

ЛАБОРАТОРНАЯ РАБОТА №6

ИСПОЛЬЗОВАНИЕ ТРИГГЕРОВ И ХРАНИМЫХ ПРОЦЕДУР В MS SQL SERVER.

Цель: изучить хранимые процедуры и триггеры в базах данных, приобрести практические навыки создания хранимых процедур и триггеров в среде MS SQL Server Management Studio.

Содержание лабораторной работы:

1. Изучить теоретические сведения лабораторной работы.
2. Создать следующие хранимые процедуры (в своей базе):
 - a. Пример из теоретической части лабораторной работы.
 - b. Хранимую процедуру для поиска информации по названию компании.
 - c. Хранимую процедуру для поиска товаров по диапазону цен.
 - d. Хранимую процедуру для поиска заказов по дате заказа и диапазону дат заказа, доставки.
 - e. Хранимые процедуры по заданию варианта (стр. 9).
3. Создать триггер INSERT.
4. Создать триггер DELETE.
5. Создать триггер UPDATE.
6. Создать триггер, который при удалении записи из таблицы STOCK сначала удаляет все связанные с ней записи из таблицы ITEM, а затем удаляет саму запись из таблицы STOCK.
7. Создать триггер, с использованием временной таблицы Inserted.
8. Создать триггер DDL, который предотвратит удаление или изменение таблиц в базе данных.
9. Подготовиться к защите лабораторной работы (уметь написать собственную процедуру/триггер).

Краткий вспомогательный материал

Хранимая процедура – SQL-запрос, который имеет параметры, то есть он выполняется как обычная процедура. В зависимости от значения параметров хранимой процедуры мы получаем тот или иной результат запроса. В SQL сервере хранимые процедуры реализуют динамические запросы, выполняемые на стороне сервера.

Рассмотрим создание хранимых процедур при помощи команд языка SQL. Чтобы отобразить хранимые процедуры рабочей БД панели «Обозреватель объектов» нужно выбрать пункт «Программирование», а в нем – «Хранимые процедуры». Чтобы создать новую процедуру при помощи команд языка SQL нужно щелкнуть левой кнопкой мыши по кнопке на панели инструментов. В рабочей области окна сервера появится вкладка SQLQuery1.sql, где нужно набрать код новой процедуры, который имеет следующий синтаксис:

```
CREATE { PROC | PROCEDURE } <procedure_name> [ ; number  
] [ { @parameter data_type }  
[ = default ] [ OUT | OUTPUT ] [READONLY]] [ ,...n ]  
[ WITH [ RECOMPILE ] [ ,...n ] ] [ FOR REPLICATION ]
```

```
AS { <sql_statement> [;][ ...n ] } [;]
```

```
<sql_statement> ::=
```

```
{ [ BEGIN ] statements [ END ] }
```

Здесь:

- *procedure_name* – имя новой хранимой процедуры. Имена процедур должны соответствовать правилам, предъявляемым к идентификаторам, и должны быть уникальными.
- *number* – необязательное целое число, используемое для группирования процедур с одним именем. Все сгруппированные процедуры можно удалить, выполнив одну инструкцию DROP PROCEDURE.
- *@parameter* – параметр процедуры. В инструкции CREATE PROCEDURE можно объявить один или более параметров. При выполнении процедуры значение каждого из объявленных параметров должно быть указано пользователем, если для параметра не определено значение по умолчанию или значение не задано равным другому параметру. Хранимая процедура может иметь не более 2100 параметров. Определяет имя параметра, используя знак @ как первый символ. Имя параметра должно соответствовать правилам для идентификаторов. Параметры являются локальными в пределах процедуры; в разных процедурах могут быть использованы одинаковые имена параметров.
- *data_type* – тип данных параметра. Все типы данных, которые могут использоваться в качестве параметра хранимой процедуры Transact-SQL. Можно использовать определяемый пользователем табличный тип, чтобы объявить возвращающий табличное значение параметр в качестве параметра хранимой процедуры Transact-SQL.
- *default* – значение параметра по умолчанию. Если значение default определено, процедуру можно выполнить без указания значения соответствующего параметра. Значение по умолчанию должно быть константой или может равняться NULL.
- *OUTPUT* показывает, что параметр процедуры является выходным. Значение этого параметра можно получить при помощи инструкции EXECUTE. Используйте параметры OUTPUT для возврата значений коду, вызвавшему процедуру.
- *READONLY* указывает, что параметр не может быть обновлен или изменен в тексте процедуры. Если тип параметра является определяемым пользователем табличным типом, должно быть указано ключевое слово READONLY.
- *RECOMPILE* показывает, что компонент Database Engine не кэширует план выполнения процедуры и что процедура компилируется во время выполнения.
- *EXECUTE AS* определяет контекст безопасности, в котором должна быть выполнена хранимая процедура.

- *<sql_statement>* – одна или несколько инструкций языка SQL, которые будут включены в состав процедуры.

Если параметры сравниваются с какими-то полями или выражениями, то они должны иметь точно такой же тип данных, как эти поля или выражения. После создания процедура помещается в раздел «Хранимые процедуры» текущей БД на панели «Обозреватель объектов». Чтобы посмотреть информацию о хранимой процедуре необходимо выполнить команду

```
EXEC SP_HELPTEXT <Имя процедуры>
```

Хранимые процедуры могут быть запущены следующей командой

```
EXEC <Имя процедуры> [<Параметр1>, <Параметр2>, ...]
```

Здесь *<Имя процедуры>* – имя выполняемой процедуры;
<Параметр1>, *<Параметр2>*, ... – значения параметров.

Пример: Создание хранимой процедуры, которая выводит имена студентов со средним баллом, большим заданной величины:

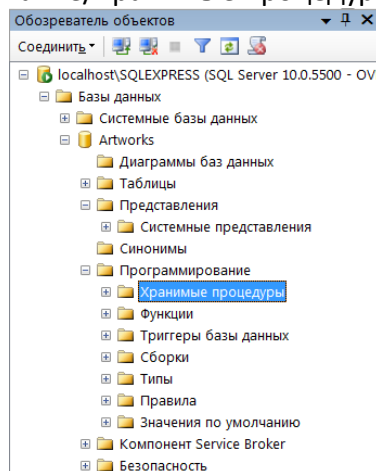
```
CREATE PROCEDURE СрБАЛЛ
@X Real AS
SELECT *
FROM Студенты
WHERE
(Оценка1 + Оценка2 + Оценка3) / 3 > @X
```

Команда вызова этой процедуры выглядит следующим образом:

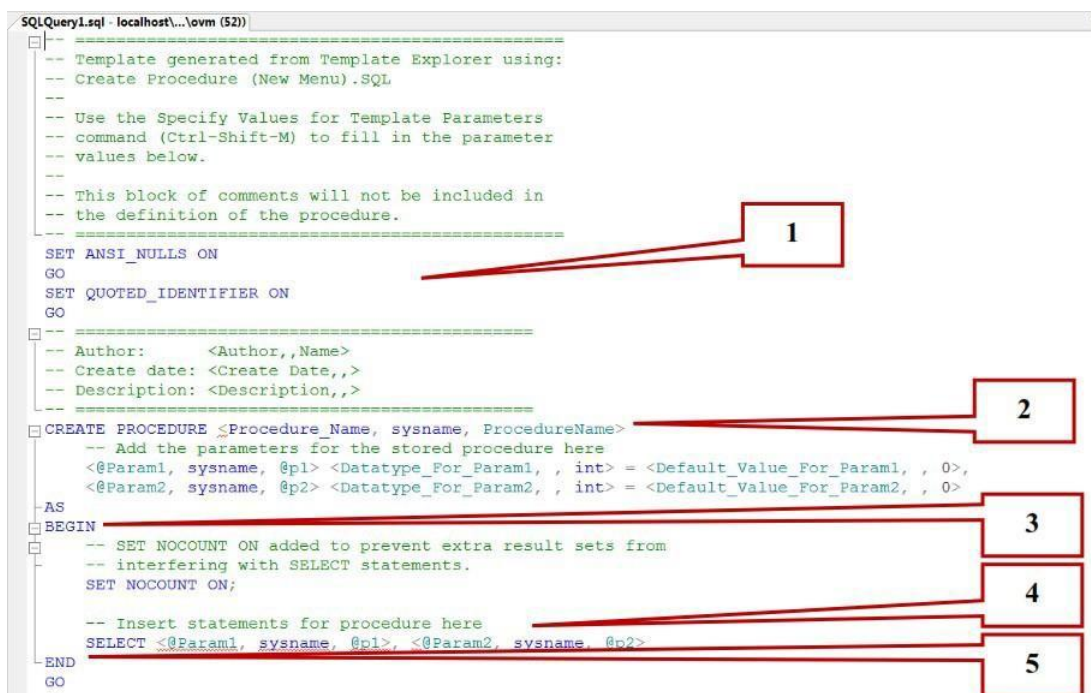
```
EXEC СрБАЛЛ 4
```

Команда выведет всех студентов, у которых средний балл больше 4.

Для работы с хранимыми процедурами в обозревателе объектов необходимо выделить папку «Программирование/Хранимые процедуры» базы данных.



Создадим процедуру, вычисляющую среднее трёх чисел. Для создания новой хранимой процедуры нужно щелкнуть правой кнопкой мыши по папке «Хранимые процедуры» и в появившемся меню выбрать пункт «Создать хранимую процедуру». Появится окно кода новой хранимой процедуры.



Хранимая процедура имеет следующую структуру:

1. Область настройки параметров синтаксиса процедуры. Позволяет настраивать некоторые синтаксические правила, используемые при наборе кода процедуры. В нашем случае это: SET ANSI_NULLS ON включает использование значений NULL в кодировке ANSI, SET QUOTED_IDENTIFIER ON включает возможность использования двойных кавычек для определения идентификаторов;
2. Область определения имени процедуры и параметров, передаваемых в процедуру. Определение параметров имеет следующий синтаксис:

@<Имя параметра> <Тип данных> = <Значение по умолчанию>

Параметры разделяются между собой запятыми;

3. Начало тела процедуры, обозначается служебным словом BEGIN;

4. Тело процедуры, содержит команды языка SQL;

5. Конец тела процедуры, обозначается служебным словом END.

В коде зелёным цветом выделяются комментарии. Они не обрабатываются сервером и выполняют функцию пояснений к коду. Строки комментариев начинаются с подстроки «--». Далее в коде, мы не будем отображать комментарии, они будут свёрнуты. Слева от раздела с комментариями будет стоять знак «+», щёлкнув по которому можно развернуть комментарий. Наберём код процедуры, вычисляющей среднее трёх чисел, как это показано на следующем рисунке.

```
SQLQuery1.sql - localhost\...\ovm (52)
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
--
CREATE PROCEDURE MeanValue
-- Add the parameters for the stored procedure here
@Value1 Real = 0,
@Value2 Real = 0,
@Value3 Real = 0
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result sets from...
SET NOCOUNT ON;

-- Insert statements for procedure here
SELECT 'MeanValue'=(@Value1 + @Value2 + @Value3) / 3
END
GO
```

1 points to `CREATE PROCEDURE MeanValue`
2 points to the parameter list: `@Value1 Real = 0, @Value2 Real = 0, @Value3 Real = 0`
3 points to the calculation: `SELECT 'MeanValue'=(@Value1 + @Value2 + @Value3) / 3`

Рассмотрим код данной процедуры более подробно:

1. `CREATE PROCEDURE MeanValue` определяет имя создаваемой процедуры как «MeanValue»;
2. `@Value1 Real = 0, @Value2 Real = 0, @Value3 Real = 0` – определение трех параметра процедуры Value1, Value2 и Value3. Тип параметров – Real, значения по умолчанию равны 0;
3. `SELECT 'Mean Value' = (@Value1+@Value2+@Value3)/3` – вычисление среднего и вывод результата с подписью «Среднее значение».

Для создания процедуры выполним ее код, нажав кнопку «Выполнить» на панели инструментов. В нижней части окна с кодом появится сообщение «Выполнение команд успешно завершено». Закройте окно с кодом, щёлкнув мышью по кнопке закрытия, расположенной в верхнем правом углу окна с кодом процедуры.

Проверим работоспособность созданной хранимой процедуры. Для запуска хранимой процедуры необходимо создать новый пустой запрос, нажав на кнопку «создать запрос» на панели инструментов. В появившемся окне с пустым запросом наберите команду `EXEC MeanValue 1, 7, 9` и нажмите кнопку «Выполнить» на панели инструментов.

```
SQLQuery2.sql - localhost\...\o... (52)*
EXEC MeanValue 1, 7, 9
```

Results pane:

MeanValue
1 5,666667

В нижней части окна с кодом появится результат выполнения новой хранимой процедуры: Среднее значение равно 5,66667.

Триггеры

При работе БД должна обеспечиваться целостность данных. При удалении записей из первичных таблиц автоматически должны удаляться связанные с ними записи из вторичных таблиц. В случае несоблюдения этого принципа со временем в БД накопится большое количество записей во вторичных таблицах, связанных с несуществующими записями в первичных таблицах, что приведёт к сбоям в работе БД и её засорению. Для обеспечения целостности данных в SQL Server используют триггеры.

Триггер – это сочетание хранимой в базе данных процедуры и события, которое заставляет ее выполняться. Такими событиями могут быть: ввод новой строки таблицы,

изменение значений одного или нескольких ее столбцов и (или) удаление строки таблицы. При любом из этих событий автоматически запускаются один или несколько заранее созданных триггеров, которые производят проверку запрограммированных в них условий, и, если они не выполняются, отменяют ввод, изменение или удаление, посылая об этом заранее подготовленное сообщение пользователю.

Триггеры похожи на процедуры и функции тем, что также являются именованными блоками и имеют раздел объявлений, выполняемый раздел и раздел обработки исключительных ситуаций. Подобно процедурам и функциям, триггеры хранятся как автономные объекты в базе данных.

Триггеры позволяют:

- реализовывать сложные ограничения целостности данных, которые невозможно реализовать через ограничения, устанавливаемые при создании таблицы;
- контролировать информацию, хранимую в таблице, посредством регистрации вносимых изменений и пользователей, производящих эти изменения;
- автоматически оповещать другие программы о том, что необходимо делать в случае изменения информации, содержащейся в таблице;
- публиковать информацию о различных событиях.

Триггеры также делятся на три основных типа.

- Триггеры DML активизируются предложениями ввода, обновления и удаления информации (INSERT, UPDATE, DELETE) до или после выполнения предложения, на уровне строки или таблицы.
- Триггеры замещения (instead of) можно создавать только для представлений (либо объектных, либо реляционных). В отличие от триггеров DML, которые выполняются в дополнение к предложениям DML, триггеры замещения выполняются вместо предложений DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами.
- Системные триггеры активизируется не на предложение DML, выполняемое над таблицей, а на системное событие, например, на запуск или останов базы данных. Системные триггеры срабатывают и на предложения DDL, такие как создание таблицы.

Триггеры INSERT запускаются (и выполняются) при каждой попытке создать новую запись в таблице с помощью команды INSERT. При попытке пользователя вставить новую запись в таблицу, SQL Server копирует эту запись в таблицу триггеров базы данных и в специальную таблицу, которая хранится в памяти и имеет имя *inserted*. Это означает, что ваша новая запись существует в двух таблицах — таблице триггеров и таблице *inserted*. Запись в таблице *inserted* должна полностью соответствовать записи в таблице триггеров.

Триггеры DELETE

При наличии триггера DELETE SQL Server переносит удаляемую запись в логическую таблицу в памяти с именем *deleted*. Таким образом, записи не исчезают полностью, и вы можете ссылаться на них в коде. Это удобно применять в бизнес-логике.

Специальную таблицу *deleted* можно сравнить с корзиной в операционной системе Windows, в которую переносятся файлы, удаляемые из системы. Основное отличие заключается в том, что после выполнения транзакции таблица *deleted* автоматически очищается от записей, а корзину требуется очищать вручную.

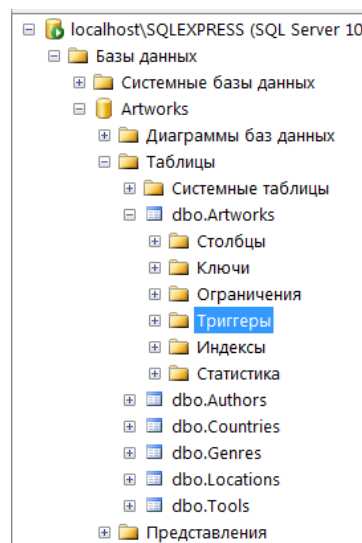
Триггеры UPDATE используются для ограничения инструкций обновления данных. Метод, используемый триггером UPDATE, представляет комбинацию методов, применяемых триггерами INSERT и DELETE. Помните, что триггер INSERT использует таблицу *inserted*, а триггер DELETE — таблицу *deleted*, триггер UPDATE использует обе таблицы.

Пример

Предположим, что существует база данных, содержащая информацию о клиентах, заказах и товарах. Каждый раз при выполнении заказа клиента, вам нужно вычитать эти товары в учете складских запасов (т.е. в таблице товаров), чтобы поддерживать правильный баланс, тогда тело триггера будет выглядеть так:

```
UPDATE Products
SET Products.instock = (Products.Instock - Inserted.Qty)
From Products Join Inserted ON Products.ProdID = Inserted.ProdID
```

Триггеры создаются отдельно для каждой таблицы и располагаются в обозревателе объектов в папке «Триггеры».



Структура триггера:

1. область определения имени триггера (Trigger_Name);
2. область, показывающая для какой таблицы, создаётся триггер (Table_Name);
3. область, показывающая, когда выполнять триггер (INSERT – при создании записи в таблице, DELETE – при удалении и UPDATE – при изменении);
4. тело триггера, содержащее команды языка T-SQL.

```
SQLQuery1.sql - localhost\...\ovm (52)

-- =====
-- Template generated from Template Explorer using:
-- Create Trigger (New Menu).SQL
--
-- Use the Specify Values for Template Parameters
-- command (Ctrl-Shift-M) to fill in the parameter
-- values below.
--
-- See additional Create Trigger templates for more
-- examples of different Trigger statements.
--
-- This block of comments will not be included in
-- the definition of the function.
-- =====

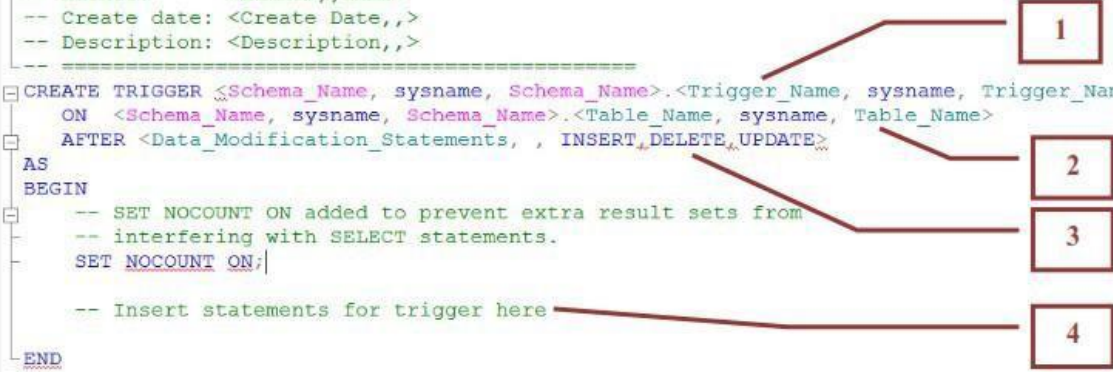
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- =====
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =====

CREATE TRIGGER <Schema_Name, sysname, Schema_Name>.<Trigger_Name, sysname, Trigger_Name>
ON <Schema_Name, sysname, Schema_Name>.<Table_Name, sysname, Table_Name>
AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE>
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here

END
GO
```



Как и триггеры DML (Data Manipulation Language), триггеры DDL (Data Definition Language) срабатывают в ответ на событие. Основное различие между триггерами DML и DDL заключается в событии, запускающем их. Триггеры DDL не срабатывают для инструкций INSERT, UPDATE и DELETE. Они предназначены для инструкций CREATE, ALTER, DROP и вообще для любых инструкций, модифицирующих структуру базы данных.

Этот новый тип триггера может оказаться полезным, когда требуется контролировать пользователей, модифицирующих структуру базы данных, их методы модификации, а также отслеживать изменения схемы. Предположим, что вы наняли временных служащих по контракту для работы с базой данных, и вам нужно, чтобы они не могли удалять столбцы, не ставя вас в известность. Для этого вы можете создать триггер DDL. Вы также можете позволить пользователю удалять любые столбцы и использовать триггер DDL для регистрации изменений в таблице.

Вариант	Задание
1-5	1. Выбрать все сведения о покупателях двух указанных компаний. Номера компаний вводятся как параметры.
	2. Получить информацию о количестве покупателей, оплативших заказ в каждом городе.
6-10	1. Выбрать всю информацию о покупателях города, которые совершили заказы. Название города вводится как параметр.
	2. Подсчитать количество покупателей в каждом городе, которые не сделали ни одного заказа
11-15	1. Выбрать всю информацию о покупателях, которые не совершили ни одного заказа.
	2. Подсчитать сумму сделок каждого покупателя из определенного города. Название города вводится как параметр.
16-20	1. Вывести список товаров (ID, описание, количество, сумма, дата доставки), которые были заказаны в двух заданных заказах. Номера заказов вводятся как параметры.
	2. Подсчитать количество каждого товара в заказах города. Название города вводится как параметр.
21-25	1. Получить сгруппированный по городу список с информацией (№заказа, дата заказа, дата доставки) за интервал времени. Отсортировать список по дате доставки. Интервал вводится как параметр.
	2. Подсчитать количество заказов по городам.
26-30	1. Получить сгруппированный по городу список с информацией (№заказа, дата заказа, дата доставки), для покупателей, сделавших больше n заказов. N вводится как параметр.
	2. Подсчитать среднюю цену заказов по городам.

Контрольные вопросы

1. Что такое хранимая процедура? Применение.
2. Как создаются хранимые процедуры?
3. Опишите синтаксис создания хранимой процедуры?
4. Как можно посмотреть информацию о хранимой процедуре?
5. Как осуществляется запуск хранимых процедур?
6. Опишите структуру хранимой процедуры?
7. Что такое триггер? Применение.
8. Как создаются триггеры?
9. Что позволяют триггеры?
10. Перечислите и опишите основные типы триггеров.
11. Опишите работу триггера.