



# Урок 5

[Агрегатные функции](#)

[Особенности функции COUNT](#)

[Группировка данных](#)

[Фильтрация по агрегатам](#)

[Вопросы на собеседовании](#)

[Резюме](#)

## Агрегатные функции



Агрегатные функции выполняют вычисление над набором значений и возвращают одиночное значение, которое называют агрегатом. Они используются для анализа и обобщения данных, группируя их по определенным критериям и возвращая статистические значения. Например, мы можем посчитать сумму всех покупок за сегодня, количество человек в департаменте или средний рост населения.

Основные агрегатные функции включают:



- `COUNT(1)` аналогично `COUNT(*)`, но СУБД не читает значения каждого поля, а использует константу;
- `COUNT(<поле>)` подсчитывает количество непустых значений в указанном поле;
- `COUNT(DISTINCT <поле>)` возвращает количество уникальных непустых значений в указанном поле.

#### Пример работы функции `COUNT` :

```
SELECT COUNT(*) AS count_,
       COUNT(1) AS count_1,
       COUNT(dimension) AS count_dimension,
       COUNT(distinct type) AS count_dist_type
FROM locations;
```

```
1 SELECT COUNT(*) AS count_,
2     COUNT(1) AS count_1,
3     COUNT(dimension) AS count_dimension,
4     COUNT(distinct type) AS count_dist_type
5 FROM locations;
```

{{ }}

≡

⚡

✓

LIMIT 1000

StartDE Theory

▼

📄\*

▶

Table

+

count\_

count\_1

count\_dimension

count\_dist\_type

126

126

95

44

Пример подсчета количества значений в колонке с использованием `COUNT`, `SUM` и `CASE` :

```
SELECT COUNT(1) AS row_cnt,
       SUM(CASE WHEN gender = 'Male' THEN 1 ELSE 0 END) AS male_cnt,
       SUM(CASE WHEN gender = 'Female' THEN 1 ELSE 0 END) AS female_cnt
FROM characters
WHERE status = 'Alive';
```

```
1 SELECT COUNT(1) AS row_cnt,
2     SUM(CASE WHEN gender = 'Male' THEN 1 ELSE 0 END) AS male_cnt,
3     SUM(CASE WHEN gender = 'Female' THEN 1 ELSE 0 END) AS female_cnt
4 FROM characters
5 WHERE status = 'Alive';
```

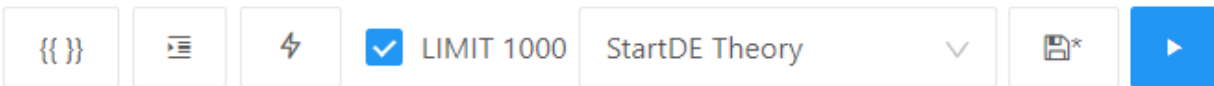


Table		
row_cnt	male_cnt	female_cnt
439	309	95

В запросе выше поле с количеством строк получаем с помощью `COUNT(1)`, а подсчет по гендерному признаку мы осуществляем с помощью суммирования результата работы функций `CASE`. В случае соответствия нужному нам гендеру мы проставляем 1, а в противоположном случае — 0. Если просуммировать эти единички, мы получим количество мужских и женских персонажей.

## Группировка данных



Для вычисления агрегатов по наборам данных используется инструкция `GROUP BY`. Она позволяет задать поле группировки и выполнять агрегирующие функции в рамках каждой группы.


Общий вид запроса с группировкой:

```
SELECT <поле группировки>, <агрегат>
  FROM table
 GROUP BY <поле группировки>;
```

### Пример использования группировки и агрегации

Нам необходимо посчитать количество потоков для каждого курса:

id	name	description
1	Data Engineer	Обучение дата-инженеров
2	Data Analyst	Обучение дата-аналитиков
3	Start ML	Начальный курс по ML

id	course_id	number	start_date
1	1	31	2024-03-07
2	1	32	2024-04-01
3	1	33	2024-04-21

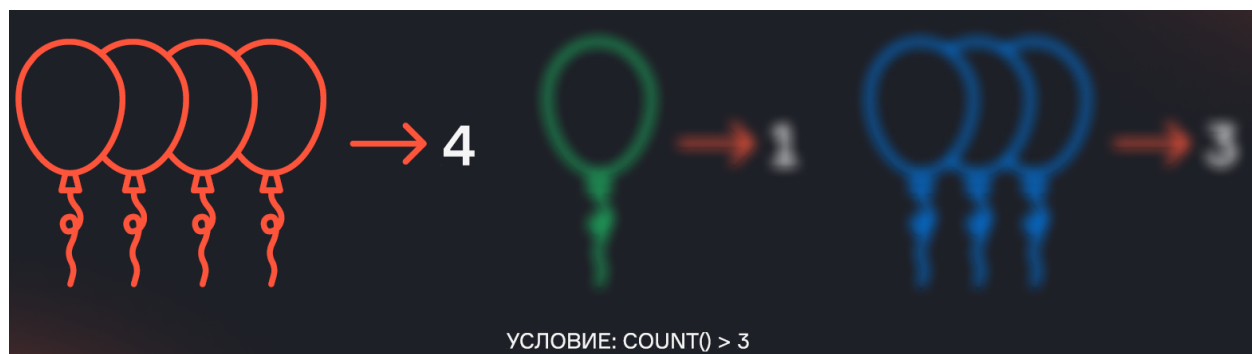
Для этого необходимо написать SQL-запрос, соединяющий таблицы курсов и потоков, а также агрегирующий поле «потоки» для каждого из курсов:

```
SELECT c.id AS course_id,
       c.name AS course_name,
       COUNT(f.id) AS flow_cnt
FROM   courses AS c
FULL JOIN flows AS f
      ON c.id = f.course_id
GROUP BY course_id, course_name
```

В запросе мы объединяем две таблицы, группируем данные по `course_id` и `course_name` и считаем количество потоков для каждого курса. В результате получаем нужную нам таблицу:

course_id	course_name	flow_cnt
1	Data Engineer	3
2	Data Analyst	0
3	Start ML	0

## Фильтрация по агрегатам




Иногда требуется отфильтровать данные после их агрегации. Для этого используется инструкция `HAVING`. Она позволяет задать условия для агрегированных данных.

Общий вид запроса с группировкой и фильтрацией по агрегату:

```
SELECT <поле группировки>, <агрегат>
  FROM table
  GROUP BY <поле группировки>
  HAVING <условие на агрегат>
```

### Пример запроса с фильтрацией по агрегату

Нам необходимо найти курсы, для которых есть хотя бы один поток.



id	name	description
1	Data Engineer	Обучение дата-инженеров
2	Data Analyst	Обучение дата-аналитиков
3	Start ML	Начальный курс по ML

id	course_id	number	start_date
1	1	31	2024-03-07
2	1	32	2024-04-01
3	1	33	2024-04-21

Для этого необходимо написать SQL-запрос, соединяющий таблицы курсов и потоков, а также агрегирующий поле «потоки» для каждого из курсов, применив условие фильтрации по агрегированному значению:



```

SELECT c.id AS course_id,
       c.name AS course_name,
       COUNT(f.id) AS flow_cnt
FROM courses AS c
FULL JOIN flows AS f
  ON c.id = f.course_id
GROUP BY course_id, course_name
HAVING COUNT(f.id) > 0

```

В запросе мы объединяем две таблицы, группируем данные по `course_id` и `course_name`, считаем количество потоков для каждого курса и затем фильтруем это количество. В результате получаем нужную нам таблицу:

course_id	course_name	flow_cnt
1	Data Engineer	3

Этот запрос возвращает только те курсы, для которых есть хотя бы один поток.

### Пример запроса с фильтрацией по агрегату:

```

SELECT char.name, COUNT(ep.name) AS ep_cnt
FROM characters AS char
FULL JOIN char_ep AS c_e
  ON char.id = c_e.character_id
FULL JOIN episodes AS ep
  ON ep.id = c_e.episode_id
GROUP BY char.name
HAVING COUNT(ep.name) > 10
ORDER BY ep_cnt DESC;

```

```

1 SELECT char.name, COUNT(ep.name) AS ep_cnt
2   FROM characters AS char
3  FULL JOIN char_ep AS c_e
4    ON char.id = c_e.character_id
5  FULL JOIN episodes AS ep
6    ON ep.id = c_e.episode_id
7  GROUP BY char.name
8  HAVING COUNT(ep.name) > 10
9  ORDER BY ep_cnt DESC;

```

{{ }}



LIMIT 1000

StartDE Theory



Table



name	ep_cnt
Morty Smith	54
Rick Sanchez	54
Beth Smith	52
Summer Smith	51

Фильтрацию данных по неагрегированным значениям можно делать как в блоке WHERE, так и в блоке HAVING. Например, результат следующих запросов будет одинаковый:

```

SELECT sex, COUNT(user_id)
FROM users
WHERE sex != 'male'
GROUP BY sex

```

```

SELECT sex, COUNT(user_id)
FROM users

```

```
GROUP BY sex  
HAVING sex != 'male'
```

Однако делать фильтрацию по неагрегированным данным рекомендуется именно в блоке WHERE, т.е. заранее. В таком случае ненужные данные убираются из расчётов ещё до группировки и не расходуются вычислительные ресурсы на подсчёт значений, которые всё равно будут отфильтрованы вами позже.

## Вопросы на собеседовании

- **Как посчитать количество сотрудников, работающих в конкретном департаменте?**

Мы считаем `COUNT(1)` из таблицы, фильтруя с помощью `WHERE` департамент по названию или ID.

- **Как одним запросом посчитать общее количество строк, количество мужчин и количество женщин в таблице?**

Используя `COUNT(1)`, а также подменяя признак гендера мужчин единицей в одном поле и признак гендера женщин единицей в другом поле, а потом суммируя эти строки.

- **Как посмотреть дубли с помощью группировки?**

Мы можем сгруппировать данные по нужным нам полям, посчитав количество строк по группам, затем отфильтровав в `HAVING` группы, содержащие более одной строки, то есть дубликаты.

- **Чем отличается `WHERE` от `HAVING`?**

- `WHERE` фильтрует исходные строки до их группировки
- `HAVING` фильтрует агрегированные данные после выполнения группировки

Последовательность выполнения: сначала `WHERE`, затем `GROUP BY`, и в конце `HAVING`.

## Резюме

- Для вычислений над набором значений используются **агрегатные функции** ( `SUM` , `MIN` , `MAX` , `AVG` , `COUNT` ). С помощью функции `COUNT` можно подсчитать количество строк в таблице, количество непустых значений в поле, а также количество уникальных непустых значений поля.
- Для применения агрегатных функций к группам строк, а не ко всему полю, используется инструкция `GROUP BY` . Она позволяет задать поля группировки, объединив строки, имеющие одинаковые значения в группы, и выполнять агрегирующие функции в рамках каждой группы.
- Для фильтрации результата вычисления агрегатных функций после группировки используется инструкция `HAVING` . При этом сначала выполняется фильтрация в `WHERE` , затем `GROUP BY` , а в конце фильтрация в `HAVING` .

**К концу текущего урока вы можете написать SQL-запрос с использованием инструкций:**

SELECT	Основной оператор для запроса данных из одной или нескольких таблиц. Позволяет выбрать конкретные столбцы и строки на основе заданных условий.
DISTINCT	Используется вместе с SELECT для удаления дублирующихся строк из результата запроса
FROM	Указывает на таблицу (таблицы) в базе данных, к которой обращается запрос
JOIN	Соединяет строки из двух или более таблиц на основе связанного столбца между ними. Различные типы соединений включают: INNER JOIN, LEFT JOIN (или LEFT OUTER JOIN), RIGHT JOIN (или RIGHT OUTER JOIN), FULL JOIN (или FULL OUTER JOIN), CROSS JOIN
WHERE	Указывает условия, по которым строки будут выбраны
GROUP BY	Группирует строки с одинаковыми значениями в указанных столбцах и позволяет выполнять агрегатные функции на каждой группе
HAVING	Используется для фильтрации групп, созданных с помощью GROUP BY, на основе заданных условий

<b>ORDER BY</b>	Упорядочивает строки в результате запроса по одному или нескольким столбцам. Может быть использован для сортировки по возрастанию (ASC) или убыванию (DESC)
<b>LIMIT и OFFSET</b>	LIMIT ограничивает количество возвращаемых строк. OFFSET пропускает указанное количество строк перед возвратом оставшихся строк
<b>UNION и UNION ALL</b>	Объединяют результаты двух или более SELECT запросов. UNION удаляет дубликаты, а UNION ALL сохраняет все строки, включая дубликаты