SQL
CHEAT SHEET

created by
Tomi Mester

BASE QUERY

SELECT * FROM table_name;

This query returns every column and every row of the table called table_name.

SELECT * FROM table_name LIMIT 10;

It returns every column and the first 10 rows from table_name.

SELECTING SPECIFIC COLUMNS

SELECT column1, column2, column3 FROM table_name;

This query returns every row of column1, column2 and column3 from table_name.

[your notes]

DATA TYPES IN SQL

In SQL we have more than 40 different data types. But these seven are the most important ones:

1. **Integer.** A whole number without a fractional part. E.g. 1, 156, 2012412
2. **Decimal.** A number with a fractional part. E.g. 3.14, 3.141592654, 961.1241250
3. **Boolean.** A binary value. It can be either TRUE or FALSE.
4. **Date.** Speaks for itself. You can also choose the format. E.g. 2017-12-31
5. **Time.** You can decide the format of this, as well. E.g. 23:59:59
6. **Timestamp.** The date and the time together. E.g. 2017-12-31 23:59:59
7. **Text.** This is the most general data type. But it can be alphabetical letters only, or a mix of letters and numbers and any other characters. E.g. hello, R2D2, Tomi, 124.56.128.41

FILTERING (the WHERE CLAUSE)

SELECT * FROM table_name WHERE column1 = 'expression';
"Horizontal filtering." This query returns every column from table_name - but only those rows where the value in column1 is 'expression'. Obviously this can be something other than text: a number (integer or decimal), date or any other data format, too.

ADVANCED FILTERING

Comparison operators help you compare two values. (Usually a value that you define in your query and values that exist in your SQL table.) Mostly, they are mathematical symbols, with a few exceptions:

Comparison operator	What does it mean?
=	Equal to
<>	Not equal to
!=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE '%expression%'	Contains 'expression'
IN ('exp1', 'exp2', 'exp3')	Contains any of 'exp1', 'exp2', or 'exp3'

A few examples:

SELECT * FROM table_name WHERE column1 != 'expression';

This query returns every column from table_name, but only those rows where the value in column1 is NOT 'expression'.

SELECT * FROM table_name WHERE column2 >= 10;

It returns every column from table_name, but only those rows where the value in column2 is greater or equal to 10.

SELECT * FROM table_name WHERE column3 LIKE '%xzy%';

It returns every column from table_name, but only those rows where the value in column3 contains the 'xyz' string.

MULTIPLE CONDITIONS

You can use more than one condition to filter. For that, we have two logical operators: OR, AND.

SELECT * FROM table_name WHERE column1 != 'expression' AND column3 LIKE '%xzy%';

This query returns every column from table_name, but only those rows where the value in column1 is NOT 'expression' AND the value in column3 contains the 'xyz' string.

SELECT * FROM table_name WHERE column1 != 'expression' OR column3 LIKE '%xzy%';

This query returns every column from table_name, but only those rows where the value in column1 is NOT 'expression' OR the value in column3 contains the 'xyz' string.

PROPER FORMATTING

You can use line breaks and indentations for nicer formatting. It won't have any effect on your output. Be careful and put a semicolon at the end of the query though!

```
SELECT *
FROM table_name
WHERE column1 != 'expression'
    AND column3 LIKE '%xzy%'
LIMIT 10;
```

SORTING VALUES

SELECT * FROM table_name ORDER BY column1;
This query returns every row and column from table_name, ordered by column1, in ascending order (by default).

SELECT * FROM table_name ORDER BY column1 DESC;
This query returns every row and column from table_name, ordered by column1, in descending order.

UNIQUE VALUES

SELECT DISTINCT(column1) FROM table_name;
It returns every unique value from column1 from table_name.

CORRECT KEYWORD ORDER

SQL is extremely sensitive to keyword order.
So make sure you keep it right:

1. **SELECT**
2. **FROM**
3. **WHERE**
4. **ORDER BY**
5. **LIMIT**

SQL FUNCTIONS FOR AGGREGATION

In SQL, there are five important aggregate functions for data analysts/scientists:

- **COUNT()**
- **SUM()**
- **AVG()**
- **MIN()**
- **MAX()**

A few examples:
SELECT COUNT(*) FROM table_name WHERE column1 = 'something';
It counts the number of rows in the SQL table in which the value in column1 is 'something'.

SELECT AVG(column1) FROM table_name WHERE column2 > 1000;
It calculates the average (mean) of the values in column1, only including rows in which the value in column2 is greater than 1000.

SQL GROUP BY

The GROUP BY clause is usually used with an aggregate function (COUNT, SUM, AVG, MIN, MAX). It groups the rows by a given column value (specified after GROUP BY) then calculates the aggregate for each group and returns that to the screen.

SELECT column1, COUNT(column2) FROM table_name GROUP BY column1;
This query counts the number of values in column2 - for each group of unique column1 values.

SELECT column1, SUM(column2) FROM table_name GROUP BY column1;
This query sums the number of values in column2 - for each group of unique column1 values.

SELECT column1, MIN(column2) FROM table_name GROUP BY column1;
This query finds the minimum value in column2 - for each group of unique column1 values.

SELECT column1, MAX(column2) FROM table_name GROUP BY column1;
This query finds the maximum value in column2 - for each group of unique column1 values.

SQL ALIASES

You can rename columns, tables, subqueries, anything.

SELECT column1, COUNT(column2) AS number_of_values FROM table_name GROUP BY column1;
This query counts the number of values in column2 - for each group of unique column1 values. Then it renames the COUNT(column2) column to number_of_values.

SQL JOIN

You can JOIN two (or more) SQL tables based on column values.

```
SELECT *
FROM table1
JOIN table2
ON table1.column1 = table2.column1;
```

This joins table1 and table2 values - for every row where the value of column1 from table1 equals the value of column1 from table2.

Detailed explanation here: https://data36.com/sql-join-data-analysis-tutorial-ep5/

SQL HAVING

The execution order of the different SQL keywords doesn't allow you to filter with the WHERE clause on the result of an aggregate function (COUNT, SUM, etc.). This is because WHERE is executed before the aggregate functions. But that's what HAVING is for:

```
SELECT column1, COUNT(column2)
FROM table_name
GROUP BY column1
HAVING COUNT(column2) > 100;
```

This query counts the number of values in column2 - for each group of unique column1 values. It returns only those results where the counted value is greater than 100.

Detailed explanation and examples here: https://data36.com/sql-data-analysis-advanced-tutorial-ep6/

CORRECT KEYWORD ORDER AGAIN

SQL is extremely sensitive to keyword order.
So make sure you keep it right:

1. SELECT
2. FROM
3. JOIN (ON)
4. WHERE
5. GROUP BY
6. HAVING
7. ORDER BY
8. LIMIT

SUBQUERIES

You can run SQL queries within SQL queries. (Called subqueries.) Even queries within queries within queries. The point is to use the result of one query as an input value of another query.
Example:

```
SELECT COUNT(*) FROM
  (SELECT column1, COUNT(column2) AS inner_number_of_values
  FROM table_name
  GROUP BY column1) AS inner_query
WHERE inner_number_of_values > 100;
```

The inner query counts the number of values in column2 - for each group of unique column1 values. Then the outer query uses the inner query's results and counts the number of values where inner_number_of_values are greater than 100. (The result is one number.)

Detailed explanation here: https://data36.com/sql-data-analysis-advanced-tutorial-ep6/