

Текстовая расшифровка видео:

АГРЕГАЦИЯ И ПОДЗАПРОСЫ

План:

- Считаем выгоду;
- Yo Dawg, мы положили запрос в твой запрос.

Считаем выгоду

Вспомним о существовании такого понятия, как «**Having**».

Когда мы делаем группировку, мы можем использовать «**When**», но он фильтрует только первую выборку. Если же мы хотим проводить фильтрацию в группах, нужно использовать «**Having**».

[Почитать про HAVING](#)

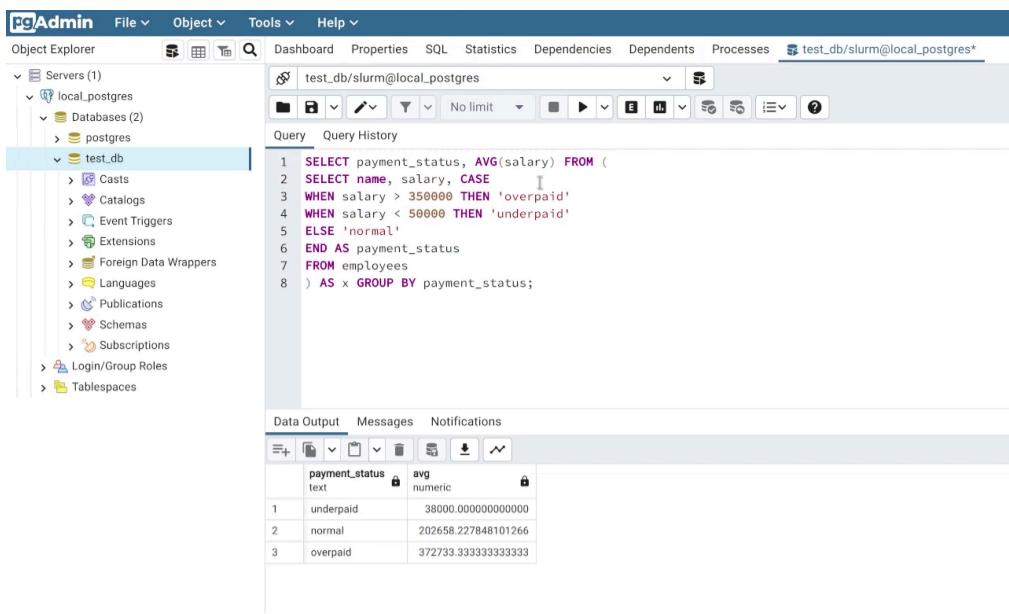
[Почему не просто WHERE](#)

Группировка и агрегация:

```
SELECT fired, SUM(salary) FROM employees GROUP BY fired HAVING fired = TRUE;
```

Мы можем использовать подзапрос. Его мы можем вставить в любое место нашего query. Его можно использовать и в «**Where**», и в «**Having**», и с ограничениями внутри агрегата, также можно использовать во «**From**» (мы генерируем временную табличку и делаем выборку напрямую из нее):





```
SELECT payment_status, AVG(salary) FROM (
    SELECT name, salary, CASE
        WHEN salary > 350000 THEN 'overpaid'
        WHEN salary < 50000 THEN 'underpaid'
        ELSE 'normal'
    END AS payment_status
    FROM employees
) AS x GROUP BY payment_status;
```

Обратите внимание, что мы группируем здесь по колонке, которую сами вычислили.

Yo Dawg, мы положили запрос в твой запрос

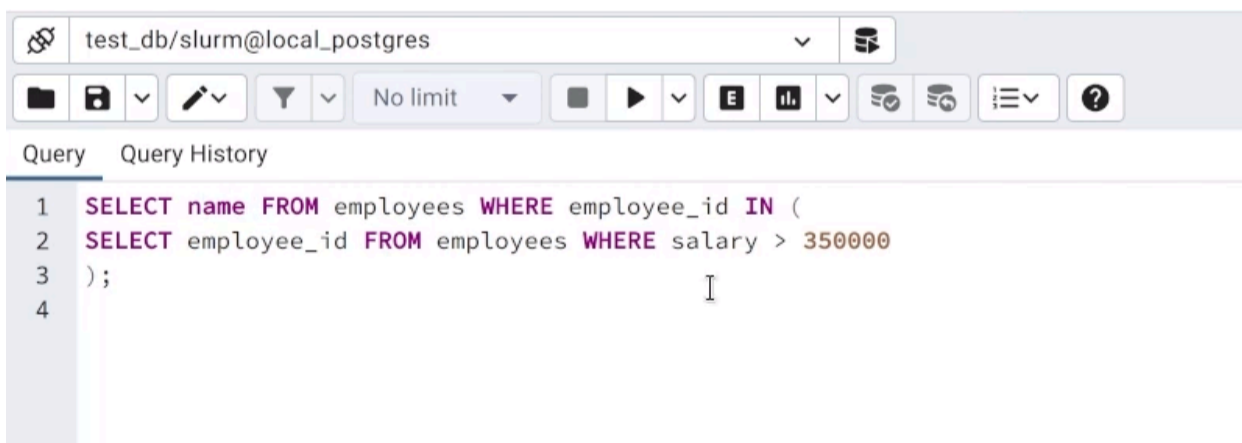
«Where» – другой вариант использования подзапроса.

Рассмотрим задачу:

```
SELECT name FROM employees WHERE employee_id IN (
    SELECT employee_id FROM employees WHERE salary > 350000
);
```

Мы достаем всех сотрудников, у которых зарплата больше 350000. При этом достать хотим только айдишник. Мы могли бы написать через «Where», но иногда в некоторых ситуациях удобнее оперировать айдишниками (в том числе при оптимизации запроса).

Выполнив данный запрос, получаем следующее:



Получаем список имен всех сотрудников, у которых зарплата достаточно высокая. Сначала мы создали виртуальную табличку и уже из нее выбрали.

Отсюда вытекает важное правило о том, как работает SQL и реляционная алгебра: **результат любого запроса – это relation.**

Можно составлять запросы большой степени множности.

Плюс:

- Некоторые задачи гораздо проще решить с использованием подзапросов.

Минус:

- Злоупотребление подзапросами может сильно затруднять работу оптимизатора.

Рассмотрим пример:

```
test_db=# ALTER TABLE employees ADD paid_status NOT NULL DEFAULT 'normal';
ERROR: column "paid_status" of relation "employees" does not exist
LINE 1: UPDATE employees SET paid_status = CASE
                        ^
test_db=# ALTER TABLE employees ADD paid_status VARCHAR(20) NOT NULL DEFAULT 'normal';
ALTER TABLE
test_db=# UPDATE employees SET paid_status = CASE
WHEN salary > 350000 THEN 'overpaid'
WHEN salary < 50000 THEN 'underpaid'
ELSE 'normal' END;
UPDATE 100
test_db=# EXPLAIN SELECT name FROM employees WHERE employee_id IN (
SELECT employee_id FROM employees WHERE salary > 350000
);
          QUERY PLAN
-----
Hash Join  (cost=7.44..14.71 rows=15 width=14)
Hash Cond: (employees.employee_id = employees_1.employee_id)
-> Seq Scan on employees  (cost=0.00..7.00 rows=100 width=18)
-> Hash  (cost=7.25..7.25 rows=15 width=4)
    -> Seq Scan on employees employees_1  (cost=0.00..7.25 rows=15 width=4)
        Filter: (salary > 350000)
(6 rows)

test_db=#
```

Работа проходит неэффективно: происходит join таблички саму на себя. Напишем запрос напрямую:

```
test_db=# ALTER TABLE employees ADD paid_status VARCHAR(20) NOT NULL DEFAULT 'normal';
ALTER TABLE
test_db=# UPDATE employees SET paid_status = CASE
WHEN salary > 350000 THEN 'overpaid'
WHEN salary < 50000 THEN 'underpaid'
ELSE 'normal' END;
UPDATE 100
test_db=# EXPLAIN SELECT name FROM employees WHERE employee_id IN (
SELECT employee_id FROM employees WHERE salary > 350000
);
          QUERY PLAN
-----
Hash Join  (cost=7.44..14.71 rows=15 width=14)
Hash Cond: (employees.employee_id = employees_1.employee_id)
-> Seq Scan on employees  (cost=0.00..7.00 rows=100 width=18)
-> Hash  (cost=7.25..7.25 rows=15 width=4)
    -> Seq Scan on employees employees_1  (cost=0.00..7.25 rows=15 width=4)
        Filter: (salary > 350000)
(6 rows)

test_db=# EXPLAIN SELECT name FROM employees WHERE salary > 350000;
          QUERY PLAN
-----
Seq Scan on employees  (cost=0.00..7.25 rows=15 width=14)
Filter: (salary > 350000)
(2 rows)

test_db=#
```

Это будет линейный скан без дополнительных сложностей.

Как вам урок?



Изучил, далее >

Слёрм ©

+7 (495) 248-05-80

[Лицензия №ДЛ-1368 от 22.08.2019](#)

[Политика конфиденциальности](#)

[Публичная оферта](#)

