

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО
ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

***Федеральное государственное автономное
образовательное учреждение
высшего образования***

**«Национальный исследовательский ядерный университет
«МИФИ»»**

Кафедра №42 «Криптология и кибербезопасность»

ЛАБОРАТОРНАЯ РАБОТА №1-3:

«Сложные запросы на выборку. Соединения»

Аверин Владислав

Группа Б19-505

Апрель, 2022

Содержание

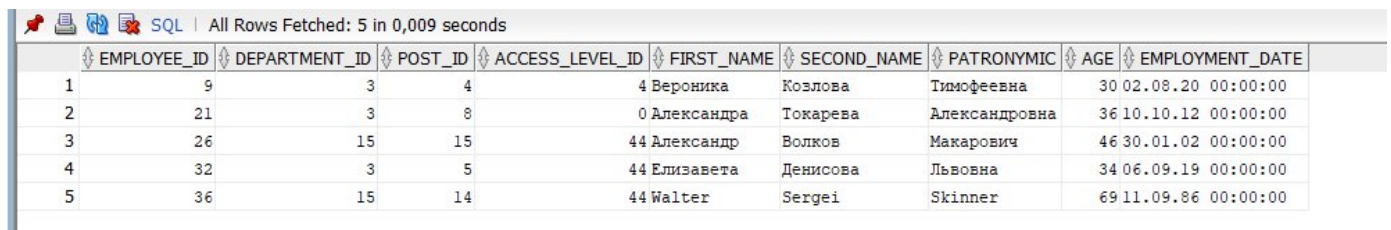
1. Подзапросы.....	3
2. Соединения.....	4
3. Иерархические запросы.....	9
4. Аналитические функции	11
Выводы.....	14

1. Подзапросы

1.1 Для начала найдем сотрудников, которые не связаны ни с одним делом (балласт, в общем):

```
SELECT * FROM Employees  
WHERE employee_id NOT IN  
(  
    SELECT DISTINCT employee_id FROM AssignedCases  
)  
ORDER BY employee_id;
```

Вывод:

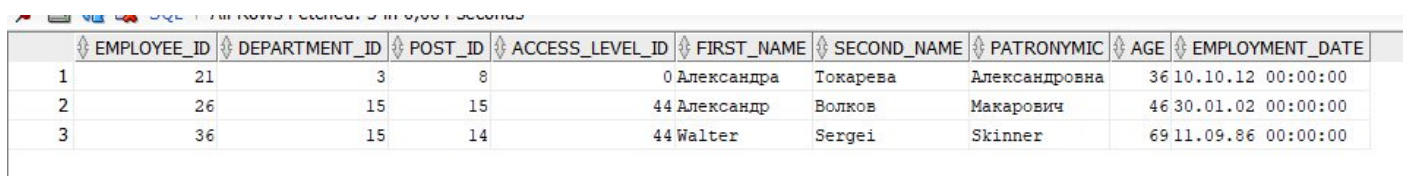


	EMPLOYEE_ID	DEPARTMENT_ID	POST_ID	ACCESS_LEVEL_ID	FIRST_NAME	SECOND_NAME	PATRONYMIC	AGE	EMPLOYMENT_DATE
1	9	3	4	4	Вероника	Козлова	Тимофеевна	30	02.08.20 00:00:00
2	21	3	8	0	Александра	Токарева	Александровна	36	10.10.12 00:00:00
3	26	15	15	44	Александр	Волков	Макарович	46	30.01.02 00:00:00
4	32	3	5	44	Елизавета	Денисова	Львовна	34	06.09.19 00:00:00
5	36	15	14	44	Walter	Sergei	Skinner	69	11.09.86 00:00:00

*(department 3 - это бухгалтерия, а 15 - высшие должностные лица)

Усложним немного запрос, добавив, например, ограничение на стаж работы больше 5 лет (просто так):

```
SELECT * FROM Employees  
WHERE employee_id NOT IN  
(  
    SELECT DISTINCT employee_id FROM AssignedCases  
) AND ( SYSDATE - employment_date ) > 365*5  
ORDER BY employee_id;
```



	EMPLOYEE_ID	DEPARTMENT_ID	POST_ID	ACCESS_LEVEL_ID	FIRST_NAME	SECOND_NAME	PATRONYMIC	AGE	EMPLOYMENT_DATE
1	21	3	8	0	Александра	Токарева	Александровна	36	10.10.12 00:00:00
2	26	15	15	44	Александр	Волков	Макарович	46	30.01.02 00:00:00
3	36	15	14	44	Walter	Sergei	Skinner	69	11.09.86 00:00:00

1.2 «Найти людей, работающих в участке с самого "открытия" (т.е. те, у которых employment_date меньше, чем у самого раннего дела в таблице Cases)»

```
SELECT * FROM Employees
WHERE employment_date <
(
    SELECT MIN(start_date) FROM Cases
);
```

Вывод:

SQL All Rows Fetched: 1 in 0,009 seconds										
EMPLOYEE_ID	DEPARTMENT_ID	POST_ID	ACCESS_LEVEL_ID	FIRST_NAME	SECOND_NAME	PATRONYMIC	AGE	EMPLOYMENT_DATE		
1	36	15	14	44 Walter	Sergei	Skinner	69	11.09.86 00:00:00		

2. Соединения

2.1 Выведем статистику штата сотрудников по отделам (кол-во человек в каждом отделе):

```
SELECT department_id, COUNT(*) AS staff FROM Employees
GROUP BY department_id
ORDER BY department_id;
```

Вывод:

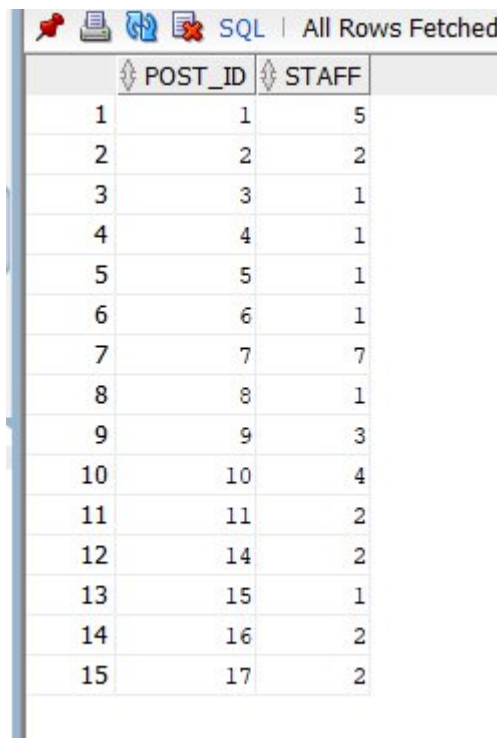
DEPARTMENT_ID	STAFF
1	2
2	2
3	3
4	2
5	2
6	2
7	1
8	2
9	2
10	2
11	2
12	2
13	3
14	2
15	3
16	2
17	1

Note: такой запрос не будет показывать отсутствие штата у каких-то отделов (логично, ведь в таблице записей о них вообще нету). В данном случае этого не видно, т.к. во всех отделах имеются сотрудники. Однако попробуем написать ту же статистику по должностям:

```
SELECT post_id, COUNT(*) AS staff FROM Employees
```

```
GROUP BY post_id
```

```
ORDER BY post_id;
```



	POST_ID	STAFF
1	1	5
2	2	2
3	3	1
4	4	1
5	5	1
6	6	1
7	7	7
8	8	1
9	9	3
10	10	4
11	11	2
12	14	2
13	15	1
14	16	2
15	17	2

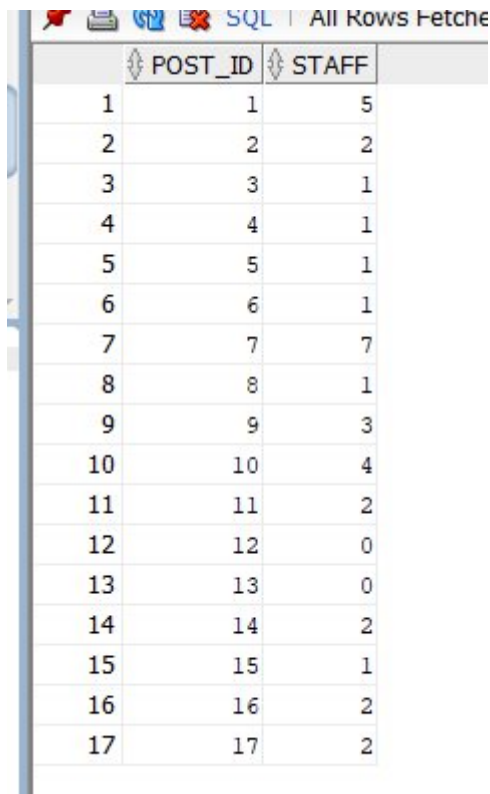
Заметим, что в запросе отсутствуют записи post_id = 12, 13. Исправим это, реализовав запрос через внешнее (left outer) соединение с таблицей Posts, чтобы в таблице были все должности (не забыв при этом, как обработать null-значения):

```
SELECT Posts.post_id, COUNT(CASE WHEN employee_id IS NOT NULL THEN 0 END) AS  
staff FROM Posts
```

```
LEFT JOIN Employees ON employees.post_id = posts.post_id
```

```
GROUP BY Posts.post_id;
```

Получаем следующее:



	POST_ID	STAFF
1	1	5
2	2	2
3	3	1
4	4	1
5	5	1
6	6	1
7	7	7
8	8	1
9	9	3
10	10	4
11	11	2
12	12	0
13	13	0
14	14	2
15	15	1
16	16	2
17	17	2

Как видим, напротив post_id = 12 и 13 расположены верные значения.

(Не важно, какое число использовать после 'then', т.к. count считает именно строки, а не значения атрибута. Если бы, к примеру, вместо COUNT было бы SUM, то надо было бы поставить 1)

Немного другой вариант запроса (который я хотел сделать первоначально), в котором сначала производится агрегирование, а уже затем - соединение с необходимой обработкой null значений:

```
SELECT Posts.post_id, COALESCE(tabl1.res, 0) AS staff FROM  
(  
    SELECT post_id, COUNT(*) AS res FROM Employees  
    GROUP BY Employees.post_id  
) tabl1  
RIGHT JOIN Posts  
ON tabl1.post_id = Posts.post_id  
ORDER BY Posts.post_id;
```

SQL | All Rows Fetched

	POST_ID	STAFF
1	1	5
2	2	2
3	3	1
4	4	1
5	5	1
6	6	1
7	7	7
8	8	1
9	9	3
10	10	4
11	11	2
12	12	0
13	13	0
14	14	2
15	15	1
16	16	2
17	17	2

Результат тот же, но второй вариант как по мне более громоздкий.

2.2 Изменим условие п. 1.1 на то, что выбранные сотрудники вместо связи с каким-то делом (что фактически означает наличие сотрудника в таблице) не должны на данный момент вести никаких дел. То есть в данном случае нам необходима дополнительная информация из таблицы Cases (состояние дела), а не только найти все employee_id из Assigned Cases:

```

SELECT * FROM Employees
WHERE employee_id NOT IN
(
    SELECT DISTINCT employee_id
    FROM Cases
    INNER JOIN AssignedCases ON AssignedCases.case_id = Cases.case_id
    WHERE status_id = 1
)
ORDER BY employee_id;

```

Вывод:

	EMPLOYEE_ID	DEPARTMENT_ID	POST_ID	ACCESS_LEVEL_ID	FIRST_NAME	SECOND_NAME	PATRONYMIC	AGE	EMPLOYMENT_DATE
1	5	13	7	24	Глеб	Желтов	Георгиевич	36	02.04.13 00:00:00
2	9	3	4	4	Вероника	Козлова	Тимофеевна	30	02.08.20 00:00:00
3	21	3	8	0	Александра	Токарева	Александровна	36	10.10.12 00:00:00
4	26	15	15	44	Александр	Волков	Макарович	46	30.01.02 00:00:00
5	28	16	16	34	Dana	Katherine	Scully	22	12.05.21 00:00:00
6	30	16	16	34	Fox	William	Mulder	24	22.10.20 00:00:00
7	32	3	5	44	Елизавета	Денисова	Львовна	34	06.09.19 00:00:00
8	36	15	14	44	Walter	Sergei	Skinner	69	11.09.86 00:00:00

Изменив немного запрос, найдем сотрудников, которые занимаются на данный момент не больше чем двумя делами одновременно. Для этого снова найдем сотрудников, которые нам не подходят, и вычтем их из всех сотрудников (используя MINUS вместо 'WHERE employee_id NOT IN' просто так, чтобы было):

SELECT employee_id **FROM** Employees

MINUS

(

SELECT employee_id **FROM**

(

SELECT employee_id, **COUNT**(*) **AS** actual_cases

FROM Cases

INNER JOIN AssignedCases **ON** AssignedCases.case_id = Cases.case_id

WHERE status_id = 1

GROUP BY employee_id

)

WHERE actual_cases > 2

)

ORDER BY employee_id;

	EMPLOYEE_ID
1	1
2	5
3	6
4	7
5	9
6	10
7	12
8	13
9	17
10	19
11	20
12	21
13	23
14	26
15	27
16	28
17	29
18	30
19	31
20	32
21	33
22	34
23	36

3. Иерархические запросы

С ними были небольшие проблемы, т.к. в моей БД всунуть иерархическое наследование было некуда (в ту структуру, которая была у меня). Поэтому, я решил просто создать специально для этого типа запросов новую таблицу иерархической системы персонала (заодно дополнив БД новой инфой). Она включает в себя поля id (для уникальности строк, т.к. отношение таблицы к Posts будет 0..n), post_id и pid (идентификатор родителя, на который ссылается данная должность в этой же таблице).

(Note: ниваажна, что оно может выглядеть не логичным; я попытался описать максимально правдоподобную структуру должностей; и я хз, нормально ли, что в одной таблице два внешних ключа, которые ссылаются на один и тот же атрибут).

Создадим саму таблицу

```
CREATE TABLE PeckingOrder
(
    post_id,
    pid NUMBER(2,0),
    CONSTRAINT PeckingOrder_pk PRIMARY KEY (post_id),
    CONSTRAINT PeckingOrder_fk_post
        FOREIGN KEY (post_id)
        REFERENCES Posts(post_id),
    CONSTRAINT PeckingOrder_fk_pid
        FOREIGN KEY (pid)
        REFERENCES Posts(post_id)
);
```

Заполним ее (я ее заполнял через *sql developer*):

	POST_ID	PID
1	1	7
2	2	7
3	3	7
4	4	5
5	5	15
6	6	13
7	7	15
8	8	15
9	9	7
10	10	7
11	11	14
12	12	17
13	13	15
14	14	15
15	15	(null)
16	16	14
17	17	7

Собственно, сам запрос (идея "отрисовки" дерева через *lpad* честно и полностью скомунизжена с Хабра):

```
SELECT LPAD(' ', 3*LEVEL)||post_name AS pecking_order
```

```
FROM
```

```
(
```

```
SELECT Posts.post_id, PeckingOrder.pid, Posts.post_name FROM Posts
```

```
INNER JOIN PeckingOrder
```

```
ON Posts.post_id = PeckingOrder.post_id
```

```
)
```

```
CONNECT BY PRIOR post_id = pid
```

```
START WITH pid IS NULL
```

```
ORDER SIBLINGS BY post_name;
```

	PECKING_ORDER
1	Начальник полиции
2	Глав.Бух
3	Бухгалтер
4	Заместитель начальника полиции
5	Испектор по связи с общественностью
6	Заместитель начальника полиции
7	спец. Агент
8	Старший дознаватель
9	Кадровик
10	Оперуполномоченный
11	Оперативник
12	Офицер
13	Следователь
14	Суд.мед.эксперт
15	Мед. Работник
16	Участковый уполномоченный
17	Эксперт-криминалист

Все же проще было эту иерархию сделать напрямую в таблице Employees по id "босса" каждого сотрудника; однако, никто же не запрещает потом поменять таблицы (хотя, по сути оба варианта не особо полезны; кроме проверки того, какую должность надо давать сотруднику при повышении, я больше назначений ей не вижу). Поэтому, пусть пока останется она, и в будущем если что использую ее для того, чтобы можно было добавить pid в таблицу Employees и наглядно расставить их значения всем сотрудникам (в зависимости от должности).

4. Аналитические функции

Предположим, нам необходимо вывести вместе с информацией о сотруднике (будем во избежание загромождения выводить только employee_id) дополнительную аналитику, например, кол-во сотрудников в его отделе. В данном случае, можно написать подзапрос со статистикой каждого отдела, и соединить его с родительской таблицей Employees:

```
SELECT employee_id, Employees.department_id, tabl2.staff FROM Employees
```

```
INNER JOIN
```

```
(
```

```
SELECT department_id, COUNT(*) AS staff FROM Employees
```

```
GROUP BY department_id
```

```
) tabl2
```

```
ON
```

```
Employees.department_id = tabl2.department_id;
```

	EMPLOYEE_ID	DEPARTMENT_ID	STAFF
1	1	5	2
2	2	1	2
3	3	6	2
4	4	11	2
5	5	13	3
6	6	14	2
7	7	13	3
8	8	10	2
9	9	3	3
10	10	8	2
11	11	9	2
12	12	12	2
13	13	7	1
14	14	10	2
15	15	15	3
16	16	4	2
17	17	8	2

Однако выглядит это довольно... Громоздко. А предположим еще, что нам нужен не один аналитический столбец, а несколько с разными механизмами (и представляем нагромождение JOIN'ов). Чтобы облегчить себе жизнь, рационально использовать аналитические функции (ORDER BY делаем, чтобы порядок строк в запросе был такой же, как в предыдущем варианте):

```
SELECT Employee_id, department_id,
COUNT(*) OVER (PARTITION BY department_id) AS staff
FROM Employees
ORDER BY employee_id;
```

	EMPLOYEE_ID	DEPARTMENT_ID	STAFF
1	1	5	2
2	2	1	2
3	3	6	2
4	4	11	2
5	5	13	3
6	6	14	2
7	7	13	3
8	8	10	2
9	9	3	3
10	10	8	2
11	11	9	2
12	12	12	2
13	13	7	1
14	14	10	2
15	15	15	3
16	16	4	2
17	17	8	2
18	18	9	2
19	19	2	2
20	20	11	2
21	21	3	3
22	22	1	2

Результаты эквивалентны, однако вторая запись более простая. Кроме того, возвращаясь к вопросу о нескольких аналитических столбцах, в одной выборке SELECT можно использовать несколько аналитических функций. Выведем, к примеру, с размером штата сотрудников отделов, кол-во сотрудников соответствующих должностей и средний возраст сотрудников отдела:

```
SELECT Employee_id, department_id,
COUNT(*) OVER (PARTITION BY department_id) AS department_staff,
AVG(age) OVER (PARTITION BY department_id) AS AVG_age,
post_id, COUNT(*) OVER (PARTITION BY post_id) AS post_staff FROM Employees
ORDER BY employee_id;
```

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_STAFF	AVG_AGE	POST_ID	POST_STAFF
1	1	5	2	29,5	1	5
2	2	1	2	30	9	3
3	3	6	2	37	2	2
4	4	11	2	39,5	9	3
5	5	13	3	30,33333333...	7	7
6	6	14	2	37	7	7
7	7	13	3	30,33333333...	7	7
8	8	10	2	36	7	7
9	9	3	3	33,33333333...	4	1
10	10	8	2	22	10	4
11	11	9	2	44,5	3	1
12	12	12	2	33,5	1	5
13	13	7	1	30	10	4
14	14	10	2	36	1	5
15	15	15	3	53	14	2
16	16	4	2	35,5	11	2
17	17	8	2	22	10	4
18	18	9	2	44,5	11	2
19	19	2	2	39,5	17	2
20	20	11	2	39,5	7	7
21	21	3	3	33,33333333...	8	1
22	22	1	2	30	9	3
23	23	6	2	37	2	2
24	24	5	2	29,5	10	4

В данных случаях мы использовали аналитические функции чтобы упростить запросы и сделать их более понятными и комплексными (ибо как мы поняли, по сути любой запрос можно представить множеством соединений).

Выводы

В ходе работы были изучены и проанализированы различные функции и методы запросов, приведены примеры их использования, а именно: агрегатные и иерархические функции, соединения и объединения, а так же механизмы подзапросов.