# ЛАБОРАТОРНАЯ РАБОТА №15

*Тема:* Оконные аналитические функции в SQL запросах .

*Цель:* Пробрести навыки составления SQL запросов использующих оконные аналитические функции.

В данной лабораторной работе используются база данных HR, созданная в рамках первой лабораторной работы. Также используется таблица TB_ELEKTROSTAL_2018, которая создана и наполнена данными в процессе выполнения лаб. работы №.1.

Дополнительно используется удаленная база данных проекта stackoverflow.com. Структура этой базы данных описана в методических указаниях к выполнению лабораторной работы № 3. Напомним, адрес ресурса http://data.stackexchange.com/stackoverflow/query/new.

В лабораторной работе 14 рассматривались особенности построения запросов с использованием аналитических функций. Многие виды функций этого типа дополнительно могут использовать специальное «окно» для доступа к данным. Такое окно может иметь как фиксированные так и плавающие границы. Поэтому окно часто называют плавающим.

Выполнение вычислений для строк в группе по плавающему окну (интервалу)

Для некоторых аналитических функций, например, агрегатных, можно дополнительно указать объем строк, участвующих в вычислении, выполняемом для каждой строки в группе. Этот объем, своего рода контекст строки, называется "окном", а границы окна могут задаваться различными способами.

```
{ROWS | RANGE} {{UNBOUNDED | выражение} PRECEDING |
CURRENT ROW }
{ROWS | RANGE}
BETWEEN
{{UNBOUNDED PRECEDING | CURRENT ROW |
{UNBOUNDED | выражение 1}{PRECEDING | FOLLOWING}}
AND
```

```
{{UNBOUNDED FOLLOWING | CURRENT ROW |
{UNBOUNDED | выражение 2}{PRECEDING | FOLLOWING}}
```

Фразы PRECEDING и FOLLOWING задают верхнюю и нижнюю границы агрегирования (то есть интервал строк, "окно" для агрегирования).

Вот поясняющий пример, воспроизводящий результат из предыдущей лабораторной работы:

```
SELECT first_name||' '||last_name
     , department_id
     , job_id
     , SUM(salary) OVER (PARTITION BY department_id,
job_id ORDER BY hire_date
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
sum_sal
        FROM employees;
```

Здесь в пределах каждой группы (использована фраза **PARTITION BY**) сотрудники упорядочиваются по времени найма на работу (фраза **ORDER BY**) и для каждого в группе вычисляется сумма зарплат: его и всех его предшественников (фраза **ROWS BETWEEN** формулирует "окошко суммирования" от первого в группе до текущего рассматриваемого).

Размеры и параметры окна, указанные в данном примере, подразумеваются по умолчанию, если описание окна отсутствует в запросе (сравните с запросом из предыдущей лабораторной работы).

Плавающий интервал задается в терминах упорядоченных строк (**ROWS**) или значений (**RANGE**), для чего фраза **ORDER BY** в определении группы обязана присутствовать.

Формирование интервалов агрегирования "по строкам" и "по значениям"

Разницу между **ROWS** и **RANGE** (определяющими, как говорится в документации, "физические" и "логические" интервалы-окна) демонстрируется следующим примером:

```
SELECT first_name||' '||last_name
     , hire_date
     , salary
     , SUM(salary) OVER (ORDER BY hire_date ROWS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) rows_sal
```

```
     , SUM(salary) OVER (ORDER BY hiredate RANGE
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) range_sal
FROM employees;
```

Работники, которые поступили на работу одновременно, с точки зрения интервала суммирования неразличимы. Поэтому суммирование "по значению" присвоило им один и тот же общий для "мини-группы", образованной ими парой, результат - максимальную сумму, которая при всех возможных порядках перечисления сотрудников внутри этой пары будет всегда одинакова. Суммирование "по строкам" (ROWS) поступило иначе: оно упорядочило сотрудников в "мини-группе", образованной равными датами (на самом деле чисто произвольно) и подсчитало суммы, как будто бы у этих сотрудников был задан порядок следования.

Функции **FIRST_VALUE** и **LAST_VALUE** для интервалов агрегирования.

Эти функции позволяют для каждой строки выдать первое значение ее окна и последнее. Пример:

```
SELECT first_name||' `||last_name
     , hire_date
     , salary
     , FIRST_VALUE(salary) OVER (ORDER BY hire_date
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)  first_rows
     , LAST_VALUE(salary) OVER (ORDER BY hire_date ROWS
BETWEEN 2 PRECEDING AND CURRENT ROW) last_rows
     , FIRST_VALUE(salary) OVER (ORDER BY hire_date
RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) first_range
     , LAST_VALUE(salary) OVER (ORDER BY hire_date
RANGE BETWEEN 2 PRECEDING AND CURRENT ROW) last_range
FROM employees;
```

Интервалы времени

Для интервалов (окон), упорядоченных внутри по значению ("логическом", RANGE) в случае, если это значение имеет тип "дата", границы интервала можно указывать выражением над датой, а не конкретными значениями из строк. Примеры таких выражений:

```
INTERVAL число {YEAR | MONTH | DAY | HOUR | MINUTE |
SECOND}
```

```
NUMTODSINTERVAL(число, '{DAY | HOUR | MINUTE |
SECOND}')
NUMTOYMINTERVAL(число, '{YEAR | MONTH}')
```

Пример вывода зарплат сотрудников и средних зарплат за последние полгода на момент приема нового сотрудника:

```
SELECT first_name||' '||last_name
     , hire_date
     , salary
     , AVG(salary) OVER (ORDER BY hire_date RANGE
BETWEEN INTERVAL '6' MONTH PRECEDING AND CURRENT ROW)
avg_sal
FROM employees;
```

Ниже приведен другой вариант записи того же запроса, позволяющий использовать для числа месяцев обычное числовое выражение:

```
SELECT first_name||' '||last_name
     , hire_date
     , salary
     , AVG(salary) OVER (ORDER BY hire_date RANGE
BETWEEN NUMTOYMINTERVAL(6, 'MONTH') PRECEDING  AND
CURRENT ROW) avg_sal
FROM employees;
```

## Порядок выполнения работы

Создайте таблицу:

```
create table tb_oper(oper_id number,
                     oper_date date,
                     amount number,
                     primary key (oper_id,
oper_date));
```

где:

oper_id – идентификатор оператора;

oper_date – дата совершения оператором операции;

amount – сумма операции.

Заполните таблицу данными:

```
insert into tb_oper (oper_id, oper_date, amount)
  values(1, to_date('2015-02-17 10:25:00', 'yyyy.mm.dd
hh24.mi.ss'), 100);
insert into tb_oper (oper_id, oper_date, amount)
  values(1, to_date('2015-02-17 10:27:00', 'yyyy.mm.dd
hh24.mi.ss'), 20);
insert into tb_oper (oper_id, oper_date, amount)
  values(1, to_date('2015-02-17 10:28:00', 'yyyy.mm.dd
hh24.mi.ss'), 30);
insert into tb_oper (oper_id, oper_date, amount)
  values(1, to_date('2015-02-17 10:29:00', 'yyyy.mm.dd
hh24.mi.ss'), 5);
insert into tb_oper (oper_id, oper_date, amount)
  values(1, to_date('2015-02-17 10:38:00', 'yyyy.mm.dd
hh24.mi.ss'), 15);
insert into tb_oper (oper_id, oper_date, amount)
  values(1, to_date('2015-02-17 10:55:00', 'yyyy.mm.dd
hh24.mi.ss'), 25);
insert into tb_oper (oper_id, oper_date, amount)
  values(2, to_date('2015-02-17 10:23:00', 'yyyy.mm.dd
hh24.mi.ss'), 50);
insert into tb_oper (oper_id, oper_date, amount)
  values(2, to_date('2015-02-17 10:27:10', 'yyyy.mm.dd
hh24.mi.ss'), 60);
insert into tb_oper (oper_id, oper_date, amount)
  values(2, to_date('2015-02-17 10:28:00', 'yyyy.mm.dd
hh24.mi.ss'), 10);
insert into tb_oper (oper_id, oper_date, amount)
  values(2, to_date('2015-02-17 10:50:00', 'yyyy.mm.dd
hh24.mi.ss'), 5);
insert into tb_oper (oper_id, oper_date, amount)
  values(2, to_date('2015-02-17 10:50:15', 'yyyy.mm.dd
hh24.mi.ss'), 85);
insert into tb_oper (oper_id, oper_date, amount)
  values(2, to_date('2015-02-17 11:00:01', 'yyyy.mm.dd
hh24.mi.ss'), 25);
```

Одним SQL-запросом выведите все подозрительные операции операторов, удовлетворяющие следующим условиям:

1. Предыдущие операции были совершены не позднее десяти минут назад от времени подозрительной операции;

2. Количество предыдущих операций, с учетом подозрительной операции, не менее трех;

3. Сумма всех предыдущих операций, с учетом суммы подозрительной операции, не менее ста.

Результат отсортировать по идентификатору оператора в порядке возрастания, дате операции в порядке убывания.

Формат результата

| OPER_ID | OPER_DATE | AMOUNT |
|---|---|---|
| 1 | 17.02.2015 10:29:00 | 5 |
| 1 | 17.02.2015 10:28:00 | 30 |
| ... | ... | ... |

**Содержание отчета:**

1. Тема, цель лабораторной работы.
2. Примеры выполнения запросов к базе данных.
3. Составленные согласно заданию запросы и скриншоты полученных результатов.
5. Выводы.

**Контрольные вопросы:**

1. Укажите назначение и основные характеристики плавающих окон в аналитических функциях.
2. Если в аналитической функции в запросе параметры плавающего окна не указаны, какие его значения принимаются по умолчанию?
3. В чем общие черты и отличию окон типа ROWS и RANGE? Где записываются аналитические функции в SQL-операторах?
4. Особенности описания окон для временных интервалов?

| | |
|---|---|
| Файл: | Лабораторная работа 15 |
| Каталог: | E:\Projects\documents\md\predmety\BD_2018 |
| Шаблон: | C:\Users\sss\AppData\Roaming\Microsoft\Шаблоны\Normal.dotm |
| Заголовок: | |
| Содержание: | |
| Автор: | sss |
| Ключевые слова: | |
| Заметки: | |
| Дата создания: | 19.11.2018 21:19:00 |
| Число сохранений: | 27 |
| Дата сохранения: | 19.11.2018 22:12:00 |
| Сохранил: | sss |
| Полное время правки: | 38 мин. |
| Дата печати: | 19.11.2018 22:12:00 |
| При последней печати | |
| страниц: | 6 |
| слов: | 1 282 (прибл.) |
| знаков: | 7 308 (прибл.) |