DML
продолжение

Формат оператора SELECT

SELECT [ALL | DISTINCT]
{*|[имя_столбца [AS новое_имя]]} [,...n]
[INTO new_table]
FROM имя_таблицы [[AS] псевдоним]
[,...n]
[WHERE <условие_поиска>]
[GROUP BY имя_столбца [,...n]]
[HAVING <критерии выбора групп>]
[ORDER BY имя_столбца [,...n]]

Вызов простейших функций

- select ‘Hello!‘
  Hello!
- select 2+3
  5
- select ASCII(‘a’) as char_code
  97
- select modul=ABS(3-5)
  2

Вложенные подзапросы в предикатах

- IN, NOT IN – принадлежность множеству
- EXISTS, NOT EXISTS - существует
- ALL, ANY – все/хоть один

Группа	Студент	Ин_язык
101	Сидоров	Английский
101	Петров	Немецкий
102	Иванов	Английский
103	Николаев	Испанский

S

Аудитория	Ин_язык
5	Английский
33	Немецкий
24	Французский

R

INNER JOIN

```
SELECT
Группа, студент, S.ин_язык, аудитория
FROM S
JOIN R
ON S. Ин_язык = R. Ин_язык
```

Группа	Студент	Ин_язык	Аудитория
101	Сидоров	Английский	5
101	Петров	Немецкий	33
102	Иванов	Английский	5

Ин_язык=‘Английский’ ?

SELECT

Группа, студент, S.ин_язык, аудитория

FROM S

JOIN R

ON S. Ин_язык = R. Ин_язык

Группа	Студент	Ин_язык	Аудитория
101	Сидоров	Английский	5
101	Петров	Немецкий	33
102	Иванов	Английский	5

Ин_язык=‘Английский’ ?

SELECT
Группа, студент, S.ин_язык, аудитория
FROM S
JOIN R
ON S. Ин_язык = R. Ин_язык
WHERE S. Ин_язык =‘Английский’

Группа	Студент	Ин_язык	Аудитория
101	Сидоров	Английский	5
102	Иванов	Английский	5

Ин_язык=‘Английский’ ?

SELECT
Группа, студент, S.ин_язык, аудитория
FROM S
JOIN R
ON S. Ин_язык = R. Ин_язык
AND S. Ин_язык =‘Английский’

Группа	Студент	Ин_язык	Аудитория
101	Сидоров	Английский	5
102	Иванов	Английский	5

WHERE или AND ?

- Одинаковый результат в **inner** joins
- Разный в left/right/outer

a. 'where' : *After joining*.
b. 'on' : *Before joining*. Строки фильтруются **before joining**, и в соединении могут быть строки с полями null

Для кого нет аудитории?

SELECT
Группа, студент, S.ин_язык, аудитория
FROM S
LEFT JOIN R
ON S. Ин_язык = R. Ин_язык

Группа	Студент	Ин_язык	Аудитория
101	Сидоров	Английский	5
101	Петров	Немецкий	33
102	Иванов	Английский	5
103	Николаев	Испанский	NULL

Для кого нет аудитории?

```
SELECT
Группа, студент, S.ин_язык, аудитория
FROM S
LEFT JOIN R
ON S. Ин_язык = R. Ин_язык
WHERE аудитория IS NULL
```

Группа	Студент	Ин_язык	Аудитория
103	Николаев	Испанский	NULL

Вложенные подзапросы в предикатах

Полусоединение:

```
SELECT *
FROM R
WHERE ин_язык NOT IN (
SELECT ин_язык FROM S)

SELECT *
FROM R
WHERE NOT EXISTS(
SELECT ин_язык FROM S WHERE ин_язык =R. ин_язык)
```

Вложенные подзапросы в предикатах: IN or EXISTS?

- EXISTS быстрее IN, когда вложенный запрос возвращает большую таблицу. (Сравнение происходит до первого совпадения)

- IN быстрее EXISTS, когда вложенный запрос возвращает маленькую таблицу. (Сравниваются все значения)

EXISTS

T (курс, группа, фамилия, ср_балл)

```
SELECT курс, группа
FROM T
WHERE NOT EXISTS
   (SELECT курс, группа
   FROM T AS T1
   WHERE T1.курс= T.курс AND T1.группа= T.группа
    AND ср_балл = 5 );
```

Результат ?

EXISTS

T (курс, группа, фамилия, ср_балл)

```
SELECT курс, группа
FROM T
WHERE NOT EXISTS
   (SELECT курс, группа
   FROM T AS T1
   WHERE T1.курс= T.курс AND T1.группа= T.группа AND
    ср_балл = 5 );
```

Результат – группы, в которых нет ни одного отличника

ANY/ALL

- SELECT фамилия FROM T
  WHERE ср_балл > ANY (SELECT ср_балл
  FROM T)

- SELECT фамилия FROM T
  WHERE ср_балл > =ALL (SELECT ср_балл
  FROM T)

ALL/ANY

if 170>any(select people.height from people)
print ‘Any people are high‘

if 170>all(select people.height from people)
print ‘All people are high'

Лучший балл в каждой группе

T (курс, группа, фамилия, ср_балл)

```
SELECT курс, группа, MAX(ср_балл) балл
    FROM T
    GROUP BY курс, группа
```

Кто лучший в каждой группе?

T (курс, группа, фамилия, ср_балл)

```
SELECT фамилия, T.курс, T.группа
FROM T
JOIN
    (SELECT курс, группа, MAX(ср_балл) балл
     FROM T
     GROUP BY курс, группа)  T1
    ON T1.курс= T.курс AND T1.группа= T.группа AND
     ср_балл=балл;
```

Теоретико-множественные операции

- UNION
- INTERSECT
- EXCEPT

} [DISTINCT | ALL]

- При этом отношения должны быть совместимы, т.е. иметь одинаковое количество полей с совместимыми типами данных.
- По умолчанию DISTINCT !

Теоретико-множественные операции - порядок

- Выражения в скобках.
- Оператор INTERSECT
- Операторы EXCEPT и UNION обрабатываются слева направо в соответствии с их позицией в выражении.

Теоретико-множественные операции - объединение

```
SELECT
*
FROM T
WHERE курс=1
UNION
SELECT * from T
WHERE курс=2
```

Объединение без дубликатов

```
SELECT
*
FROM T
WHERE курс=1
UNION
SELECT * from T
WHERE группа=2
```

Объединение с дубликатами

```
SELECT
*
FROM T
WHERE курс=1
UNION ALL
SELECT * from T
WHERE группа=2
```

Пересечение

```
SELECT
*
FROM T
WHERE курс=1
INTERSECT
SELECT * from T
WHERE группа=2
```

Разность

```
SELECT
*
FROM T
WHERE курс=1
EXCEPT
SELECT * from T
WHERE группа=2
```

Мухи

Название	Размах крыльев
Жужжало	9
Большеголовка	6
Мошка	3
Златоглазик	13
Гессенская муха	6
Овод желудочный	17
Слепень бычий	22
Рунец овечий	3
Комар	6
Пятнокрылка	13

Название	Размах крыльев
Комар	6
Пятнокрылка	13
Ктырь	25
Овод подкожный	20
Журчалка	25
Ежемуха	6
Американская меромиза	6

Мухи с котлетами

Название	Вес
Котлета пожарская	80
Котлета по-киевски	130
Котлета куриная	70

Название	Размах крыльев
Комар	6
Пятнокрылка	13
Ктырь	25
Овод подкожный	20
Журчалка	25
Ежемуха	6
Американская меромиза	6

Мухи-2

Название	Размах крыльев
Жужжало	9
Большеголовка	6
Мошка	3
Златоглазик	13
Гессенская муха	6
Овод желудочный	17
Слепень бычий	22
Рунец овечий	3
Комар	6
Пятнокрылка	13

Название	Размах крыльев
Комар	6,5
Пятнокрылка	13,7
Ктырь	25,0
Овод подкожный	20,1
Журчалка	25,2
Ежемуха	6,3
Американская меромиза	6,0

Преобразование типов данных

- Явное
- Неявное

Явное преобразование типов

- CAST (expression AS data_type [(length)])
- CONVERT (data_type [(length)] , expression [, style])
- style – количество разрядов числа, знаков после запятой, формат даты/времени

- SELECT CAST(10.6496 AS int)
- SELECT CONVERT(int, 10.6496)

Преобразование типов для даты

- SELECT GETDATE()
  2016-03-14 09:58:04.570

- SELECT CAST(GETDATE() AS nvarchar(30))
  Mar 14 2016 9:58AM

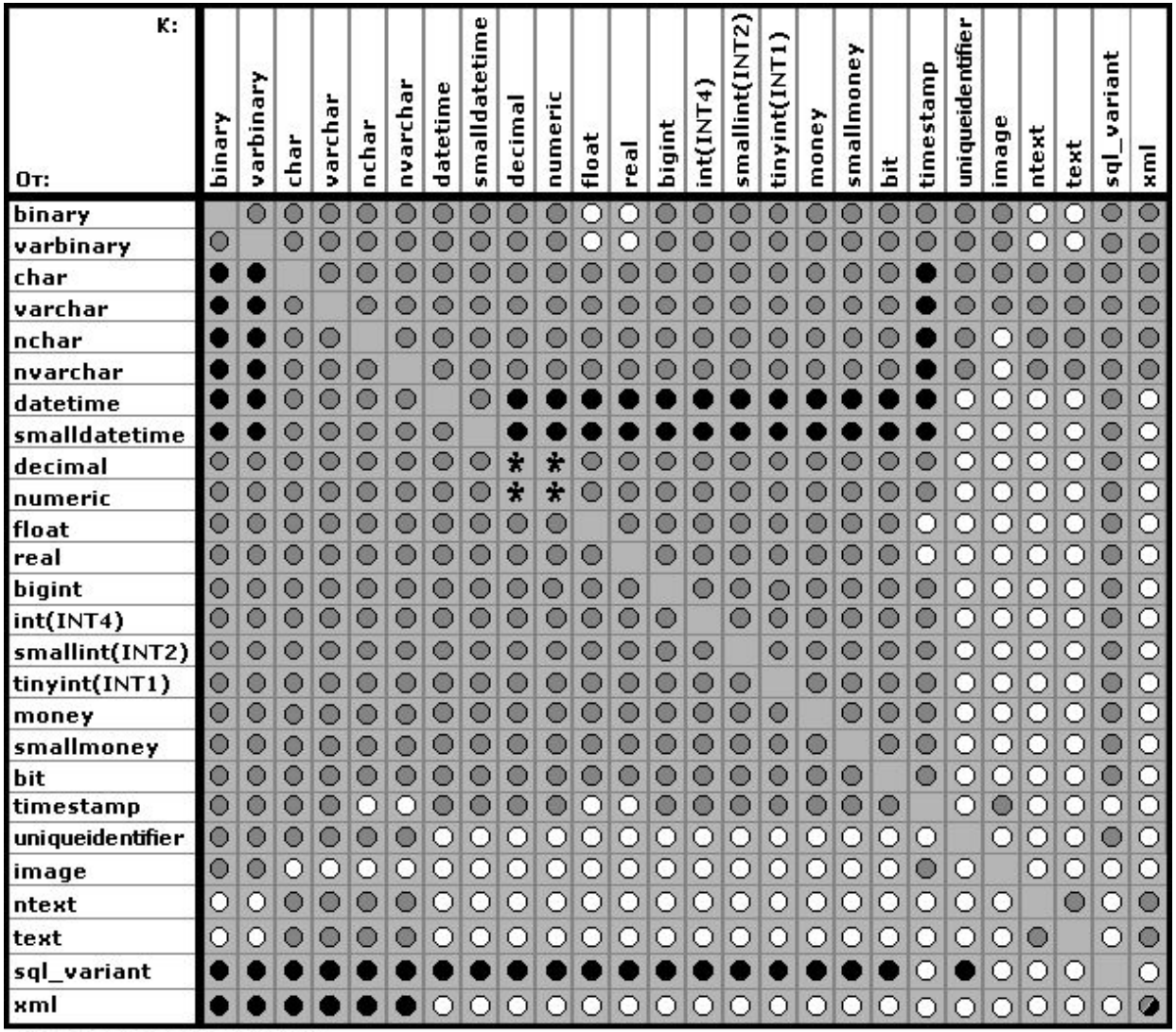
- SELECT CONVERT(nvarchar(30), GETDATE(), 126)
  2016-03-14T09:58:04.570

Двойное преобразование:

- Один в поле не воин => *eng*
  - There is safety in numbers => *rus*
    - Безопаснее действовать сообща ☺

Неявное преобразование типов данных происходит:

- При перемещении, сравнении или объединении данных одного объекта с данными другого объекта эти данные могут преобразовываться из одного типа в другой.
- При передаче в переменную программы данных из результирующего столбца Transact-SQL, кодов возврата или выходных параметров эти данные должны преобразовываться из системного типа данных SQL Server в тип данных переменной.
- При взаимных преобразованиях переменных приложения и столбцов результирующих наборов SQL Server, кодов возврата, параметров и маркеров параметров поддерживаемые преобразования типов данных определяются API базы данных.



● Явное преобразование

◍ Неявное преобразование

○ Преобразование недопустимо

* Требуется явное CAST, чтобы предотвратить потерю точности или масштаба, возможную при неявном преобразовании

◍ Неявные преобразования между типами XML-данных поддеживаются только если исходный или целевой тип - нетипизированный xml. Иначе - преобразование должно быть явным

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

Преобразование типов данных

- select ‘Hello!‘
  Hello!
- select 2+3
  5
- select ‘Маша’+’ ’+’Иванова’
  Маша Иванова

Преобразование типов данных

- select 2+’3’ ?
- select ‘Маша’+1 ?
- select 3/2 ?

Преобразование типов данных

- select 3/2 => 1.5 ?
- select 3/cast(2 as real)
- select cast(3 as real)/2

Преобразование типов данных

- SELECT 'Средний балл= ' + AVG(ср_балл)
  FROM T
  сообщение об ошибке

- SELECT 'Средний балл= ' +
  CAST(AVG(ср_балл) AS CHAR(5)) FROM T
  Средний балл= 4

Преобразование типов данных

- SELECT AVG(ср_балл) FROM T
  4
- Результат с заданной точностью (до двух десятичных знаков)?
  SELECT CAST(AVG(ср_балл) AS NUMERIC(6,2)) FROM T
  4.00
- SELECT AVG(CAST(ср_балл AS NUMERIC(6,2))) FROM T;
  4.248095.
- SELECT CAST(AVG(CAST(ср_балл AS NUMERIC(6,2))) AS NUMERIC(6,2)) FROM T;
- SELECT CAST(AVG(ср_балл*1.0) AS NUMERIC(6,2)) FROM T;

Выражение CASE

- Простое выражение

```
CASE input_expression
    WHEN when_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
   END
```

- Поисковое выражение

```
CASE
    WHEN Boolean_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
```

Простое выражение CASE

T (курс, группа, фамилия, ср_балл)

```
SELECT фамилия, характеристика=
  CASE ср_балл
    WHEN 5 THEN ‘отличник’
    WHEN 4 THEN ‘хорошист’
    ELSE ‘плохой студент’
  END
 FROM T
```

Поисковое выражение CASE

T (курс, группа, фамилия, ср_балл)

```
SELECT фамилия, характеристика=
   CASE WHEN (ср_балл=5 OR ср_балл=4) THEN
‘хороший’
   ELSE ‘плохой’
   END
 FROM T
```

Создание представлений

```
SELECT КодКлиента, Фамилия,
    ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва‘
```

Создание представлений

```
CREATE  VIEW ViewName AS
SELECT КодКлиента, Фамилия,
   ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва‘
```

Создание представлений

```
CREATE  VIEW ViewName AS
SELECT КодКлиента, Фамилия,
   ГородКлиента
FROM Клиент
WHERE ГородКлиента='Москва‘

SELECT * FROM ViewName
INSERT INTO ViewName VALUES (12,'Петров',
   'Самара')
```

Создание представлений (виртуальных таблиц)

CREATE VIEW view_name [(column [,...n])]

[WITH SCHEMABINDING]

AS select_statement

[WITH CHECK OPTION]

Создание представлений

- Для упрощения и настройки восприятия информации в базе данных каждым пользователем.
- В качестве механизма безопасности, позволяющего пользователям обращаться к данным через представления, но не предоставляя им разрешений на непосредственный доступ к базовым таблицам.
- Для предоставления интерфейса обратной совместимости, моделирующего таблицу, схема которой изменилась.

Использование представлений

= таблица	≠ таблица
обращение к представлениям осуществляется также как и к таблицам;	для представления невозможно определить ограничения целостности и первичный ключ;
ко всем представлениям применим оператор **SELECT**;	в операторе **SELECT**, на базе которого создается представление, нельзя устанавливать сортировку его результатов
для некоторых представлений могут применяться операторы **INSERT, UPDATE** и **DELETE**.	не ко всем представлениям могут применяться операторы **INSERT, UPDATE** и **DELETE**
	запрос, именованный через представление выполняется только в момент обращения к представлению

Изменение данных в представлении возможно, если для оператора **SELECT:**

- не используется служебное слово **DISTINCT**;
- при выполнении запроса данные извлекаются только из одной таблицы
- в списка полей этого оператора отсутствуют арифметические выражения
- в запросе не применяются подзапросы
- для результирующих данных не определено группирование.

Обобщенные табличные выражения

- Задается временно именованный результирующий набор (ОТВ - CTE).
- Может включать ссылки на само себя – рекурсия
- За CTE должны следовать одиночные инструкции SELECT, INSERT, UPDATE или DELETE

Структура CTE

WITH expression_name (column_name [,...n])
AS
(CTE_query_definition)

Инструкция для обращения к CTE:

SELECT <column_list>
FROM expression_name;

CTE

Вспомним пример про шахматы:

Каких фигур на доске больше всего?

Группируем

```
SELECT type_fig, COUNT(*) AS Amount
FROM Chessman
GROUP BY type_fig
```

Группируем

```
SELECT Top (1) WITH TIES type_fig, COUNT(*) AS Amount
FROM Chessman
GROUP BY type_fig
ORDER BY Amount
```

CTE

```
WITH gr_chess AS
(SELECT type_fig, COUNT(*) AS Amount
FROM Chessman
GROUP BY type_fig)
SELECT MAX(Amount) FROM gr_chess
```

Строки таблицы – это не записи

```
UPDATE Table1
SET a = b, b = a;
```

- В любом языке программирования:

```
BEGIN
SET a = b;
SET b = a;
END;
```

Сводная таблица

- PIVOT
- UNPIVOT

PIVOT

maker	device	price
B	PC	100
A	PC	110
A	PC	90
E	PC	99
A	Printer	50
D	Printer	45
A	Laptop	200
C	Laptop	220
A	Printer	40
A	Printer	45
D	Printer	55
E	Printer	50
B	Laptop	210
A	Laptop	200
E	PC	90
E	PC	85

	Типы продукции	
Производители	Laptop	PC
	Printer	
	A	
	B	
	C	
	D	
	E	

Сводная таблица - PIVOT

SELECT *столбец для группировки, [значения по горизонтали],…*

FROM *таблица или подзапрос*

PIVOT(*агрегатная функция*

FOR *столбец, содержащий значения, которые станут именами столбцов*

IN (*[значения по горизонтали],…*)

)AS *псевдоним таблицы (обязательно)*

в случае необходимости ORDER BY;

Сводная таблица - PIVOT

```
select maker, [Laptop],[PC], [Printer]
from T
PIVOT (sum(price) for device in ([Laptop],[PC], [Printer]
)
   ) as test_pivot
```

Предложение OVER

- Определяет секционирование и упорядочение набора строк до применения соответствующей оконной функции.
- OVER определяет окно или определяемый пользователем набор строк внутри результирующего набора запроса.
- OVER (

  [<PARTITION BY clause>]

  [<ORDER BY clause>]

)

Сумма нарастающим итогом

```
SELECT
    id, dept, salary
    , SUM(salary) OVER (ORDER BY id)  AS Running_Sum
 from Employees
```

id	dept	salary	Running_Sum
1	1	100	100
2	2	150	250
3	2	110	360
4	2	100	460
5	3	200	660

Сумма с группировкой

```
SELECT
    id, dept, salary
    , SUM(salary) OVER (partition by dept)  AS Dept_Sum
    , AVG(salary) OVER (partition by dept)  AS Dept_AVG
from Employees
```

id	dept	salary	Dept_Sum	Dept_AVG
1	1	100	100	100
2	2	150	360	120
3	2	110	360	120
4	2	100	360	120
5	3	200	200	200

Сумма с группировкой

```
SELECT
    id, dept, salary
    , SUM(salary) OVER (partition by dept ORDER by id)  AS Dept_Sum
    , AVG(salary) OVER (partition by dept)  AS Dept_AVG
 from Employees
```

id	dept	salary	Dept_Run_Sum	Dept_AVG
1	1	100	100	100
2	2	150	150	120
3	2	110	260	120
4	2	100	360	120
5	3	200	200	200

ROW_NUMBER()

```
SELECT
  S.*, ROW_NUMBER() OVER (ORDER BY empName) AS RowNum
FROM Employees S
```

Номер строки

```
SELECT
    id, dept, salary
    , ROW_NUMBER() OVER (ORDER BY id) AS RowNum
 from Employees
```

id	dept	salary	RowNum
1	1	100	1
2	2	150	2
3	2	110	3
4	2	100	4
5	3	200	5

ROW_NUMBER() + PARTITION

```
SELECT
 S.*, ROW_NUMBER() OVER
(PARTITION BY S.mgrid
ORDER BY S.empName) AS LocalRowNum
FROM Employees S
```

Номер строки с группировкой

```
SELECT
    id, dept, salary
    , ROW_NUMBER() OVER (PARTITION BY dept ORDER BY id) AS RowNum
 from Employees
```

id	dept	salary	RowNum
1	1	100	1
2	2	150	1
3	2	110	2
4	2	100	3
5	3	200	1

RANK ( ) / DENSE_RANK ( )

- Rank - возвращает ранг каждой строки в секции результирующего набора. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки.(1, 1, 1, 4)
- Dense_rank - возвращает ранг строк в секции результирующего набора без промежутков в ранжировании. Ранг строки равен количеству различных значений рангов, предшествующих строке, увеличенному на единицу. (1, 1, 2)

RANK ( ) OVER (ORDER by smth)

- Распределяет строки упорядоченной секции в заданное количество групп.

```
SELECT
  S.*
  , RANK ( ) OVER (ORDER by salary desc) AS Gr
FROM Employees S
```

NTILE (N)

- Распределяет строки упорядоченной секции в заданное количество групп.

```
SELECT
  S.*
  , NTILE(3) OVER (ORDER BY S.salary) AS Gr
FROM Employees S
```

```
SELECT
  S.*
  , ROW_NUMBER() OVER
     (PARTITION BY S.mgrid ORDER BY S.empName)  AS
LocalRowNum
  , RANK() OVER (ORDER BY S.salary) AS Rank
  , COUNT(*) OVER
     (PARTITION BY S.mgrid ) AS Amount
FROM Employees S
```

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

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;
```

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).

Временные таблицы

```
CREATE TABLE #TestTableLocal
    (  id INT PRIMARY KEY );

CREATE TABLE ##TestTableGlobal
    (  id INT PRIMARY KEY );
```

Временные таблицы локальные

```
CREATE TABLE #TestTable
 (
  id INT PRIMARY KEY
 )
```

Таблица будет существовать только во время выполнения одной сессии, и работать с ней сможете только вы.

БД tempdb

Временные таблицы глобальные

```
CREATE TABLE ##TestTable
 (
  id INT PRIMARY KEY
 )
```

Таблица будет видна всем. Уничтожается после закрытия создавшей ее сессии /окончания работы с ней другими пользователями

Группировка

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

Условный оператор

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

Условный оператор

IF Boolean_expression { sql_statement | statement_block }

[ELSE { sql_statement | statement_block }]

Метки

- Определение метки

label:

- Переход

GOTO label

Метки

```
DECLARE @i int =0;
label:
INSERT Table1 (id) VALUES (@i);
SET @i+ = 1;

IF @i<5
    GOTO label;
```

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 …';
```

- SQL Server содержит множество встроенных функций, а также поддерживает создание определяемых пользователем функций.

Категории встроенных функций

Function	Описание
Функции, возвращающие наборы строк.	Возвращают объект, который можно использовать так же, как табличные ссылки в SQL-инструкции.
Агрегатные функции	Обрабатывают коллекцию значений и возвращают одно результирующее значение.
Ранжирующие функции	Возвращают ранжирующее значение для каждой строки в секции.
Скалярная функция	Обрабатывают и возвращают одиночное значение. Скалярные функции можно применять везде, где выражение допустимо.

Скалярные функции

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.

Для чего нужны функции

- Для реализации логики приложения на стороне базы данных
  - Создание хранимых процедур и функций
  - Создание триггеров
- Позволяют использовать повторно написанный код, реализующий бизнес-логику
- Облегчают поддержку

Функции – ограничения (1)

- Определяемые пользователем функции не могут выполнять действия, изменяющие состояние базы данных.
- Определяемые пользователем функции не могут возвращать несколько результирующих наборов. Используйте хранимую процедуру, если нужно возвращать несколько результирующих наборов.
- Обработка ошибок в функциях, определяемых пользователем, ограниченна. UDF не поддерживает инструкции TRY…CATCH, @ERROR и RAISERROR.

Функции – ограничения (2)

- Определяемые пользователем функции не могут вызывать хранимую процедуру.
- Определяемые пользователем функции не могут использовать динамический SQL и временные таблицы.
- Определяемые пользователем функции не могут использовать внутри себя недетерминированные функции (GETDATE)
- Нельзя создавать временные таблицы внутри функций

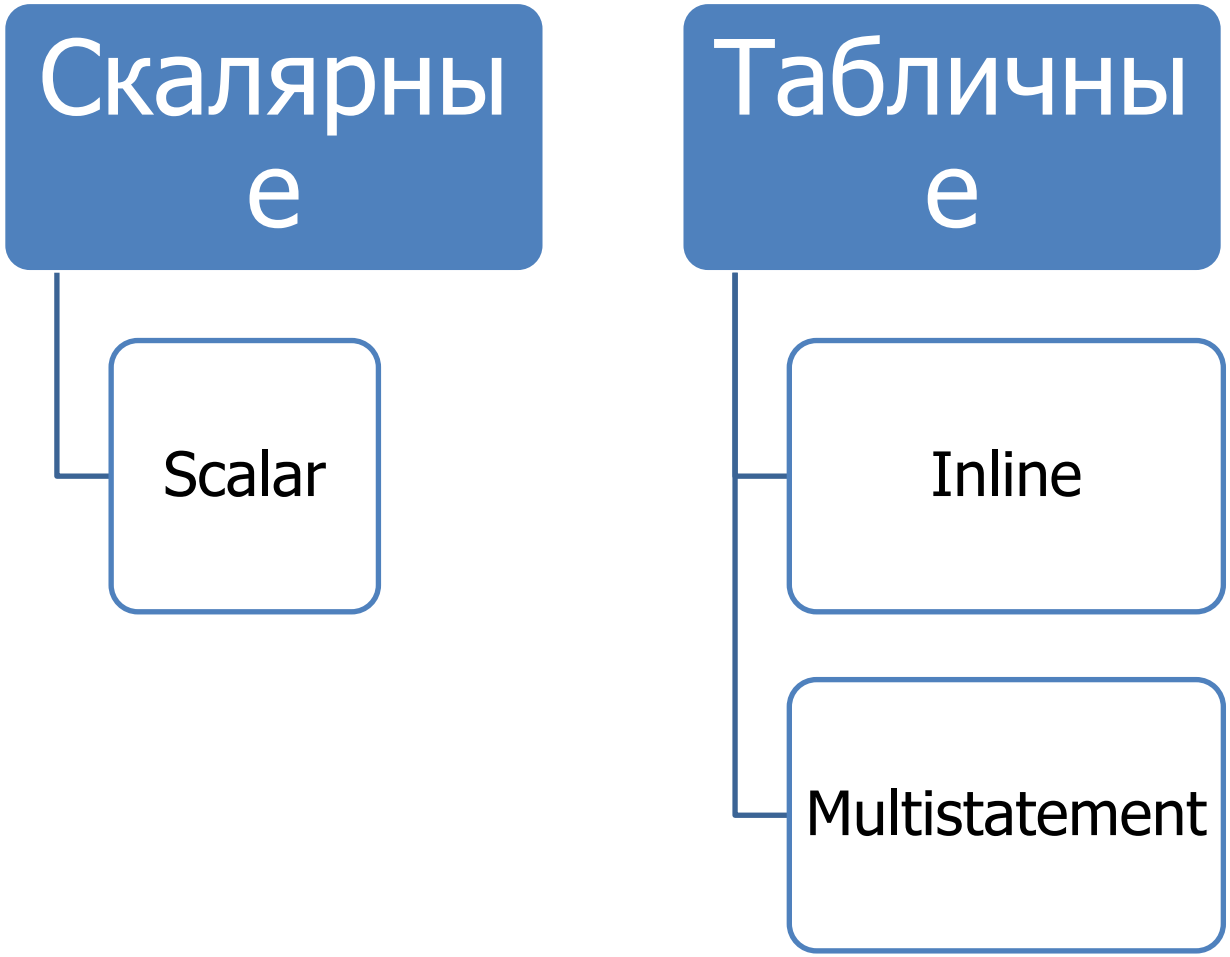
Функции – разрешения (1)

- Определяемые пользователем функции могут быть вложенными, то есть из одной функции может быть вызвана другая. Вложенность определяемых пользователем функций не может превышать 32 уровней.
- UDF могут быть вызваны через Select
- Скалярные функции могут быть использованы после SELECT, WHERE, HAVING
- Табличные функции могут быть использованы после FROM, JOIN, CROSS APPLY.

Функции – разрешения

- Инструкции присваивания.
- Инструкции DECLARE, объявляющие локальные переменные и локальные курсоры.
- Инструкции SELECT, которые содержат списки выбора с выражениями, присваивающими значения локальным переменным.
- Операции над локальными курсорами, которые объявляются, открываются, закрываются и освобождаются в теле функции. Допустимы только те инструкции FETCH, которые предложением INTO присваивают значения локальным переменным. Инструкции FETCH, возвращающие данные клиенту, недопустимы.
- Инструкции INSERT, UPDATE и DELETE, которые изменяют локальные табличные переменные.

Виды функций



Функции, возвращающие значение (скалярные)

```
CREATE  FUNCTION function_name (
    [@parameter scalar_parameter_data_type
    [ = default ] [ ,...n ] ] )
RETURNS scalar_return_data_type
[ AS ]
BEGIN
        function_body
        RETURN scalar_expression
END
```

Значение

```
CREATE  FUNCTION f2 (@num int)
RETURNS  INT
 AS BEGIN
  RETURN ( select count(id)
        from chess
        where id>@num)
  END;
```

```
CREATE FUNCTION testF(@n1 int, @n2 int)
RETURNS int
AS
BEGIN
Return (@n1*@n2)
END

Select testF(5, 8)
```