Практическая работа 55
Создание хранимых процедур.
Создание диаграмм и триггеров, применяемых для обеспечения целостности данных

Цель работы: Получить практический опыт создания хранимых процедур.

Перечень оборудования и программного обеспечения

Персональный компьютер
Microsoft Office (Word, Visio, Access)
MS SQL Server 2008 R2

Краткие теоретические сведения

Функциональность скриптов T-SQL, которые выполняются на SQL Server, может быть очень сильно расширена при помощи набора хранимых процедур SP_OA (от OleAutomation - SP_OACREATE, SP_OAMETHOD, SP_OASETPROPERTY, SP_OAGETPROPERTY 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 = Create Object("Scripting.FileSystemObject")
'Создаем объект TextStream (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 @Varnvarchar(40)
Select @Var = CompanyName FROM Customers WHERE CustomerId = 'ALFKI'
```

2) затем создаем объект файловой системы и получаем на него ссылку (handleid) 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 @hrint
DECLARE @srcvarchar(255)
DECLARE @descvarchar(255)
DECLARE @oFileint
 -- 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', @oFileOUT, '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, каки 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;

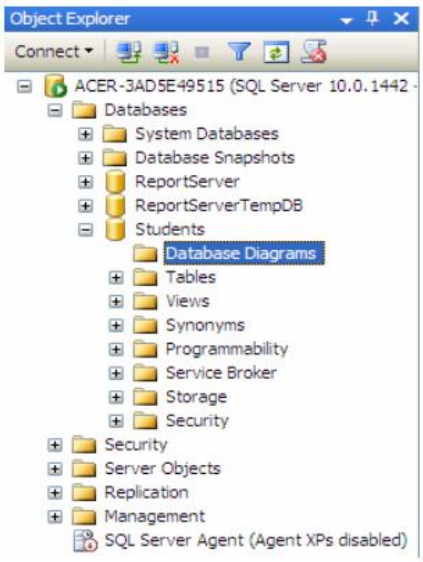
– SP_OADestroy можно, в принципе, не вызывать - созданные при помощи SP_OACreate программные объекты автоматически удаляются из памяти при завершении работы скрипта;

– при первом запуске SP_OACreate в оперативной памяти выделяется специальная программная область - так называемая shared OLE Automationstoredprocedureexecutionenvironment, 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;

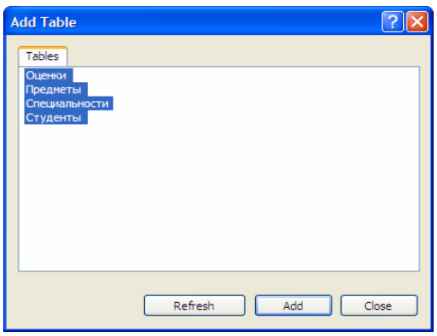
– по умолчанию программный объект работает в контексте учетной записи SQL Server и обладает полными правами на все ресурсы, к которым имеет доступ SQL Server (в том числе выделенной SQL Server оперативной памяти). При вызове SP_OACreate при помощи необязательного параметра можно изменить эти права на более ограничительные.

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



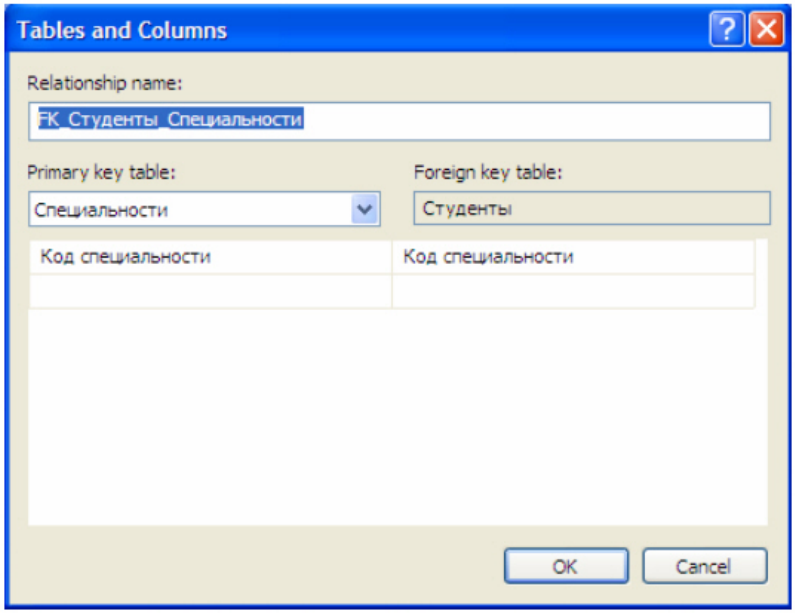
Создадим диаграмму, обеспечивающую целостность данных БД "Students". Для создания новой диаграммы в БД "Students" щелкните ПКМ по папке "DatabaseDiagrams" и в появившемся меню выберем пункт

"NewDatabaseDiagram". Сначала появится окно с вопросом о добавлении нового объекта "Диаграмма". В этом окне нужно нажать кнопку "Yes". Затем появится окно "AddTable" предназначенное для добавления таблиц в новую диаграмму.

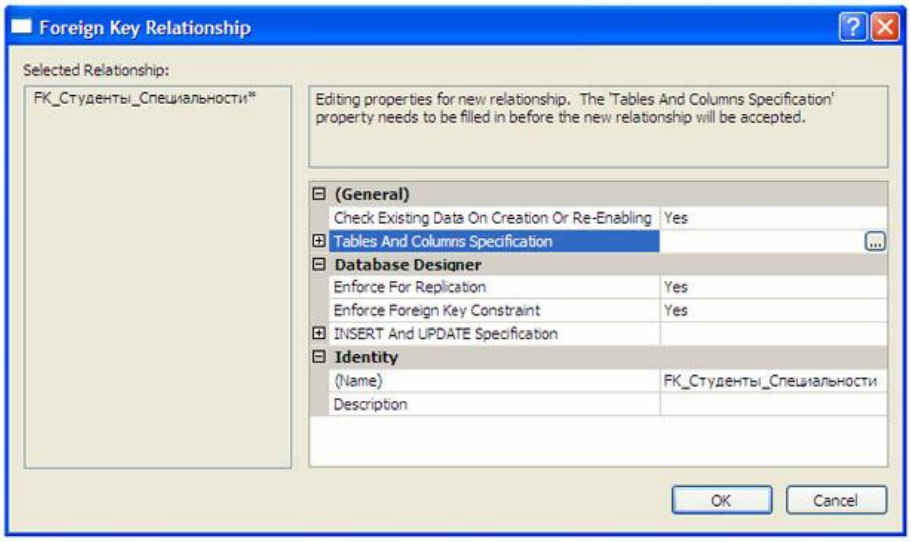


В окне добавления таблиц выделите все таблицы нашей БД и нажмите кнопку "Add". Закройте окно "AddTable" нажатием на кнопку "Close".

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 "TablesandColumns".

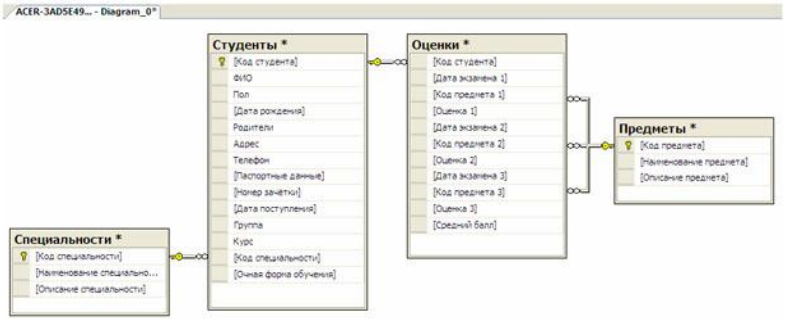


В окне создания связи нажмите кнопку "Ok". Появится окно настройки свойств связи "ForeignKeyRelationship".

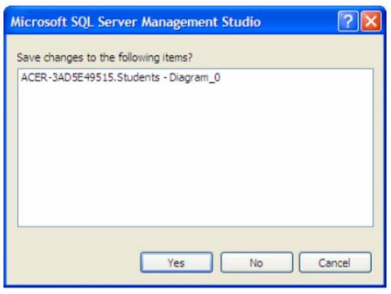


Оставьте свойства связи без изменений и в окне свойств связи нажмите кнопку "Ok".В диаграмме между таблицами "Студенты" и "Специальности" появится связь в виде ломанной линии.

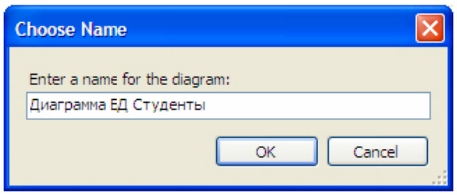
Аналогичным образом создайте связь таблицы "Студенты" с таблицей "Оценки", перетащив поле "Код студента" из таблицы "Студенты" на одноименное поле в таблице "Оценки". Затем, свяжите таблицы "Предметы" и "Оценки", перетащив поле "Код предмета" из таблицы "Предметы" на поля "Код предмета 1", "Код предмета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



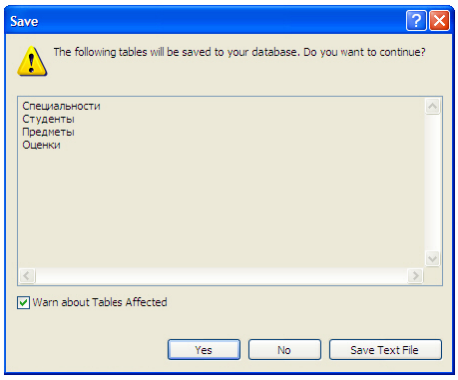
Закройте окно с диаграммой, щелкнув мышью по кнопке закрытия, расположенной в верхнем правом углу окна с диаграммой. Появится окно с вопросом о сохранении новой диаграммы, где необходимо нажать кнопку "Yes".



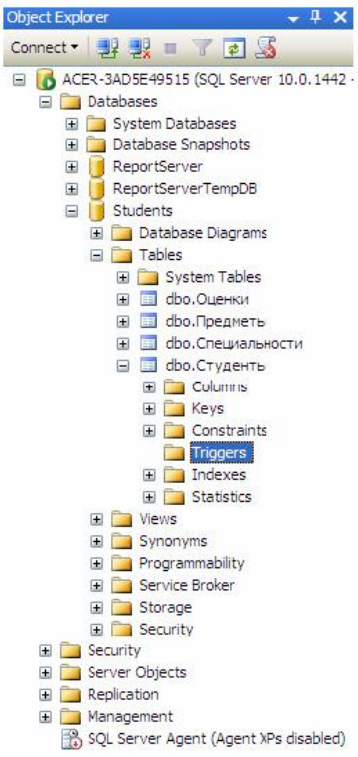
Появится окно определения имени новой диаграммы "ChooseName". В окне определения имени, задайте имя диаграммы как "Диаграмма БД Студенты" и нажмите кнопку "Ok".



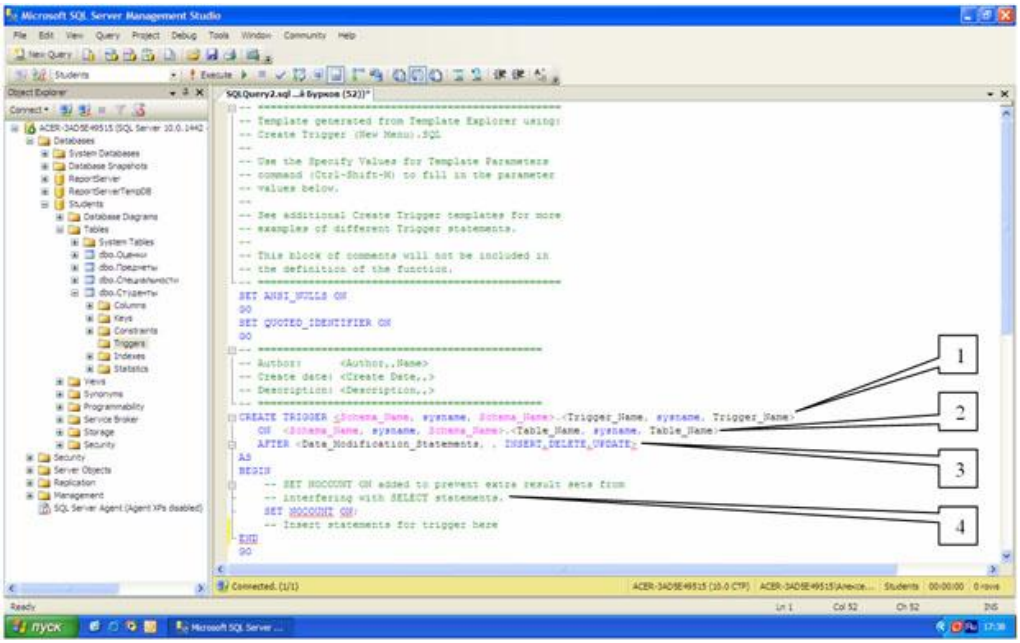
Появится окно "Save" с запросом сохранения таблиц, входящих в диаграмму. В данном окне необходимо нажать кнопку "Yes".



Перейдем к созданию триггеров. 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".



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 "NewTrigger". Появится следующее окно с новым триггером:



Рассмотрим структуру триггеров:

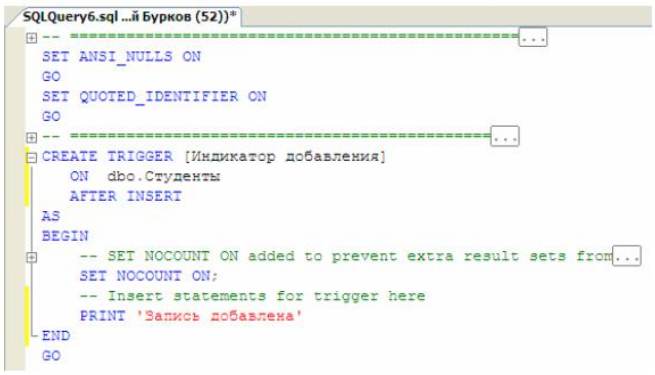
Область определения имени функции (Trigger_Name);

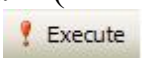
Область, показывающая для какой таблицы создается триггер (Table_Name);

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.

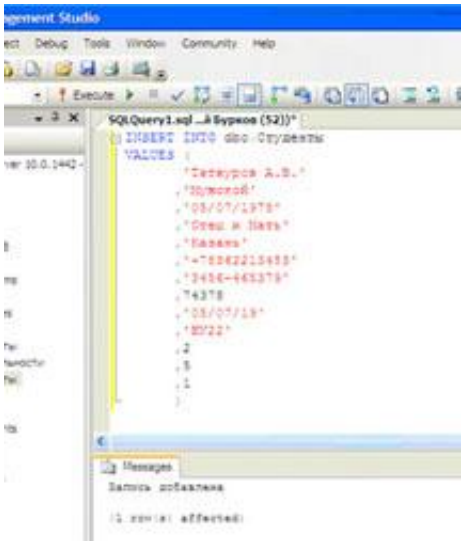
В окне нового триггера наберите код как показано на рисунке.



Из рисунка 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 

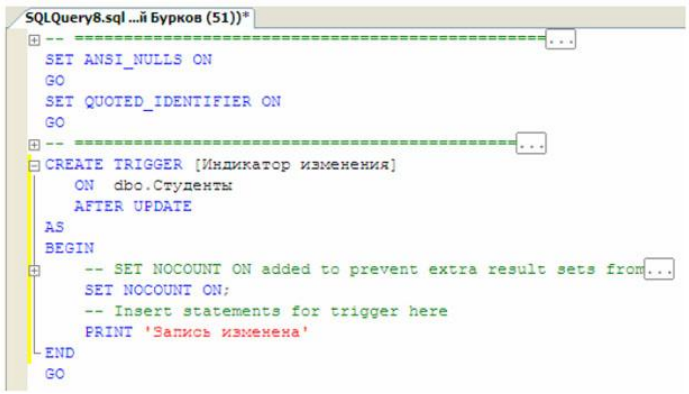
на панели инструментов. В нижней части окна с кодом появится сообщение "Command(s) completedsuccessfully.".

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



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

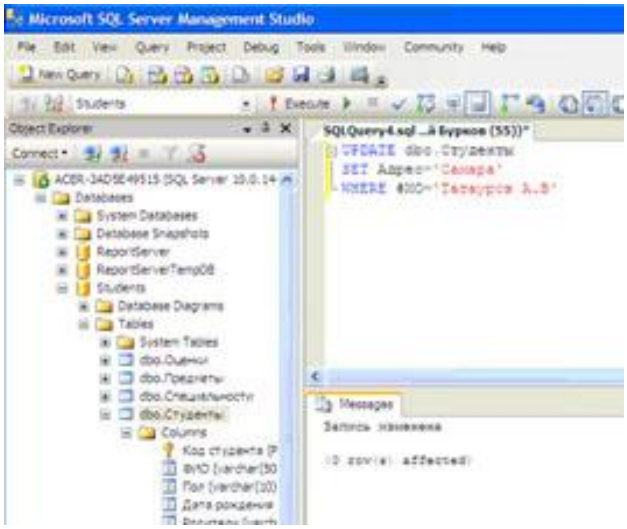
Теперь создадим триггер отображающий сообщение"Запись изменена". Создайте новый триггер, как в предыдущем случае. В окне нового триггера наберите следующий код:



Из рисунка видно, что новый триггер "Индикатор изменения" выполняется после изменения записи (AFTER UPDATE) в таблице "Студенты" (ON dbo.Студенты). После изменения записи триггер выведет на

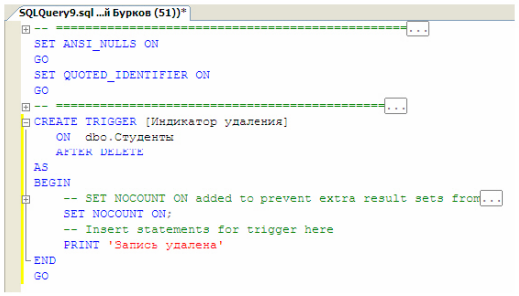
экран сообщение "Запись изменена" (PRINT 'Запись изменена'). Выполните набранный код. В нижней части окна с кодом появится сообщение "Command(s) completedsuccessfully.".

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



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

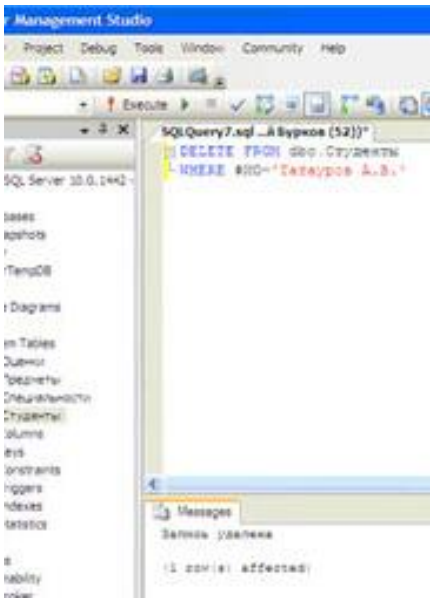
Для полноты картины создадим триггер, выводящий сообщение при удалении записи из таблицы "Студенты". Создайте новый триггер и в нем наберите код, показанный на рисунке.



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 рисунке. В нижней части окна с кодом появится сообщение "Command(s) completedsuccessfully.".

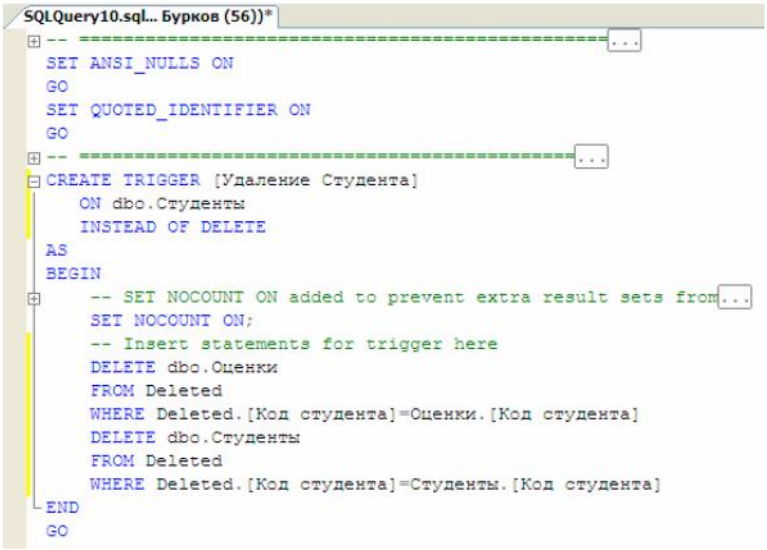
Проверим работу триггера "Индикатор удаления" удалив созданную ранее запись из таблицы "Студенты". Для этого создайте новый запрос и в нем наберите следующую команду:



Выполните вышеприведенную команду. После удаления записи триггер "Индикатор удаления" отобразит сообщение "Запись удалена".

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:



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:

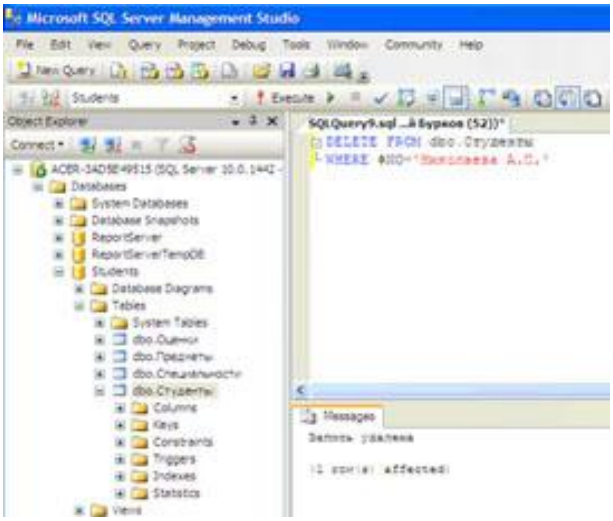
```
DELETE dbo.Оценки FROM Deleted
WHERE Deleted.[Код студента] = Оценки.[Код студента]
```

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 предыдущем рисунке. 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:



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.

Задания

1 Изучить теоретические сведения.

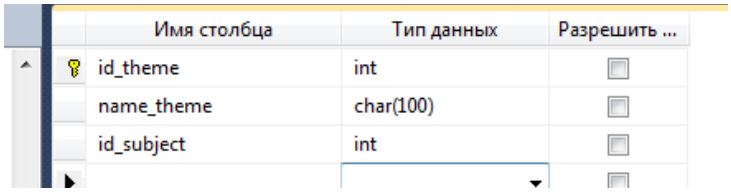
2 В соответствии с вариантом задания выполнить создание хранимых процедур.

3 В соответствии с вариантом задания выполнить создание триггеров:

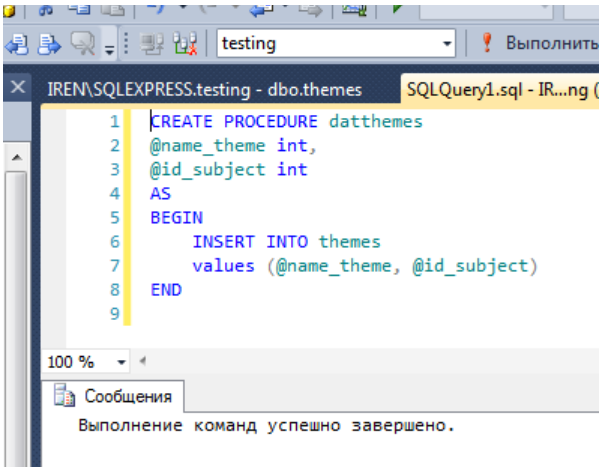
- на добавление данных;
- обновление данных;
- для обеспечения целостности данных.

Порядок выполнения работы

1 Создадим хранимую процедуру, например, для добавления данных в таблицу Темы (themes), следующей структуры:

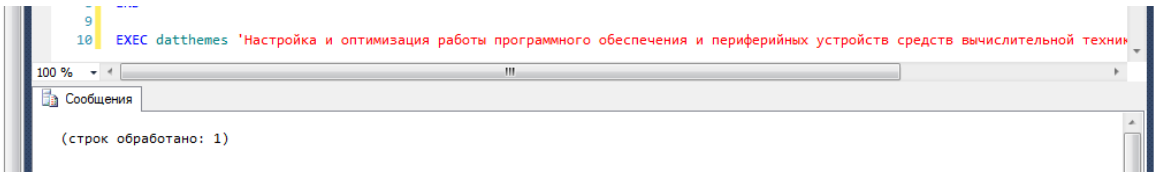


С помощью команды CREATE определим процедуру datthemes с параметрами, имена которых совпадают с именами полей таблицы, а тело процедуры состоит из команды ввода данных:



Если обновить раздел Хранимые процедуры, то в нем появится новый элемент.

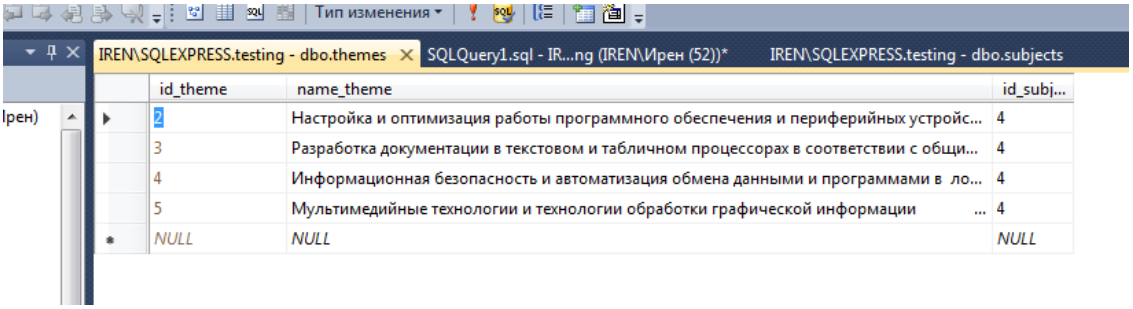
Запишем команды на выполнение процедуры:



```
EXEC datthemes 'Разработка документации в текстовом и табличном процессорах в соответствии с общими требованиями к оформлению'
EXEC datthemes 'Информационная безопасность и автоматизация обмена данными и программами в  локальной  и глобальной сети',4;
EXEC datthemes 'Мультимедийные технологии и технологии обработки графической информации',4;
```

```
100 %
Сообщения
(строк обработано: 1)
(строк обработано: 1)
(строк обработано: 1)
```

В результате данные размещены в таблице, причем код темы заполняется автоматически, т.к. при создании таблицы это поле было задано со свойством IDENTITY, он автоматически генерирует идентификаторы, поэтому в такой столбец вставить данные просто не получится:



id_theme	name_theme	id_subj...
2	Настройка и оптимизация работы программного обеспечения и периферийных устройс...	4
3	Разработка документации в текстовом и табличном процессорах в соответствии с общи...	4
4	Информационная безопасность и автоматизация обмена данными и программами в ло...	4
5	Мультимедийные технологии и технологии обработки графической информации ...	4
NULL	NULL	NULL

2 Перейдем к созданию **триггер**ов. Создадим триггер для таблицы themes. Определим триггер, который будет срабатывать при добавлении данных:

```
CREATE TRIGGER themes_INSERT_UPDATE
ON themes
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
PRINT 'Добавлена тема '
END;
```

При выполнении команды INSERT триггер выдает сообщение о добавлении темы.

Инструкция SET NOCOUNT ON запрещает всем инструкциям хранимой процедуры отправлять клиенту сообщения DONE_IN_PROC, установка в инструкции SET NOCOUNT параметра ON может значительно повысить производительность за счет существенного снижения объема сетевого трафика. Инструкция SET NOCOUNT устанавливается во время выполнения хранимой процедуры.

```
24
25  EXEC datthemes 'Базы данных на Access',2;
```

```
100 %
Сообщения
Добавлена тема

(строк обработано: 1)
```

Создадим триггер обновления данных в таблице theme:

```
26
27  CREATE TRIGGER themes_UPDATE
28  ON themes
29  AFTER UPDATE
30  AS
31  BEGIN
32  SET NOCOUNT ON;
33  PRINT 'Обновлена тема '
34  END;
```

```
100 %
Сообщения
Выполнение команд успешно завершено.
```

Изменим только что созданную запись, а затем снова запустим процедуру создания новой темы. Теперь запишем команду обновления данных:

```
35
36  UPDATE themes SET name_theme='Работа с базами данных в SQL Server'
37  WHERE name_theme='Базы данных на Access';
```

```
100 %
Сообщения
Обновлена тема

(строк обработано: 1)
```

В результате появилось сообщение об обновлении темы.

3 Рассмотрим пример триггера, отвечающего, за сохранение целостности базы данных при удалении данных. Пусть в базе данных имеются связанные таблицы Предметы1 и Темы, предположим необходимо удалить данные о предмете из таблицы Предметы1. Запишем процедуру создания триггера, который выполняет удаление соответствующих данных из связанных таблиц:

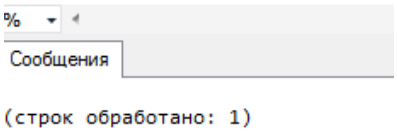
```
8
9   CREATE TRIGGER AAA
10  ON Предметы1
11  INSTEAD OF DELETE
12  AS
13  BEGIN
14  SET NOCOUNT ON;
15  DELETE Темы
16  FROM Deleted
17  WHERE Deleted.id_subject=Темы.id_subject
18  DELETE Предметы1
19  FROM Deleted
20  WHERE Deleted.id_subject= Предметы1.id_subject
21  END;
```

Если связь между таблицами содержит каскадное удаление и каскадное обновление, будет выдано сообщение об ошибке:

```
Невозможно создать INSTEAD OF DELETE или INSTEAD OF UPDATE TRIGGER "subject_DEL" для таблицы
"subjects", так как таблица содержит FOREIGN KEY с каскадными инструкциями DELETE или UPDATE.
```

Это значит, что каскадные инструкции уже обеспечили целостность данных, но если эти характеристики внешних ключей отключить, то при удалении записи о предмете триггер удалит записи в связанных таблицах.

```
23  DELETE FROM Предметы1
24  WHERE id_subject=10;
```




```
(строк обработано: 1)
```

Содержание отчета

1 Название работы
2 Цель работы
3 Перечень технических средств обучения
4 Порядок выполнения работы
5 Ответы на контрольные вопросы
6 Вывод

Варианты заданий

Варианты заданий представлены в практической работе № 39

Контрольные вопросы:

1. Что такое и для чего предназначены хранимые процедуры MS SQL Server?
2. Какая команда используется для создания хранимой процедуры?
3. Что такое триггер?
4. Отличия между хранимой процедурой и триггером?
5. Типы триггеров.
6. В чем различия триггеров DML и DDL?

Используемая литература

– Г.Н.Федорова Основы проектирования баз данных. М.: Академия, 2020

– Г.Н.Федорова Разработка, администрирование и защита баз данных. М.: Академия, 2018

– http://www.ikasteko.ru/page/filtracija-dannyh-sql-server

– http://palien.narod.ru/Documents/DB/Transact_SQL/page0016.htm