Практическая работа 15
Использование языка SQL для реализации запросов

Цель занятия: Получить практический опыт построения запросов на языке SQL

Перечень оборудования и программного обеспечения

Персональныйкомпьютер
Microsoft Office (Word, Visio, Access)
Microsoft SQL Server Management Studio

Краткие теоретические сведения

Функциональность скриптов TSQL, которые выполняются на SQL Server, может быть очень сильно расширена при помощи набора хранимых процедур SP_OA (от Ole Automation - SP_OA CREATE, SP_OA METHOD, SP_OA SETPROPERTY, SP_OA GETPROPERTY и остальных). При использовании этих хранимых процедур вы можете во время выполнения скрипта создавать программные объекты (COM-серверы, имеющиеся на вашем компьютере), вызывать их свойства и методы, а затем удалять. Что чаще всего делается при помощи хранимых процедур SP_OA:

- запись результатов выполнения запросов в файл (например, очень удобно выгружать в файл данные в XML), выполнение других дисковых операций;
- запуск внешних программ для загрузки/выгрузки данных (например, пакетов DTS);
- изменение рабочей среды Windows: подключение сетевых дисков и принтеров, создание переменных окружения, изменение параметров в реестре;
- проведение резервного копирования нестандартными способами;
- отслеживание событий операционной системы и приложений на своем и чужих компьютерах;
- работа со службой каталогов (создание/изменение учетных записей пользователей, групп, компьютеров в домене и на локальном компьютере);
- отправка сообщений электронной почты по SMTP (при помощи объектов CDO.Configuration и CDO.Message), что стандартными способами SQL Server делать не умеет;
- использование различных возможностей SQL-DMO (подробнее в соответствующем модуле).

Очень часто также разработчики пишут свои COM-серверы и вызывают их из кода Transact-SQL, таким образом, расширяя возможности SQL Server.

Некоторые из этих возможностей доступны при помощи bat - файлов и просто вызова внешних исполняемых файлов при помощи расширенной хранимой процедуры XP_CMDSHELL, однако при помощи SP_OA работать во многих ситуациях удобнее:

– больше функциональность - из скрипта всегда можно вызвать внешний исполняемый файл;

– можно использовать различные свойства и методы программных объектов - нет необходимости ограничиваться только передачей параметров исполняемому файлу;

– нет альтернативы при необходимости получения дополнительной информации по работе операционной системы - пакетными файлами вернуть эту информацию на SQL Server очень сложно (например, информация о свободном месте на диске, наличии учетной записи/членстве ее в группе, информация о работающих программах и службах и т.п.)

Поскольку синтаксис SP_OACREATE нельзя признать самым удобным для написания сложных конструкций и их отладки, то рекомендуется вначале проверять работоспособность программных объектов, доступность их свойств и методов при помощи более специализированных средств, таких, как PrimalScript, а затем уже переносить в код Transact-SQL.

Рассмотрим работу с SP_OA на простом примере.

Предположим, что нам необходимо записать результаты выполнения запроса к базе данных на SQL Server в файл output.txt в корневом каталоге диска C:. Запрос будет самый обычный -

SELECT companyname FROM northwind.dbo.customers WHERE CustomerID = 'ALKFI'

Если нужно скачать множество значений, то придется использовать цикл с курсорами. Поскольку это сильно усложнит код, а для нашего примера это непринципиально, то мы будем записывать только одно значение.

Пример обычного скрипта Windows, который записывает информацию в текстовый файл на диске:

```
'Создаем объект файловой системы - FSO
Set FSO = CreateObject("Scripting.FileSystemObject")
'Создаем объект Text Stream (у нас он называется oFile) для записи
'текстовых данных в файл. 8 - значит открыть на добавление данных, True -
'создать, если еще нет, -1 - писать в Unicode
Set oFile = FSO.OpenTextFile("C:\output.txt", 8, True, -1)
'вызываем метод Write для записи информации в файл
oFile.Write("Наши данные")
'Удаляем созданные нами программные объекты
Set oFile = Nothing
Set FSO = Nothing
```

А теперь переводим то же самое на язык TSQL:

1) вначале - небольшая подготовка: скачиваем результаты запроса в переменную:

```
DECLARE @Var nvarchar(40)
Select @Var = CompanyName FROM Customers WHERE CustomerId = 'ALFKI'
```

2) затем создаем объект файловой системы и получаем на него ссылку (handle id) в формате Int - то, что пойдет в переменную FSO. Создание объекта производится при помощи хранимой процедуры SP_OACreate.

```
-- Объявляем переменные (сразу, чтобы не забыть)
DECLARE @FSO int
DECLARE @hr int
DECLARE @src varchar(255)
DECLARE @desc varchar(255)
DECLARE @oFile int
 -- Создаем сам объект FSO
EXEC @hr = sp_OACreate 'Scripting.FileSystemObject', @FSO OUT
```

-- и - в соответствии с рекомендациями Microsoft ловим ошибки при помощи --- стандартной конструкции:

```
IF @hr <> 0
BEGIN
 EXEC sp_OAGetErrorInfo @FSO, @src OUT, @desc OUT
 SELECT hr=convert(varbinary(4),@hr), Source=@src, Description=@desc
 RETURN
END
```

3) теперь при помощи метода OpenTextFile объекта FSO получаем ссылку на еще один объект (типа TxtStream), который будет называться oFile. Необходимый инструмент - хранимая процедура SP_OAMethod:

```
EXEC @hr = sp_OAMethod @FSO, 'OpenTextFile', @oFile OUT, 'C:\output.txt', 8 , True, -1
IF @hr <> 0
BEGIN
 EXEC sp_OAGetErrorInfo @FSO, @src OUT, @desc OUT
 SELECT hr=convert(varbinary(4),@hr), Source=@src, Description=@desc
 RETURN
END
```

Обратить внимание на синтаксис процедуры SP_OAMethod:

- первый параметр - всегда указатель на созданный объект в виде локальной целочисленной переменной;
- второй параметр - всегда название вызываемого метода;
- третий параметр - только исходящий, то, что вернет данный метод. Указывается всегда, если есть хотя бы один входящий параметр. Если метод объекта ничего возвращать не собирается, то обязательно нужно указать NULL.

Далее через запятую, в обычном формате указываем остальные входящие и исходящие параметры.

4) далее еще раз используем процедуру SP_OAMethod, но уже для вызова метода Write объекта oFile. Null для исходящего параметра обязателен!

```
EXEC @hr = sp_OAMethod @oFile, 'Write', NULL, @Var
IF @hr <> 0
BEGIN
 EXEC sp_OAGetErrorInfo @FSO, @src OUT, @desc OUT
 SELECT hr=convert(varbinary(4),@hr), Source=@src, Description=@desc
 RETURN
END
```

Осталось в соответствии с правилами хорошего тона убрать за собой мусор из оперативной памяти:

```
EXEC @hr = sp_OADestroy @FSO
 IF @hr <> 0
 BEGIN
 EXEC sp_OAGetErrorInfo @FSO, @src OUT, @desc OUT
 SELECT hr=convert(varbinary(4),@hr), Source=@src, Description=@desc
 RETURN
 END
EXEC @hr = sp_OADestroy @oFile
 IF @hr <> 0
 BEGIN
 EXEC sp_OAGetErrorInfo @oFile, @src OUT, @desc OUT
 SELECT hr=convert(varbinary(4),@hr), Source=@src, Description=@desc
 RETURN
 END
```

Все, результат запроса добавлен в текстовый файл.

Некоторые дополнительные моменты по SP_OA:

- получить значение свойства/изменить значение свойства созданного объекта можно при помощи хранимых процедур SP_OAGetProperty и SP_OASetProperty соответственно. Работа с ними выглядит так же, как и работа с SP_OAMethod, и сложностей не представляет;
- хранимая процедура SP_OAGetErrorInfo позволяет получить информацию об ошибках, которые могут возникнуть при выполнении операций с внешними программными объектами. Рекомендованный Microsoft способ применения был приведен выше;
- для всех процедур SP_OA передавать параметры можно только по позиции, но не по их имени;
- SP_OADestroy можно, в принципе, не вызывать - созданные при помощи SP_OACreate программные объекты автоматически удаляются из памяти при завершении работы скрипта;

– при первом запуске SP_OACreate в оперативной памяти выделяется специальная программная область - так называемая shared OLE Automation stored procedure execution environment, которая будет жить до перезагрузки сервера. Если по каким-то причинам эту память нужно освободить, то можно выполнить хранимую процедуру SP_OAStop. Однако нужно быть осторожным - если это время выполняется любая другая процедура SP_OA, то она вернет ошибку. Память будет снова выделена автоматически при следующем запуске SP_OACreate.
– правами на выполнение любых хранимых процедур SP_OA обладают только системные администраторы SQL Server. Эти процедуры входят в группу "повышенного риска" с точки зрения безопасности и дополнительных прав на них предоставлять пользователям категорически не рекомендуется;
– по умолчанию программный объект работает в контексте учетной записи SQL Server и обладает полными правами на все ресурсы, к которым имеет доступ SQL Server (в том числе выделенной SQL Server оперативной памяти). При вызове SP_OACreate при помощи необязательного параметра можно изменить эти права на более ограничительные.

В БД "Microsoft SQL Server" все диаграммы находятся в папке "Database Diagrams" обозревателя объектов

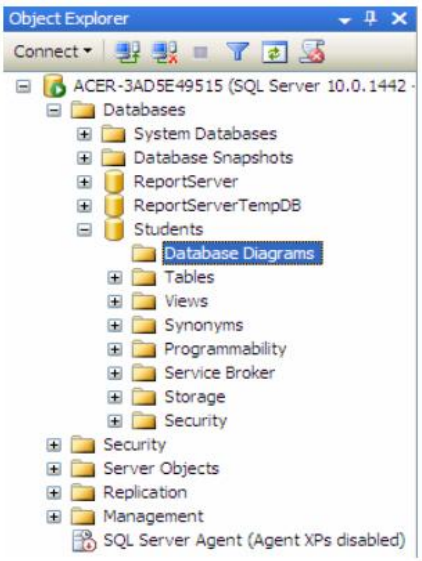


Рис. 1

Создадим диаграмму, обеспечивающую целостность данных БД "Students". Для создания новой диаграммы в БД "Students" щелкните ПКМ по папке "Database Diagrams" и в появившемся меню выберем пункт "New Database Diagram". Сначала появится окно с вопросом о добавлении нового объекта "Диаграмма". В этом окне нужно нажать кнопку "Yes". Затем появится окно "Add Table" предназначенное для добавления таблиц в новую диаграмму.

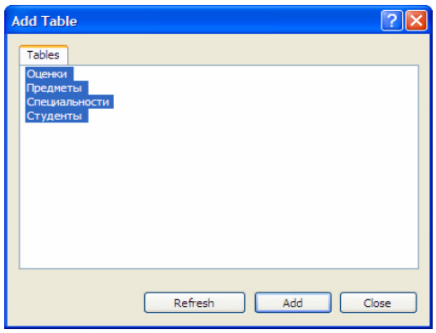


Рис. 2

В окне добавления таблиц выделите все таблицы нашей БД и нажмите кнопку "Add"(рис. 2). Закройте окно "Add Table" нажатием на кнопку "Close".

Появится окно диаграммы, где будут отображены отобранные таблицы. Теперь необходимо определить связи между таблицами. Перетащите поле "Код специальности" из таблицы "Специальности" на такое же поле в таблице "Студенты". Появится окно создания связи между таблицами "Tables and Columns" (рис. 3).

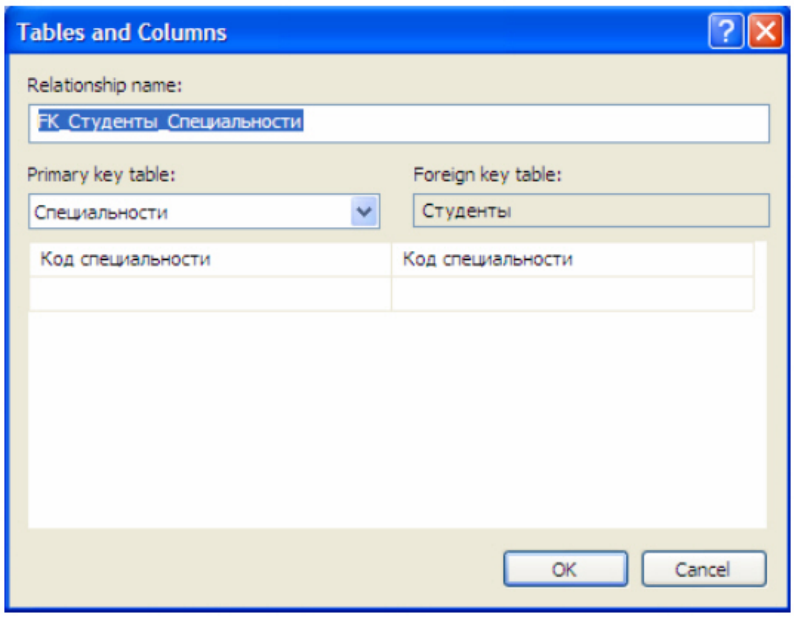


Рис. 3

В окне создания связи нажмите кнопку "Ok". Появится окно настройки свойств связи "Foreign Key Relationship" (рис. 4).

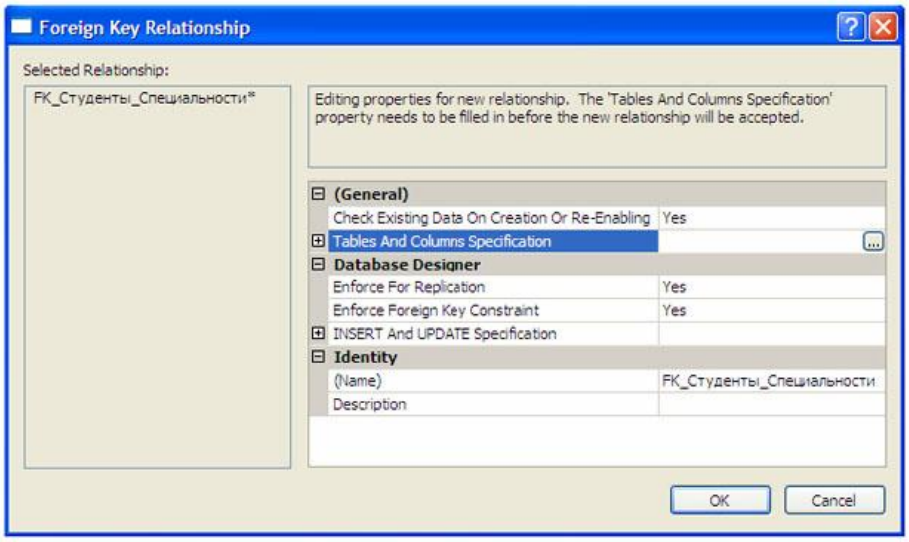


Рис. 4

Оставьте свойства связи без изменений и в окне свойств связи нажмите кнопку "Ok".В диаграмме между таблицами "Студенты" и "Специальности" появится связь в виде ломанной линии (рис. 5).

Аналогичным образом создайте связь таблицы "Студенты" с таблицей "Оценки", перетащив поле "Код студента" из таблицы "Студенты" на одноименное поле в таблице "Оценки". Затем, свяжите таблицы "Предметы" и "Оценки", перетащив поле "Код предмета" из таблицы "Предметы" на поля "Код предмета 1", "Код предмета 2" и "Код предмета 3" таблицы "Оценки". После выполнения вышеперечисленных действий диаграмма примет следующий вид (рис. 5).

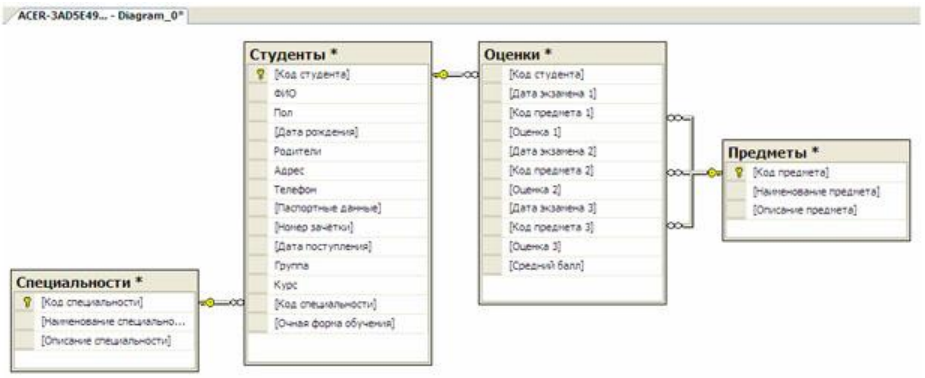


Рис. 5

Закройте окно с диаграммой, щелкнув мышью по кнопке закрытия, расположенной в верхнем правом углу окна с диаграммой. Появится окно с вопросом о сохранении новой диаграммы, где необходимо нажать кнопку "Yes" (рис. 6).

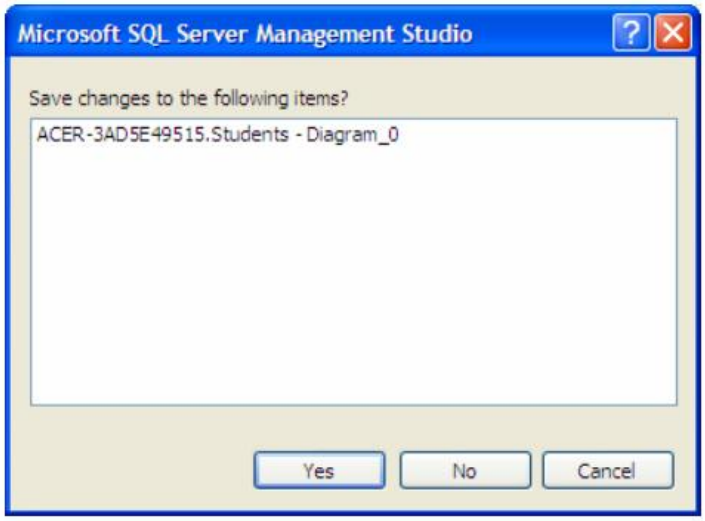


Рис. 6.

Появится окно определения имени новой диаграммы "Choose Name". В окне определения имени, задайте имя диаграммы как "Диаграмма БД Студенты" и нажмите кнопку "Ok" (рис. 7).

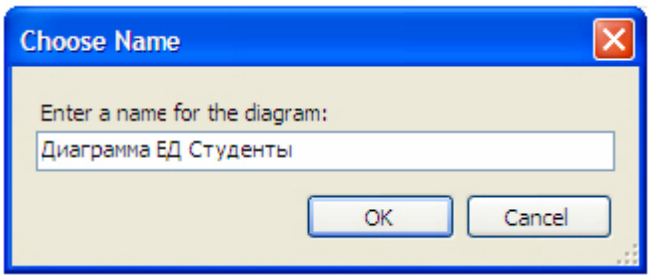


Рис. 7.

Появится окно "Save" с запросом сохранения таблиц, входящих в диаграмму. В данном окне необходимо нажать кнопку "Yes" (рис. 8).

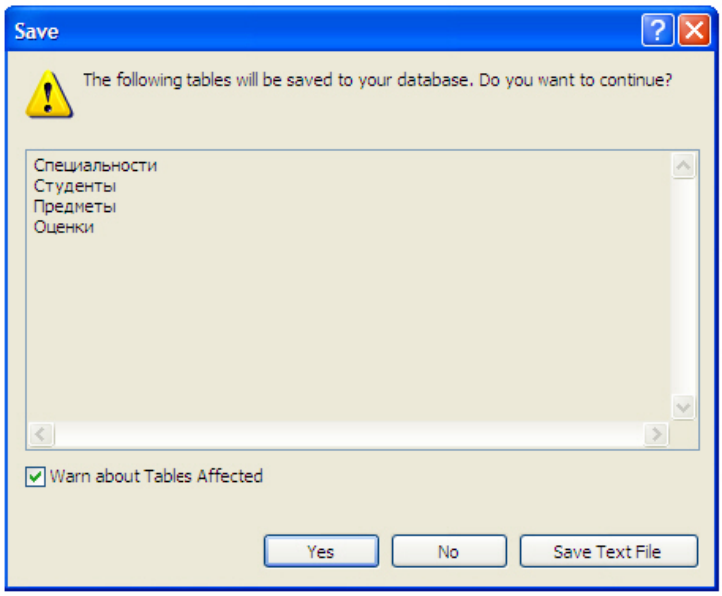


Рис. 8.

Создадим триггеры для таблицы "Студенты". Триггеры создаются отдельно для каждой таблицы и располагаются в обозревателе объектов в папке "Triggers". В нашем случае, папка "Triggers" входит в состав таблицы "Студенты" (рис. 9).

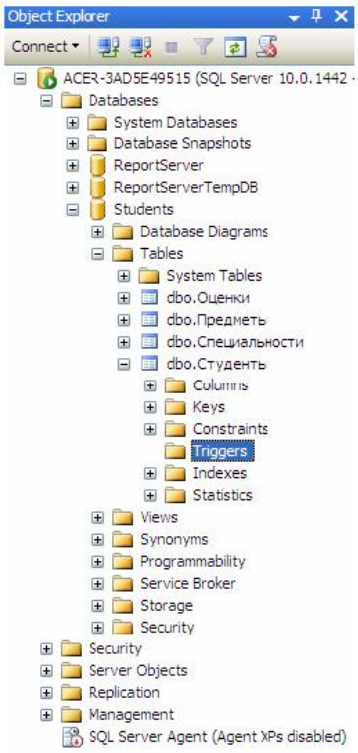


Рис. 9.

Для начала создадим триггер, выводящий сообщение "Запись добавлена" при добавлении записи в таблицу "Студенты". Создадим новый триггер, щелкнув ПКМ по папке "Triggers" в таблице "Студенты" и в появившемся меню выбрав пункт "New Trigger". Появится следующее окно с новым триггером (рис. 10):

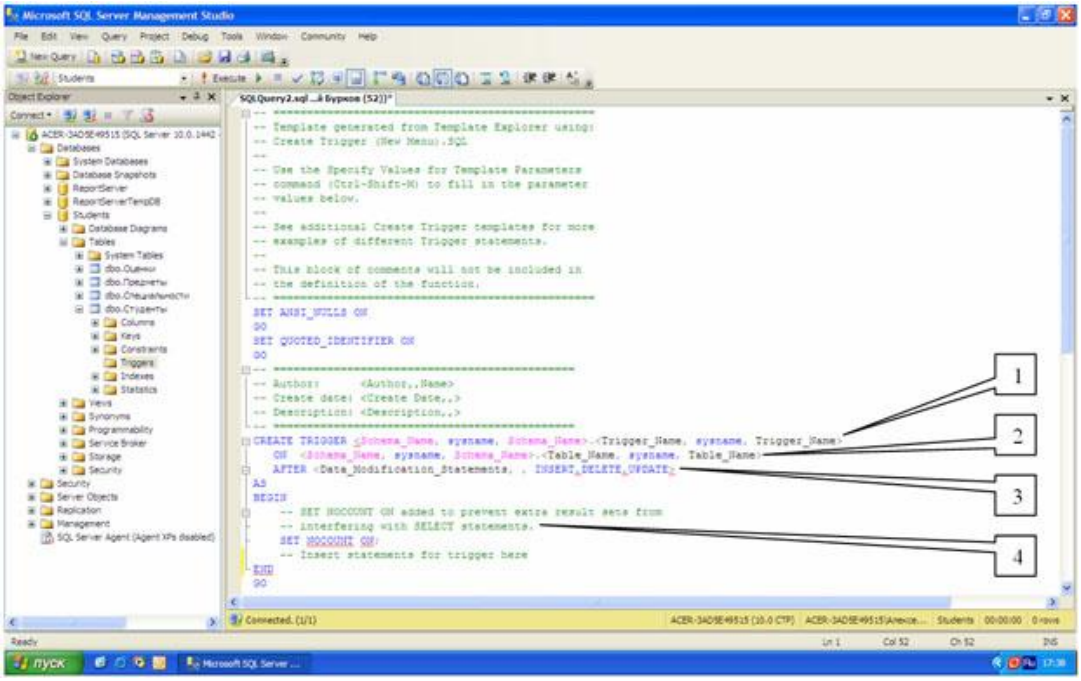


Рис. 10

Рассмотрим структуру триггеров:

Область определения имени функции (Trigger_Name);

Область, показывающая для какой таблицы создается триггер (Table_Name);

Область, показывающая когда выполнять триггер (INSERT - при создании записи в таблице, DELETE - при удалении и UPDATE - при изменении) и как его выполнять (AFTER - после выполнения операции, INSTEAD OF - вместо выполнения операции);

Тело триггера, содержит команды языка программирования запросов T-SQL.

В окне нового триггера наберите код как показано на рис. 11.

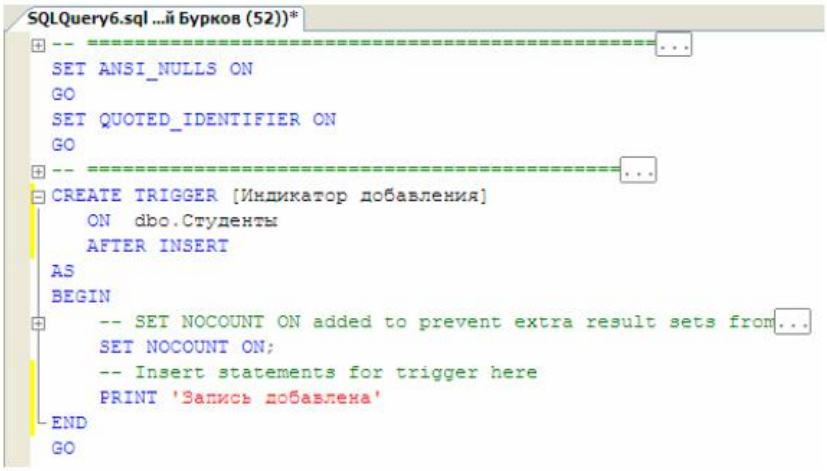
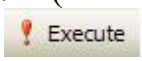


Рис. 11.

Из рис. 11 видно, что создаваемый триггер "Индикатор добавления" выполняется после добавления записи (AFTER INSERT) в таблицу

"Студенты" (ON dbo.Студенты). После добавления записи триггер выведет на экран сообщение "Запись добавлена" (PRINT 'Запись добавлена'). Выполните набранный код, нажав кнопку 

на панели инструментов. В нижней части окна с кодом появится сообщение "Command(s) completed successfully.".

Проверим, как работает новый триггер. Создайте новый пустой запрос и в нем наберите следующую команду для добавления новой записи в таблицу "Студенты" (рис. 12):

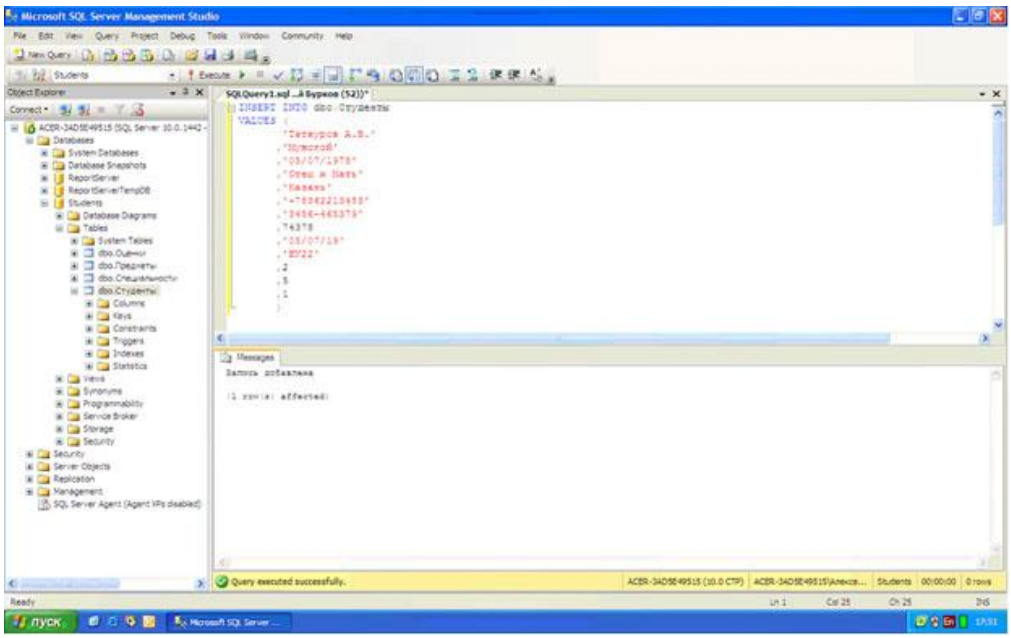


Рис. 12

Выполните набранную команду, нажав кнопку на панели инструментов. В таблицу будет добавлена новая запись, и триггер выведет сообщение "Запись добавлена" (рис. 12).

Теперь создадим триггер отображающий сообщение"Запись изменена". Создайте новый триггер, как в предыдущем случае. В окне нового триггера наберите следующий код (рис. 13):

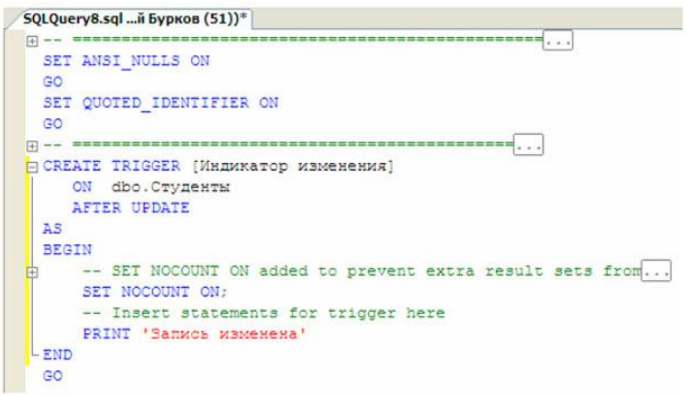


Рис. 13

Из рис. 13 видно, что новый триггер "Индикатор изменения" выполняется после изменения записи (AFTER UPDATE) в таблице "Студенты" (ON dbo.Студенты). После изменения записи триггер выведет на экран сообщение "Запись изменена" (PRINT 'Запись изменена'). Выполните набранный код. В нижней части окна с кодом появится сообщение "Command(s) completed successfully.".

Проверим работоспособность созданного триггера. Создайте новый запрос и в нем наберите команду, представленную на рис. 14.

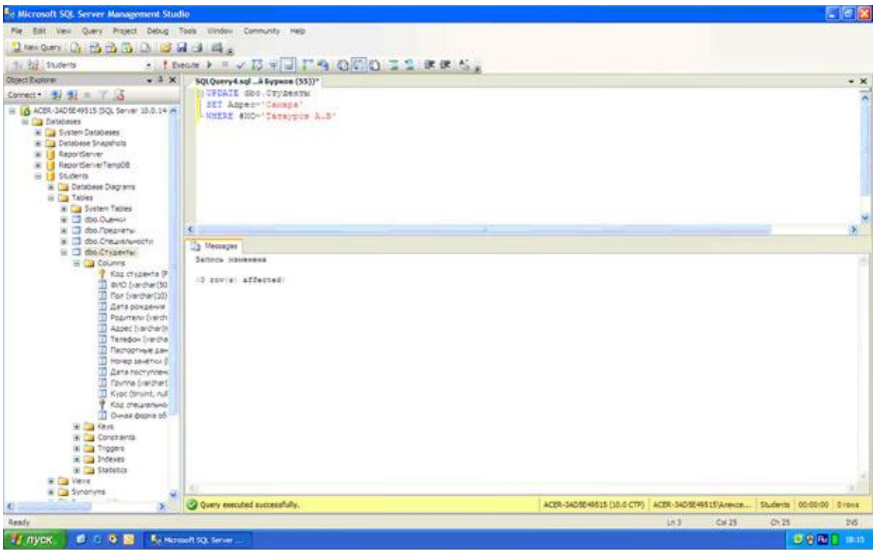


Рис. 14

Выполните набранную команду, нажав кнопку на панели инструментов. В таблицу будет добавлена новая запись, и триггер выведет сообщение "Запись изменена" (рис. 14).

Для полноты картины создадим триггер, выводящий сообщение при удалении записи из таблицы "Студенты". Создайте новый триггер и в нем наберите код, показанный на рис. 15.

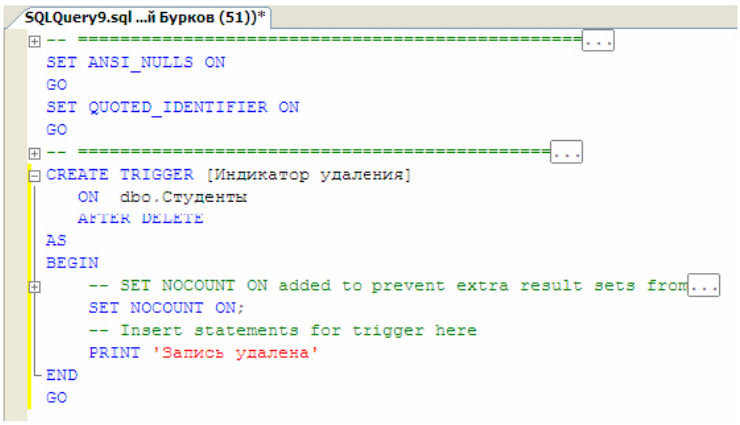


Рис. 15

Создаваемый триггер "Индикатор удаления" выполняется после удаления записи (AFTER DELETE) из таблицы студенты (ON

dbo.Студенты). После удаления записи триггер выводит сообщение "Запись удалена" (PRINT 'Запись удалена').

Выполните код, представленный рис. 15. В нижней части окна с кодом появится сообщение "Command(s) completed successfully.".

Проверим работу триггера "Индикатор удаления" удалив созданную ранее запись из таблицы "Студенты". Для этого создайте новый запрос и в нем наберите следующую команду (рис. 16):

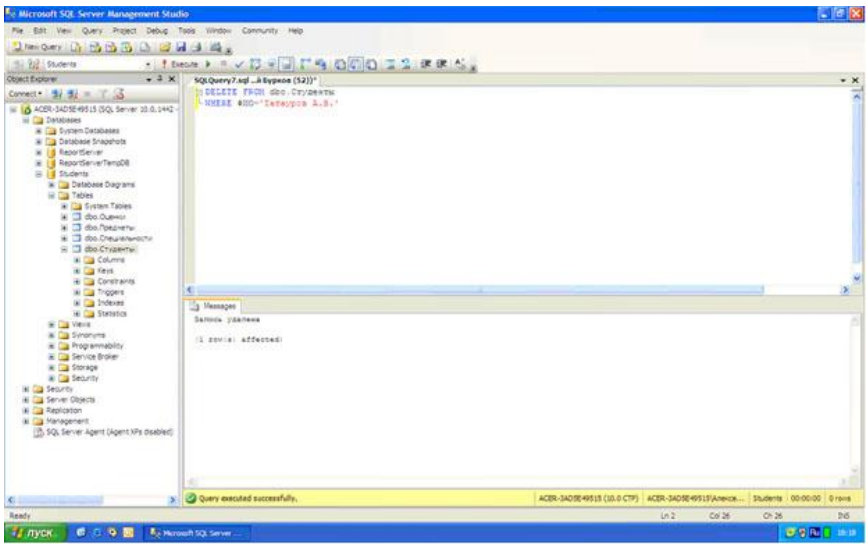


Рис. 16

Выполните вышеприведенную команду. После удаления записи триггер "Индикатор удаления" отобразит сообщение "Запись удалена" (рис. 16).

В заключение рассмотрим пример применения триггеров для обеспечения целостности данных. Создадим триггер "Удаление студента", который при удалении записи из таблицы студенты сначала удаляет все связанные с ней записи из таблицы "Оценки", а затем удаляет саму запись из таблицы "Студенты", тем самым обеспечивается целостность данных.

Создайте новый триггер и в нем наберите следующий код (рис. 17):

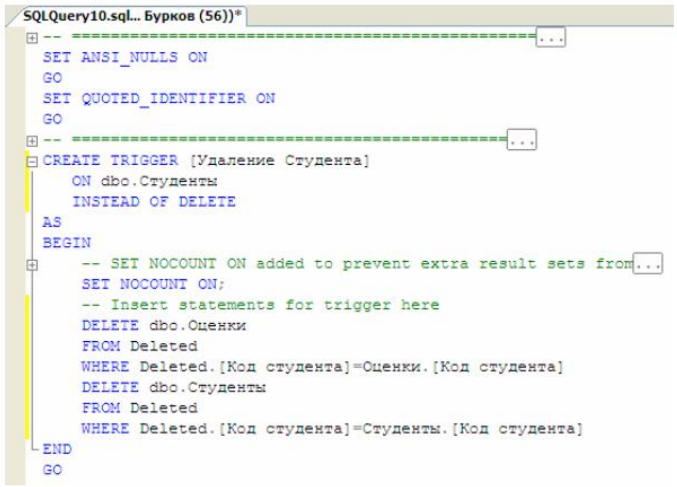


Рис. 17

Создаваемый триггер "Удаление студента" выполняется вместо удаления записи (INSTEAD OF DELETE) из таблицы "Студенты" (ON dbo.Студенты).

Замечание: При срабатывании триггера вместо удаления записи создается временная константа Deleted, содержащая имя таблицы из которой должно было быть произведено удаление.

После срабатывания триггера из таблицы "Оценки" удаляется запись, у которой значение поля "Код студента" равно значению такого же поля у удаляемой записи из таблицы "Студенты". Эту операцию выполняют следующие команды:

```
DELETE dbo.Оценки FROM Deleted
WHERE Deleted.[Код студента] = Оценки.[Код студента]
```

Затем удаляется запись из таблицы "Студенты", которую удаляли до срабатывания триггера. Удаление выполняется следующими командами:

```
DELETE dbo.Студенты
FROM Deleted
WHERE Deleted.[Код студента] = Студенты.[Код студента]
```

Выполните код, представленный на рис. 14.17. В нижней части окна с кодом появиться сообщение "Command(s) completed successfully.".

Проверим, как работает триггер "Удаление студента". Для этого создайте новый запрос и в нем наберите следующий код (рис. 18):

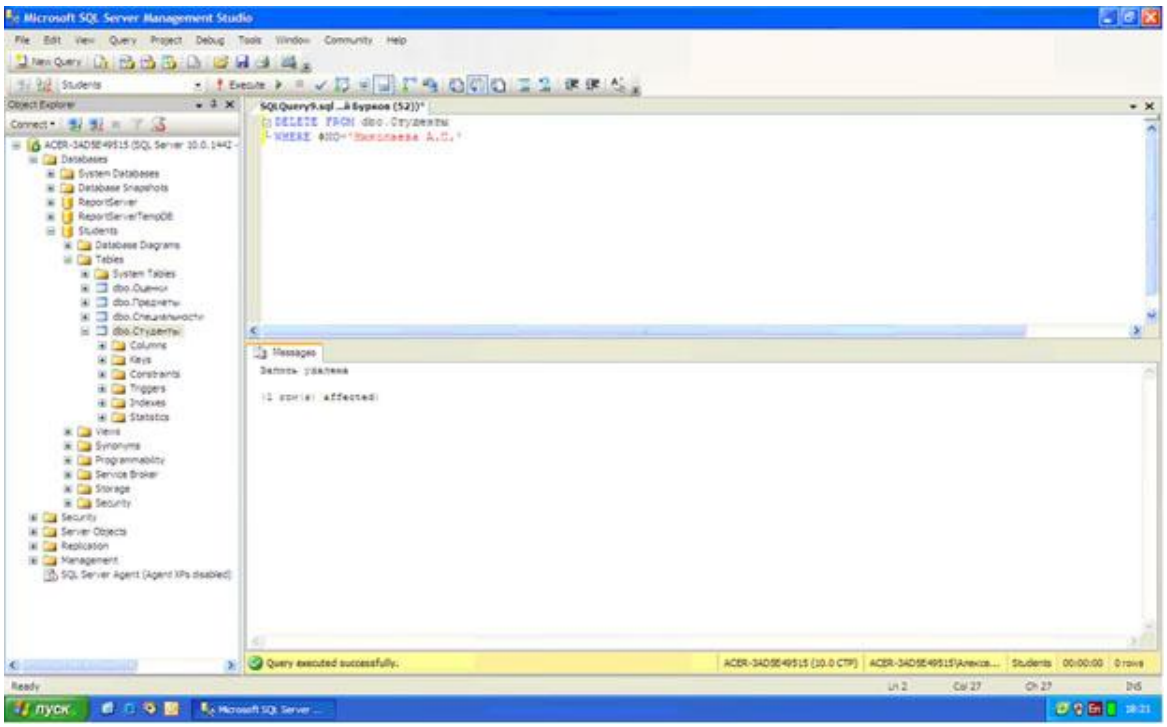


Рис. 18

При срабатывании триггера сначала из таблицы "Оценки" удалятся все связанные с удаляемой записью записи, а затем удаляется сама удаляемая запись из таблицы "Студенты", при этом сохраняется целостность данных.

Замечание: Хотелось бы заметить, что без использования триггера "Удаление студента" нам бы не удалось удалить запись из таблицы "Студенты". Команда удаления была бы заблокирована диаграммой "Диаграмма БД Студенты" во избежание нарушения целостности данных.

Команды языка DML

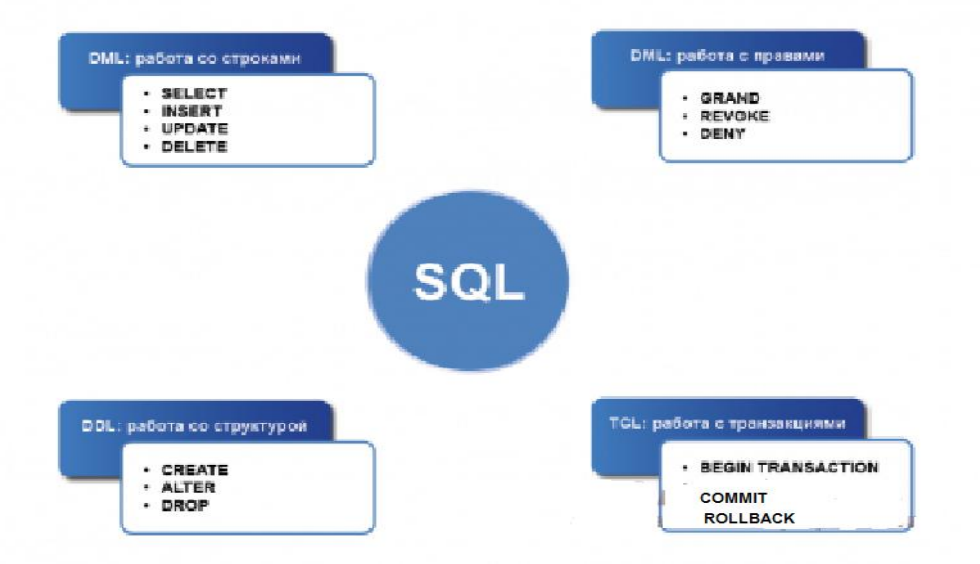
DataManipulationLanguage (работа со строками)

Приведенные ниже команды группы DML работают исключительно со строками и выполняются преимущественно клиентом:

INSERT — добавление строк(и);

SELECT — выборка строк(и);

UPDATE — изменение строк(и);



Пример: Вставка строки в таблицу books

```
INSERT INTO books (title, author, `year`, `description`)
VALUES ('Лабиринт тайных книг', 'Флавия Эрметес', 2001, 'Книга о книгах');
```

Пример: Вставка изображения в таблицу Images

```
CREATE TABLE Images
(
Id int,
Name varchar(50) not null,
Photo varbinary(max) not null
)
```

```
INSERT INTO Images (Id, Name, Photo)
SELECT 10, 'John', BulkColumn
FROM Openrowset ( Bulk 'C:\photo.bmp', Single_Blob) as Picture
```

Команды языка TCL (работа с транзакциями)

Группа команд по работе с транзакциями/

BEGIN TRANSACTION — начать транзакцию;

COMMIT — принять изменения, внесенные текущей транзакцией;

ROLLBACK — откат;

Пример: Вставка строки в таблицу books. Принять изменения.

```
BEGIN TRANSACTION;
INSERT INTO books (title, author, year, description)
VALUES ('Новая книга', 'Василий Ежиков', 2005, 'Книга о новом');
COMMIT WORK;
```

Пример: Добавление новой строки в таблицу books. Удаление записи. Откатить все изменения.

```
BEGIN TRANSACTION;
INSERT INTO books (title, author, year, description)
VALUES ('Новая книга', 'Василий Ежиков', 2005, 'Книга о новом');
DELETE FROM books WHERE id=3;
ROLLBACK WORK;
```

Роль триггеров в обеспечении целостности

Триггеры являются одной из разновидностей хранимых процедур. Их исполнение происходит при выполнении для таблицы какого-либо оператора языка манипулирования данными (DML). Триггеры используются для проверки целостности данных, а также для отката транзакций.

Триггер – это откомпилированная SQL-процедура, исполнение которой обусловлено наступлением определенных событий внутри реляционной базы данных. Применение триггеров большей частью весьма удобно для пользователей базы данных.

С помощью триггеров достигаются следующие цели:

- проверка корректности введенных данных и выполнение сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью ограничений целостности, установленных для таблицы;
- выдача предупреждений, напоминающих о необходимости выполнения некоторых действий при обновлении таблицы, реализованном определенным образом;
- накопление аудиторской информации посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили;

– поддержка репликации.

Триггер INSERT

Выполнение данного вида триггеров.

- пользователем выполняется оператор INSERT для добавления записей;
- сервер сохраняет информацию о запросе в журнале транзакций;
- вызывается триггер;
- подтверждение изменений и физическое изменение данных.

Во время вызова триггера, физического изменения в базе еще не произошло. В теле триггера вы можете увидеть добавляемые записи в виде таблицы inserted. Такой таблицы в базе данных не существует, inserted – это логическая таблица, которая содержит копию строк, которые должны быть вставлены в таблицу. Если быть точнее, она содержит журнал активности оператора INSERT. Вы можете использовать данные из этой таблицы для определения вставляемых данных. Строки из таблицы inserted всегда дублируют одну или несколько строк таблицы триггера.

Вся активность по изменению данных записываются в журнал, но информация в журнале транзакций не читаема. Однако таблица inserted позволяет вам ссылаться и определить изменения.

Таблица inserted всегда содержит такую же структуру, что и у таблицы, на которую установлен триггер.

Пример. С помощью триггера запретить добавление записей, в которых фамилия работника равно Иванов.

Использование таблицы inserted

```
CREATE TRIGGER i_tbPeoples ON dbo.tbPeoples
FOR INSERT
AS
 DECLARE @Fam varchar(50)
 SELECT @Fam= Familiy
 FROM inserted
 IF @Fam='Иванов'
  BEGIN
   PRINT 'ОШИБКА'
ROLLBACK TRANSACTION
END
```

В данном примере мы создаем триггер на добавление записей. Внутри триггера мы объявляем переменную @Fam типа varchar длиной в 50 символов. В эту переменную мы сохраняем содержимое поля "Familiy" таблицы inserted. Далее проверяем, если фамилия равно Иванов, то сообщаем об ошибке и откатываем транзакцию. Иначе, строка будет удачно добавлена.

Пример. Запрет нулевых значений в поле с помощью триггера

```
CREATE TRIGGER i_tbPeoples ON dbo.tbPeoples
FOR INSERT
AS
 IF EXISTS (SELECT *
          FROM inserted
          WHERE Familiy is NULL)
BEGIN
   PRINT 'ОШИБКА, вы должны заполнить поле Familiy '
   ROLLBACK TRANSACTION
  END
```

В этом примере мы проверяем, если в таблице inserted есть записи с нулевым значением поля "Familiy ", то откатываем попытку добавления.

Пример**.**Использование триггера для реализации ограничений на значение. В добавляемой в таблицу Сделка записи количество проданного товара должно быть не меньше, чем его остаток из таблицы Склад.

Команда вставки записи в таблицу Сделка может быть, например, такой:

```
INSERT INTO Сделка VALUES (3,1,-299,'01/08/2012')
```

Создаваемый триггер должен отреагировать на ее выполнение следующим образом: необходимо отменить команду, если в таблице Склад величина остатка товара оказалась меньше продаваемого количества товара с введенным кодом (в примере код товара=3). Во вставляемой записи количество товара указывается со знаком "+", если товар поставляется, и со знаком "-", если он продается. Представленный триггер настроен на обработку только одной добавляемой записи.

```
CREATE TRIGGER Триггер_ins
ON Сделка FOR INSERT
AS
IF @@ROWCOUNT=1
BEGIN
IF NOT EXISTS (SELECT *
FROM inserted
WHERE -inserted.количество<= ALL(SELECT
Склад.Остаток
FROM Склад, Сделка
WHERE Склад.КодТовара=Сделка.КодТовара))
BEGIN
ROLLBACK TRAN
```

```
PRINT
'Отмена поставки: товара на складе нет'
   END
END
```

Примечание:

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию:@@ROWCOUNT, которая возвращает количество строк, изменённое последней командой. Для каждого триггера создается свой комплект таблиц inserted и deleted, так что никакой другой триггер не сможет получить к ним доступ. В зависимости от команды, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным.

Пример. Использование триггера для реализации ограничений на значение. Если значение добавляемого поля TEW меньше нуля, происходит откат транзакции.

```
ALTER trigger [dbo].[ins1] on [dbo].[new]
After insert
as
declare @s int
select @s=tew from inserted
if @s < 0
 BEGIN
print 'NO'
ROLLBACK TRANSACTION;
RETURN
END
 print @s
```

Пример. Использование триггера для реализации ограничений на значение. Если значение добавляемого поля datw равно текущему году, происходит откат транзакции.

```
Create trigger [dbo].[ins1] on [dbo].[new]
after insert
as
declare @s int,
@d date
select @d=datw from inserted
set @s = DATEDIFF(YYYY,@d,GETDATE());
print @s
if @s <= 0
 BEGIN
```

```
print 'NO'
ROLLBACK TRANSACTION;
RETURN
END
```

Импорт данных

Для импорта и экспорта данных в SQL Server доступны разнообразные методы. Сюда входят инструкции Transact-SQL, программы командной строки и мастеры.

Кроме того, можно импортировать и экспортировать данные в разных форматах. Эти форматы включают неструктурированные файлы, файлы Excel, основные типы реляционных баз данных и форматы различных облачных служб.

Импорт данных осуществляется с помощью команд BULK INSERT или OPENROWSET(BULK...). Обычно эти команды выполняются в SQL Server Management Studio (SSMS). Кроме того, можно импортировать и экспортировать данные с помощью служебной программы командной строки BCP, а также воспользоваться мастером экспорта и импорта неструктурированных файлов в SQL Server Management Studio (SSMS). Мастер импорта и экспорта SQL Server позволяет экспортировать данные из самых разных источников и импортировать их во множество различных назначений. Чтобы использовать мастер, необходимо установить SQL Server Integration Services (SSIS) или SQL Server Data Tools (SSDT).

Можно выполнять импорт и экспорт данных в виде неструктурированных файлов, а также во множестве других форматов, в виде реляционных баз данных и облачных служб.

Оператор BULK INSERT

Оператор T-SQL BULK INSERT можно использовать для массового копирования данных из файла данных в базу данных SQL Server. Но нельзя использовать для извлечения данных из баз данных SQL Server. Это ограничение уменьшает его функциональные возможности, но поскольку оператор BULK INSERT выполняется как поток внутри SQL Server, это устраняет необходимость передачи данных из одной программы в другую, что повышает производительность при загрузке данных. Таким образом, оператор BULK INSERT загружает данные более эффективно, чем другие.

Синтаксис BULK INSERT

Оператор BULK INSERT имеет несколько обязательных параметров и много необязательных. Вызов BULK INSERT из SQL Server (с помощью ISQL, OSQL или анализатора запросов [Query Analyzer]) происходит с помощью следующего оператора. (Здесь приводятся все обязательные и необязательные параметры.)

```
BULK INSERT   [['имя_базы_данных'.]['владелец'].]
                    {'имя_таблицы' | 'имя_представления' FROM 'файл_данных' }
```

```
[WITH  (
        [BATCHSIZE [ = размер_группы ]]
        [[,] CHECK_CONSTRAINTS ]
        [[,] CODEPAGE [ = 'ACP' | 'OEM' | 'RAW' | 'кодовая_страница']]
        [[,] DATAFILETYPE [ = {'char'|'native'|
                               'widechar'|'widenative'}]]
        [[,] FIELDTERMINATOR [ = 'ограничитель_полей' ]]
        [[,] FIRSTROW [ = первая_строка ]]
        [[,] FIRETRIGGERS [ = триггеры ]]
        [[,] FORMATFILE [ = 'путь_к_форматному_файлу' ]]
        [[,] KEEPIDENTITY ]
        [[,] KEEPNULLS ]
        [[,] KILOBYTES_PER_BATCH [ = килобайт_на_группу ]]
        [[,] LASTROW [ = последняя_строка ]]
        [[,] MAXERRORS [ = максимум_ошибок ]]
        [[,] ORDER ( { колонка [ ASC | DESC ]}[ ,...n ])]
        [[,] ROWS_PER_BATCH [ = строк_на_группу ]]
        [[,] ROWTERMINATOR [ = 'разделитель_строк' ]]
        [[,] TABLOCK ]
         )]
```

Обязательные параметры

Местоположение файла данных указывается параметром файл_данных. Это должен быть допустимый путь доступа к файлу.

Местоположение базы данных, в которую помещаются данные, задается определением таблицы или представлением. Как видно из определения синтаксиса оператора, вы можете также указывать владельца таблицы или представления и/или имя базы данных. Если использовать эту команду массовой вставки для вставки данных в какое-либо представление, то вы можете затрагивать только одну из *базовых таблиц*, указываемую в предложении FROM этого представления.

Необязательные параметры

Вы можете использовать необязательные параметры и ключевые слова, которые перечислены в таблице, чтобы модифицировать поведение BULK INSERT. Как вы увидите из описания, параметры, которые можно использовать с оператором BULK INSERT, аналогичны параметрам программы *BCP*.

Необязательные параметры для оператора BULK INSERT

Необязательный параметр	Описание
BATCHSIZE = размер	Указывает количество строк в группе (пакетном задании). Каждая группа обрабатывается как одна транзакция
CHECK_CONSTRAINTS	Указывает на то, что будет выполняться проверка ограничений. По умолчанию

	ограничения игнорируются.
CODEPAGE [= '*ACP*' | '*OEM*' | 'RAW' | 'кодовая_страница']	Указывает кодовую страницу данных в файле данных. Этот параметр полезно использовать только с типами данных char, varchar и text
ATAFILETYPE [= 'char' | 'native' | '*widechar*' | 'widenative']	Указывает тип данных в файле данных; по умолчанию это тип char. Другие типы: native (собственные типы данных базы данных), *widechar* (символы Unicode) и widenative (то же, что и native, но типы char, varchar и text сохраняются как Unicode)
FIELDTERMINATOR [= ограничитель_полей]	Указывает ограничитель полей, используемый с типами данных char и *widechar*. По умолчанию это символ табуляции (\t)
FIRSTROW [= первая_строка]	Номер первой строки для копирования. По умолчанию 1. Этот параметр полезно использовать, если вы хотите пропустить заголовочную информацию в файле данных.
FORMATFILE [= форматный_файл]	Указывает путь доступа к форматному файлу
KEEPIDENTITY	Указывает, что в импортируемых файлах данных присутствуют значения для колонки со свойством identity
KEEPNULLS	Указывает, что в пустых колонках сохраняется значение null
KILOBYTES_PER_BATCH [= число]	Указывает приблизительное количество килобайт на одну группу (пакетное задание), используемую при массовом копировании
LASTROW [= последняя_строка]	Указывает последнюю строку для выполнения массового копирования. По умолчанию используется значение 0. Этот параметр полезно использовать, если вы хотите вставить только определенное количество строк
MAXERRORS [= максимум_ошибок]	Указывает, сколько ошибок должно произойти, чтобы прекратить вставку. Значение по умолчанию равно 10
ORDER (колонка [*ASC* | DESC])	Задает, что данные в указанной колонке должны быть отсортированы в указанном порядке (по возрастанию или убыванию)
ROWS_PER_BATCH [= строк_на_группу]	Указывает количество строк на одну группу (пакетное задание). Каждая группа копируется в виде одной транзакции. По умолчанию вставка всех строк файла данных

	выполняется как одна транзакция с использованием одной фиксации. Этот параметр может понадобиться, когда вам нужно выполнять массовые вставки, чтобы освобождать блокировки таблиц, когда идет обработка групп, что позволит выполнять другую обработку
ROWTERMINATOR [= разделитель_строк]	Указывает разделитель строк для данных типа char и *widechar*. По умолчанию используется символ новой строки (\n)

Использование BULK INSERT

Рассмотрим два примера использования оператора BULK INSERT.

В обоих примерах мы будем загружать данные из файла с символьными данными data.file (который использовали в предыдущих примерах) в таблицу Customers базы данных Northwind.

Примечание. Оператор BULK INSERT можно использовать только для вставки данных в базу данных; его нельзя использовать для извлечения данных.

Для загрузки данных в базу данных используйте следующий оператор T-SQL:

```
BULK INSERT Northwind..Customers FROM 'C:\data.file'
WITH
   (
   DATAFILETYPE = 'char'
   )
GO
```

Можно добавлять любое количество параметров. В следующем примере используется большее количество необязательных параметров:

```
BULK INSERT Northwind..Customers FROM 'C:\data.file'
WITH
   (
   BATCHSIZE = 5,
   CHECK_CONSTRAINTS,
   DATAFILETYPE = 'char',
   FIELDTERMINATOR = '\t',
   FIRSTROW = 5,
   LASTROW = 20,
   TABLOCK
   )
```

Этот оператор будет загружать из файла данных только строки 5-20. Указывается, что *разделителем полей* будет символ табуляции (несмотря на то, что он используется по умолчанию). В этом примере также указано, что во время процесса массовой вставки будут проверяться ограничения, а также задается блокировка таблицы на период загрузки. Транзакции, выполняющие загрузку, будут выполняться группами по пять строк.

Задания

1 Изучить теоретические сведения.

2 В соответствии с вариантом задания создать запросы на вставку значений в базу данных с помощью:

– запросов;

– хранимых процедур;

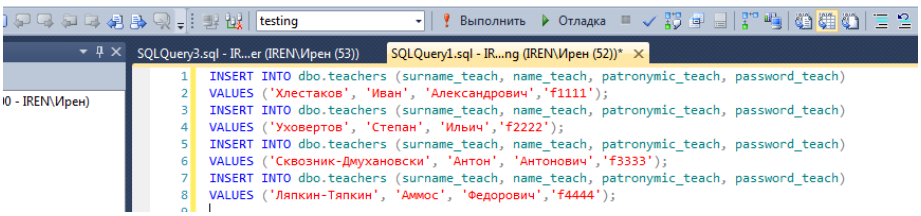
– импорта.

3 Для обеспечения целостности и непротиворечивости значений создать триггеры ограничений работы с данными;

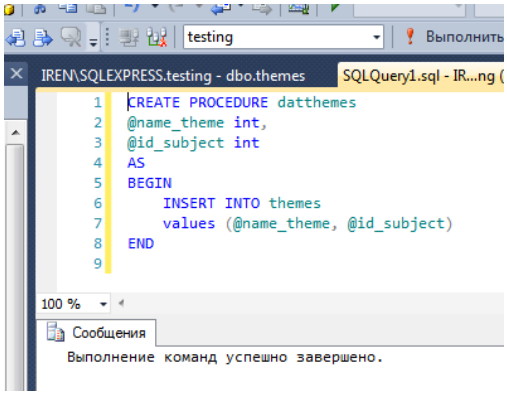
Порядок выполнения работы

Вспомним, что для ввода данных в таблицу используется команда INSERT INTO непосредственного ввода:

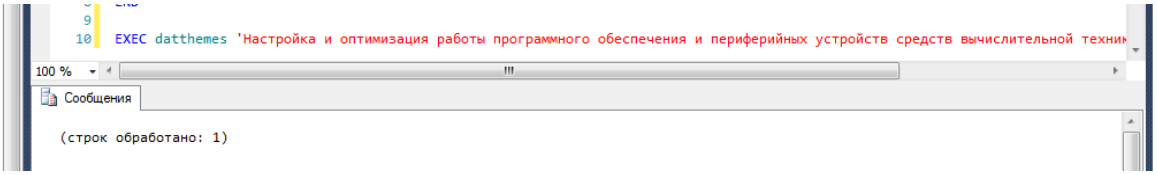
INSERT INTO dbo.teachers (surname_teach, name_teach, patronymic_teach, password_teach)
VALUES ('Хлестаков', 'Иван', 'Александрович','f1111'); …



Или с помощью процедуры. Определим процедуру datthemes:

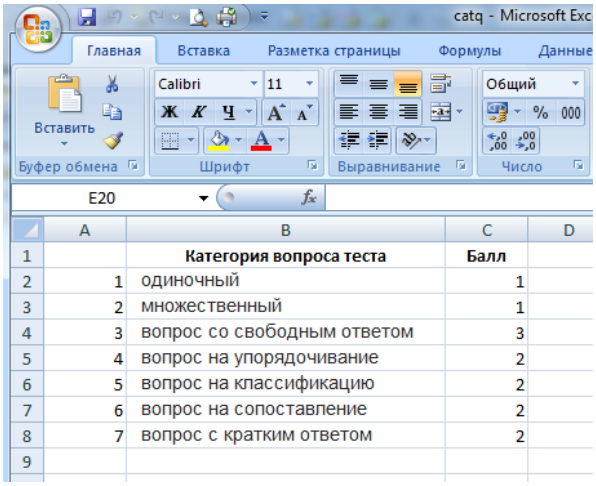


И команды на выполнение процедуры:



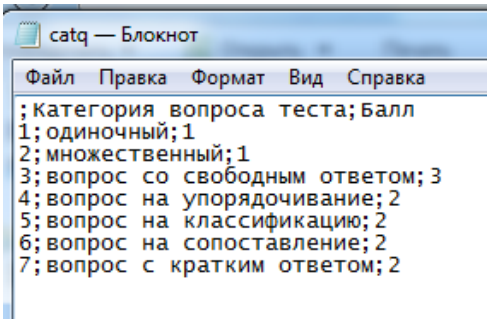
Импорт данных

Часто пользователям непривычно работать с базой данных, но зато большинство работников достаточно хорошо справляются с созданием таблиц в Ms Excel. Воспользуемся оператором массового импорта данных BULK INSERT. Для этого создадим таблицу, в которой количество столбцов **обязательно** совпадает с количеством полей таблицы базы данных. Например, таблицу категорий вопросов для теста.



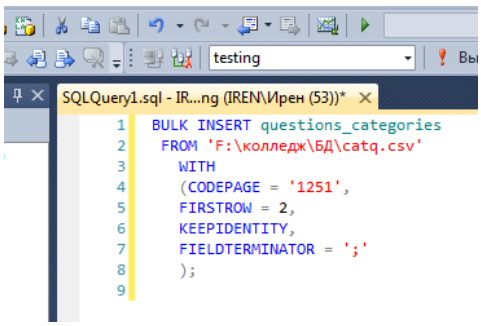
	A	B	C	D
1		**Категория вопроса теста**	**Балл**	
2	1	одиночный	1	
3	2	множественный	1	
4	3	вопрос со свободным ответом	3	
5	4	вопрос на упорядочивание	2	
6	5	вопрос на классификацию	2	
7	6	вопрос на сопоставление	2	
8	7	вопрос с кратким ответом	2	
9				

Преобразуем файл с расширением xlsx в формат CSV (разделители – запятые), после чего откроем его в блокноте. CSV представляет собой компактный текстовый формат для хранения табличных данных. С файлами в формате CSV способны работать (экспортировать/импортировать данные) все современные приложения для работы с таблицами (например, Excel).

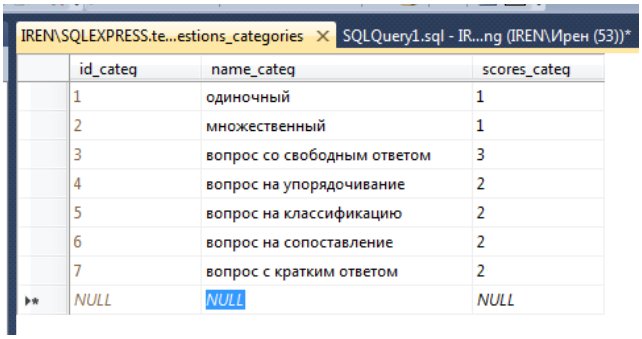


Можно заметить, что разделитель между колонками «;». Кроме того, первая строка – шапка таблицы, которую необходимо пропустить при импорте данных. Первый столбец – это коды категорий вопросов, в таблице базы данных они заполняются автоматически, но для файлов данных, содержащих значения идентификаторов, можно выполнить массовый импорт в экземпляр Microsoft SQL Server. По умолчанию значения столбца идентификаторов в импортируемом файле данных не учитываются, и SQL Server автоматически присваивает им уникальные значения на основе начального значения и значения приращения, указанных при создании таблицы. Запишем запрос для импорта данных в таблицу questions_categories из файла catq.csv, в котором укажем следующие параметры:

– начальная строка 2;
– разделитель столбцов «;»;
– автоматическая нумерация KEEPIDENTITY;
– кодировка для кириллицы «1251».



Обновим таблицы и посмотрим результат:

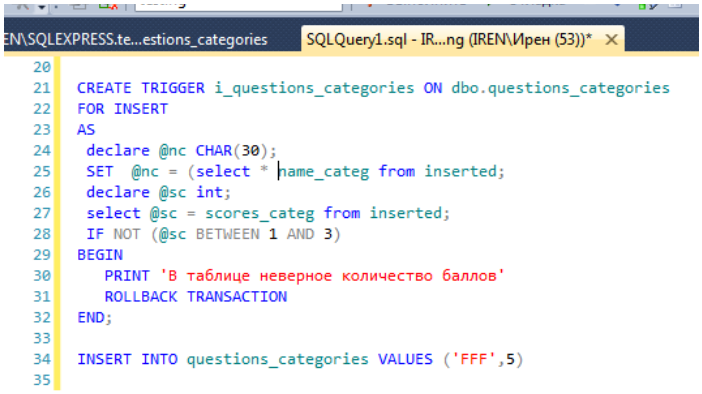


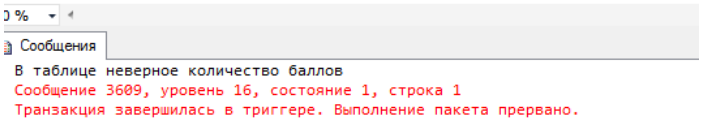
id_categ	name_categ	scores_categ
1	одиночный	1
2	множественный	1
3	вопрос со свободным ответом	3
4	вопрос на упорядочивание	2
5	вопрос на классификацию	2
6	вопрос на сопоставление	2
7	вопрос с кратким ответом	2
NULL	NULL	NULL

Особенно полезно использовать импорт для больших таблиц.

Ограничения на значения

Создадим триггер для реализации ограничений на значения. В таблице базы данных в столбце балл значения должны быть в диапазоне от 1 до 3. Если значение добавляемого поля не входит в этот диапазон, происходит откат транзакции. При добавлении записи, в которой количество баллов за ответ 5, произойдет завершение транзакции и появится сообщение об ошибке.





Таким образом, использование возможностей языка SQL для реализации запросов позволяет …

Содержание отчета

1 Название работы
2 Цель работы
3 Перечень технических средств обучения
4 Порядок выполнения работы
5 Вывод

Варианты заданий

Варианты заданий представлены в практической работе № 13.

Используемая литература

– Г.Н.Федорова Основы проектирования баз данных. М.: Академия, 2020
– Г.Н.Федорова Разработка, администрирование и защита баз данных. М.: Академия, 2018
– https://metanit.com/sql/sqlserver/12.1.php
– https://info-comp.ru/obucheniest/361-trigger-in-transact-sql.html