SQL Basics Cheat Sheet

**SQL**

SQL, or Structured Query Language, is a language to talk to databases. It allows you to select specific data and to build complex reports. Today, SQL is a universal language of data. It is used in practically all technologies that process data.

**SAMPLE DATA**

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66690008</td>
<td>646680</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>86700808</td>
<td>357808</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**JOIN**

JOIN country ON ci.country_id = co.id;

SELECT co.name, ci.name
FROM city;

**SELECT**

SELECT name
FROM city;

**ALTER OPERATORS**

Fetch names of cities that have a population between 500K and 5M:

SELECT name
FROM city
WHERE population BETWEEN 500000 AND 5000000;

**FILTERING THE OUTPUT**

**JOIN**

INNER JOIN country ON city.country_id = country.id;

SELECT city.name, country.name
FROM city
INNER JOIN country
ON city.country_id = country.id;

**LEFT JOIN**

LEFT JOIN country ON city.country_id = country.id;

SELECT city.name, country.name
FROM city
LEFT JOIN country
ON city.country_id = country.id;

**CROSS JOIN**

CROSS JOIN country;

SELECT city.name, country.name
FROM city
CROSS JOIN country;

**RIGHT JOIN**

RIGHT JOIN country ON city.country_id = country.id;

SELECT city.name, country.name
FROM city
RIGHT JOIN country
ON city.country_id = country.id;

**FULL JOIN**

FULL JOIN country ON city.country_id = country.id;

SELECT city.name, country.name
FROM city
FULL JOIN country
ON city.country_id = country.id;

**FULL OUTER JOIN**

FULL OUTER JOIN country ON city.country_id = country.id;

**QUREYING MULTIPLE TABLES**

**TEXT OPERATORS**

Fetch names of cities that start with a 'P' or end with an 's':

SELECT name
FROM city
WHERE name LIKE 'P%'
OR name LIKE '%s';

**OTHER OPERATORS**

Fetch names of cities that have a rating above 3:

SELECT name
FROM city
WHERE rating > 3;

**COMPARISON OPERATORS**

Fetch names of cities that have a rating above 3:

SELECT name
FROM city
WHERE rating IS NOT NULL;

SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);

**OTHER OPERATORS**

Fetch names of cities that don't miss a rating value:

SELECT name
FROM city
WHERE rating IS NOT NULL;

Fetch names of cities that are in countries with IDs 1, 4, 7, or 8:

SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);

**SAMPLE DATA**

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66690008</td>
<td>646680</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>86700808</td>
<td>357808</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**SAMPLE DATA**

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66690008</td>
<td>646680</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>86700808</td>
<td>357808</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**SAMPLE DATA**

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66690008</td>
<td>646680</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>86700808</td>
<td>357808</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**SAMPLE DATA**

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66690008</td>
<td>646680</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>86700808</td>
<td>357808</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**SAMPLE DATA**

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66690008</td>
<td>646680</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>86700808</td>
<td>357808</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SQL Basics Cheat Sheet

**AGGREGATION AND GROUPING**

GROUP BY groups together rows that have the same values in specified columns. It computes summaries (aggregates) for each unique combination of values.

<table>
<thead>
<tr>
<th>CITY</th>
<th>id</th>
<th>name</th>
<th>country_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A1</td>
<td>Paris</td>
<td>1</td>
</tr>
<tr>
<td>101</td>
<td>A2</td>
<td>Marseille</td>
<td>1</td>
</tr>
<tr>
<td>102</td>
<td>A3</td>
<td>Lyon</td>
<td>1</td>
</tr>
<tr>
<td>103</td>
<td>A4</td>
<td>Berlin</td>
<td>2</td>
</tr>
<tr>
<td>104</td>
<td>A5</td>
<td>Munich</td>
<td>2</td>
</tr>
<tr>
<td>105</td>
<td>A6</td>
<td>Cracow</td>
<td>4</td>
</tr>
</tbody>
</table>

**AGGREGATE FUNCTIONS**

- `avg(expr)` - average value for rows within the group
- `count(expr)` - count of values for rows within the group
- `max(expr)` - maximum value within the group
- `min(expr)` - minimum value within the group
- `sum(expr)` - sum of values within the group

**EXAMPLE QUERIES**

Find out the number of cities:

```
SELECT COUNT(*)
FROM city;
```

Find out the number of cities with non-null ratings:

```
SELECT COUNT(rating)
FROM city;
```

Find out the number of distinctive country values:

```
SELECT COUNT(DISTINCT country_id)
FROM city;
```

Find out the smallest and the greatest country populations:

```
SELECT MIN(population), MAX(population)
FROM country;
```

Find out the total population of cities in respective countries:

```
SELECT country_id, SUM(population)
FROM city
GROUP BY country_id;
```

Find out the average rating for cities in respective countries if the average is above 3.0:

```
SELECT country_id, AVG(rating)
FROM city
GROUP BY country_id
HAVING AVG(rating) > 3.0;
```

**SUBQUERIES**

A subquery is a query that is nested inside another query, or inside another subquery. There are different types of subqueries.

**SINGLE VALUE**

The simplest subquery returns exactly one column and exactly one row. It can be used with comparison operators `=`, `<`, `>`, or `>=`.

This query finds cities with the same rating as Paris:

```
SELECT name FROM city
WHERE rating = (SELECT rating
FROM city
WHERE name = 'Paris');
```

**MULTIPLE VALUES**

A subquery can also return multiple columns or multiple rows. Such subqueries can be used with operators IN, EXISTS, ALL, or ANY.

This query finds cities in countries that have a population above 20M:

```
SELECT country_id
FROM country
WHERE country_id IN (
    SELECT country_id
    FROM country
    WHERE population > 20000000
);
```

**CORRELATED**

A correlated subquery refers to the tables introduced in the outer query. A correlated subquery depends on the outer query. It cannot be run independently from the outer query.

This query finds cities with a population greater than the average population in the country:

```
SELECT * FROM city main_city
WHERE population > (SELECT AVG(population)
FROM city average_city
WHERE average_city.country_id = main_city.country_id);
```

This query finds countries that have at least one city:

```
SELECT name
FROM country
WHERE EXISTS (SELECT * FROM city
WHERE country_id = country.id);
```

**SET OPERATIONS**

Set operations are used to combine the results of two or more queries into a single result. The combined queries must return the same number of columns and compatible data types. The names of the corresponding columns can be different.

**UNION**

UNION combines the results of two result sets and removes duplicates. UNION ALL doesn’t remove duplicate rows.

This query displays German cyclists together with German skaters:

```
SELECT name
FROM cycling
WHERE country = 'DE'
UNION
SELECT name
FROM skating
WHERE country = 'DE';
```

**INTERSECT**

INTERSECT returns only rows that appear in both result sets.

This query displays German cyclists who are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
INTERSECT
SELECT name
FROM skating
WHERE country = 'DE';
```

**EXCEPT**

EXCEPT returns only the rows that appear in the first result set but do not appear in the second result set.

This query displays German cyclists unless they are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
EXCEPT
SELECT name
FROM skating
WHERE country = 'DE';
```

---

Try out the interactive [SQL Basics](https://www.learnsql.com) course at LearnSQL.com, and check out our other SQL courses.