ОАО Электрон')

Если список полей опущен, то в списке значений должны быть перечислены значения всех полей в порядке их создания.

Параметр **<Список полей>** указывает, в каких столбцах следует разместить информацию. Например:

INSERT INTO Sklad (Nom\_Num, Name, Kolvo) VALUES (1001, 'Транзистор КТ315А', 100)

Обратите внимание, что столбцы **Price** и **Postav** отсутствуют. Это значит, что эти строки автоматически установлены в значение – по умолчанию. По умолчанию может быть введено или значение **NULL**, или другое значение, определяемое как по умолчанию. Если ограничение запрещает использование значения **NULL** в данном столбце и этот столбец не установлен как по умолчанию, то он должен быть обеспечен значением.

Если в инструкции **INSERT** указывается столбец, но никакая информация в него не добавляется, а в столбце не допускаются значения **NULL**, то весь процесс записи информации завершается неудачно.

Кроме того, если поле автоинкрементное (автоматически заполняемое), то вы не сможете вставить в него данные. В этом случае запрос на ставку данных будет иметь вид:

INSERT INTO customers (name, city, raiting) VALUES ('Иванов', 'Борисоглебск',55)

Значение поля **id** будет добавлено автоматически.

Вы можете также использовать команду **INSERT**, чтобы получать или выбирать значения из одной таблицы и помещать их в другую. Для этого необходимо в предложение **VALUES** вставить соответствующий запрос. Например:

INSERT INTO WladimirCity (name, raiting)

VALUES (SELECT name, raiting

FROM Sales\_people

WHERE city = 'Владимир')

Здесь выбираются все значения, произведенные запросом, т. е. все строки из таблицы "Продавцы" со значениями city = 'Владимир', и помещаются в таблицу, называемую WladimirCity. Чтобы это работало, таблица WladimirCity должна отвечать следующим условиям:

• Она должна уже быть создана командой CREATE TABLE.

• Она должна иметь два столбца, которые совпадают с таблицей ''Продавцы'' в терминах типа данных, т. е. соответствующие столбцы обеих таблиц должны иметь одинаковый тип данных (но не должны иметь одинаковых имен).

**3. DELETE-запросы**

Запросы на удаление данных могут реализовать удаление как одной записи, так и нескольких записей, удовлетворяющих данному условию. Синтаксис запроса имеет следующий вид:

DELETE FROM <Имя таблицы>

[WHERE <Условие>]

Например,

DELETE FROM Sales\_people

WHERE id = 1003;

Если условие не указано, то будут удалены все записи. Например:

DELETE FROM Sales\_people

Следует учитывать, что SQL Server не удаляет записи, а лишь делает их невидимыми для обработки. Этот факт является одним из явных недостатков SQL Server, потому что могут возникнуть проблемы двух типов:

1. Постепенно таблица разрастается до больших размеров.

2. Если в удаленной записи есть автоинкрементное поле, то новая запись будет иметь значение на единицу большее, чем в поле удаленной записи.

**4. UPDATE-запросы**

С целью обновления записей таблицы используются **UPDATE**-запросы. Синтаксис такого запроса имеет следующий вид:

UPDATE <Имя таблицы>

SET Столбец1 = Выражение1

[, Столбец2 = Выражение2,…]

[WHERE <Условие>]

Например:

UPDATE Sales\_people

SET raiting = 15

WHERE name = 'Борисов Л.П' AND city = 'Владимир'

Если условие отсутствует, то будут модифицированы все записи.

**5. Подзапросы и структурированные запросы**

**Подзапрос** (subquery) – дополнительный метод манипулирования несколькими таблицами, проще говоря, это инструкция **SELECT**, вложенная:

– в предложение WHERE, HAVING или SELECT другой инструкции SELECT;

– инструкцию INSERT, UPDATE ИЛИ DELETE;

– другой подзапрос.

Именно возможность вложения инструкций SQL друг в друга является причиной, по которой SQL получил свое название – Structured Query Language (структурированный язык запросов). Каждая включающая инструкция – следующий по старшинству уровень в подзапросе – представляет собой внешний уровень для внутреннего подзапроса.

Достаточно упрощенный синтаксис подзапроса иллюстрирует вложение подзапроса в инструкции SELECT:

SELECT [DISTINCT] <список выбора>

FROM <список\_таблиц>

WHERE

<выражение> [[NОТ] IN | оператор\_сравнения [ANY ALL] |

[NOT] EXISTS]

( SELECT [DISTINCT] <список\_выбора\_подзапроса>

FROM <список\_таблиц>

WHERE <условия>)

[GROUP BY <список\_группировки>]

[HAVING <условия>]

[ORDER BY <порядок сортировки>]

В зависимости от элементов в предложении WHERE внешнего запроса существует три типа подзапросов:

– не возвращающие ни одного или возвращающие несколько элементов. Они начинаются с предложения IN или оператора сравнения и содержат ключевые слова ANY или ALL.

– возвращающие единственное значение. Они начинаются с простого оператора сравнения.

– представляющие собой проверку (тест) на существование. Такие подзапросы начинаются с ключевого слова EXISTS.

**Порядок выполнения работы**

Откройте SQL Server Management Studio и выберите базу данных, созданную в предыдущей лабораторной работе.

1. Напишите команду SELECT, которая бы вывела номер заказа, сумму и дату для всех строк из таблицы "Заказы".

2. Напишите запрос, который вывел бы все строки из таблицы "Заказчики", для которых номер продавца = 1001.

3. Напишите запрос, который вывел бы коды всех продавцов в текущем порядке из таблицы "Заказы" без каких бы то ни было повторений.

4. Напишите запрос к таблице"Заказчики", который содержит всех заказчиков с рейтингом =< 100, если они не находятся во Владимире.

5. Напишите запрос, который мог бы вывести все заказы на 3 или 4 апреля 2014 г.

6. Напишите запрос, который может вывести всех заказчиков, чьи имена начинаются с буквы, попадающей в диапазон от A до П.

7. Напишите запрос, который выбрал бы наименьшую сумму для каждого заказчика.

8. Напишите запрос, который выбрал бы высший рейтинг в каждом городе.

9. Напишите запрос, который выводил бы список заказчиков в нисходящем порядке. Вывод поля рейтинга должен сопровождаться именем заказчика и его номером.

10. Напишите запрос, который бы выдавал имена продавца и заказчика для каждого заказа после номера заказа.

11. Напишите запрос, который бы выводил всех заказчиков, обслуживаемых продавцом с комиссионными выше 12 %. Выведите имя заказчика, имя продавца и ставку его комиссионных.

12. Напишите запрос, который вычислил бы сумму комиссионных продавца для каждого заказа заказчика с рейтингом выше 100.

13. Напишите запрос, который бы вывел все пары продавцов, живущих в одном и том же городе. Исключите комбинации продавцов с ними же, а также дубликаты строк, выводимых в обратном порядке.

14. Напишите запрос с подзапросом, который вывел бы фамилии и рейтинги всех заказчиков, у которых имеют усредненные порядки.

15. Напишите запрос, который бы выбрал общую сумму всех заказов для каждого продавца, у которого эта общая сумма больше, чем сумма наибольшего заказа в таблице "Заказы".

16. Напишите команду SELECT, использующую подзапрос, которая выберет фамилии и коды всех заказчиков с максимальными для их городов рейтингами.

17. Напишите два запроса, которые выберут всех продавцов (по фамилии и коду), имеющих в своих городах заказчиков, которых они не обслуживают. Один запрос – с использованием объединения и один – с подзапросом. (Подсказка: один из способов это сделать состоит в том, чтобы найти всех заказчиков, не обслуживаемых данным продавцом, и определить, находится ли каждый из них в городе продавца.)

18. Напишите запрос, который бы использовал оператор EXISTS для извлечения всех продавцов, которые имеют заказчиков с рейтингом 300.

19. Напишите запрос, использующий оператор EXISTS, который выберет всех продавцов с заказчиками, размещенными в их городах, которые ими не обслуживаются.

20. Напишите запрос, который бы выбирал всех заказчиков, чьи рейтинги равны или больше, чем любой (ANY) рейтинг заказчика.

21. Напишите запрос, использующий ANY или ALL, который бы находил всех продавцов, которые не имеют никаких заказчиков, размещенных в их городе.

22. Напишите команду, которая бы удаляла всех заказчиков, не имеющих текущих порядков.

23. Напишите команду, которая бы увеличила на двадцать процентов комиссионные всех продавцов, имеющих общие текущие порядки выше, чем 3000.

24. Напишите команду, которая бы удалила все заказы заданного заказчика из таблицы "Заказы".

25. Напишите команду, которая бы увеличила рейтинг всех заказчиков в Москве на 100.

26. Продавец X оставил компанию. Переназначьте его заказчиков продавцу Y.

**Содержание отчета**

1. Цель работы.

2. Описание схемы базы данных

3. Описание запросов и результатов запросов.

**Контрольные вопросы**

1. Дайте общую характеристику языку запросов Transact-SQL.

2. Запишите базовую конструкцию команды SELECT.

3. Какие операторы используются в предложении WHERE для отбора записей?

4. Дайте определение подзапроса.

5. Какие типы подзапросов вы знаете?

6. Укажите правила формирования запросов.

**Лабораторная работа №3**

**ПРЕДСТАВЛЕНИЯ, ХРАНИМЫЕ ПРОЦЕДУРЫ И ТРИГГЕРЫ**

***Цель работы:*** практическое освоение методов создания и использования в MS SQL SERVER представлений, хранимых процедур и триггеров.

**1. Представления**

Типы таблиц, с которыми вы имели дело до сих пор, назывались **базовыми таблицами**. Это таблицы, которые содержат данные. Однако имеется другой вид таблиц – представления. **Представление** – это "виртуальная таблица", содержимое которой определяется запросом. Представление позволяет пользователю увидеть результаты сохраненного запроса, а SQL обеспечивает доступ к этим результатам таким образом, как если бы они были простой таблицей.

С технической точки зрения представление – это хранимое в базе данных определение инструкции SELECT с заданными в ней строками и столбцами, которые при последующем вызове вида должны считываться.

Вы можете использовать представление точно так же, как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена из и соединена с другими таблицами и представлениями.

Преимущество использования представления по сравнению с основной таблицы в том, что представление будет модифицировано автоматически всякий раз, когда таблица, лежащая в его основе, изменяется.

Представления значительно расширяют управление вашими данными. Это превосходный способ дать публичный доступ к некоторой, но не всей информации в таблице.

Если раскрыть вкладку **Представления** в программе **Management Studio**, то откроются все имеющиеся в используемой базе данных представления (рис. 21).

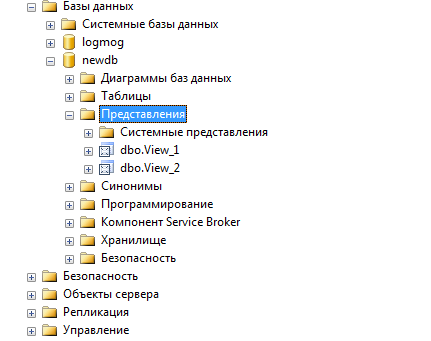


Рисунок 21

Для того чтобы просмотреть содержимое представления, щелкните правой кнопкой мыши на интересующем вас представлении и в появившемся контекстном меню выберите команду **Изменить первые 200 строк** или команду **Выбрать первые 1000 строк**.

Если щелкнуть правой кнопкой на представлении, а затем выбрать команду **Проект**, то откроется окно редактирования (рис. 22).

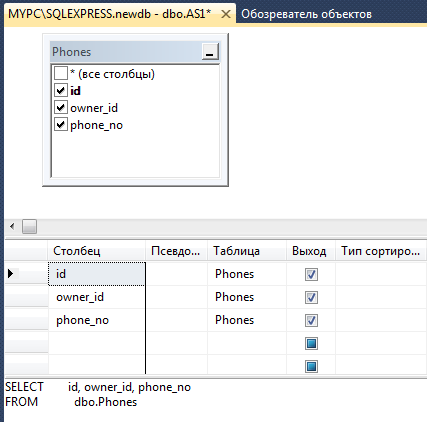


Рисунок 22

При нажатии кнопки **Проверить синтаксис SQL** на панели инструментов происходит проверка синтаксиса написанного представления.

Для создания нового представления используется команда **CREATE VIEW**:

CREATE VIEW имя\_представления

AS

*Инструкция* SELECT

Например, следующее представление с именем **Predstav** содержит все записи таблицы **customers**:

CREATE VIEW Predstav

AS

SELECT \*

FROM customers

Перейдем к рассмотрению вопроса создания представлений с помощью средств **SQL Server Management Studio**. Для этого выполните следующие действия: внутри текущей базы данных выделите папку **Представления** и в контекстном меню этой папки выберите пункт **Создать представление**. Откроется новая вкладка и окно (рис. 23) для создания нового представления.

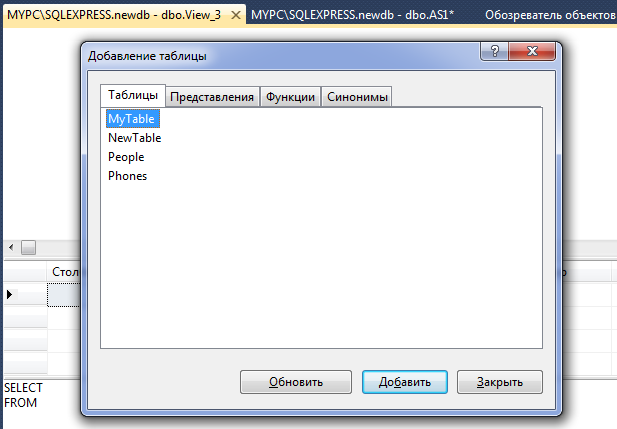


Рисунок 23

Для создания представления можно воспользоваться либо уже имеющейся инструкцией и просто ввести ее в окно SQL (оно открыто и подготовлено двумя предложениями SELECT и FROM), а также можно создать представление без использования команд SQL. Для этого необходимо выполнить следующие действия:

1. В окне **Добавление таблицы** поочередно добавить необходимые таблицы или другие представления, столбцы которых будут использовать для создания нового представления. Закрыть окно.

2. В появившейся диаграмме таблиц выбрать необходимые столбцы, которые должны быть включены в представление. При этом столбцы будут автоматически отображаться в сетке, а также будет изменяться SQL код в нижней части вкладки.

3. В сетке укажите порядок и тип сортировки, а также критерии отбора для нужных полей.

4. Если в столбце **Выход** сбросить в соответствующей строке автоматически появляющийся флажок, то этот столбец не будет отображаться в результирующем наборе.

5. Для сохранения представления нажмите на панели инструментов кнопку **Сохранить**. Если представление не может быть создано, появится сообщение об ошибке. Это произойдет, например, в случае, когда уже существует представление с таким именем. В появляющемся диалоговом окне **Сохранить как** введите имя созданного представления и нажмите кнопку ОК.

Если вы не хотите сохранять представление до того, как убедитесь еще раз, что оно выдает именно то, что вам нужно, можно предварительно нажать кнопку **Выполнить код SQL** (с восклицательным знаком) на панели инструментов, чтобы увидеть результаты работы представления.

Для считывания определенных в представлении столбцов и строк можно обратиться к нему как к таблице с помощью самой простой инструкции SELECT. Например, чтобы показать столбцы и строки, определенные в предыдущем представлении, можно использовать следующую инструкцию:

SELECT \*

FROM Predstav

Посредством представлений можно изменять данные. Представление разрешается обновлять в том случае, если система управления базами данных может для каждой строки представления найти исходную строку в исходной таблице, а для каждого обновляемого столбца представления – исходный столбец в исходной таблице. Если представление соответствует этим требованиям, то над ним и над исходной таблицей можно выполнять имеющие смысл операции вставки, обновления и удаления.

Добавление строк в таблицы, на которых представление определено, делается с помощью инструкции INSERT, в которой указано представление. Добавить строку гораздо проще, если представление ссылается на все столбцы таблицы.

С помощью представлений можно удалять строки, даже если в этом представлении происходит обращение не ко всем столбцам таблицы. Для этого используется команда DELETE.

Инструкция UPDATE позволяет посредством представления изменить один или несколько столбцов или строк. Любые изменения, сделанные с помощью представления, выполняются в таблице, указанной в определении представления.

Удаляется представление командой

**DROP VIEW** имя представления.

1. **Хранимые процедуры SQL Server**

**Хранимая процедура** – это набор скомпилированных команд SQL, доступных напрямую SQL - серверу. Хранимые процедуры находятся на сервере СУБД и используются, когда необходимо часто выполнять повторяющиеся в определенном порядке запросы к серверу. Команды размещаются в хранимой процедуре и выполняются как одно целое или подпрограмма по аналогии с другими языками программирования. Расширенные хранимые процедуры – это разновидность обычных хранимых процедур, но в них можно использовать обращения к подпрограммам, написанным на языке С или С++, что позволяет расширить возможности SQL. Обычно расширенные хранимые процедуры представляются в виде динамических библиотек. Имена этих процедур, как правило, начинаются с префикса **xp** (eXtended Procedure).

В хранимые процедуры можно передавать значения и получать из них значения. Среди достоинств использования хранимых процедур можно выделить увеличение скорости выполнения операций по причине их работы на достаточно мощном сервере. Другим достоинством является то, что хранимые процедуры дают большие возможности при реализации sql-запросов, которые нужно выполнить после проверки некоторого условия или при проверке выполнения условий целостности данных. В SQL Server имеются **системные хранимые процедуры**, размещаемые в базе данных **master**, пользовательские размещаются в базах данных пользователей.

Для создания хранимой процедуры выберите необходимую базу данных и раскройте закладку **Программирование**, затем **Хранимые процедуры**. Перед вами отобразятся хранимые процедуры выбранной базы данных (рис.24).

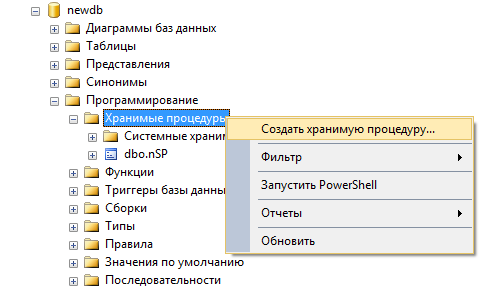


Рисунок 24

Щелкните правой кнопкой мыши на вкладке и нажмите **Создать хранимую процедуру** (рис. 25).

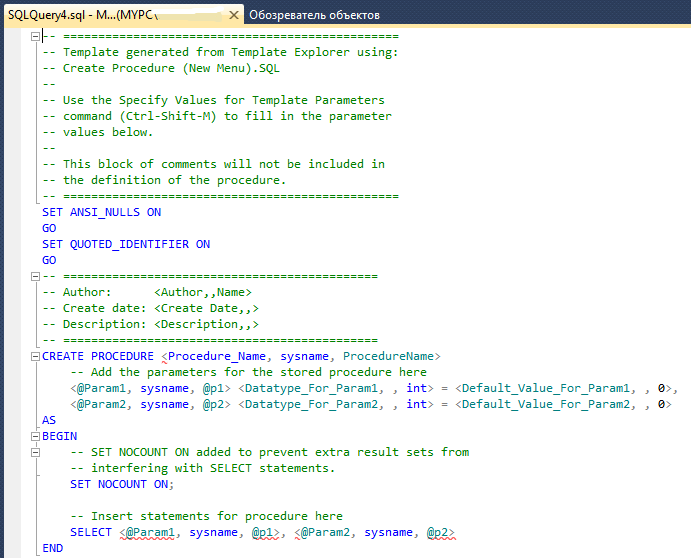


Рисунок 25

В данном окне можно набирать текст хранимой процедуры. Неполный синтаксис хранимой процедуры имеет вид:

CREATE PROCEDURE имя процедуры

[(@имя переменной1 тип, …@имя переменнойN тип)]

As

*Инструкции* SQL

С появлением хранимых процедур язык SQL приобрел ряд дополнительных возможностей, которые присущи всем языкам программирования:

• реализация условного перехода:

IF логическое выражение

*Инструкция* SQL

[ELSE

*Инструкция* SQL]

• реализация цикла:

WHELE логическое выражение

*Инструкция* SQL

• реализация блока инструкций:

BEGIN

*Инструкции* SQL

END

• использование переменных: в **Transact-SQL** они могут быть локальными или глобальными. Обращение к локальным переменным всегда происходит с помощью символа @, предваряющего их имена. Локальные переменные определяются с помощью инструкции **DECLARE**, где указывается тип данных переменной:

DECLARE @имя переменной тип данных

Для определения значений локальных переменных применяется инструкция SELECT, ее синтаксис:

SELECT @имя\_переменной = выражение | *инструкция* SELECT

Глобальные переменные определены на уровне сервера, поэтому они не определяются в вашей программе. При обращении к глобальной переменной нужно предварять ее имя символами @@. С помощью глобальных переменных можно получить доступ к информации сервера и информации о выполняемых операциях. Например, глобальная переменная @@ERROR возвращает номер последней системной ошибки (0 – ошибки нет), а переменная @@ROWCOUNT – количество строк, вовлеченных в последний запрос;

• использование инструкции **PRINT**: она может быть использована для вывода текста или значений переменных типов char, varchar или глобальной переменной @@VERSION. (Глобальная переменная @@VERSION содержит информацию о дате и версии SQL Server).

Имеются и другие инструкции, с которыми можно ознакомиться в соответствующей литературе [1].

Рассмотрим пример использования хранимой процедуры.

CREATE PROCEDURE nSP

(@id integer, @n varchar(20))

AS

if @id<10

begin

select \* from People

end

else

select \* from People where name=@n

GO

Если передаваемый процедуре параметр @id<10, то выводятся все поля таблицы People. В противном случае выводятся все поля, где имя равно параметру @n.

Нажатием кнопки **Синтаксический анализ** (в виде галочки) (рис. 26) можно проверить синтаксис хранимой процедуры.



Рисунок 26

Для запуска процедуры необходимо написать SQL запрос. Синтаксис вызова процедуры имеет следующий вид:

Имя процедуры [Параметр1, Параметр2, ...ПараметрN]

или

EXEC Имя процедуры [Параметр1, Параметр2, ...ПараметрN]

Например, при вызове процедуры в виде

nSP 11, 'Петров'

будет получен результат, представленный на рис. 27.

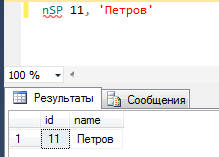


Рисунок 27

При другом типе вызова

EXEC nSP 9,’sdfvsdv’

будут получены результаты, приведенные на рис. 28.

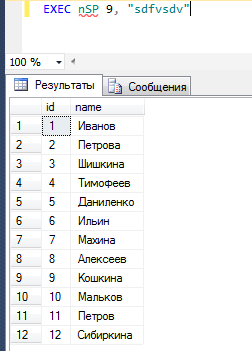


Рисунок 28

Для удаления хранимой процедуры используется оператор

**DROP PROCEDURE** имя процедуры.

1. **Триггеры**

**Триггеры** – это вид системной процедуры, автоматически осуществляемый при выполнении каких-либо действий с таблицами базы данных. Это необходимо, например, для реализации сложных условий проверки целостности данных. Неполный синтаксис текста команды, создающей триггер, выглядит следующим образом:

CREATE TRIGGER имя триггера

ON имя таблицы

FOR [DELETE] [,] [INSERT] [,] [UPDATE]

AS

*Инструкции* SQL

Триггеры пишутся c помощью SQL запросов.

Специально для триггеров **Transact-SQL** определяет две виртуальных таблицы, структура которых идентична структуре таблицы, с которой связан триггер. Эти таблицы называются **DELETED** и **INSERTED**. Они заполняются строками из модифицируемой таблицы, причем их конкретное содержимое зависит от выполняемой операции:

• Удаление (DELETE) – все строки, удаленные из связанной таблицы, помещаются в таблицу **DELETED**, таблица **INSERTED** пуста.

• Вставка (INSERT) – все строки, добавленные в связанную таблицу, помещаются в таблицу **INSERTED**, таблица **DELETED** пуста.

• Обновление (UPDATE) – для каждой строки связанной таблицы, измененной инструкцией **UPDATE**, ее исходная версия помещается в таблицу **DELETED**, а новая версия – в таблицу **INSERTED**.

К этим двум виртуальным таблицам можно обращаться из тела триггера, и их данные можно использовать в триггере наряду с данными всех остальных таблиц.

Удаляется триггер командой

**DROP TRIGGER** имя триггера.

1. **Курсоры**

Очень часто требуется просмотреть или обработать не весь результат запроса, а каждую или некоторые из его строк. Для этого используются курсоры. Они позволяют просматривать содержимое строк, передавать значения его полей в переменные. Сам курсор представляет собой временную таблицу и может быть уничтожен при его закрытии. Для работы с курсором следует пройти следующие шаги:

1. Объявить курсор. Курсор объявляется командой

DECLARE имя курсора CURSOR FOR (SELECT-запрос).

2. Для работы с содержимым курсора его нужно открыть командой

OPEN имя курсора.

3. Переход по записям курсора осуществляется командой **FETCH** [ориентация] FROM имя курсора INTO имя переменная. В качестве параметра [ориентация] используется значение **NEXT**, т. е. построчный переход по записям.

Курсоры создаются с помощью SQL запросов. Например, создадим курсор, содержащий все записи таблицы **People** (рис. 29).

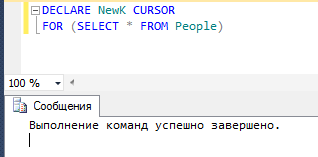


Рисунок 29

Теперь открываем созданный курсор командой **OPEN** NewK, которую необходимо вводить в той же вкладке, в которой мы его объявили. Оператор, приведенный на рис. 30, также вводится в той же вкладке и производит последовательный отбор записей из курсора.

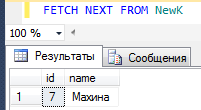


Рисунок 30

После завершения работы с курсором его закрывают командой

**CLOSE** имя курсора.

**Порядок выполнения работы**

1. Откройте SQL Server Management Studio и выберите базу данных, созданную в предыдущей лабораторной работе.

2. Создайте представление, которое бы показывало всех заказчиков, имеющих самые высокие оценки.

3. Создайте представление, которое бы показывало номер продавца в каждом городе.

4. Создайте представление, которое бы показывало каждого продавца с многочисленными заказчиками.

5. Создайте представление, которое показывало фамилии продавцов, не выполняющих план продаж.

6. Создайте представление, которое бы показывало всех заказчиков из указанного города с рейтингом больше 100.

7. Создайте представление, которое бы показывало фамилии продавцов и их общую сумму комиссионных.

8. Создайте представление, которое бы показывало сумму заказов по каждому городу заказчика.

9. Создайте представление таблицы "Продавцы" с именем Commissions (Комиссионные). Это представление должно включать только поля Код\_продавца, ФИО\_продавца и Комиссион-ные\_продавца. С помощью этого представления можно будет вводить или изменять комиссионные, но только для значений между 0,1 и 0,2.

10. Напишите хранимую процедуру, которая увеличивает комиссионные у тех продавцов, сумма продаж которых превышает среднюю сумму продаж всей компании.

11. Напишите хранимую процедуру с параметрами, показывающую фамилии и сумму продаж продавцов, работающих под руководством указанного начальника.

12. Напишите хранимую процедуру, которая при выполнении заказа автоматически изменяла количество товара в таблице "Товар".

13. Напишите триггер, который при добавлении или обновлении таблицы "Заказчики" проверял условие: значение рейтинга должно быть положительным и не превосходить 500.

14. Напишите триггер, который при добавлении или обновлении заказа в кредит сравнивал итоговую сумму заказа с максимально допустимым кредитом для данного заказчика.

15. Напишите триггер, который запрещает удалять запись из таблицы "Заказы", если с ней связаны записи из других таблиц.

**Содержание отчета**

1. Цель работы.

2. Описание схемы базы данных.

3. Описание представлений и результатов выполнения представлений.

4. Описание хранимых процедур и результатов выполнения хранимых процедур.

5. Описание триггеров и результатов выполнения триггеров.

6. Описание курсоров и результатов выполнения курсоров.

**Контрольные вопросы и задания**

1. С какой целью в базах данных используются представления?

2. Как создаются представления в SQL Server?

3. Какие операции можно выполнять при работе с представлениями?

4. С какой целью в базах данных используются хранимые процедуры?

5. Поясните, в чем состоит достоинство применения хранимых процедур?

6. Как осуществляется создание, вызов и удаление хранимых процедур?

7. Что представляют собой триггеры, как они создаются?

8. Что представляют собой курсоры, как они создаются?

9. Как осуществляется работа с курсорами в SQL Server?

**Лабораторная работа №4**

**АДМИНИСТРИРОВАНИЕ И ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ**

**БАЗ ДАННЫХ В MS SQL SERVER**

***Цель работы:*** практическое освоение методов администрирования и обеспечения безопасности баз данных в MS SQL SERVER.

1. **Резервирование баз данных в MS SQL Server**

Одной из важнейших задач при использовании и работы с базами данных является резервное сохранение баз данных на постоянных и съемных носителях информации. В системе Microsoft SQL Server имеются возможности резервирования баз данных (например, с целью сохранения базы данных от повреждений) и восстановление (с целью восстановления поврежденной базы данных). Эти возможности реализуются через основное меню программы Management Studio или через SQL-запрос. Резервирование баз данных осуществляется только в рабочие папки сервера. Таким образом, восстановление может осуществляться из файлов, размещенных только на сервере.

В программе Management Studio эта задача решается с использованием функции **Резервное копирование базы данных**. Операция резервирования может выполняться при работе с текущей базой данных. Щелкните правой кнопкой мыши на используемой базе данных и выберите команду «Задачи – Создать резервную копию».

После этого открывается окно **Резервное копирование базы данных**. Возможно создание двух видов копий резервных файлов:

1. **Полная копия базы данных** (полное резервирование – database complete) – при этом сохраняется вся база данных и все журналы транзакций.

2. **Разностная копия** (database-differential) – сохраняются изменения в базе данных и незавершенные журналы транзакций.

Выбор типов резервирования осуществляется в поле **Тип резервной копии** при помощи переключателя (рис. 31).

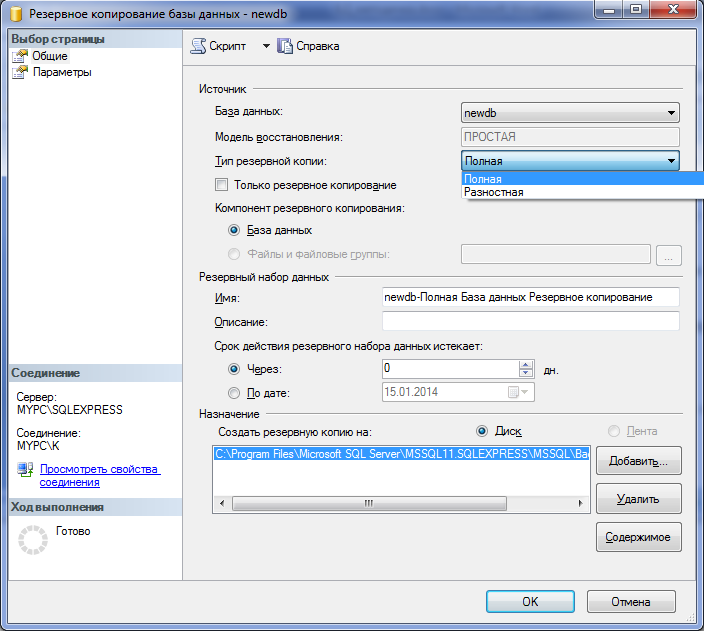


Рисунок 31

В разделе **Назначение** (см. рис. 31) после нажатия кнопки **Добавить** открывается окно **Выбор места расположения резервной копии** (рис. 32).

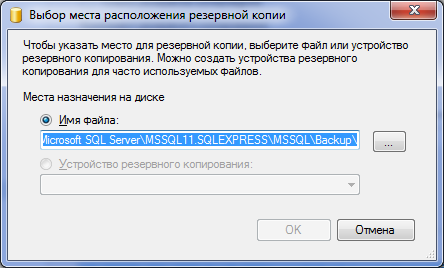


Рисунок 32

Возможно резервирование в файл (поле **Имя файла**) или в созданное ранее устройство резервирования. **Устройство резервирования копирования** является файлом, предназначенным для сохранения в него баз данных. Оно создается пользователем и хранится в базе данных **master** в таблице **sysdevices**.

Например, резервируем базу данных в файл D:\newdb.new (путь набирается "вручную" в поле **Имя файла**), выбирая создание полной копии базы данных. Нажимаем кнопку OK. После этого происходит процесс резервирования. После завершения резервирования появляется сообщение об успешном завершении процесса.

Создание резервной копии возможно с использованием SQL-запроса, который можно создать в **Management Studio**. Запрос, создающий приведенную в примере резервную копию, имеет следующий

вид:

USE master

BACKUP DATABASE newdb to disk='d:\newdb.new'.

1. **Восстановление баз данных**

Очень часто при аварийных сбоях в технических или программных средствах базам данных может быть причинен значительный вред. При этом при наличии резервных копий позволяет восстанавливать базы данных (восстановление также возможно и при неповрежденной базе данных). Щелкните правой кнопкой мыши на используемой базе данных и выберите команду «Задачи – Восстановить – База данных».

Открывается окно **Восстановление базы данных** (рис. 33). В окне указывается название базы данных, а также имя последней по времени созданной пользователем резервной копии.

Далее нажатием кнопки **OK** утверждаем ранее выбранный файл резервной копии и запускаем процесс восстановления.

После завершения процесса восстановления появляется окно об успешном окончании процесса (Восстановление базы данных “newdb” успешно завершено).

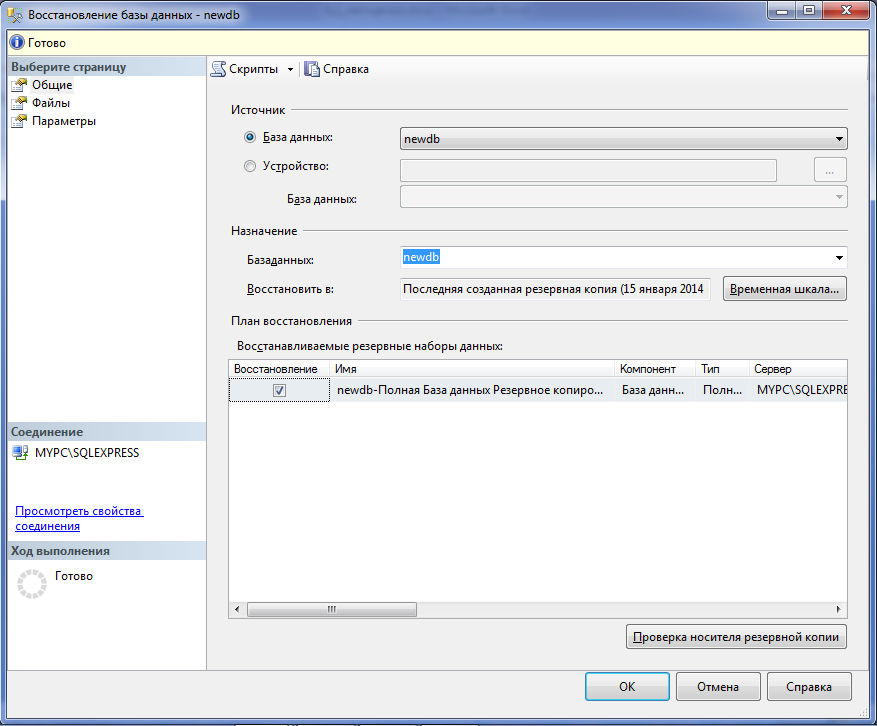


Рисунок 33

Если база данных удалена, то устанавливаем переключатель на **Устройство.**

После нажатия кнопки рядом со строкой с именем устройства (выглядит как троеточие) появляется окно выбора устройства резервирования. Нажимаем кнопку **Добавить** и выбираем файл с копией базы данных (рис. 34). Имя также можно прописать "вручную".

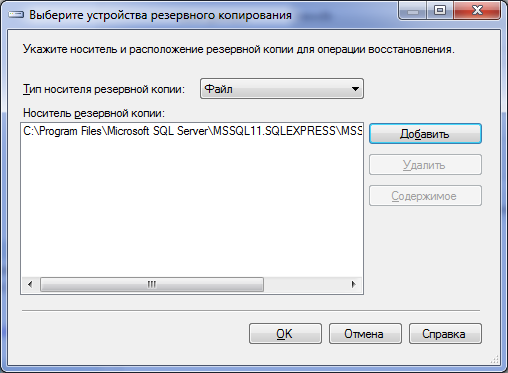


Рисунок 34

При нажатии в левом меню пункта **Параметры** при активном переключателе можно задать параметры восстановления, такие как RESTORE WITH RECOVERY и др.

Возможно восстановление базы данных с использованием SQL-запроса при условии, что в данный момент времени база данных неактивна. Например:

USE master

RESTORE DATABASE newdb FROM DISK=’d:\newdb.new’ WITH replaсe, norecovery.

1. **"Отделение-прикрепление" баз данных**

Возможен случай, когда необходимо перенести базу данных с одного сервера на другой. При работе SQL Server не позволяет копировать базы данных на носители. Эта задача решается использованием команды **DETACH** (**Отсоединить**), которая отсоединяет базу данных от сервера и позволяет ее перекопировать.

Нажмите правой кнопкой мыши на используемой базе данных и выберите команду «Задачи – Отсоединить». Появляется окно **Отсоединение базы данных** (рис. 35). Нажимаем кнопку OK.

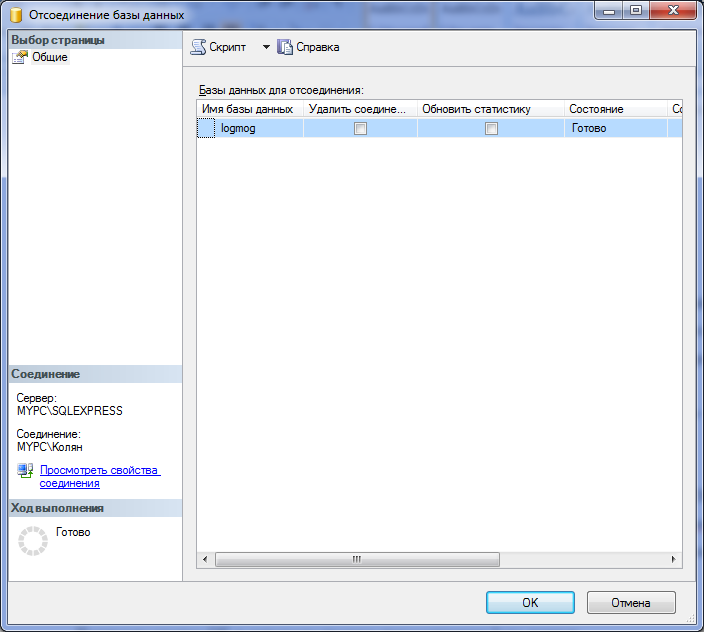


Рисунок 35

Присоединение базы данных к серверу осуществляется с помощью команды **ATTACH** (**Присоединение**) (команда «**Присоединить**» в контекстном меню). Появляется окно «Присоединение баз данных» (рис. 36). В нем указывается путь к расположению файлов базы данных\*.mdf (журнал транзакций \*.ldf добавляется автоматически).

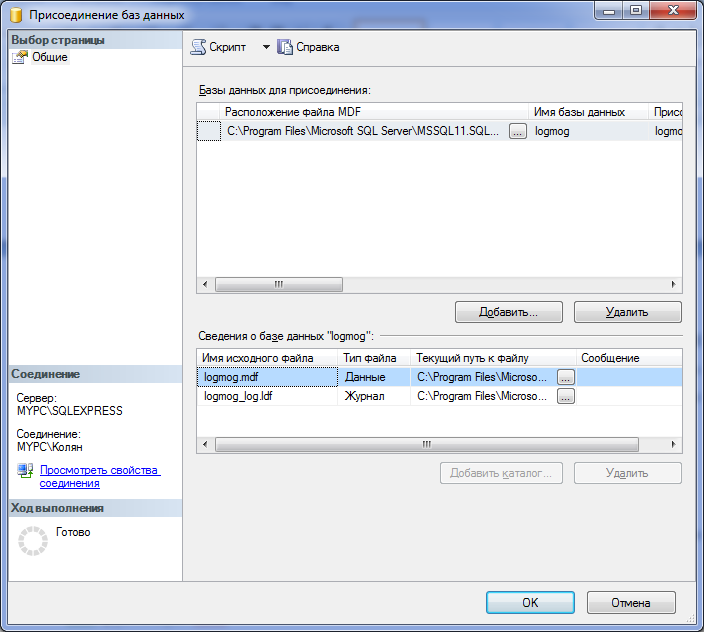


Рисунок 36

Нажимаем кнопку OK и запускаем процесс присоединения базы данных.

Операции **ATTACH** и **DETACH** можно реализовать и с использованием SQL-запросов. Для операции **DETACH** запрос имеет вид:

Use master

exec sp\_detach\_db ‘mydatabase'

Для операции ATTACH SQL-запрос имеет следующий вид:

use master

sp\_attach\_db @dbname ='mydatabase',

@filename1 = 'c:\downloads\rinok\_Data.MDF',

@filename2 = 'c:\downloads\rinok\_Log.LDF'

1. **Физическое сжатие базы данных**

Базы данных могут разрастаться до размеров в несколько гигабайт. В таких случаях помогает операция физического сжатия базы данных (SHRINK). Щелкните правой кнопкой мыши на используемой базе данных и запустите команду ''Все задачи – Shrink Database''. В окне SHRINK DATABASE (рис. 37) указывается "коэффициент сжатия" базы данных.

Рис. 37

Нажимаем OK. После этого появляется сообщение об успешном окончании процесса SHRINK.

1. **Назначение прав пользователей**

При использовании баз данных в многопользовательском режиме актуальной становится задача определения действий пользователей при работе с данными. Таким образом, в серверах баз данных должна решаться задача по назначению прав пользователей. Этими правами могут быть запрет или разрешение на просмотр, изменение, удаление данных, выполнение процедур. Права пользователей делятся:

– на разрешающие – пользователь использует только назначенные ему права;

– запрещающие (отзывающие) – у пользователя отзываются ранее назначенные ему права.

Разрешение прав пользователей производится через вкладку каждой базы данных Users. Во вкладке отображаются все пользователи базы данных (рис. 38).

Рис. 38

После нажатия на кнопку Permissions в открывшемся диалоговом окне (рис. 39) можно настраивать права пользователей (select, update, delete, insert для таблиц и execute для хранимых процедур).

Рис. 39

При активном отображении таблицы нажатием кнопки Columns можно настраивать права на обновление или просмотр столбцов таблицы (рис. 40).

Рис. 40

Нажатием кнопки OK права пользователей утверждаются.

Настройка прав пользователей возможна с использованием SQL-запросов.

Инструкция GRANT используется для задания прав и имеет следующий синтаксис:

GRANT разрешения

ON объект (таблица, хранимая процедура и др.)

TO пользователи

Инструкция REVOKE отзывает полномочия у пользователей. Синтаксис этой инструкции следующий:

REVOKE полномочия

ON объект

FROM пользователи.

1. **Создание ролей пользователей**

Для определения одинаковых полномочий для группы пользователей можно создать роль, определяющую набор полномочий и отнести пользователей к этой роли.

Щелкните на вкладке базы данных Roles. Откроется список ролей базы данных. Нажатием правой кнопки мыши в поле создадим новую роль.

В появившемся окне вводится имя роли (рис. 41). Нажатием кнопки Add можно добавить пользователей к роли.

Рис. 41

Создав роль, два раза щелкните по ней. В появившемся окне после нажатия кнопки Permissions можно определить полномочия роли (рис. 42).

Рис. 42

1. **Создание учетной записи пользователя в Microsoft SQL Server**

При установке Microsoft SQL Server может иметь в качестве учетной записи как Windows -аутентификацию, так и создание учетной записи непосредственно в сервере. Рассмотрим этот процесс.

Раскройте свою базу данных и выберите вкладку Users. Щелкните в поле, отображающем всех пользователей. Щелкнув правой кнопкой мыши, можно создать нового пользователя (рис. 43). В поле Login name выберите New и щелкните по нему.

Рис. 43

В поле name введите имя нового пользователя, переключитесь на SQL Server Autentification и наберите пароль (рис. 44). Нажав OK, подтвердите пароль. Создание пользователя закончено.

Созданный таким образом пользователь может быть добавлен к любой базе данных сервера.

Рис. 44

Откройте SQL Query Analyzer, выполните команду File – Connect. В открывшемся окне выберите SQL Server Autentification, введите имя нового пользователя и его пароль. Нажмите OK. Вход в систему осуществлен.

Созданного таким образом пользователя можно добавить к любой базе данных сервера. Возможность создания роли и отнесение пользователя к данной роли помогает при определении полномочий пользователя при работе с базой данных. Добавление пользователя к базе данных осуществляется следующим образом: щелкните правой кнопкой мыши во вкладке Users используемой базы данных и выберите New Database Users. Раскройте поле Login и выберите NewUs (рис. 45). Нажмите OK.

Рис. 45

Теперь перейдите во вкладку Roles и щелкните два раза по созданной ранее роли NewRole. В появившемся окне щелкните кнопу Permissions. В данном окне расставим новые разрешения: запрет на вставку новых записей в определенные таблицы базы данных (двойной щелчок в столбце Insert) (рис. 46). Щелкните OK.

Рис. 46

В окне Database Role Properties в поле User нажмите кнопку Add, осуществляющую добавление пользователя к этой роли (рис. 47). Выберем пользователя базы данных NewUs.

Рис. 47

Выбрав пользователя, щелкните OK в окне Add Role Members, щелкните OK в окне Database Role Properties. Откроем теперь SQL Query Analyzer и войдем в него под учетной записью NewUs. Это реализуется следующим образом. Выполните команду File – Connect. Откроется окно подключения к серверу. В поле Login Name вводим NewUs, в поле Password – пароль учетной записи (рис. 48). Нажимаем OK и ожидаем подключения к серверу.

Рис. 48

Введем теперь запрос на вставку записи в таблицу Customers. Нажав на кнопку выполнения запроса, SQL Server выдаст сообщение о невозможности внесения записи в таблицу пользователем данной учетной записи (рис. 49).

Рис. 49

**Порядок выполнения работы**

1. Создайте резервную копию БД с помощью команд языка SQL.

2. Сохраните файлы вашей БД на каком-либо носителе. Для этого, используя команду Detach Database, отключите вашу базу от сервера.

3. Удалите вашу БД с сервера и восстановите ее двумя способами (используя резервную копию БД с применением команды Attach), используя стандартные функции Enterprise Manager.

4. Проделайте всё указанное в п. 4, используя команды языка SQL.

5. Создайте в вашей БД пользователя, который имеет право только просматривать содержимое таблиц. Проверьте это с помощью Query Analyzer.

6. Создайте пользователя, которому запрещено просматривать данные из таблиц ''Продавцы'' и ''Заказчики'' и удалять данные из таблиц товары и заказы. Проверьте это с помощью запросов, используя Query Analyzer.

**Содержание отчета**

1. Цель работы.

2. Описание процесса создания резервной копии БД.

3. Описание процесса отключения/подключения БД.

4. Описание процесса восстановления БД из резервной копии.

5. Описание процесса, создания пользователей в БД и описание прав пользователей, созданных в вашей БД.

6. Используя SQL запросы, продемонстрировать результаты работы в БД под разными пользователями.

**Контрольные вопросы**

1. Команды SQL для создания резервной копии БД.

2. Команды SQL для создания восстановления БД из резервной копии.

3. Какой командой можно изменить размер БД?

4. Что такое роль в SQL server?

5. Команды для физического удаления данных из БД.

6. Опишите процесс создания нового пользователя в БД.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Тихомиров, Ю. В.Microsoft SQL Server 7.0 / Ю. В. Тихомиров. –

СПб. : БХВ-Санкт-Петербург, 1999. – 720 с. – ISBN 5-8206-0032-0.

2. ЭнциклопедияSQL / сост. Дж. Грофф, П. Вайнберг. – 3-е изд. –

СПб. : Питер, 2003. – 896 с. – ISBN 5-88782-007-2.

3. Хоменко, А. Д. Базы данных/ А. Д. Хоменко, В. М. Цыганков,

М. Г. Мальцев; под ред. А. Д. Хоменко. – СПб. : КОРОНА-принт,

2003. – 672 с. – ISBN 5-7931-0168-3.

ОГЛАВЛЕНИЕ

Лабораторная работа №1. СОЗДАНИЕ БАЗЫ ДАННЫХ

MS SQL SERVER........................................................................ 3

Лабораторная работа №2. МАНИПУЛИРОВАНИЕ ДАННЫМИ

ВMS SQL SERVER .................................................................. 22

Лабораторная работа №3. ПРЕДСТАВЛЕНИЯ, ХРАНИМЫЕ

ПРОЦЕДУРЫ И ТРИГГЕРЫ................................................... 32

Лабораторная работа №4. АДМИНИСТРИРОВАНИЕ

И ОБЕСПЕЧЕНИЕ БЕЗОБАСНОСТИ БАЗ ДАННЫХ

ВMS SQL SERVER .................................................................. 48

Библиографический список.................................................................... 67

68

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

ПО ДИСЦИПЛИНЕ«БАЗЫ ДАННЫХ»

Составители:

ГРАДУСОВ Александр Борисович

ГРАДУСОВ Денис Александрович

ГАЛКИН Анатолий Александрович

Ответственный за выпуск– зав. кафедрой доцент А.А. Галкин

Подписано в печать13.04.07.

Формат60х84/16. Усл. печ. л. 3,95. Тираж100 экз.

Заказ

Издательство

Владимирского государственного университета.

600000, Владимир, ул. Горького, 87.