SQL Cheatsheet

Understanding data with SQL

Clauses

Clauses are distinct parts of an SQL statement. Put each on its own line and capitalize as below to increase legibility. Here are the five you will find most useful for understanding data:

`SELECT`	List the columns you want to show. `*` selects all columns.
`FROM` *`table`*	Specify the table you want. For example, `FROM citibike`
`WHERE` *`conditions`*	Place conditions on the rows that will be shown. Combine conditions with `AND` and `OR` (for example, `bikes >= 5 AND bikes < 10` would choose rows where `bikes` is greater than or equal to five and less than ten). Negate them with `NOT` (for example `NOT bikes = 3`) would choose rows where `bikes` is not three.
`GROUP BY` *`column`*	Group the output by the column. See **Count in groups** below for an example.
`ORDER BY` *`column`*	Order the output by the column. Add `ASC` to order ascending (lowest to highest value), `DESC` to order descending (highest to lowest value). For example, `ORDER BY bikes DESC` would order rows by `bikes`, highest to lowest.

Common Queries

View all

This is the default statement in applications such as CartoDB.

```
SELECT *
FROM table
```

For example:

```
SELECT *
FROM citibike
```

Filter

*Only show the rows that match the given conditions. See **Clauses**, above, for more details on conditions.*

```
SELECT *
FROM table
WHERE conditions
```

For example:

```
SELECT *
FROM citibike
WHERE bikes >= 5
```

Count

`SELECT` can do more than pick columns. It can also aggregate columns. Get the number of rows in a table:

```
SELECT COUNT(*)
FROM table
```

For example:

```
SELECT COUNT(*)
FROM citibike
```

Count and filter

Add a `WHERE` clause to the above to count only the rows you are interested in.

```
SELECT COUNT(*)
FROM table
WHERE conditions
```

For example:

```
SELECT COUNT(*)
FROM citibike
WHERE bikes >= 5
```

Count in groups

Group rows by their value in a column, then count the number of rows in each group. Handy for answering questions like "How many stations are there in each borough?"

```
SELECT column, COUNT(*)
FROM table
GROUP BY column
```

For example:

```
SELECT borough, COUNT(*)
FROM citibike
GROUP BY borough
```

Count in groups and filter

Group filtered rows by their value in a column, then count the number of rows in each group. "How many stations are there in each borough that meet my criteria?"

```
SELECT column, COUNT(*)
FROM table
WHERE conditions
GROUP BY column
```

For example:

```
SELECT borough, COUNT(*)
FROM citibike
WHERE bikes > 1
GROUP BY borough
```

Find unique values in a column

`SELECT` has more tricks up its sleeve. Here it is used to quickly give us all of the unique values in a column by using `DISTINCT`. This is handy for understanding a column in a database that is new to you. "What's in here?"

```
SELECT DISTINCT(column)
FROM table
```

For example:

```
SELECT DISTINCT(borough)
FROM citibike
```

Find the range of a column

More `SELECT` fun. Get the range of values in a column with `MIN` and `MAX`. As above, this is useful for understanding a column in a database that is new to you.

```
SELECT MIN(column),
MAX(column)
FROM table
```

For example:

```
SELECT MIN(bikes), MAX(bikes)
FROM citibike
```

Find unique values in a column, filter

As with most statements, you can add a `WHERE` clause after the `FROM` clause to restrict the rows that you are querying. Here you can get the unique values in a column while only looking at certain rows.

```
SELECT DISTINCT(column)
FROM table
WHERE conditions
```

For example:

```
SELECT DISTINCT(borough)
FROM citibike
WHERE bikes > 10
```

Ordering rows

Add an `ORDER BY` clause after a `FROM` clause (or a `WHERE` clause, if you are filtering) to sort rows.

```
SELECT *
FROM table
WHERE conditions
ORDER BY column
```

For example:

```
SELECT DISTINCT(borough)
FROM citibike
WHERE bikes > 10
ORDER BY bikes
```

Condition operators

WHERE clause conditions can contain the following operators:

`>`	Greater than. Eg, `bikes > 10` selects rows with `bikes` over 10.
`<`	Less than. Eg, `bikes < 10` selects rows with `bikes` under 10.
`=`	Equal. Eg, `bikes = 10` selects rows with `bikes` equal to 10.
`!=`	Not equal. Eg, `bikes != 10` selects rows with `bikes` not equal to 10.
`>=`	Greater or equal. `bikes >= 10` selects rows with `bikes` greater or equal to 10.
`<=`	Less than or equal. `bikes <= 10` selects rows with `bikes` less than or equal to 10.
`IN`	In. Eg, `bikes in (8, 9, 10)`, selecting rows with `bikes` equal to 8, 9, or 10.
`IS NULL`	Is null. Eg, `bikes IS NULL`, selecting rows where `bikes` is not set.
`IS NOT NULL`	Is not null. Eg, `bikes IS NOT NULL`, selecting rows where `bikes` is set.

Working with strings

If you are working with a column that is a string, your conditions sometimes have to take that into account. For example, when using =, you have to put the string you are comparing the column to in quotes:

```
SELECT *
FROM citibike
WHERE borough = 'Brooklyn'
```

More likely you will be searching within a column, which you can do using the `ILIKE` operator:

```
SELECT *
FROM citibike
WHERE address ILIKE '%fulton%'
```

The above searches the address column for any values containing “`fulton`”. The `%` means “match anything,” so this translates to “fulton preceded by anything and followed by anything else.”