Лекция 6

Оконные функции

Использование GROUP BY

Правило использования *GROUP BY* :

В списке вывода при использовании *GROUP BY* могут быть указаны только **функции агрегирования, константы и поля, перечисленные в *GROUP BY***.

Например, нельзя получить сведения о том, у каких сотрудников самая высокая зарплата в своём отделе с помощью такого запроса:

```
select  depno,  name,  max(salary) as max_sal
        from  emp
        group by  depno;
```

Этот запрос синтаксически неверен!

depno	name	salary
1	Белов С.В.	58000
1	Иванова К.Е.	28000
1	Седов О.Л.	41000
2	Волков Н.Е.	40000
2	Рогов И.Л.	32000
3	Санина В.П.	**47000**
3	Дымова С.Т.	29000
3	Павлов К.Д.	**47000**
3	Орлов Т.Ф.	30000

Таблица БД для примера

```
--создание таблицы
create table student_grades
( name varchar,
subject varchar,
grade int);
-- наполнение таблицы данными insert into
student_grades ( values ('Петя', 'русский',
4), ('Петя', 'физика', 5), ('Петя', 'история', 4),
('Маша', 'математика', 4), ('Маша',
'русский', 3), ('Маша', 'физика', 5), ('Маша',
'история', 3) ('Петя', 'математика', 4));
--запрос всех данных из таблицы
 select * from student_grades;
```

name character varying	subject character varying	grade integer
Петя	русский	4
Петя	физика	5
Петя	история	4
Маша	математика	4
Маша	русский	3
Маша	физика	5
Маша	история	3

Определения

SQL часто используется для вычислений в данных различных метрик или агрегаций значений по измерениям. Помимо функций агрегации для этого широко используются **оконные функции**, их удобно применять для **всякой аналитики, отчетов** и так далее.
Окно — это некоторое выражение, описывающее набор строк, которые будет обрабатывать функция и порядок этой обработки (синтаксис: *функция* **OVER()** *окно*)
Причем окно может быть просто задано пустыми скобками **( )**, т.е. окном являются все строки результата запроса.
Оконная функция в SQL - функция, которая работает с выделенным набором строк (***окном, партицией***) и выполняет вычисление для этого набора строк в отдельном столбце.
Партиции (окна из набора строк) - это набор строк, указанный для оконной функции по одному из столбцов или группе столбцов таблицы. Партиции для каждой оконной функции в запросе могут быть разделены по различным колонкам таблицы.

Партиции
оконной функции
(в данном примере
по полю **Имя**)

Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4
Маша	рус	3
Маша	физ	5
Маша	история	3

Пример

```
select name as Имя,subject as Предмет,grade as Оценка,
 row_number() over () AS Номер from student_grades;
```

Имя character varying	Предмет character varying	Оценка integer	Номер bigint
Петя	русский	4	1
Петя	физика	5	2
Петя	история	4	3
Маша	математика	4	4
Маша	русский	3	5
Маша	физика	5	6
Маша	история	3	7
Петя	математика	3	8

окном являются все строки результата запроса.

```
select name as Имя,subject as Предмет,grade as Оценка, row_number() over
(partition by name ORDER BY grade DESC) from student_grades;
```

Имя character varying	Предмет character varying	Оценка integer	row_number bigint
Маша	физика	5	1
Маша	математика	4	2
Маша	история	3	3
Маша	русский	3	4
Петя	физика	5	1
Петя	русский	4	2
Петя	история	4	3
Петя	математика	3	4

В оконное выражение можно добавить слово **PARTITION BY [expression]**, например **row_number() OVER (PARTITION BY section)**, тогда подсчет будет идти в каждой группе отдельно, если добавить **ORDER BY**, тогда можно изменить порядок обработки

Отличие оконных функций от функций агрегации с группировкой

Применение функции агрегации и команды GROUP BY

Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4
Маша	рус	3
Маша	физ	5
Маша	история	3

"Имя"	"Средняя_оценка"
"Маша"	3.75
"Петя"	4.33

Имя	Средняя оценка
Петя	4
Маша	3,75

```
select name as Имя, avg(grade) as Средняя оценка
from student_grades group by name;
```

Применение Оконной функции

Имя	Предмет	Оценка
Петя	матем	3
Петя	рус	4
Петя	физ	5
Петя	история	4
Маша	матем	4

Имя	Предмет	Оценка	Средняя оценка
Петя	матем	3	4
Петя	рус	4	4
Петя	физ	5	4
Петя	история	4	4
Маша	матем	4	3,75

```
select name, subject, grade, avg(grade)
over (partition by name) as avg_grade from student_grades;
```

При использовании оконных функций количество строк в запросе не уменьшается по сравнении с исходной таблицей.

Синтаксис оконных функций

Имя оконной функции одного из классов	→	FUNCTION_NAME(column_name)
Необязательное выражение фильтрации	→	[FILTER (WHERE filter_clause)]
Ключевое слово определения оконной ф.	→	OVER
		(
Определение партиций по колонкам	→	PARTITION BY (column_names),
Сортировка вычисления оконной функции	→	ORDER BY (column_names),
Указание фрейма для партиции	→	[frame_clause]
		)

Оконные функции можно прописывать как под командой SELECT, так и в отдельном ключевом слове WINDOW, где окну дается алиас (псевдоним), к которому можно обращаться в SELECT выборке.

дубли определения окна

```
select name, subject, grade,
row_number() over (partition by name order by grade desc),
rank() over (partition by name order by grade desc),
dense_rank() over (partition by name order by grade desc)
from student_grades;
```

=

```
select name, subject, grade,
row_number() over name_grade,
rank() over name_grade,
dense_rank() over name_grade
from student_grades
window name_grade as (partition by name order by grade desc);
```

Классы Оконных функций

Агрегирующие (Aggregate); Ранжирующие (Ranking); Функции смещения (Value)
PostgreSQL https://postgrespro.ru/docs/postgrespro/14/functions-window

Можно применять любую из агрегирующих функций - SUM, AVG, COUNT, MIN, MAX:

```
select name, subject, grade,
        sum(grade) over (partition by name) as sum_grade,
        avg(grade) over (partition by name) as avg_grade,
        count(grade) over (partition by name) as count_grade,
        min(grade) over (partition by name) as min_grade,
        max(grade) over (partition by name) as max_grade
from student_grades;
```

Применение агрегирующих функций

name	subject	grade	sum_grade	avg_grade	count_grade	min_grade	max_grade
Маша	история	3	15	3,75	4	3	5
Маша	математика	4	15	3,75	4	3	5
Маша	русский	3	15	3,75	4	3	5
Маша	физика	5	15	3,75	4	3	5
Петя	математика	3	16	4	4	3	5
Петя	русский	4	16	4	4	3	5
Петя	физика	5	16	4	4	3	5
Петя	история	4	16	4	4	3	5

name character varying	subject character varying	grade integer	sum_grade bigint	avg_grade numeric	count_grade bigint	min_grade integer	max_grade integer
Маша	математика	4	15	3.7500000000	4	3	5
Маша	русский	3	15	3.7500000000	4	3	5
Маша	физика	5	15	3.7500000000	4	3	5
Маша	история	3	15	3.7500000000	4	3	5
Петя	физика	5	13	4.3333333333	3	4	5
Петя	история	4	13	4.3333333333	3	4	5
Петя	русский	4	13	4.3333333333	3	4	5

Применение агрегирующих функций (2)

1. Если не задан **ORDER BY** в окне, идет подсчет по всей партиции один раз, и результат пишется во все строки (одинаков для всех строк партиции).
2. Если же **ORDER BY** задан, то подсчет в каждой строке идет от начала партиции до этой строки.

```
select name, subject, grade,
sum(grade) over w as sum_grade,
avg(grade) over w as avg_grade,
count(grade) over w as count_grade,
min(grade) over w as min_grade,
max(grade) over w as max_grade     from student_grades
  window w as (partition by name ORDER BY grade);
```

Здесь для каждой строки идет подсчет в отдельном *фрейме*. **Фрейм(рамка окна)** - это набор строк от начала до текущей строки (если есть PARTITION BY, то от начала

name character varying	subject character varying	grade integer	sum_grade bigint	avg_grade numeric	count_grade bigint	min_grade integer	max_grade integer
Маша	русский	3	6	3.0000000000	2	3	3
Маша	история	3	6	3.0000000000	2	3	3
Маша	математика	4	10	3.3333333333	3	3	4
Маша	физика	5	15	3.7500000000	4	3	5
Петя	математика	3	3	3.0000000000	1	3	3
Петя	история	4	11	3.6666666666	3	3	4
Петя	русский	4	11	3.6666666666	3	3	4
Петя	физика	5	16	4.0000000000	4	3	5

Применение агрегирующих функций (3)

1. Оконные функции можно использовать сразу по несколько штук, они друг другу ничуть не мешают, чтобы вы там в них не написали.
2. Подсчитать процент различных оценок в общем количестве оценок.

```
select name, subject, grade,
sum(grade) OVER (PARTITION BY grade ORDER BY grade) as sum_window,-- сумма оценок в окне оценок
 sum(grade) OVER () as sum_total, --общая сумма оценок
round(100.0 * sum(grade) OVER (PARTITION BY grade ORDER BY grade)/sum(grade) OVER (),2) AS percent_of_total,
count(*) OVER () as grade_count – всего строк  from student_grades;
```

name character varying	subject character varying	grade integer	sum_window bigint	sum_total bigint	percent_of_total numeric	grade_count bigint
Петя	математика	3	9	31	29.03	8
Маша	русский	3	9	31	29.03	8
Маша	история	3	9	31	29.03	8
Маша	математика	4	12	31	38.71	8
Петя	русский	4	12	31	38.71	8
Петя	история	4	12	31	38.71	8
Маша	физика	5	10	31	32.26	8
Петя	физика	5	10	31	32.26	8

Ранжирующие оконные функции

Ранжирующие функции – это функции, которые ранжируют значение для каждой строки в окне. Например, их можно использовать для того, чтобы присвоить порядковый номер строке или составить рейтинг.

В ранжирующих функциях под ключевым словом **OVER** обязательным идет указание условия **ORDER BY**, по которому будет происходить сортировка ранжирования.

1. **ROW_NUMBER** – функция возвращает номер строки и используется для нумерации;
2. **RANK** — функция возвращает ранг каждой строки. В данном случае значения уже анализируются и, в случае нахождения одинаковых, возвращает одинаковый ранг с пропуском следующего значения;
3. **DENSE_RANK** — функция возвращает ранг каждой строки. Но в отличие от функции RANK, она для одинаковых значений возвращает ранг, не пропуская следующий;
4. **NTILE** – это функция, которая позволяет определить к какой группе относится текущая строка. Количество групп задается в скобках.

Применение ранжирующих функций

```
select name, subject, grade,
        row_number() over (partition by name order by grade desc),
        rank() over (partition by name order by grade desc),
        dense_rank() over (partition by name order by grade desc)
from student_grades;
```

name character varying	subject character varying	grade integer	row_number bigint	rank bigint	dense_rank bigint
Маша	физика	5	1	1	1
Маша	математика	4	2	2	2
Маша	история	3	3	3	3
Маша	русский	3	4	3	3
Петя	физика	5	1	1	1
Петя	русский	4	2	2	2
Петя	история	4	3	2	2
Петя	математика	3	4	4	3

Пропуск значения «3»

Без пропуска значения

Про NULL в случае ранжирования:

Для SQL пустые NULL значения будут определяться одинаковым рангом

Функции смещения

Это функции, которые позволяют перемещаясь по выделенной партиции таблицы обращаться к предыдущему значению строки или крайним значениям строк в партиции.

1. **LAG** или **LEAD** – функция **LAG** обращается к данным из предыдущей строки окна, а **LEAD** к данным из следующей строки. Функцию можно использовать для того, чтобы сравнивать текущее значение строки с предыдущим или следующим. Имеет три параметра: столбец, значение которого необходимо вернуть, количество строк для смещения (по умолчанию 1), значение, которое необходимо вернуть если после смещения возвращается значение **NULL**;
2. **FIRST_VALUE**()/**LAST_VALUE**() - функции возвращающие первое или последнее значение столбца в указанной партиции. В качестве аргумента указывает столбец, значение которого нужно вернуть. В оконной функции под словом **OVER** обязательное указание **ORDER BY** условия.

Таблица для примера

```
--создание таблицы
create table grades_quartal
      ( name varchar,
      quartal varchar,
      subject varchar,
      grade int);
--наполнение таблицы данными
insert into grades_quartal
( values ('Петя', '1 четверть', 'физика', 4),
('Петя', '2 четверть', 'физика', 3),
('Петя', '3 четверть', 'физика', 4),
('Петя', '4 четверть', 'физика', 5) );
```

name	quartal	subject	grade
Петя	1 четверть	физика	4
Петя	2 четверть	физика	3
Петя	3 четверть	физика	4
Петя	4 четверть	физика	5

Применение функции смещения

На простом примере видно, как можно в одной строке получить текущую оценку, предыдущую и следующую оценки Пети в четвертях.

```
select name, quartal, subject, grade,
lag(grade) over (order by quartal) as previous_grade,
lead(grade) over (order by quartal) as next_grade
from grades_quartal;
```

name	quartal	subject	grade	previous_grade	next_grade
Петя	1 четверть	физика	4	[NULL]	3
Петя	2 четверть	физика	3	4	4
Петя	3 четверть	физика	4	3	5
Петя	4 четверть	физика	5	4	[NULL]

Аналитические функции

Аналитические функции — это функции которые возвращают информацию о распределении данных и используются для статистического анализа.

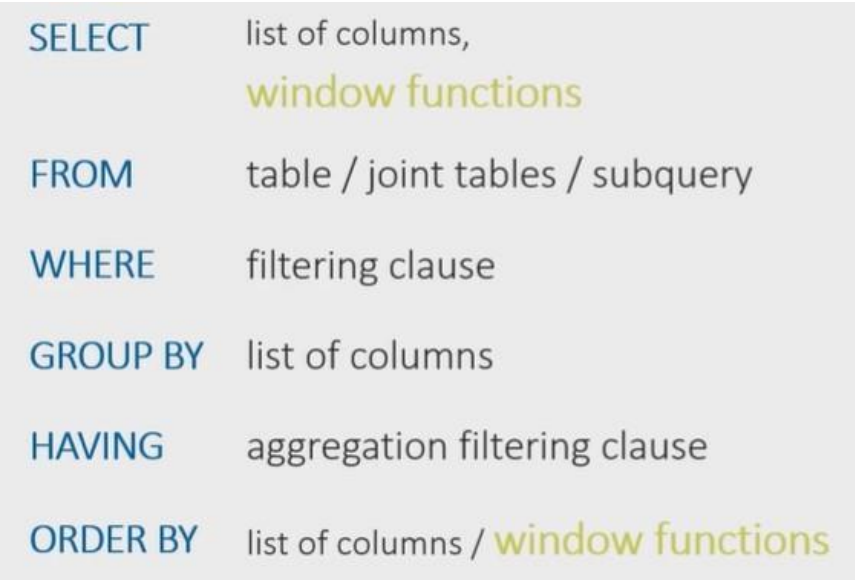
1. **CUME_DIST** — вычисляет интегральное распределение (относительное положение) значений в окне;
2. **PERCENT_RANK** — вычисляет относительный ранг строки в окне;
3. **PERCENTILE_CONT** — вычисляет процентиль на основе постоянного распределения значения столбца. В качестве параметра принимает процентиль, который необходимо вычислить;
4. **PERCENTILE_DISC** — вычисляет определенный процентиль для отсортированных значений в наборе данных. В качестве параметра принимает процентиль, который необходимо вычислить.

Важно! У функций PERCENTILE_CONT и PERCENTILE_DISC, столбец, по которому будет происходить сортировка, указывается с помощью ключевого слова WITHIN GROUP.

Процентиль — это *значение, которое заданная случайная величина не превышает с фиксированной вероятностью, заданной в процентах.*

Порядок расчета оконных функций

1. Сначала выполняется команда выборки таблиц, их объединения и возможные подзапросы под командой **FROM**.
2. Далее выполняются условия фильтрации **WHERE**, группировки **GROUP BY** и возможная фильтрация с **HAVING**
3. Только потом применяется команда выборки столбцов **SELECT** и расчет **оконных функций** под выборкой.
4. После этого идет условие сортировки **ORDER BY**, где тоже можно указать столбец расчета **оконной функции** для сортировки.
5. Здесь важно уточнить, что партиции или окна оконных функций создаются после разделения таблицы на группы с помощью команды **GROUP BY**, если эта команда используется в запросе.



Порядок расчета оконных функций

• Оконные функции разрешается использовать в запросе только в списке **SELECT** и предложении **ORDER BY**.
• Во всех остальных предложениях, включая **GROUP BY**, **HAVING** и **WHERE**, они запрещены.
• Это объясняется тем, что логически они выполняются после этих предложений, а также после неоконных агрегатных функций, и значит агрегатную функцию можно вызывать в аргументах оконной, но не наоборот.
• Если вам нужно отфильтровать или сгруппировать строки после вычисления оконных функций, вы можете использовать вложенный запрос.
Например:

```
SELECT depname, empno, salary, enroll_date
FROM
        (SELECT depname, empno, salary, enroll_date,
        rank() OVER (PARTITION BY depname ORDER BY salary DESC, empno)
        AS pos FROM empsalary ) AS ss
WHERE pos < 3;
```

Данный запрос покажет только те строки внутреннего запроса, у которых rank (порядковый номер) меньше 3.

Самосоединение отношений

Подзапросы

Самосоединение

В команде SELECT можно обратиться к одной и той же таблице несколько раз. А для того чтобы исключить соединение записи таблицы с самой собой в запросе на самосоединение необходимо также указывать условие типа "не равно" (<>, >, <).

Пример использования самосоединения:

Вывести список детей сотрудников, у которых есть младшие братья или сёстры:

```
SELECT e.name, c1.name AS child1, c1.born AS born1,
    c2.name AS child2, c2.born AS born2
FROM children c1, children c2, emp e
    WHERE c1.tabno=e.tabno -- первое условие соединения
    AND c1.tabno=c2.tabno  -- второе условие соединения
    AND c1.born<c2.born    -- условие исключения
ORDER BY 1, 3;
```

Результат самосоединения

TabNo	**Name**	Born	Sex
988	Вадим	03.05.1995	м
110	Ольга	18.07.2001	ж
023	Илья	19.02.1987	м
023	Анна	26.12.1989	ж
909	Инна	25.01.2008	ж
909	Роман	21.11.2006	м
909	Антон	06.03.2009	м

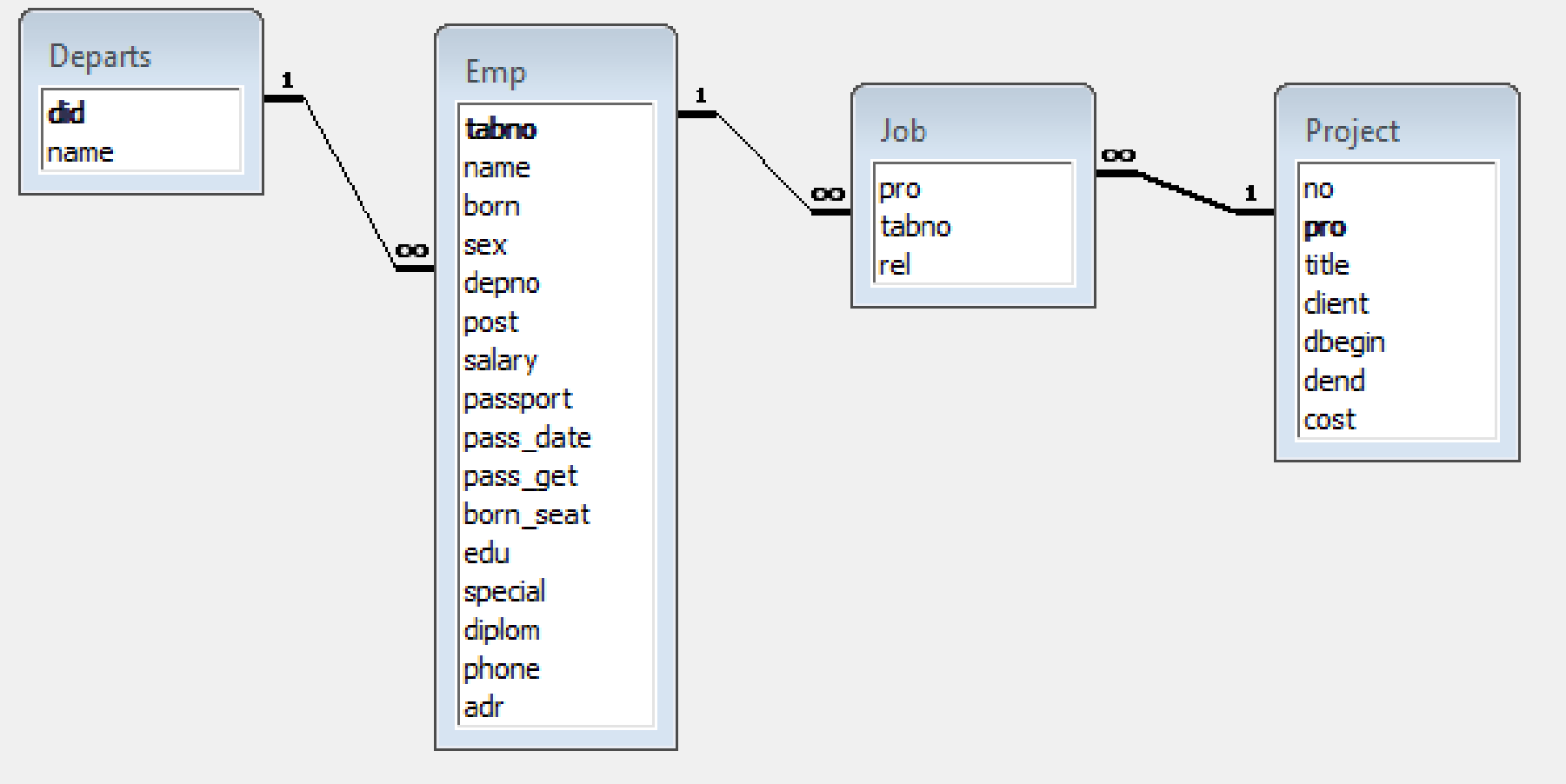
NAME	CHILD1	BORN1	CHILD2	BORN2
Малова Л.А.	Илья	19.02.1987	Анна	26.12.1989
Серова Т.В.	Роман	21.11.2006	Инна	25.01.2008
Серова Т.В.	Роман	21.11.2006	Антон	06.03.2009
Серова Т.В.	Инна	25.01.2008	Антон	06.03.2009

Подзапросы

Подзапрос – это запрос SELECT, расположенный внутри другой команды.
Подзапросы можно разделить на следующие группы в зависимости от возвращаемых результатов:
- ✓**скалярные**
- ✓**векторные**
- ✓**табличные**

Подзапросы бывают:
- ✓ **некоррелированные** – не содержат ссылки на запрос верхнего уровня; вычисляются один раз для запроса верхнего уровня;
- ✓**коррелированные –** содержат условия, зависящие от значений полей в основном запросе; вычисляются для каждой строки запроса верхнего уровня.

Пример БД: проектная организация



Departs – отделы, Project – проекты,

Emp – сотрудники, Job – участие в проектах.

Данные таблицы Emp (сотрудники)

TabNo	*DepNo*	Name	Post	Salary	Born	Phone
988	*1*	Рюмин В.П.	начальник отдела	4850.00	01.02.1970	115-26-12
909	*1*	Серова Т.В.	вед. программист	4850.00	20.10.1981	115-91-19
829	*1*	Дурова А.В.	экономист	4350.00	03.10.1978	115-26-12
819	*1*	Тамм Л.В.	экономист	4350.00	13.11.1985	115-91-19
100	*2*	Волков Л.Д.	программист	4650.00	16.10.1982	null
110	*2*	Буров Г.О.	бухгалтер	4288.00	22.05.1975	115-46-32
023	*2*	Малова Л.А.	гл. бухгалтер	5924.00	24.11.1954	114-24-55
130	*2*	Лукина Н.Н.	бухгалтер	4288.00	12.07.1979	115-46-32
034	*3*	Перова К.В.	делопроизводитель	3200.00	24.04.1988	null
002	*3*	Сухова К.А.	начальник отдела	4850.00	08.06.1948	115-12-69
056	*5*	Павлов А.А.	директор	8000.00	05.05.1968	115-33-44
087	*5*	Котова И.М.	секретарь	3500.00	16.09.1990	115-33-65
088	*5*	Кроль А.П.	зам.директора	7000.00	18.04.1974	115-33-01

Расположение подзапросов в команде select

- Чаще всего подзапрос располагается в части **WHERE.**

Пример 1:

```
select * from emp
    where salary > (select avg(salary) from emp);
```

DEPNO	NAME	POST	SALARY
2	Малова Л.А.	гл. бухгалтер	59240
5	Павлов А.А.	директор	80000
5	Кроль А.П.	зам. директора	70000

Пример 2. :

```
select * from emp
    where salary > ALL (select avg(salary) from emp group by depno);
```

Примеры использования подзапросов в части WHERE

Выдать список сотрудников, имеющих детей:

а) с помощью операции соединения таблиц:

```
SELECT  e.*
FROM  emp e, children c
WHERE e.tabno=c.tabno;
```

б) с помощью некоррелированного векторного подзапроса:

```
SELECT  *
FROM  emp
WHERE  tabno  IN  (SELECT  tabno FROM children);
```

в) с помощью коррелированного табличного подзапроса:

```
SELECT  *
FROM  emp  e
WHERE  EXISTS (SELECT * FROM children c
                    WHERE e.tabno=c.tabno);
```

Расположение подзапросов в команде select

- Подзапрос в части **FROM.**

Например,

```
select * from emp e
    where salary > (select avg(salary) from emp m
                    where m.depno = e.depno);
```

Это работает долго, т.к. коррелированный подзапрос вычисляется для каждой строки основного запроса. Можно ускорить выполнение данного запроса:

```
select *
    from emp e,
    (select depno, avg(salary) sal
            from emp
            group by depno) m   -- подзапрос вычисляется 1 раз
    where m.depno = e.depno
            and salary > sal;
```

Расположение подзапросов в команде select

- Подзапрос в части **HAVING.**
  Например,

```
select depno, avg(salary) sal
    from emp
    group by depno
    having avg(salary) < (select avg(salary) from emp);
```

- Подзапрос в части **SELECT.**
  Например,

```
 select depno, name,
    (select count(*) from job j where j.tabno = e.tabno)
cnt
from emp e;
```

Этот запрос выведет даже тех сотрудников, которые не участвуют в проектах
(для них **cnt** будет равен 0).

Подробнее о подзапросах

Подзапросы

Язык **SQL** разрешает использовать в других операторах языка **DML** подзапросы, которые являются внутренними запросами, определяемыми оператором **SELECT**.

Подзапрос - очень мощное средство языка **SQL**. Он позволяет строить сложные иерархии запросов, многократно выполняемые в процессе построения результирующего набора или выполнения одного из операторов изменения данных (**DELETE**, **INSERT**, **UPDATE**).

Условно подзапросы иногда подразделяют на три типа, каждый из которых является сужением предыдущего:

- **табличный подзапрос**, возвращающий набор строк и столбцов;
- **подзапрос строки**, возвращающий только одну строку, но, возможно, несколько столбцов (такие подзапросы часто используются во встроенном **SQL**);
- **скалярный подзапрос**, возвращающий значение одного столбца в одной строке.

Подзапросы

Подзапрос позволяет решать следующие задачи:

- определять набор строк, добавляемый в таблицу на одно выполнение оператора **INSERT**;
- определять данные, включаемые в представление, создаваемое оператором **CREATE VIEW** ;
- определять значения, модифицируемые оператором **UPDATE**;
- указывать одно или несколько значений во фразах **WHERE** и **HAVING** оператора **SELECT**;
- определять во фразе **FROM** таблицу как результат выполнения подзапроса;
- применять коррелированные подзапросы. Подзапрос называется ***коррелированным***, если запрос, содержащийся в предикате, имеет ссылку на значение из таблицы (внешней к данному запросу), которая проверяется посредством данного предиката.

Подзапрос

Некоторые **СУБД** (например, **СУБД Oracle**) позволяют на основе *подзапроса* создавать новые таблицы с помощью оператора **CREATE TABLE**.

Простым примером использования *подзапроса* может служить следующий оператор:

SELECT * from tbl1 WHERE f2=(SELECT f2 FROM tbl2 WHERE f1=1);

В данном операторе *подзапрос* всегда должен возвращать **единственное** значение, которое будет проверяться в предикате. Если *подзапрос* вернет более одного значения, то СУБД выдаст сообщение об ошибке выполнения SQL-оператора.

В случае если *подзапрос* не выберет ни одной строки, то предикат будет равен **UNKNOWN**, что большинством СУБД интерпретируется как **FALSE**.

Подзапрос

Стандарт определяет запись предиката в форме "значение оператор *подзапрос*". Однако некоторые **СУБД** также позволяют записывать предикат в форме, указывающей *подзапрос* слева от оператора сравнения.

Например:

SELECT * from tbl1 WHERE (SELECT f2 FROM tbl2 WHERE f1=1) = f2;

Очень часто с *подзапросами* используются агрегирующие функции, предоставляющие возможность сформулировать условие типа "больше, чем среднее по группе".

Например:

SELECT f1,f2,f3 FROM tbl1 WHERE f2> (SELECT AVG(f2) FROM tbl1);

Подзапрос

Если результатом *подзапроса* становится группа строк (это случается всегда, когда условие не гарантирует уникальности значения проверяемого предикатом внутреннего запроса), то следует использовать оператор **IN**, осуществляющий выбор одного значения из указываемого множества.

Например:

SELECT * from tbl1 WHERE f2 **IN** (SELECT f2 FROM tbl2 WHERE f1=1);

В этом случае предикат принимает значение TRUE, если хотя бы одно из значений, возвращаемых *подзапросом*, удовлетворяет условию.

Однако применение оператора **IN** имеет и некоторые смысловые недостатки: в запросе четко не определяется, сколько строк должны быть результатом выполнения запроса.

При построении отношений для реальной модели данных это может приводить к некоторой неоднозначности и зависимости от самих данных. Если модель данных предполагает в качестве постоянного результата *подзапроса* наличие только одной строки и, соответственно, использует оператор сравнения =, а структура данных позволяет ввод значений, когда в результате *подзапроса* будет более одной строки, то при использовании такого SQL-оператора в какой-то момент времени может проявиться ошибка.

Подзапрос

Если в запросе участвуют более двух таблиц, то для большей наглядности имена полей иногда квалифицируют именами таблиц, указывая их через точку. Стандарт позволяет не квалифицировать имя поля именем таблицы в том случае, если не возникает неоднозначности (поле сначала ищется в таблице, указанной фразой **FROM** текущего запроса, затем внешнего запроса и т.д.).

*Очень часто вместо записи оператора **SELECT** с использованием подзапроса можно применять соединения. Однако на практике большинство СУБД подзапросы выполняют более эффективно. ?*

При проектировании комплекса программ с критичными требованиями по быстродействию, разработчик должен проанализировать план выполнения SQL-оператора для конкретной СУБД.

Наиболее продвинутые СУБД, такие как Oracle, предоставляют ряд SQL-операторов, позволяющих оценить производительность выполнения конкретного оператора языка SQL, а также определить уровень оптимизации, применяемый для данного оператора.

Подзапрос может быть указан как в предикате, определяемом фразой **WHERE**, так и в предикате по группам, определяемом фразой **HAVING**:

SELECT avg_f1, COUNT (f2) from tbl1 GROUP BY avg_f1 **HAVING** avg_f1 >(SELECT f1 FROM tbl1 WHERE f3='a1');

Коррелированные подзапросы

В операторе **SELECT** из внутреннего *подзапроса* можно ссылаться на столбцы внешнего запроса, указанного во фразе **SELECT**. Такой *подзапрос* выполняется для каждой строки таблицы, определяя условие ее вхождения в формируемый результирующий набор.

Например:

```
SELECT * from tbl1 t1 WHERE f2 IN (SELECT f2
FROM tbl2 t2 WHERE t1.f3=t2.f3);
```

В данном случае для каждой строки таблицы **tbl1** будет проверяться условие, что значение поля **f2** совпадает со значением строки **таблицы tbl2**, где значение поля **f3** равно значению поля **f3** внешней таблицы (**tbl1**).

Очень часто требуется, чтобы *подзапрос* использовал те же данные, что и внешняя таблица. В этом случае обязательно применение **алиасов**.

Например:

```
SELECT * from tbl1 t_out WHERE f2< (SELECT AVG(f2) FROM
  tbl1 t_in WHERE t_out.f1= t_in.f1);
```

В случае *коррелированного подзапроса* во фразе **HAVING** можно использовать только агрегирующие функции, так как каждый раз на момент выполнения *подзапроса* в качестве проверяемой строки, к значениям которой имеет доступ *подзапрос*, выступает результат группирования строк на основе агрегирующих функций основного запроса.

Например:

```
SELECT f1, COUNT(*), SUM(f2) from tbl1 t1
GROUP BY f1 HAVING SUM(f2)> (SELECT MIN(f2)*4 FROM tbl1
  t1_in WHERE t1.f1=t1_in.f1);
```

Построение предиката для подзапроса, возвращающего несколько строк

Если в предикате надо сравнить значение с некоторым множеством, то, как было показано выше, можно использовать оператор **IN**.

Для того чтобы проверить, существуют ли строки, удовлетворяющие конкретному условию *подзапроса*, применяется оператор **EXISTS (существования)**.

Например:

```
SELECT f1,f2,f3 from tbl1 WHERE EXISTS (SELECT * FROM
  tbl1 WHERE f4='10/11/2003');
```

Этот запрос будет формировать не пустой результирующий набор только в том случае, если в какое-либо значение столбца f4 таблицы была занесена дата, например: '10/11/2003‘ **(подзапрос должен выдавать хотя бы одну строку).**

Преимущество применения оператора **EXISTS** с результатами *подзапроса* состоит в том, что *подзапрос* может возвращать как множество строк, так и множество столбцов.

При *коррелированном подзапросе* оператор **EXISTS** будет вычисляться каждый раз для каждой строки внешнего запроса.

ANY и ALL

В стандарте SQL-92 не предусмотрено использование в *подзапросах*, к которым применяется оператор **EXISTS** агрегирующих функций. Однако некоторые СУБД позволяют такой вид *подзапросов*.

Для использования результата *подзапроса* в предикате также применяются операторы **ANY** и **ALL**.

Пример использования оператора **ANY**:

```
SELECT f1,f2,f3 from tbl1 WHERE f3 = ANY
  (SELECT f3 FROM tbl2);--хотя бы 1 значение
  tbl1.f3=tbl2.f3
```

Данный оператор определяет, что в результирующий набор будут включены все строки, значение столбца **f3 таблицы tbl1** которых присутствует в таблице **tbl2**.

Применение подзапросов в операторах изменения данных

К операторам языка **DML**, кроме оператора **SELECT**, относятся операторы, позволяющие изменять данные в таблицах:

- оператор **INSERT**, выполняющий добавление одной или нескольких строк в таблицу

```
INSERT INTO table_name [ (field .,:) ] { VALUES (value .,:) }|subquery |{DEFAULT VALUES};
```

- оператор **DELETE**, удаляющий из таблицы одну или несколько строк

```
DELETE FROM table_name [{ WHERE condition }|{ WHERE CURRENT OF cursor_name }];
```

- оператор **UPDATE**, изменяющий значения столбцов таблицы.

```
UPDATE table_name SET {field={expr|NULL|DEFAULT }}.,
[{WHERE condition}|{WHERE CURRENT OF cursor_name }];
```

Выражение **expr**, используемое для вычисления значения столбца, может быть как простым выражением, так и ***подзапросом***, возвращающим единственное значение. В выражении можно ссылаться на старое значение изменяемого столбца и других столбцов текущей записи.

Расположение подзапросов в командах DML

В команде **INSERT**:

- Вместо **VALUES**, например, добавление данных из одной таблицы в другую:

```
insert into emp select * from new_emp;
```

В команде **UPDATE**:

- в части **WHERE** для вычисления условий, например, повышение зарплаты на 10% всем участникам проектов:

```
update emp set salary = salary*1.1
    where tabNo IN (select tabNo from job);
```

- в части **SET** для вычисления значений полей, например, повышение зарплаты на 10% за каждое участие сотрудника в проекте:

```
update emp e set salary = salary*(1+(select count(*)/10 from job j
                                    where j.tabNo = e.tabNo) );
```

В команде **DELETE**:

- в части **WHERE** для вычисления условий, например, удаление сведений об участии в закончившихся проектах:

```
delete from job
    where pro IN (select pro from project where dend < sysdate);
```