Процедуры и функции

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

Категория функции	Описание
Функции конфигурации	Возвращают сведения о текущей конфигурации.
Функции преобразования	Поддержка приведения и преобразования типов данных.
Функции работы с курсорами	Возвращают сведения о курсорах.
Функции и типы данных даты и времени	Выполняют операции над исходными значениями даты и времени, возвращают строковые и числовые значения, а также значения даты и времени.
Логические функции	Выполнение логических операций.
Математические функции	Выполняют вычисления, основанные на числовых значениях, переданных функции в виде аргументов, и возвращают числовые значения.
Функции метаданных	Возвращают сведения о базах данных и объектах баз данных.
Функции безопасности	Возвращают данные о пользователях и ролях.
Строковые функции	Выполняют операции со строковым (**char** или **varchar**) исходным значением и возвращают строковое или числовое значение.
Системные функции	Выполняют операции над значениями, объектами и параметрами экземпляра SQL Server и возвращают сведения о них.
Системные статистические функции	Возвращают статистические сведения о системе.
Функции обработки текста и изображений	Выполняют операции над текстовыми или графическими исходными значениями или столбцами и возвращают сведения о значении.

Вызов встроенной функции

```
SELECT SQRT(5)
```

Преобразование типов данных

- CAST (expression AS data_type [(length)])
- CAST ($10.50 AS VARCHAR(10))

- CONVERT (data_type [(length)] , expression [, style])
- CONVERT(VARCHAR(12), GETDATE(), 1)

- '2'+4

Примеры

- CAST (expression AS data_type [(length)])

- select cast('12' as int)+45
  57
- select cast(27 as char(2))+'R‘
  27R
- select cast(27 as char(1))+'R‘
  *R

Код символа

- Select ASCII('A'), ASCII('a')
  97 65
- select CHAR(98)
  B
- select CHAR(ASCII('B')+2)
  D

Группировка

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

Условный оператор

IF Boolean_expression { sql_statement | statement_block }

[ELSE { sql_statement | statement_block }]

Метки

Определение метки:

label:

Переход:

GOTO label

Оператор цикл

WHILE Boolean_expression

{ sql_statement | statement_block | BREAK | CONTINUE }

BREAK

Приводит к выходу из ближайшего цикла WHILE.

CONTINUE

Выполняет цикл WHILE для перезагрузки, не учитывая все инструкции, следующие после ключевого слова CONTINUE.

Выражение CASE

```
SELECT   ProductNumber, Category =
      CASE ProductLine
        WHEN 'R' THEN 'Road'
        WHEN 'M' THEN 'Mountain'
        WHEN 'T' THEN 'Touring'
        WHEN 'S' THEN 'Other sale items'
        ELSE 'Not for sale'
      END,
   Name
FROM Production.Product
ORDER BY ProductNumber;
```

Выражение CASE

```
SELECT a,
  CASE
    WHEN a = 1 THEN b
    WHEN a=2 THEN c
    WHEN a>2 THEN b+c
  END
 FROM t2
```

Обработка исключений

```
CREATE TABLE dbo.TestRethrow
(    ID INT PRIMARY KEY);

BEGIN TRY
   INSERT dbo.TestRethrow(ID) VALUES(1);
--  Force error 2627, Violation of PRIMARY KEY constraint to be raised.
   INSERT dbo.TestRethrow(ID) VALUES(1);
END TRY
BEGIN CATCH
   PRINT 'In catch block.';
   THROW;
END CATCH;
```

Процедуры

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

Переменные

```
DECLARE @var_name var_type, …

SET @var_name = var_value;

SELECT @var_name;
```

Процедуры

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

Создание простой процедуры

```
CREATE PROCEDURE SimpleProc AS
UPDATE TOP (5) students
    SET salary=salary*1.5;
```

Изменение простой процедуры

```
ALTER PROCEDURE SimpleProc AS
UPDATE TOP (5) students
    SET salary=salary*1.5;
```

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
```

Переменные

```
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
```

Переменные

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

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

- 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 - это противоположность статических курсоров. 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*]] }]

Последовательный курсор

```
DECLARE employee_cursor CURSOR FOR
SELECT id, name FROM employee
DECLARE @emp_id INT, @emp_name VARCHAR(32)
OPEN employee_cursor
FETCH employee_cursor INTO @emp_id, @emp_name
WHILE (@@FETCH_STATUS = 0) BEGIN
    <do something>
    FETCH employee_cursor INTO @emp_id,
   @emp_name
END
CLOSE employee_cursor
DEALLOCATE employee_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*]

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!