Сценарии и пакеты

Общие сведения о сценариях

В общем случае **сценарии** (**script**) пишутся для решения от начала и до конца какой-то определенной задачи. С этой целью в один сценарий объединяются несколько команд. Примерами могут служить сценарии, используемые для построения баз данных (они часто применяются при установке систем). Сценарий не является сценарием до тех пор, пока он не сохранен в файле, из которого он может быть извлечен и использован в дальнейшем. SQL-сценарии хранятся в текстовых файлах. Любой сценарий всегда рассматривается как единое целое. Либо выполняется весь сценарий целиком, либо совсем ничего не выполняется. В сценариях могут использоваться системные функции и локальные переменные.

Пример

```
USE Northwind
DECLARE @Ident int
INSERT INTO Orders (CustomerID,OrderDate)
VALUES ('ALFKI', DATEADD(day,-1,GETDATE()))
SELECT @Ident = @@IDENTITY
INSERT INTO [Order Details] (OrderID, ProductID, UnitPrice, Quantity)
VALUES (@Ident, 1, 50, 25)
SELECT 'The OrderID of the INSERTed row is ' + CONVERT(varchar(8),@Ident)
```

Здесь для решения задачи целиком объединено шесть команд. Используются системные функции, локальные переменные, операторы USE и INSERT. Оператор SELECT был использован двумя способами - в рамках присваивания и обычным способом (для выборки). Все эти команды выполняют одну задачу - добавляют записи в базу данных.

Замечание. То, что раньше в SQL Server называлось глобальными переменными (да и сейчас довольно часто именно так и называется в документации) на самом деле не является переменными вовсе (в них нельзя хранить все, что угодно), и будет более корректным называть их "системными функциями" или "системными функциями без аргументов". Везде далее будет использоваться термин "системные функции".

Оператор USE

С помощью оператора USE устанавливается текущая база данных. В дальнейшем при выполнении операций над объектами этой базы данных уже не нужно указывать ее имя. После использования оператора USE эта база становится используемой по умолчанию. Это совсем не означает, что оператор USE всегда надо включать в сценарий. Если создается сценарий общего назначения (т. е. предназначенный для использования с произвольной базой), тогда вполне разумно будет не включать в него оператор USE.

Объявление переменных

Для объявления переменных используется оператор DECLARE, который имеет следующий синтаксис:

```
DECLARE список-объявлений-переменных
```

где объявление переменной имеет вид

```
@имя-переменной тип-переменной
```

Можно объявить одну или несколько переменных одновременно. Использование отдельного оператора DECLARE для каждой объявляемой переменной дает тот же результат, что и разделение переменных запятыми в общем операторе. Применяемый стиль зависит предпочтений программиста, но независимо от выбранного способа объявления значение переменной будет равно NULL до тех пор, пока ей не будет присвоено значение.

Присвоение значений переменным

Существует два способа присваивания значений переменным. Можно использовать операторы SELECT или SET. Функционально они практически эквивалентны, за исключением того, что прямо внутри оператора SELECT можно использовать в качестве присваиваемых значений содержимое полей.

Оператор SET традиционно используется для присвоения значений переменным подобно
тому, как это делается во многих процедурных языках. Типичным примером может служить следующий фрагмент кода:

```
USE Northwind
SET @Tota1Cost = 10
SET @Tota1Cost = @UnitCost * 1.1
```

При помощи оператора SET нельзя присвоить переменной значение, полученное непосредственно из запроса – нужно отделить запрос от SET. Например, следующий сценарий:

```
DECLARE @Test mînåy
SET @Test = MAX (UnitPrice) FROM [Order Detai1s]
SELECT @Test
```

вызовет ошибку, а сценарий

```
USE Northwind
DECLARE @Test money
SET @Test = (SELECT MAX (UnitPrice) FROM [Order Details])
SELECT @Test
```

будет работать.

SELECT обычно используется в том случае, когда переменной непосредственно нужно присвоить значение, полученное в результате запроса. Такой пример, как предыдущий, чаще всего реализуются с использованием SELECT:

```
USE Northwind
DECLARE @Test mînåó
SELECT @Test = MAX (UnitPrice) FROM [Order Details]
SELECT @Test
```

Таким образом, можно сделать следующие выводы:

- SET используется в случае, если переменной просто присваивается известное на момент операции значение (константа) или же одной переменной присваивается значение другой;
- SELECT лучше использовать в том случае, когда переменной необходимо присвоить результат выполнения запроса.

Обзор системных функций

Существует около 30 системных функций без параметров. Некоторые из них являются особо интересными и полезными:

- @@ERROR. Возвращает номер ошибки последнего оператора Transact-SQL, выполненного в текущем подключении. При отсутствии ошибки возвращает 0.
- @@IDENTITY. Возвращает последнее identity-значение, которое было вставлено в результате выполнения последнего оператора INSERT или SELECT INTO.
- @@ROWCOUNT. Возвращает число строк, полученных в результате выполнения последнего оператора.

Общие сведения о пакетах

Пакет (batch) - это несколько объединенных в одну логическую группу операторов Transact-SQL. Все операторы в рамках пакета комбинируются в единый план исполнения (execution plan) таким образом, что пока все операторы не будут успешно проанализированы синтаксическим анализатором, ни один из операторов пакета не будет исполняться. Если на этапе синтаксического анализа какой-либо оператор из пакета оказывается ошибочным, то ни один оператор пакета не выполняется. Если какой-либо оператор пакета вызывает ошибку на этапе выполнения программы, то выполняются все операторы пакета вплоть до ошибочного.

Для того чтобы разделить сценарий на несколько пакетов, необходимо использовать оператор GO. Оператор GO:

- должен писаться в отдельной строке (ничего, кроме комментариев, не должно следовать за ним в этой же строке); есть исключения, которые будут рассматриваться ниже;
- все операторы от начала сценария (или ближайшего предыдущего оператора GO) и до данного оператора GO компилируются в один план исполнения и пересылаются на сервер отдельно от других пакетов, т. е. ошибка в одном пакете не может помешать выполнению другого пакета;
- это не команда Transact-SQL, а директива, которую распознают различные утилиты с командным интерфейсом SQL Server (OSQL, ISQL и Анализатор Запросов).

Сценарий можно построить и таким образом, чтобы выполнение одного пакета зависело от завершения другого.

Ошибки, встречающиеся в пакетах, можно разделить на две группы:
- синтаксические ошибки. Если синтаксический анализатор находит синтаксическую ошибку в запросе, обработка данного пакета сразу же прекращается. Синтаксический анализ выполняется перед компиляцией пакета и его запуском на выполнение, наличие ошибки при этом означает, что никакая часть пакета не будет запущена на выполнение, пока ошибка не будет устранена.
- ошибки, возникающие при выполнении программы. Эти ошибки имеют несколько другие свойства. Любой оператор, который выполнился до возникновения ошибки, считается уже выполненным и все, что было сделано в результате его выполнения остается действительным, конечно с учетом того, что он является частью незавершенной транзакции.

Пример

```
USE Northwind
DECLARE @MyVarchar varchar(50) -- Только этот DECLARE остается от данного пакета!
SELECT @MyVarchar = 'Привет, я дома ...'
PRINT 'Выполним первый пакет ...'
GO
PRINT @MyVarchar  -- Приводит к ошибке, поскольку переменная @MyVarchar
                  -- в этом пакете не объявлена
PRINT 'Выполним второй пакет ...'
GO
PRINT 'Выполним третий пакет ...'   -- Этот пакет выполняется
                                    -- несмотря на предшествующие ошибки
GO
```

Когда используются пакеты?

Цели применения пакетов различны, но обычно они используются в ситуациях, когда в сценарии необходимо что-нибудь выполнить либо перед, либо отдельно от всего остального. Чаще всего пакеты используются для явного задания предшествования, – когда необходимо полностью завершить решение одной задачи перед тем, как начнет решаться другая.

Правила использования пакетов

Для использования пакетов применяются следующие правила.

1. Инструкции CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE SCHEMA, CREATE TRIGGER и CREATE VIEW не могут быть объединены с другими инструкциями в пакете. Инструкция CREATE должна быть первой инструкцией в пакете. Все последующие инструкции в этом пакете рассматриваются как часть определения первой инструкции CREATE.
2. В одном и том же пакете нельзя сначала изменить таблицу и затем обратиться к новым столбцам.
3. Если инструкция EXECUTE — первая инструкция в пакете, ключевое слово EXECUTE не требуется. Ключевое слово EXECUTE требуется, если инструкция EXECUTE не является первой инструкцией в пакете.
4. Если необходимо удалить объект, то следует поместить оператор DROP в отдельный пакет или, по крайней мере, в пакет с другими операторами DROP. Объяснение этому следующее. Если необходимо создать объект с тем именем, которое перед этим удалили, то CREATE вызовет ошибку во время синтаксического анализа, несмотря на то, что оператор DROP уже был выполнен. Это означает, что следует запускать DROP в отдельном пакете, который предшествует CREATE, чтобы удаление старого объекта завершилось к тому моменту, когда начнет выполняться создание нового объекта с таким же именем.

Пример пакетного задания.

```
USE AdventureWorks;
GO
CREATE VIEW dbo.vProduct
AS
SELECT ProductNumber, Name
FROM Production.Product;
GO
SELECT *
FROM dbo.vProduct;
GO
```

В данном примере создается представление. Так как инструкция CREATE VIEW должна быть единственной в пакете, требуются команды GO, чтобы изолировать инструкцию CREATE VIEW от предшествующей и последующей инструкций USE и SELECT.