Процедуры

- Для реализации логики приложения на стороне базы данных
  - Создание хранимых процедур и функций
  - Создание триггеров

Процедуры

Хранимая процедура – это набор операторов T-SQL, который компилируется системой SQL Server в единый "*план исполнения*".

Переменные

- Имя переменной начинается со знака @
- ~~DECLARE @a, @b, @c int~~
- DECLARE @a int, @b int, @c int
- DECLARE @a int = 5, @b int = 0, @c int

Типы данных, определяемые пользователем

```
CREATE TYPE my_type
FROM varchar(11) NOT NULL ;

DECLARE @a my_type;
```

Скалярные переменные

```
DECLARE @var_name var_type, …
SET @var_name = var_value;
SELECT @var_name = var_value;
SELECT @var_name;
SELECT @var_name=id FROM Table1;
```

(последнее значение)

Скалярные переменные

```
DECLARE @var int;
SET @var = 5;
SELECT @var = 31;
SELECT @var;
SELECT @var=id FROM Table1;
```

(последнее значение)

Скалярные переменные

```
SELECT { @local_variable
    { = | += | -= | *= | /= | %= | &= | ^= | |= }
   expression } [ ,...n ] [ ; ]

SELECT @id+ = 2;
```

Составной оператор присваивания

+= сложить и присвоить

-= вычесть и присвоить

*= умножить и присвоить

/= разделить и присвоить

%= получить остаток от деления и присвоить

&= выполнить побитовое И и присвоить

^= выполнить побитовое исключающее ИЛИ и присвоить

|= выполнить побитовое ИЛИ и присвоить

SET vs SELECT

SELECT @var=Field FROM T

SET @var=(SELECT Field FROM T)

Табличные переменные

```
CREATE TYPE Location AS TABLE
    ( LocationName VARCHAR(50)
    , CostRate INT );

DECLARE @table1 Location;

DECLARE @table_var table(
    id int
    , name char(20));
```

Табличные переменные

```
SET @table_name = Table1;
SELECT @table_name = var_value;
SELECT @table_name;
```

Табличные переменные

```
INSERT @table_name SELECT * FROM Table1;
SELECT * FROM @table_name;
```

Использование псевдонима

```
SELECT EmployeeID, DepartmentID
FROM @MyTableVar m
JOIN Employee on (m.EmployeeID
   =Employee.EmployeeID AND
   m.DepartmentID =
    Employee.DepartmentID);
```

Табличные переменные

- Автоматически очищаются в конце функции, хранимой процедуры или пакета, где они были определены
- Табличная переменная не участвует в транзакции.
- Не подходят для хранения значительных объёмов данных (>100 строк).

Группировка

```
BEGIN
    {
   sql_statement | statement_block
    }
END;
```

Условный оператор

IF Boolean_expression { sql_statement | statement_block }

[ELSE { sql_statement | statement_block }]

Условный оператор

```
IF (SELECT MAX(id) FROM Table)<32
    SELECT ‘Можно еще добавить’
ELSE SELECT ‘Больше уже нельзя’;
```

Оператор цикла

WHILE Boolean_expression

{ sql_statement | statement_block | BREAK | CONTINUE }

BREAK

Приводит к выходу из ближайшего цикла WHILE.

CONTINUE

Выполняет новый шаг цикла WHILE, не учитывая все команды, следующие после ключевого слова CONTINUE.

Оператор цикла

```
WHILE (SELECT AVG(Price) FROM Product) < $300
BEGIN
   UPDATE Product
      SET Price = Price * 2;
IF (SELECT MAX(Price) FROM Product) > $500
      BREAK
   ELSE
      CONTINUE
END
PRINT 'Too much …';
```

Обработка ошибок

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
        [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

Обработка ошибок

```
BEGIN TRY
    SELECT 1/0;
END TRY
BEGIN CATCH
   SELECT 'На ноль делить нельзя!';
END CATCH;
```

Процедуры

CREATE PROC [EDURE] *procedure_name*
    [{ *@parameter data_type* }
        [**=** *default*] [OUTPUT]
    ] [**,**...*n*]
AS *sql_statement*

Создание простой процедуры

```
CREATE PROCEDURE SimpleProc AS
UPDATE students
    SET salary=salary*1.5;
```

Изменение простой процедуры

```
ALTER PROCEDURE SimpleProc AS
UPDATE students
    SET salary=salary*1.7;
```

Создание процедуры с удалением

```
IF OBJECT_ID (' SimpleProc ') IS NOT NULL
DROP PROCEDURE  SimpleProc;

CREATE PROCEDURE SimpleProc AS
UPDATE students
   SET salary=salary*1.5;
```

Процедуры: несколько действий

```
CREATE PROCEDURE ExampleProc AS
BEGIN
   DECLARE @default_salary INT
   SET @default_salary = (SELECT …)
END
```

Создание процедуры с параметрами

```
CREATE PROCEDURE ExampleProc (
    @id INT,
    @name VARCHAR(32)
) AS
BEGIN
    DECLARE @default_salary INT
    SET @salary = (SELECT …)
END
```

Вызов процедур

- Без параметров

  EXECUTE SimpleProc

  EXEC SimpleProc
- С параметрами

  EXECUTE ExampleProc 1, ‘string’

Параметры по умолчанию и внешние

```
CREATE PROCEDURE ExampleProc (
    @id INT = 0,
    @name VARCHAR(32) = '',
    @salary INT OUTPUT
) AS
BEGIN
    DECLARE @default_salary INT
    SET @salary = (SELECT …)
END
```

Создание процедуры с параметрами

```
CREATE PROCEDURE GetUnitPrice @prod_id int,
   @unit_price money OUTPUT
   AS SELECT @unit_price = UnitPrice
   FROM Products WHERE ProductID = @prod_id


DECLARE @price money
EXECUTE GetUnitPrice 77, @price OUTPUT
SELECT @price
```

Параметры: внутренние и внешние

```
CREATE PROCEDURE ExampleProc (
    @salary INT OUTPUT,
    @id INT = 0,
    @name VARCHAR(32) = '',
```

```
DECLARE @s int;
EXEC ExampleProc @s OUTPUT, 3, ‘any_string‘
EXEC ExampleProc @s OUTPUT
```

Параметры

```
CREATE PROCEDURE ExampleProc (
   @id INT = 0,
   @name VARCHAR(32) = '',
   @salary INT OUTPUT


EXEC PROCEDURE ExampleProc 3
DECLARE @proc_name varchar(30) SET
   @proc_name = 'sp_who' EXEC
   @proc_name
```

Процедура с циклом

```
CREATE TABLE mytable (
    column1 int,
    column2 char(10) )

CREATE PROCEDURE InsertRows @start_value int
    AS BEGIN DECLARE @loop_counter int,
    @start int
    SET @start = @start_value – 1
    SET @loop_counter = 0
    WHILE (@loop_counter < 5) BEGIN
    INSERT INTO mytable VALUES (@start + 1, ‘new row’)
    PRINT (@start)
    SET @start = @start + 1
    SET @loop_counter = @loop_counter + 1
    END END
```

Процедура с циклом

- EXECUTE InsertRows 1 GO
- SELECT * FROM mytable

column1 column2
---
1 new row
2 new row
3 new row
4 new row
5 new row

Выход из процедуры RETURN

```
CREATE PROCEDURE GetUnitPrice
   @prod_id int
   AS
   IF @prod_id IS NULL
    BEGIN PRINT ‘Enter a product ID number’
   RETURN
   END
   ELSE …
```

Передача имени таблицы

```
DECLARE @SQL varchar(8000),
@table_name varchar(20)='dbo.Employees'

SET @SQL = 'SELECT * FROM ' + @table_name
exec(@SQL)
```

Имя таблицы – параметр процедуры

```
CREATE PROCEDURE dbo.mysample (
    @tabname varchar(50)
    ,@somevalue char(3) )
AS
begin
declare @sql varchar(400)
set @sql='DELETE FROM '+ @tabname + ' where id>'+ CHAR(39) + @somevalue + CHAR(39)
exec(@sql);
end
```

SELECT-выражения в блоках

- Должны возвращать только одно значение!

```
SET var_name = (SELECT column_name
  FROM …)
```

- При необходимости работать со множеством записей используйте курсор.

Курсоры

- ***Курсор*** в SQL – это область в памяти базы данных, которая предназначена для хранения запроса SQL.
- В памяти сохраняется и строка данных запроса, называемая текущим значением, или текущей строкой *курсора*.
- Указанная область в памяти поименована и доступна для прикладных программ.

Курсоры

- DECLARE – создание или *объявление курсора* ;
- OPEN – *открытие курсора*, т.е. наполнение его данными;
- FETCH – *выборка из курсора* и *изменение* строк данных с помощью курсора;
- CLOSE – *закрытие курсора* ;
- DEALLOCATE – *освобождение курсора*, т.е. удаление курсора как объекта.

Создание курсора

DECLARE имя_курсора

[INSENSITIVE][SCROLL] CURSOR FOR

SELECT_оператор

[FOR { READ_ONLY | UPDATE

[OF имя_столбца[,...n]]}]

Курсоры

```
DECLARE cursor_name CURSOR FOR
   select_statement
OPEN cursor_name
FETCH [NEXT] cursor_name [INTO variable_list]
CLOSE cursor_name
DEALLOCATE cursor_name
```

Виды курсоров

- *последовательные*
- *прокручиваемые*

- Статические
- Динамические

Статический курсор

- В схеме со ***статическим курсором*** информация читается из базы данных один раз и хранится в виде моментального снимка (по состоянию на некоторый момент времени), поэтому изменения, внесенные в базу данных другим пользователем, не видны. На время *открытия курсора* сервер устанавливает блокировку на все строки, включенные в его полный результирующий набор.
- *Статический курсор* не изменяется после создания и всегда отображает тот набор данных, который существовал на момент его *открытия*.

Создаем статический курсор

DECLARE *cursor_name*
INSENSITIVE [SCROLL]
CURSOR FOR *select_statement*

Динамический курсор

- Динамические курсоры отражают все изменения строк в результирующем наборе при прокрутке курсора. Значения типа данных, порядок и членство строк в результирующем наборе могут меняться для каждой выборки. Все инструкции UPDATE, INSERT и DELETE, выполняемые пользователями, видимы посредством курсора. Обновление видимы сразу, если они сделаны посредством курсора.

Создаем динамический курсор

DECLARE *cursor_name* [SCROLL]

CURSOR FOR *select_statement*

[FOR { READ ONLY | UPDATE

[OF *column_name* [,...*n*]] }]

Создаем и открываем курсор

```
DECLARE my_cursor CURSOR FOR

  SELECT id, name FROM Table1;


OPEN  my_cursor
```

Считываем текущую строку в перменные

```
DECLARE @id INT, @name VARCHAR(32);

FETCH FROM my_cursor INTO @id, @name
```

Функция @@FETCH_STATUS

Функция @@FETCH_STATUS возвращает:

- 0, если выборка завершилась успешно;
- -1, если выборка завершилась неудачно вследствие попытки выборки строки, находящейся за пределами курсора ;
- -2, если выборка завершилась неудачно вследствие попытки обращения к удаленной или измененной строке.

Проходим по всему курсору

```
FETCH my_cursor INTO @id, @name

WHILE (@@FETCH_STATUS = 0) BEGIN
   <do something>
   FETCH FROM my_cursor INTO @id, @name
END
```

Закрываем курсор и освобождаем память

```
CLOSE my_cursor

DEALLOCATE my_cursor
```

Последовательный курсор

```
DECLARE Employee_Cursor CURSOR FOR
SELECT EmployeeID, Title
FROM AdventureWorks2012.HumanResources.Employee
WHERE JobTitle = 'Marketing Specialist';
OPEN Employee_Cursor;
FETCH NEXT FROM Employee_Cursor;
WHILE @@FETCH_STATUS = 0
  BEGIN
    FETCH NEXT FROM Employee_Cursor;
  END;
CLOSE Employee_Cursor;
DEALLOCATE Employee_Cursor;
```

Прокручиваемый курсор

DECLARE *cursor_name* [INSENSITIVE]
SCROLL CURSOR
FOR *select_statement*

SCROLL – свобода для FETCH

FETCH [[NEXT | PRIOR | FIRST | LAST
| ABSOLUTE { *n* | **@***nvar* }
| RELATIVE { *n* | **@***nvar* }]
FROM]
cursor_name
[INTO **@***variable_name* [**,**...*n*]

Прокручиваемый курсор

FETCH

NEXT -- следующая

PRIOR – предыдущая

FIRST – первая

LAST -- последняя

ABSOLUTE { *n* | **@***nvar* } -- номер строки

RELATIVE { *n* | **@***nvar* } -- относит. текущей строки

FROM *cursor_name*

[INTO **@***variable_name* [**,**...*n*]

Курсоры: усложним

DECLARE *cursor_name* [SCROLL] CURSOR
FOR *select_statement*
FOR UPDATE [OF *column_name* [,...*n*]] }
]

UPDATE – возможность вносить изменения

FETCH …

UPDATE table_name
SET id=@id+2
WHERE CURRENT OF *cursor_name*;

- Курсор – это почти всегда дополнительные ресурсы сервера и резкое падение производительности по сравнению с другими решениями!