Оператор выборки данных в SQL

```
(5)     SELECT [ ALL | DISTINCT ] select_item_list
(1)             FROM table_reference_list
(2)             [ WHERE logical_expression ]
(3)             [ GROUP BY column_name_list ]
(4)             [ HAVING logical_expression ]
(6)             [ ORDER BY order_item_list ]
```

табличное выражение

Выполнение запроса состоит из нескольких шагов, соответствующих разделам оператора выборки.
СУБД обычно не придерживаются данной схемы выполнения, но результат должен получиться таким, как если бы он получался при точном следовании данной схеме.

Семантика оператора выборки

Шаг 1: FROM table_reference_list
Выполняется операция расширенного декартова произведения таблиц, указанных в списке FROM: $T = T_1$ TIMES T_2 TIMES … TIMES T_n
Аналогом операции переименования RENAME алгебры Кодда в SQL служат псевдонимы таблиц (пример: EMPLOYEE E) и квалифицированные имена столбцов (пример: E.EMP_BDATE)

Шаг 2: [WHERE logical_expression]
Выполняется операция ограничения таблицы T, сформированной на предыдущем шаге, по заданному логическому условию:
T1 = T WHERE logical_expression
T1 содержит только те строки таблицы T, для которых результатом вычисления логического выражения является TRUE (FALSE и UNKNOWN не являются разрешающими).
Отсутствие раздела WHERE в операторе выборки означает WHERE TRUE (T1 ≡ T)

Семантика оператора выборки

Шаг 3: [GROUP BY column_name_list]
На основе таблицы T1, полученной на предыдущем шаге, формируется сгруппированная таблица T2 (строки расставляются в минимальное число групп, таких что во всех строках одной группы значения указанных столбцов одинаковы.
SELECT … FROM EMPLOYEE GROUP BY DEPT_ID;

…	DEPT_ID	…
…	…	…
…	10	…
…	10	…
…	11	…
…	11	…
…	11	…
…	…	…

Шаг 4: [HAVING logical_expression]
Строится таблица T3, содержащая только те группы строк таблицы T2, для которых результатом вычисления логического выражения является TRUE. Логическое выражение HAVING может включать только имена столбцов группировки или агрегатные функции от других столбцов (инварианты группы).
Отсутствие раздела HAVING в операторе выборки означает HAVING TRUE (T3 ≡ T2)
Наличие HAVING при отсутствии GROUP BY означает, что таблица T1 состоит из единственной группы строк, тогда T3 ≡ T1 ∨ T3 ≡ Ø

Семантика оператора выборки

Шаг 5: SELECT [ALL | DISTINCT] select_item_list

Результирующая таблица T4 содержит столько строк, сколько и таблица T1 (в случае отсутствия GROUP BY и HAVING) или же сколько групп строк содержится в таблице T3 (в случае наличия разделов группировки). Число столбцов в таблице T4 зависит от числа элементов в списке выборки:

1. Элемент выборки имеет вид: $Z_i.c_j$ (квалифицированное имя столбца таблицы с псевдонимом Z_i) или $Z_i.*$(все столбцы таблицы с псевдонимом Z_i), для сгруппированных таблиц допустимо указывать только имена столбцов группировки – аналог операции проекции в реляционной алгебре
2. Элемент выборки имеет вид: value_expression [AS column_name], выражение может включать литералы, вызовы функций, имена столбцов таблиц T1 или T3 (в последнем случае – только столбцов группировки, другие столбцы могут являться только аргументами агрегатных функций) – указанное выражение вычисляется для каждой строки таблицы T1 или группы строк таблицы T3, и результат включается в соответствующую строку таблицы T4

DISTINCT – на завершающей стадии из T4 удаляются строки-дубликаты.

Семантика оператора выборки

Шаг 6: [ORDER BY order_item_list]
order_item ::= value_expression [ASC | DESC]
На завершающей стадии выполнения выборки данных производится сортировка строк результирующей таблицы в соответствии со списком элементов сортировки.
В общем случае выражение, входящее в элемент списка сортировки, основывается на именах столбцов результирующей таблицы T4 и именах столбцов таблицы, над которой вычислялся раздел SELECT (T1 или T3). Идея состоит в том, что если некоторое выражение могло бы быть использовано в элементе списка выборки SELECT, то его можно использовать и в элементе списка сортировки.

Конструкция TABLE table_name является сокращенной формой следующего оператора выборки данных: SELECT * FROM table_name

Агрегатные и кванторные функции

Function([DISTINCT | ALL] value_expression)

Исходное мультимножество строк – вся таблица или группа строк в случае сгруппированной таблицы. Для агрегатных функций на основании аргумента (как правило, имени столбца) из заданного мультимножества строк производится мультимножество значений. При наличии DISTINCT из мультимножества значений удаляются дубликаты и NULL. Затем производится вычисление.

COUNT – число строк или значений

MAX – максимальное значение

MIN – минимальное значение

AVG – среднее значение

SUM – суммарное значение

EVERY – квантор всеобщности (TRUE, если вычисление аргумента равно TRUE для каждой строки исходного мультимножества)

SOME или ANY – квантор существования (TRUE, если вычисление аргумента равно TRUE хотя бы для одной строки исходного мультимножества)

Важный частный случай: COUNT(*) – подсчет строк в мультимножестве (при этом все строки считаются различными)

Ссылки на таблицы раздела FROM

FROM table_reference_list

В качестве ссылок на таблицы могут использоваться не только имена базовых таблиц, но и различные выражения запросов. В рамках данного курса будут рассматриваться только наиболее простые случаи.

В самом простом случае в качестве ссылки на таблицу может использоваться:

- имя базовой таблицы (созданной оператором CREATE TABLE)
- имя представления (виртуальной таблицы)
- порождаемая таблица (выражение запроса, заключенное в круглые скобки)
- имя запроса, присоединенного к данному запросу с помощью раздела WITH

Представления (виртуальные таблицы)

Определение представления:
CREATE VIEW view_name [column_name_list] AS query_expression
Имя представления существует в том же пространстве имен, что и имена базовых таблиц. Явное указание имен столбцов представляемой таблицы требуется в том случае, когда эти имена невозможно вывести из соответствующего выражения запроса.
Пример: создать представление с данными о начальниках отделов
CREATE VIEW DEPARTMENT_MANAGER AS SELECT E.EMP_ID, E.EMP_NAME, E.EMP_BDATE, E.EMP_SALARY, D.DEPT_ID, D.DEPT_NAME FROM EMPLOYEE E, DEPARTMENT D WHERE E.EMP_ID = D.DEPT_MANAGER;
Отмена определения представления:
DROP VIEW view_name { RESTRICT | CASCADE }
Из представляемых таблиц можно выполнять выборку данных с помощью оператора SELECT. В общем случае представляемые таблицы не модифицируемы. Стандарт допускает создание представлений с объявлением WITH CHECK OPTION, через которые возможно выполнять операции обновления базы данных (в курсе лекций не рассматриваются).

Порождаемые таблицы и присоединенные запросы

Семантически порождаемые таблицы соответствуют временным представлениям, которые существуют лишь в момент исполнения запроса. Явное указание псевдонима порождаемой таблицы и имен ее столбцов требуется в том случае, когда эти имена невозможно вывести из соответствующего выражения запроса.

Пример: Найти общее число служащих и максимальный размер зарплаты в отделах с одинаковым максимальным размером зарплаты.

```
SELECT SUM( TOTAL_EMP ), MAX_SAL FROM ( SELECT MAX( EMP_SALARY ),
COUNT(*) FROM EMPLOYEE WHERE DEPT_ID IS NOT NULL GROUP BY DEPT_ID ) AS
DEPT_MAX_SAL( MAX_SAL, TOTAL_EMP ) GROUP BY MAX_SAL;
```

Тот же самый запрос можно сформулировать в виде присоединенного запроса с использованием раздела WITH:

```
WITH DEPT_MAX_SAL( MAX_SAL, TOTAL_EMP ) AS ( SELECT MAX( EMP_SALARY ),
COUNT(*) FROM EMPLOYEE WHERE DEPT_ID IS NOT NULL GROUP BY DEPT_ID )
SELECT SUM( TOTAL_EMP ), MAX_SAL FROM DEPT_MAX_SAL GROUP BY MAX_SAL;
```

Задание для размышления

a	b	c	d

Таблица T
Все столбцы определены на типе INTEGER

Укажите, какие из перечисленных ниже операторов выборки данных являются допустимыми в языке SQL:

1. SELECT MIN(a) FROM T WHERE b > 10 HAVING SUM(c) > 100;
2. SELECT b, c, d FROM T GROUP BY b, c;
3. SELECT SUM(a), b FROM T WHERE c = d GROUP BY b;
4. SELECT a, MAX(b), AVG(d) FROM T WHERE c < a GROUP BY d;
5. SELECT COUNT(*) FROM T WHERE a < 0 GROUP BY b HAVING d <> 1;
6. SELECT c FROM T WHERE c >= 5 GROUP BY c HAVING MIN(a) < c;
7. SELECT a, c FROM T WHERE b > -10 GROUP BY b, d;
8. SELECT MAX(a), SUM(b) FROM T GROUP BY d;
9. SELECT a, c+d, AVG(b) FROM T WHERE b > c AND a < d;

Логические выражения в языке SQL

Синтаксически логическое выражение SQL определяется как булевское выражение, которое строится на основе предикатов с использованием логических операций AND, OR и NOT, а также круглых скобок.

В дальнейшем будем использовать следующие обозначения:
s – скалярная величина
R – строковое значение
|R| – степень строки (количество скалярных значений в ней)
T – табличное значение
|T| - количество строк в таблице

Для предикатов выполняются следующие правила:

1. Совместимость типов операндов
2. Равенство степеней строк-операндов ($|R_X| = |R_Y|$ или $|R_X| = |R_T|$, $R_T \in T$)
3. Существование отрицательной формы предиката NOT pred ≡ NOT(pred) – имеются исключения из данного правила

Предикат сравнения, предикат BETWEEN

Предикат сравнения: s_X op s_Y или R_X op R_Y
op::= = | <> | < | > | <= | >=
NULL op s = UNKNOWN, s op NULL = UNKNOWN, NULL op NULL = UNKNOWN

Предикат BETWEEN (проверка вхождения в диапазон значений):
s_X BETWEEN s_Y AND s_Z или R_X BETWEEN R_Y AND R_Z, $|R_X| = |R_Y| = |R_Z|$
Эквивалентные формы (выражение через предикаты сравнения):
$s_X >= s_Y$ AND $s_X <= s_Z$ или $R_X >= R_Y$ AND $R_X <= R_Z$

Пример: найти номера, имена и размер зарплаты служащих, получающих зарплату, размер которой не меньше средней зарплаты служащих своего отдела и не больше зарплаты начальника отдела

```
SELECT E1.EMP_ID, E1.EMP_NAME, E1.EMP_SALARY FROM EMPLOYEE E1 WHERE
E1.EMP_SALARY BETWEEN ( SELECT AVG( E2.EMP_SALARY ) FROM EMPLOYEE E2
WHERE  E2.DEPT_ID = E1.DEPT_ID ) AND ( SELECT MNG.EMP_SALARY FROM
DEPARTMENT_MANAGER MNG WHERE MNG.DEPT_ID = E1.DEPT_ID );
```

Предикат сравнения с квантором

Квантифицированное сравнение строчного значения с табличным запросом:

$$R_X \text{ op ALL } T = \begin{cases} \text{FALSE} \Leftrightarrow \exists R_T \in T : (R_X \text{ op } R_T) = \text{FALSE} \\ \text{TRUE} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) = \text{TRUE} \vee T \equiv \varnothing \\ \text{UNKNOWN} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) \neq \text{FALSE} \wedge \exists R_T \in T : (R_X \text{ op } R_T) = \text{UNKNOWN} \end{cases}$$

$$R_X \text{ op SOME } T = \begin{cases} \text{TRUE} \Leftrightarrow \exists R_T \in T : (R_X \text{ op } R_T) = \text{TRUE} \\ \text{FASLE} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) = \text{FALSE} \vee T \equiv \varnothing \\ \text{UNKNOWN} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X \text{ op } R_T) \neq \text{TRUE} \wedge \exists R_T \in T : (R_X \text{ op } R_T) = \text{UNKNOWN} \end{cases}$$

R_X op ANY T ≡ R_X op SOME T

Пример: найти номера сотрудников отдела 12, зарплата которых в этом отделе не является минимальной

```
SELECT EMP_ID FROM EMPLOYEE WHERE DEPT_ID = 12 AND EMP_SALARY >
SOME( SELECT E1.EMP_SALARY FROM EMPLOYEE E1 WHERE EMPLOYEE.DEPT_ID =
E1.DEPT_ID );
```

Выполнение корреляционного запроса

Пример: найти номера сотрудников отдела 12, зарплата которых в этом отделе не является минимальной

```
SELECT EMP_ID FROM EMPLOYEE WHERE DEPT_ID = 12 AND EMP_SALARY >
SOME( SELECT E1.EMP_SALARY FROM EMPLOYEE E1 WHERE EMPLOYEE.DEPT_ID =
E1.DEPT_ID );
```

EMPLOYEE

EMP_ID	…	EMP_SALARY	DEPT_ID	…
100		10000.00	10	
101		15000.00	12	
102		12000.00	10	
103		12000.00	11	
104		14000.00	12	
105		20000.00	12	
106		22000.00	11	
107		11000.00	10	

E1

E1

EMP_SALARY
15000.00
14000.00
20000.00

RESULT

EMP_ID
101
105

Предикат IS NULL

Проверяет, являются ли неопределенными значения всех элементов строки-операнда: R_X IS NULL

	R_X IS NULL	R_X IS NOT NULL	NOT R_X IS NULL	NOT R_X IS NOT NULL
$\|R_X\|=1$, $s=NULL$	TRUE	FALSE	FALSE	TRUE
$\|R_X\|=1$, $s\neq NULL$	FALSE	TRUE	TRUE	FALSE
$\|R_X\|>1$, $\forall s\in R_X \Rightarrow s=NULL$	TRUE	FALSE	FALSE	TRUE
$\|R_X\|>1$, $\forall i\neq j \ \exists s_i\in R_X : s_i=NULL \wedge \exists s_j\in R_X : s_j\neq NULL$	FALSE	FALSE	TRUE	TRUE
$\|R_X\|>1$, $\forall s\in R_X \Rightarrow s\neq NULL$	FALSE	TRUE	TRUE	FALSE

Пример: найти номера и имена служащих, не задействованных в проектах
SELECT EMP_ID, EMP_NAME FROM EMPLOYEE WHERE PRO_ID IS NULL;

Предикат IN

Проверяет факт вхождения скалярного значения или строки в указанное множество: s IN (s_1, s_2, …, s_n) или R_X IN T

$$R_X \text{ IN } T = \begin{cases} \text{TRUE} \Leftrightarrow \exists R_T \in T : (R_X = R_T) = \text{TRUE} \\ \text{FASLE} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X = R_T) = \text{FALSE} \vee T \equiv \varnothing \\ \text{UNKNOWN} \Leftrightarrow \forall R_T \in T \Rightarrow (R_X = R_T) \neq \text{TRUE} \wedge \exists R_T \in T : (R_X = R_T) = \text{UNKNOWN} \end{cases}$$

Пример: найти номера сотрудников, не являющихся начальниками отделов и получающих зарплату, размер которой равен размеру зарплаты какого-либо начальника отдела

```
SELECT EMP_ID FROM EMPLOYEE WHERE EMP_ID NOT IN ( SELECT
DEPT_MANAGER FROM DEPARTMENT ) AND EMP_SALARY IN ( SELECT
EMP_SALARY FROM DEPARTMENT_MANAGER );
```

Предикаты LIKE и SIMILAR

Сопоставление символьных и битовых строк с заданным шаблоном:
source_string LIKE pattern_string [ESCAPE symbol]
В шаблоне могут использоваться два специальных символа:
‘_’ (подчеркивание) – интерпретируется как произвольный одиночный символ
‘%’ (процент) – интерпретируется как произвольная подстрока произвольной длины
Для битовых строк используются коды соответствующих символов (X’5F’ и X’25’).
Раздел ESCAPE специфицирует одиночный символ (например, ‘\’), используемый для изменения способа интерпретации ‘_’ и ‘%’ (пары символов ‘_’ и ‘\%’ будут интерпретироваться как одиночные символы ‘_’ и ‘%’ соответственно).
Основное отличие SIMILAR от LIKE состоит в возможности использования регулярных выражений внутри шаблона (в курсе лекций не рассматривается).
Пример: подсчитать количество проектов, в названии которых присутствует слово ‘software’
SELECT COUNT(*) FROM PROJECT WHERE PRO_TITLE LIKE ‘%software%’ OR PRO_TITLE LIKE ‘%Software%’;

Предикат OVERLAPS

Служит для проверки перекрытия по времени двух событий: R_X OVERLAPS R_Y, $|R_X|=|R_Y|=2$, строки задаются в виде: (t_S, t_F) или (t_S, interval), где $t_F = t_S$ + interval

Эквивалентное представление в виде предикатов сравнения:

($t_{SX} = t_{SY}$) OR

($t_{SX} > t_{SY}$) AND ($t_{SX} <= t_{FY}$ OR $t_{FX} <= t_{FY}$) OR

($t_{SY} > t_{SX}$) AND ($t_{SY} <= t_{FX}$ OR $t_{FY} <= t_{FX}$)

Пример: найти названия проектов, которые будут выполняться в течение следующего года

SELECT PRO_TITLE FROM PROJECT WHERE (PRO_SDATE, PRO_DURATION) OVERLAPS (CURRENT_DATE, INTERVAL ‘1’ YEAR)

Предикат EXISTS

Проверяет наличие строк в результате запроса

$$\text{EXISTS}(T) = \begin{cases} \text{TRUE} \Leftrightarrow |T| > 0 \\ \\ \text{FALSE} \Leftrightarrow |T| = 0 \end{cases}$$

Запросы с предикатом EXISTS можно переформулировать в виде запросов с предикатом сравнения и агрегатной функцией COUNT(*).

Пример: найти номера отделов, среди служащих которых есть руководители проектов

```
SELECT D.DEPT_ID FROM DEPARTMENT D WHERE EXISTS( SELECT
E.EMP_ID FROM EMPLOYEE E WHERE E.DEPT_ID = D.DEPT_ID AND
EXISTS( SELECT P.PRO_MANAGER FROM PROJECT P WHERE
P.PRO_MANAGER = E.EMP_ID ) );
```

Предикат IS DISTINCT FROM

Позволяет проверить, являются ли две строки дубликатами.

R_X IS DISTINCT FROM R_Y =

$$\begin{cases} \text{FALSE} \Leftrightarrow \forall i = \overline{1, |R_X|} \Rightarrow (s_{Xi} = s_{Yi}) = \text{TRUE} \\ \quad \vee\ s_{Xi} = \text{NULL} \wedge s_{Yi} = \text{NULL},\ s_{Xi} \in R_X,\ s_{Yi} \in R_Y \\ \\ \text{TRUE} \Leftrightarrow \exists i : (s_{Xi} = s_{Yi}) = \text{FALSE} \vee s_{Xi} = \text{NULL} \\ \quad \wedge\ s_{Yi} \neq \text{NULL} \vee s_{Xi} \neq \text{NULL} \wedge s_{Yi} = \text{NULL} \end{cases}$$

Отрицательная форма предиката (IS NOT DISTINCT FROM) в языке SQL отсутствует.

Пример: найти пары номеров проектов, выполняющихся в организации в одно и то же время

```
SELECT P1.PRO_ID, P2.PRO_ID FROM PROJECT P1, PROJECT P2
WHERE P1.PRO_ID <> P2.PRO_ID AND NOT( ( P1.PRO_SDATE,
P1.PRO_DURATION ) IS DISTINCT FROM ( P2.PRO_SDATE,
P2.PRO_DURATION ) );
```

Предикат UNIQUE

Служит для проверки факта отсутствия строк-дубликатов в результате запроса.
UNIQUE(T) = TRUE тогда и только тогда, когда в таблице T отсутствуют строки-дубликаты, в противном случае возвращается FALSE.

Пример: найти названия отделов, служащих которых можно различить по размеру получаемой зарплаты
SELECT DEPT_NAME FROM DEPARTMENT WHERE UNIQUE(SELECT EMP_SALARY FROM EMPLOYEE WHERE EMPLOYEE.DEPT_ID = DEPARTMENT.DEPT_ID);

Предикат MATCH

Условие соответствия строчного значения результату подзапроса:

R_X MATCH [UNIQUE] SIMPLE T = TRUE $\Leftrightarrow \exists s \in R_X : s=NULL \vee \exists R_T \in T : (R_X = R_T)=TRUE$

R_X MATCH [UNIQUE] PARTIAL T = TRUE $\Leftrightarrow \forall s \in R_X \Rightarrow s=NULL \vee \exists R_T \in T : \forall s_{Xi} \neq NULL \Rightarrow (s_{Xi} = s_{Ti})=TRUE, s_{Xi} \in R_X, s_{Ti} \in R_T$

R_X MATCH [UNIQUE] FULL T = TRUE $\Leftrightarrow \forall s \in R_X \Rightarrow s=NULL \vee \forall s \in R_X \Rightarrow s \neq NULL \wedge \exists R_T \in T : (R_X = R_T)=TRUE$

Если указано UNIQUE, то дополнительно проверяется, является ли найденная строка R_T уникальной в T (FALSE, если неуникальная).

Пример: найти номера служащих, зачисленных в один из отделов, для которых в их отделах работают служащие с той же самой датой рождения

```
SELECT EMP_ID FROM EMPLOYEE WHERE ( DEPT_ID, EMP_BDATE ) MATCH FULL (
SELECT E1.DEPT_ID, E1.EMP_BDATE FROM EMPLOYEE E1 WHERE E1.EMP_ID <>
EMPLOYEE.EMP_ID );
```