Лабораторная работа № 11

Тема: Хранимые процедуры. Триггеры (типа AFTER)

1. Хранимые процедуры.

Хранимая процедура — это сохраненная коллекция инструкций языка SQL, которая может принимать и возвращать предоставленные пользователем параметры. Процедуры можно создавать для постоянного использования, для временного использования в одном сеансе (локальная временная процедура) или для временного использования во всех сеансах (глобальная временная процедура).

Хранимые процедуры могут также выполняться автоматически при запуске экземпляра SQL Server.

Стандартный максимальный размер хранимой процедуры не установлен.

Пользовательская хранимая процедура может быть создана только в текущей базе данных. Временные процедуры представляют собой исключение из этого правила, потому что они всегда создаются в базе данных tempdb. Если имя схемы не указано, используется схема по умолчанию, связанная с пользователем, который создает процедуру.

Инструкцию CREATE PROCEDURE нельзя объединять с другими инструкциями Transact-SQL в одном пакете.

По умолчанию параметры могут принимать значения NULL. Если параметр, имеющий значение NULL, используется в инструкции CREATE TABLE или ALTER TABLE при обращении к столбцу, не поддерживающему значения NULL, то компонент Database Engine возвращает ошибку. Чтобы предотвратить передачу значений NULL столбцу, который их не поддерживает, следует реализовать в процедуре соответствующую логику или передать столбцу значение по умолчанию при помощи ключевого слова DEFAULT инструкции CREATE TABLE или ALTER TABLE.

Для каждого столбца во временной таблице рекомендуется явно указывать атрибут NULL или NOT NULL. Если атрибуты NULL или NOT NULL не указаны в инструкции CREATE TABLE или ALTER TABLE, то способ назначения этих атрибутов столбцам компонентом Database Engine определяется параметрами ANSI_DFLT_ON и ANSI_DFLT_OFF. Если в контексте соединения выполняется хранимая процедура с настройками этих параметров, отличными от настроек соединения, в котором была создана процедура, столбцы таблицы, созданной для второго соединения, могут отличаться по признаку поддержки допустимости значений NULL и работать иначе. Если атрибут NULL или NOT NULL явно задан для каждого столбца, временные таблицы создаются с одним и тем же признаком поддержки допустимости значений NULL во всех соединениях, в которых выполняется хранимая процедура.

Имя хранимой процедуры может иметь до 128 символов. В хранимой процедуре могут использоваться все операторы SQL, кроме CREATE. По умолчанию разрешение на выполнение хранимой процедуры получает владелец базы данных, который может предоставлять эти права другим пользователям. Оператор создания хранимой процедуры имеет следующий формат:

```
CREATE PROCEDURE [владелец.]имя_процедуры [;версия]
[(@имя_параметра тип [=default] [OUTPUT]
[@имя_параметра тип [=default[ [OUTPUT]])]
[FOR REPLICATION | WITH RECOMPILE]!, ENCRYPTION]
AS
<операторы_Transact-SQL>
```

Использование в имени процедуры составляющей «;версия» (задаваемой целыми числами) позволяет получить группу одноименных процедур с одинаковыми именами и разными версиями, например: proc;1, proc;2, proc;3 и т.д. Это удобно тем, что всю такую группу процедур можно удалить одной следующей командой

```
DROP PROCEDURE proc
```

Вызов хранимой процедуры для выполнения осуществляется по ее имени.

Если хранимая процедура является частью группы, то вызов ее осуществляют с помощью команды EXEC. Например,

```
EXEC proc; 3
```

Пример 1. Создание и использование хранимой процедуры.

Создадим процедуру, выполняющую отбор записей из таблицы Employee, в которой условия отбора определяются с помощью двух параметров.

```
CREATE PROCEDURE procSelect
(@pl char(30), @p2 char (20))
AS
SELECT Name, Department, Cost FROM Employee
WHERE @pl=@p2
```

Обращение к процедуре может иметь следующий вид:

```
procSelect Name, 'Kooper'
```

Возможным результатом такого обращения будет строка вида:

```
Name Department Cost
Kooper Trade 3000
```

Пример 2:

```
CREATE PROCEDURE HumanResources.usp_GetEmployeesName
@NamePrefix char(1)
AS
BEGIN
SELECT BusinessEntityID, FirstName, LastName, EmailAddress
FROM HumanResources.vEmployee
WHERE FirstName LIKE @NamePrefix + '%'
ORDER BY FirstName
END
```

EXECUTE HumanResources.usp_GetEmployeesName 'A‘

Создание хранимой процедуры с помощью программы SQL Server Management Studio включает следующие действия.

1. Запуск этой программы из группы программ Microsoft SQL Server.

2. Выбор в открывшемся диалоговом окне программы сервера баз данных и базы данных.

3. Выбор элемента Stored Procedures (Хранимые процедуры) и выполнение команды его контекстного меню New Stored Procedure (Создать хранимую процедуру).

4. В открывшемся диалоговом окне Stored Procedure Properties (Свойства хранимой процедуры) в поле Text (Текст) ввод операторов Transact-SQL создаваемой процедуры и указание имени процедуры на месте фразы <PROCEDURE NAME> (рис. 13.3).

5. Нажатие кнопки Check Syntax (Проверка синтаксиса) для проверки отсутствия синтаксических ошибок и при необходимости корректировка операторов для устранения ошибок.

6. Нажатие ОК, в результате чего хранимая процедура создается и сохраняется.

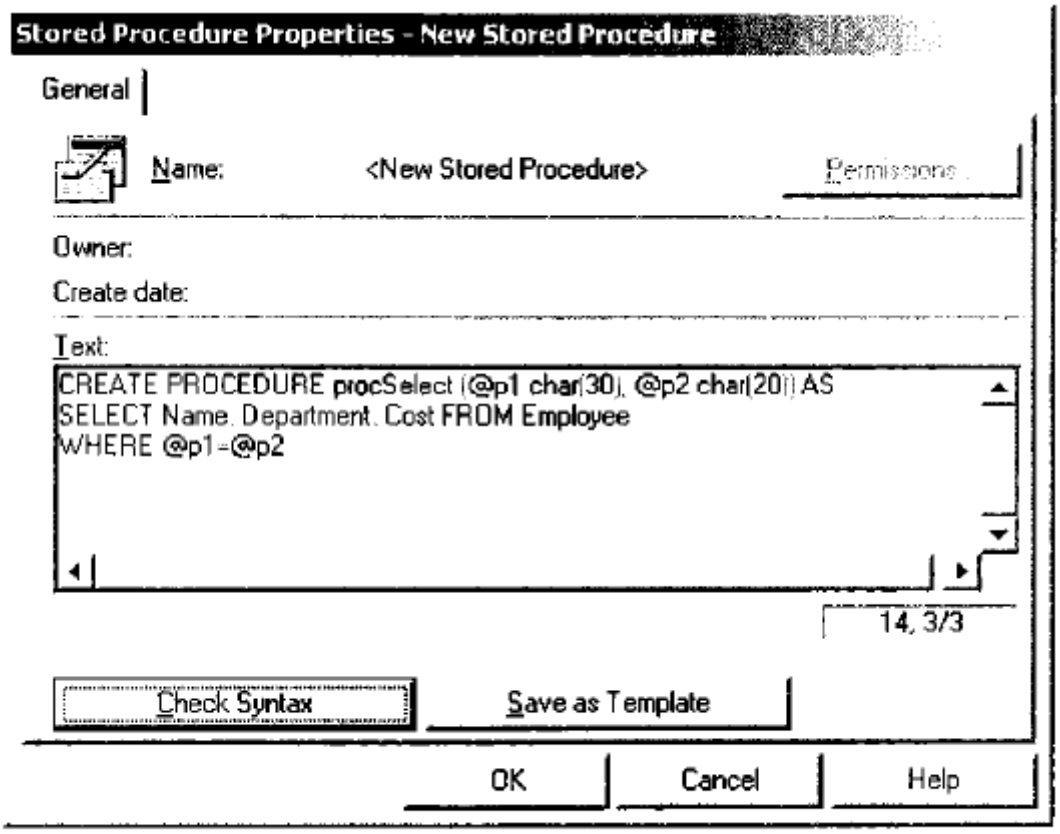


Рис. 13.3. Окно создания хранимой процедуры

2. Триггеры

Триггер — это особая разновидность хранимой процедуры, выполняемая автоматически при возникновении события на сервере базы данных.

Триггеры языка обработки данных **(триггеры DML)** выполняются по событиям, вызванным попыткой пользователя изменить данные с помощью языка обработки данных. Событиями DML являются процедуры INSERT, UPDATE или DELETE, применяемые к таблице или представлению.

Триггеры DDL срабатывают в ответ на ряд событий языка определения данных (DDL). Эти события прежде всего соответствуют инструкциям Transact-SQL CREATE, ALTER, DROP и некоторым системным хранимым процедурам, которые выполняют схожие с DDL операции.

Триггеры входа могут срабатывать в ответ на событие LOGON, возникающее при установке пользовательских сеансов.

Триггеры могут быть созданы непосредственно из инструкций Transact-SQL или из методов сборок, созданных в среде выполнения CLR платформы Microsoft.NET Framework, и переданы экземпляру SQL Server. SQL Server допускает создание нескольких триггеров для любой указанной инструкции.

Триггеры DML часто используются для соблюдения бизнес-правил и целостности данных. В SQL Server декларативное ограничение ссылочной целостности обеспечивается инструкциями ALTER TABLE и CREATE TABLE. Однако декларативное ограничение ссылочной целостности не обеспечивает ссылочную целостность между базами данных. Ограничение ссылочной целостности подразумевает выполнение правил связи между первичными и внешними ключами таблиц. Для обеспечения ограничений ссылочной целостности используйте в инструкциях ALTER TABLE и CREATE TABLE ограничения PRIMARY KEY и FOREIGN KEY. Если ограничения распространяются на таблицу триггера, они проверяются после срабатывания триггера INSTEAD OF и до выполнения триггера AFTER. В случае нарушения ограничения выполняется откат действий триггера INSTEAD OF, и триггер AFTER не срабатывает.

Для создания триггеров используется оператор CREATE, упрощенный формат которого имеет следующий вид:

```
CREATE TRIGGER [владелец, ] имя триггера
ON [владелец.]{имя_ таблицы | имя представления}
[WITH ENCRYPTION]
FOR [{AFTER | INSTEAD OF}]
{INSERT, UPDATE, DELETE}
AS
<операторы_SQL>
```

Параметр WITH ENCRYPTION (С шифрованием) служит для предотвращения возможности прочтения текста триггера после помещения его на сервер.

После ключевого слова FOR указывается тип триггера: стандартный (AFTER), запускаемый после выполнения пользователем изменений данных, либо выполняемый взамен команды, приведшей к запуску триггера (INSTEAD OF).

По умолчанию считается заданным тип AFTER.

Ключевые слова INSERT (Вставить), UPDATE (Обновить) и DELETE (Удалить) определяют операции, которые инициируют выполнение триггера.

SQL Server сохраняет текст триггера в таблице системного каталога syscomments.

Пример 3. Создание триггера вставки.

Рассмотрим создание триггера, который выполняется при вставке записи в таблицу schet. В таблице schet хранится информация о покупках и есть поле, где указывается номер накладной о покупке одной или нескольких единиц товара. Общая сумма стоимости по накладной хранится в отдельной таблице prod. Назначением создаваемого нами триггера является срабатывание на добавление записи в таблицу schet и выполнение добавления записи в таблицу prod, причем номер накладной в обеих таблицах должен совпадать.

```
CREATE TRIGGER schet_insert
ON schet
FOR INSERT
AS
INSERT INTO prod (schet, sum)
VALUES (vstavka.schet, vstavka.sum)
```

Пример 4:

```
CREATE TRIGGER Sales.trigCurrency
ON Sales.Currency
AFTER INSERT
AS
BEGIN
DECLARE @name nvarchar(50)
SELECT @name = Name
FROM inserted
IF len(@name) < 5
BEGIN
ROLLBACK TRANSACTION
END
END
```

Триггер к таблице Sales.Currency будет выполняться после вставки, проверяет значение вставляемое в столбце Name и убеждается, что имя составляет менее 5 символов. Если это правило нарушается, то откат транзакции, которая удаляет вновь вставленные строки.

Работа триггера

При работе триггера учитываются следующие условия и ограничения:

• Триггеры могут выполнять откат транзакции, когда не выполняются бизнес-правила

• Когда триггер выполняет откат транзакции, отменяется весь пакет

• Любые инструкции ROLLBACK TRANSACTION будут выполнены
• Любые изменения, которые происходят после отката, не откатываются
• Триггер может применяться только к одной таблице
• Триггеры выполняются только в текущей базе данных
• Триггеры должны принадлежать к той же схеме, что и целевая таблица
• Триггеры INSTEAD OF DELETE / UPDATE не могут быть созданы для таблицы, которая имеет каскадное определение внешнего ключа.

Типы триггеров

AFTER

AFTER указывает, что DML триггер срабатывает только тогда, когда все операции, указанные в SQL выражении, были успешно выполнены. Все ссылочные действия и проверки ограничений также должны быть выполнены.

INSTEAD OF

Указывает, что триггер DML выполняется вместо триггера SQL. INSTEAD OF не может быть указан для DDL или триггеров входа.

Замечание.

Для обеспечения высокой надежности работы с БД и возможности восстановления баз данных и приложений целесообразно иметь резервные копии хранимых процедур, триггеров, определений таблиц и общей структуры серверной части приложений SQL Server.

Триггеры представляют собой весьма полезное и в то же время опасное средство. Так, при неправильной логике работы триггера можно легко уничтожить целую базу данных. Поэтому требуется очень тщательно отлаживать триггеры и проверять логику их работы.

Задания для самостоятельного выполнения

⇒ **Подготовительное задание:**

Средствами SQL Server Management Studio(SSMS) создайте **новую БД** со следующими нижеуказанными таблицами (связи не устанавливать)

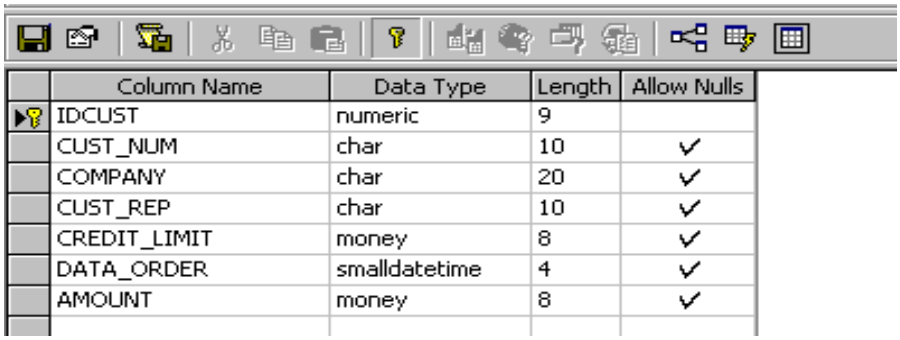
Внимание! Для быстроты таблицы создавать через конструктор (интерфейс) SSMS а не через запросы типа Create Table !

Название БД должно содержать Группу и Фамилию + ЛР_11
например :(***01_Ivanov_LR_11***)

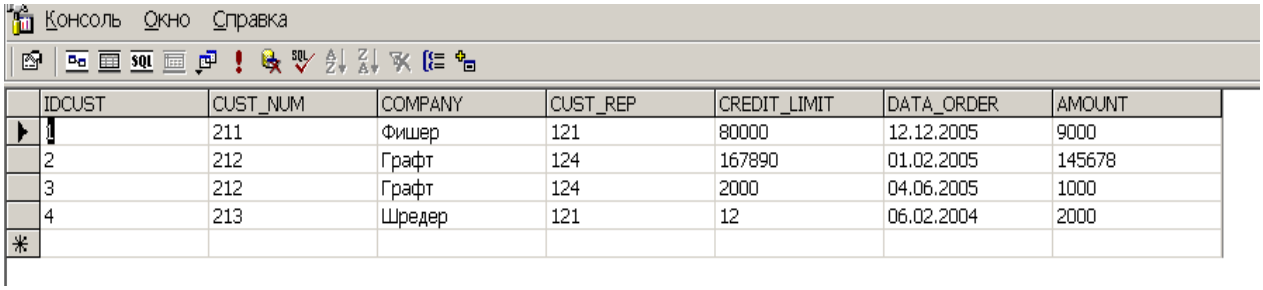
Итак, требуется создать следующие таблицы:

CUSTOMERS

IDCUST	**CUST_ NUM**	**COMPANY**	**CUST_REP**	**CREDIT_ LIMIT**	**DATA_ ORDER**	**AMOUNT**
Номер записи	Идентификатор клиента	Имя клиента	Идентификатор служащего	Лимит кредита	Дата заказа	Сумма заказа



Column Name	Data Type	Length	Allow Nulls
IDCUST	numeric	9	
CUST_NUM	char	10	✓
COMPANY	char	20	✓
CUST_REP	char	10	✓
CREDIT_LIMIT	money	8	✓
DATA_ORDER	smalldatetime	4	✓
AMOUNT	money	8	✓

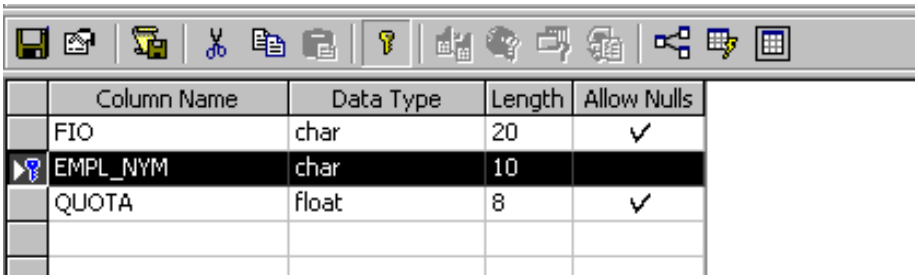


Консоль Окно Справка

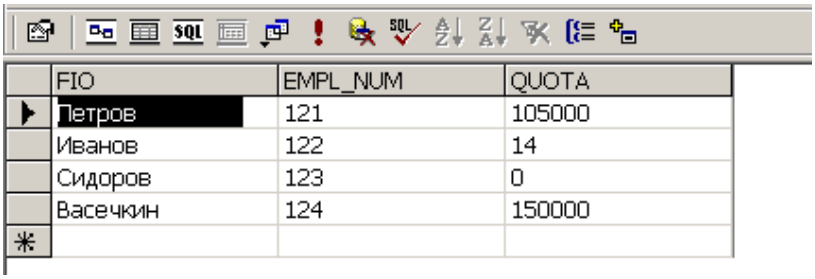
IDCUST	CUST_NUM	COMPANY	CUST_REP	CREDIT_LIMIT	DATA_ORDER	AMOUNT
1	211	Фишер	121	80000	12.12.2005	9000
2	212	Графт	124	167890	01.02.2005	145678
3	212	Графт	124	2000	04.06.2005	1000
4	213	Шредер	121	12	06.02.2004	2000

SALESPERS

FIO	**EMPL_NUM**	**QUOTA**
Имя служащего	Идентификатор служащего	Плановый объем продаж служащего



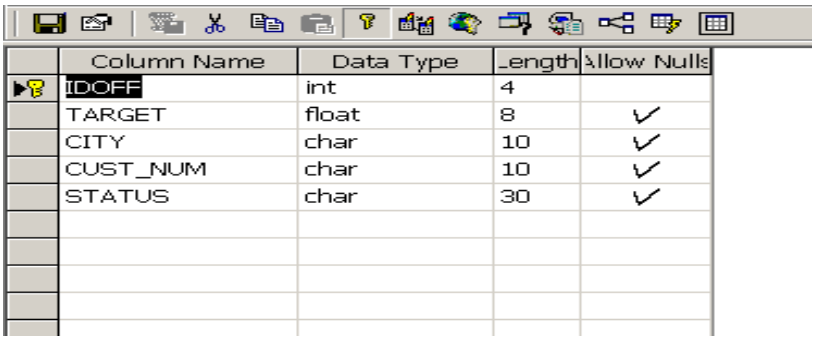
Column Name	Data Type	Length	Allow Nulls
FIO	char	20	✓
EMPL_NYM	char	10	
QUOTA	float	8	✓



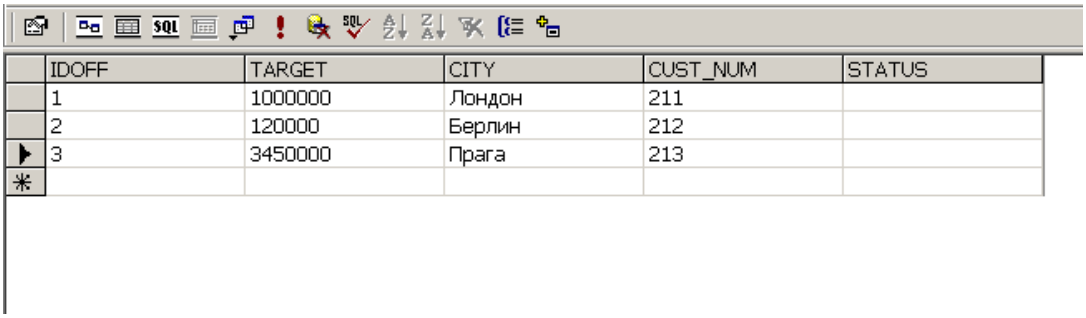
FIO	EMPL_NUM	QUOTA
Петров	121	105000
Иванов	122	14
Сидоров	123	0
Васечкин	124	150000

OFFICES

IDOFF	**TARGET**	**CITY**	**CUST_NUM**	**STATUS**
Номер записи	Объем продаж	Город, в котором расположен офис клиента	Идентификатор клиента	Примечание **(не заполнять)**



Column Name	Data Type	_ength	llow Nulls
IDOFF	int	4	
TARGET	float	8	✓
CITY	char	10	✓
CUST_NUM	char	10	✓
STATUS	char	30	✓



IDOFF	TARGET	CITY	CUST_NUM	STATUS
1	1000000	Лондон	211	
2	120000	Берлин	212	
3	3450000	Прага	213	

⇒ **Основные задания:**

1. Написать хранимую процедуру GET_CUST, которая получает идентификатор клиента и возвращает его имя (поле COMPANY таблицы CUSTOMERS), имя закрепленного за ним служащего (поле FIO таблицы SALESPERS) и название города, в котором расположен офис этого клиента (поле CITY таблицы OFFICES). Первый из передаваемых параметров - входной, остальные три – выходные (используются для передачи запрошенных данных вызывающей процедуре). Продемонстрировать вызов данной процедуры из другой процедуры Transact-SQL. Например, GET_CUST1 (идентификатор_клиента).

2. Написать хранимую процедуру CHK_TOT, которая получает идентификатор клиента и вычисляет общую стоимость его заказов (поле AMOUNT таблицы CUSTOMERS) и в зависимости от того, превысит ли эта сумма 30 000$, заносит в поле STATUS таблицы OFFICES одно из двух примечаний – “большой объем заказов”, ”малый объем заказов”.

3. Написать хранимую процедуру для добавления данных о новом клиенте в таблицу OFFICES.

- Добавить новую строку в таблицу OFFICES. Разрешается добовление только для клиентов, уже имеющих запись в таблице CUSTOMERS.

- Обновить запись в SALESPERS, увеличив поле QUOTA для соответствующего служащего (для каждого, с которым работает клиент) на величину объема продаж добавленной в предыдущем пункте записи. Плановый объем продаж служащего не может быть увеличен более чем на определенную величину.

Если сумма заказов рассматриваемого клиента составляет менее 20 000$ (вызов процедуры CHK_TOT, которая должна возвращать эту сумму, например, в качестве выходного параметра), то величина объема продаж (поле TARGET таблицы OFFICES) будет добавлена к плану служащего. Если сумма заказов рассматриваемого клиента равна 20 000$ к плану будут добавлены фиксированные 20 000$. В противном случае – запретить добавление новой записи в OFFICES и обновление в SALESPERS. Предусмотреть промежуточный вывод суммы заказов рассматриваемого клиента.

4. Создать триггер, распространяющий любое обновление столбца EMPL_NUM в таблице SALESPERS на столбец CUST_REP таблицы CUSTOMERS.

5. Создать триггер, который запускается каждый раз, когда запись вставляется в таблицу CUSTOMERS или модифицируется. Если заказ сделан не в первые 15 дней месяца, то запись не принимается.

Для выделения определенной части даты (дня) в виде целого значения используйте функцию DATEPART(date_part, date), где date_part- **day|month|year.**

Например:

```
SELECT DATEPART(month, GETDATE()) AS 'Month Number'
GO

Month Number
------------
11
```

Литература:

1. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. - Базы данных. Учебник для высших учебных заведений (6-е изд.) - 2009.