# 2. Запросы SELECT в однотабличную базу данных

## Операторы HAVING, GROUP BY, LIKE

### Оператор GROUP BY

`GROUP BY` используется для группировки строк, имеющих одинаковые значения в указанных столбцах, в агрегированные данные. Когда данные группируются, агрегатные функции могут быть применены к каждой группе.

Пример:

```
SELECT align, COUNT(*) AS count
FROM MarvelCharacters
GROUP BY align;
```

Этот запрос сгруппирует всех персонажей по их мировоззрению (`align`) и посчитает количество персонажей в каждой группе.

### Оператор HAVING

`HAVING` используется для фильтрации записей после применения оператора `GROUP BY`. В отличие от `WHERE`, который фильтрует строки до группировки, `HAVING` фильтрует группы после того, как они были созданы.

Пример:

```
SELECT align, COUNT(*) AS count
FROM MarvelCharacters
GROUP BY align
HAVING COUNT(*) > 3000;
```

Этот запрос сгруппирует всех персонажей по их мировоззрению, посчитает количество персонажей в каждой группе и покажет только те группы, в которых больше 3000 персонажей.

### Оператор LIKE

`LIKE` используется в сочетании с `WHERE` для поиска строк, соответствующих определенному шаблону. Символы подстановки `%` и `_` используются для обозначения любого количества символов или одного символа соответственно.

Пример:

```
SELECT name, align, appearances
FROM MarvelCharacters
WHERE name LIKE "Spider%";
```

Этот запрос найдет всех персонажей, имена которых начинаются с "Spider".

## Агрегационные функции

### Функция AVG

`AVG` вычисляет среднее значение для набора значений.

Пример:

```
SELECT align, AVG(appearances) AS average_appearances
FROM MarvelCharacters
GROUP BY align;
```

Этот запрос покажет среднее количество появлений персонажей для каждого типа мировоззрения.

### Функция MAX

`MAX` возвращает максимальное значение из набора значений.

Пример:

```
SELECT align, MAX(appearances) AS max_appearances
FROM MarvelCharacters
GROUP BY align;
```

Этот запрос покажет максимальное количество появлений персонажей для каждого типа мировоззрения.

### Функция MIN

`MIN` возвращает минимальное значение из набора значений.

Пример:

```
SELECT align, MIN(appearances) AS min_appearances
FROM MarvelCharacters
GROUP BY align;
```

Этот запрос покажет минимальное количество появлений персонажей для каждого типа мировоззрения.

### Функция COUNT

`COUNT` возвращает количество строк в наборе данных.

Пример:

```
SELECT align, COUNT(*) AS count
FROM MarvelCharacters
GROUP BY align;
```

Этот запрос покажет количество персонажей для каждого типа мировоззрения.

### Функция SUM

`SUM` возвращает сумму значений из набора значений.

Пример:

```
SELECT align, SUM(appearances) AS total_appearances
FROM MarvelCharacters
GROUP BY align;
```

Этот запрос покажет общее количество появлений персонажей для каждого типа мировоззрения.

## Порядок выполнения операций в SQL

1. **FROM**: Первая операция - выбор базовой таблицы или таблиц.
2. **WHERE**: Фильтрация строк на основании условий до группировки.
3. **GROUP BY**: Группировка строк, основанная на одном или нескольких столбцах.
4. **HAVING**: Фильтрация групп, основанная на условиях после группировки.
5. **SELECT**: Выборка столбцов, включая агрегатные функции, применяется после выполнения всех вышеперечисленных операций.
6. **ORDER BY**: Сортировка результирующих строк (опционально).

Пример более сложного запроса, который включает все вышеперечисленные операторы и функции:

```sql
SELECT
    align,
    AVG(appearances) AS average_appearances,
    MAX(appearances) AS max_appearances
FROM
    MarvelCharacters
WHERE
    sex = "Female Characters"
GROUP BY
    align
HAVING
    AVG(appearances) > 5
ORDER BY
    average_appearances DESC;
```

Этот запрос выполняет следующие действия:

1. Выбирает строки из таблицы `MarvelCharacters`, где `sex` равно 'Female Characters'.
2. Группирует эти строки по столбцу `align`.
3. Применяет агрегатные функции `AVG` и `MAX` к каждой группе.
4. Фильтрует группы, где среднее количество появлений больше 5.
5. Сортирует результаты по убыванию среднего количества появлений.

## Сводная таблица встроенных функций SQLite

Создадим более подробную таблицу встроенных функций SQLite, включая описание использования, примеры синтаксиса и примеры того, как данные могут измениться до и после применения функции:
Первые пять используются с GROUP BY

| Функция | Описание и использование | Пример синтаксиса | Было | Стало |
|---|---|---|---|---|
| `COUNT()` | Подсчет количества строк или уникальных значений | `SELECT COUNT(*) FROM table;` | 5 строк в таблице | 5 |
| `SUM()` | Суммирование числовых значений в столбце | `SELECT SUM(price) FROM sales;` | 10, 20, 30 | 60 |
| `AVG()` | Вычисление среднего значения | `SELECT AVG(price) FROM sales;` | 10, 20, 30 | 20 |
| `MAX()` | Поиск максимального значения | `SELECT MAX(age) FROM users;` | 25, 35, 45 | 45 |
| `MIN()` | Поиск минимального значения | `SELECT MIN(age) FROM users;` | 25, 35, 45 | 25 |
| `GROUP_CONCAT()` | Объединение строковых значений | `SELECT GROUP_CONCAT(name) FROM users;` | Анна, Борис | Анна,Борис |
| `UPPER()` | Преобразование строки в верхний регистр | `SELECT UPPER('abc');` | abc | ABC |
| `LOWER()` | Преобразование строки в нижний регистр | `SELECT LOWER('ABC');` | ABC | abc |
| `LENGTH()` | Получение длины строки | `SELECT LENGTH('Hello');` | Hello | 5 |
| `SUBSTR()` | Извлечение подстроки | `SELECT SUBSTR('Hello', 1, 2);` | Hello | He |
| `TRIM()` | Удаление пробелов | `SELECT TRIM(' Hello ');` | Hello | Hello |
| `ROUND()` | Округление числа | `SELECT ROUND(123.456, 2);` | 123.456 | 123.46 |
| `RANDOM()` | Генерация случайного числа | `SELECT RANDOM();` | - | случайное число |
| `DATETIME()` | Получение текущей даты и времени | `SELECT DATETIME('now');` | - | 2021-01-01 10:00:00 |
| `DATE()` | Получение текущей даты | `SELECT DATE('now');` | - | 2021-01-01 |

| Функция | Описание и использование | Пример синтаксиса | Было | Стало |
|---|---|---|---|---|
| `TIME()` | Получение текущего времени | `SELECT TIME('now');` | - | 10:00:00 |
| `JULIANDAY()` | Дни с начала юлианского периода | `SELECT JULIANDAY('2021-01-01');` | - | 2459216.5 |
| `STRFTIME()` | Форматирование даты/времени | `SELECT STRFTIME('%Y-%m-%d', 'now');` | - | 2021-01-01 |
| `ABS()` | Абсолютное значение числа | `SELECT ABS(-5);` | -5 | 5 |
| `COALESCE()` | Возвращает первое ненулевое значение | `SELECT COALESCE(NULL, NULL, 5);` | NULL, NULL, 5 | 5 |
| `NULLIF()` | Возвращает NULL, если значения равны | `SELECT NULLIF(5, 5);` | 5 | NULL |
| `INSTR()` | Поиск подстроки | `SELECT INSTR('Hello', 'e');` | Hello | 2 |
| `REPLACE()` | Замена подстроки | `SELECT REPLACE('Hello', 'e', 'a');` | Hello | Hallo |
| `LAST_INSERT_ROWID()` | ID последней вставленной строки | `INSERT INTO users (name) VALUES ('Анна'); SELECT LAST_INSERT_ROWID();` | - | ID новой записи |
| `TYPEOF()` | Тип данных аргумента | `SELECT TYPEOF('abc');` | abc | text |

Эта таблица охватывает многие популярные функции SQLite, их типичное использование, примеры синтаксиса, а также примеры того, как данные могут измениться до и после применения функции. Она может служить полезным ресурсом для понимания и применения этих функций в SQL-запросах.

Агрегирующие функции в SQLite, как и в других системах управления базами данных (СУБД), используются для выполнения вычислений на наборе значений и возвращают одно суммарное значение. Эти функции имеют особое значение при работе с большими объемами данных, так как позволяют проводить статистический анализ и получать обобщенную информацию по заданным критериям.

## Важные Моменты

- **Одновременное использование агрегирующих функций и обычных столбцов:** При использовании `GROUP BY`, любой столбец, который не включен в агрегирующую функцию, должен быть указан в `GROUP BY`. Например, `SELECT category_id, COUNT(*) FROM sales GROUP BY category_id;`.
- **Фильтрация групп с помощью HAVING:** Если необходимо отфильтровать группы после агрегирования, используйте `HAVING`. Например, `SELECT category_id, SUM(sales) FROM sales GROUP BY category_id HAVING SUM(sales) > 1000;`.
- **Порядок выполнения:** Важно помнить, что сначала выполняется `GROUP BY`, затем агрегирующие функции, и после этого `HAVING`.

## Заключение

Агрегирующие функции в сочетании с группировкой в SQLite позволяют эффективно анализировать и обрабатывать большие объемы данных, получая суммарную, среднюю, максимальную и минимальную информацию по группам данных. Это особенно полезно для создания сводных отчетов, анализа данных и принятия на их основе информированных решений.