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>France</td>
<td>1</td>
<td>Paris</td>
<td>2240300</td>
<td>6.77</td>
</tr>
<tr>
<td>Germany</td>
<td>2</td>
<td>Berlin</td>
<td>2460000</td>
<td>35700</td>
</tr>
<tr>
<td>Iceland</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

**COLUMNS**

```
<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Paris</td>
<td>2240300</td>
<td>6.77</td>
</tr>
<tr>
<td>Germany</td>
<td>Berlin</td>
<td>2460000</td>
<td>35700</td>
</tr>
<tr>
<td>Iceland</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

**ALIASES**

```
SELECT name AS city_name
FROM city;
```

**TABLES**

```
SELECT co.name, ci.name
FROM city AS co
JOIN country AS co
ON co.country_id = co.id;
```

**TEXT OPERATORS**

- **Filtering the output**

  **Comparison operators**

  Fetch names of cities that have a rating above 3:

  ```sql
  SELECT name
  FROM city
  WHERE rating > 3;
  ```

  Fetch names of cities that are neither Berlin nor Madrid:

  ```sql
  SELECT name
  FROM city
  WHERE name != 'Berlin'
  AND name != 'Madrid';
  ```

- **Other operators**

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

  ```sql
  SELECT name
  FROM city
  WHERE population BETWEEN 500000 AND 5000000;
  ```

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

  ```sql
  SELECT name
  FROM city
  WHERE rating IS NOT NULL;
  ```

- **JOIN**

  **Inner join**

  ```sql
  JOIN (or explicitly INNER JOIN) returns rows that have matching values in both tables.
  ```

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

  **Left join**

  ```sql
  LEFT JOIN (or explicitly LEFT JOIN) returns all rows from the left table with corresponding rows from the right table. If there's no matching row, NULLs are returned as values from the second table.
  ```

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

  **Right join**

  ```sql
  RIGHT JOIN (or explicitly RIGHT JOIN) returns all rows from the right table with corresponding rows from the left table. If there's no matching row, NULLs are returned as values from the left table.
  ```

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

  **Full join**

  ```sql
  FULL JOIN (or explicitly FULL OUTER JOIN) returns all rows from both tables – if there’s no matching row in the second table, NULLs are returned.
  ```

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

  **Natural join**

  ```sql
  NATURAL JOIN will join tables by all columns with the same name.
  ```

  ```sql
  SELECT city.name, country.name
  FROM city
  NATURAL JOIN country;
  ```

**Filtering the Output**

- **Comparison operators**

  Fetch names of cities that have a rating above 3:

  ```sql
  SELECT name
  FROM city
  WHERE rating > 3;
  ```

- **Other operators**

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

  ```sql
  SELECT name
  FROM city
  WHERE population BETWEEN 500000 AND 5000000;
  ```

**Text operators**

- **Filtering the output**

  - **Comparison operators**

    Fetch names of cities that have a rating above 3:

    ```sql
    SELECT name
    FROM city
    WHERE rating > 3;
    ```

    Fetch names of cities that are neither Berlin nor Madrid:

    ```sql
    SELECT name
    FROM city
    WHERE name != 'Berlin'
    AND name != 'Madrid';
    ```

- **Other operators**

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

  ```sql
  SELECT name
  FROM city
  WHERE population BETWEEN 500000 AND 5000000;
  ```

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

  ```sql
  SELECT name
  FROM city
  WHERE rating IS NOT NULL;
  ```

**Join**

- **Inner join**

  ```sql
  JOIN (or explicitly INNER JOIN) returns rows that have matching values in both tables.
  ```

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

- **Left join**

  ```sql
  LEFT JOIN (or explicitly LEFT JOIN) returns all rows from the left table with corresponding rows from the right table. If there's no matching row, NULLs are returned as values from the second table.
  ```

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

- **Right join**

  ```sql
  RIGHT JOIN (or explicitly RIGHT JOIN) returns all rows from the right table with corresponding rows from the left table. If there's no matching row, NULLs are returned as values from the left table.
  ```

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

- **Full join**

  ```sql
  FULL JOIN (or explicitly FULL OUTER JOIN) returns all rows from both tables – if there’s no matching row in the second table, NULLs are returned.
  ```

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

- **Natural join**

  ```sql
  NATURAL JOIN will join tables by all columns with the same name.
  ```

  ```sql
  SELECT city.name, country.name
  FROM city
  NATURAL JOIN country;
  ```

**Querying single table**

- Fetch all columns from the country table:

  ```sql
  SELECT *
  FROM country;
  ```

- Fetch id and name columns from the city table:

  ```sql
  SELECT id, name
  FROM city;
  ```

- Fetch city names sorted by the rating column in the default ASCending order:

  ```sql
  SELECT name
  FROM city
  ORDER BY rating [ASC];
  ```

- Fetch city names sorted by the rating column in the DESCending order:

  ```sql
  SELECT name
  FROM city
  ORDER BY rating DESC;
  ```

**Columns**

- Fetch city names that start with a 'P' or end with an 's':

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

- Fetch names of cities that start with any letter followed by 'ublin' [like Dublin in Ireland or Lublin in Poland]:

  ```sql
  SELECT name
  FROM city
  WHERE name LIKE '_ublin';
  ```

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

  ```sql
  SELECT name
  FROM city
  WHERE population BETWEEN 500000 AND 5000000;
  ```

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

  ```sql
  SELECT name
  FROM city
  WHERE rating IS NOT NULL;
  ```

**Tables**

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

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

**Full join**

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

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

**Natural join**

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

  ```sql
  SELECT name
  FROM city
  NATURAL JOIN country;
  ```

**Cross join**

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

  ```sql
  SELECT name
  FROM city
  CROSS JOIN country;
  ```

**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>France</td>
<td>1</td>
<td>Paris</td>
<td>2240300</td>
<td>6.77</td>
</tr>
<tr>
<td>Germany</td>
<td>2</td>
<td>Berlin</td>
<td>2460000</td>
<td>35700</td>
</tr>
<tr>
<td>Iceland</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>CITIES</th>
<th>name</th>
<th>country_id</th>
<th>population</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>1</td>
<td>1</td>
<td>2240300</td>
<td>5</td>
</tr>
<tr>
<td>Berlin</td>
<td>2</td>
<td>2</td>
<td>2460000</td>
<td>4</td>
</tr>
<tr>
<td>Warsaw</td>
<td>3</td>
<td>4</td>
<td>100000</td>
<td>3</td>
</tr>
<tr>
<td>Madrid</td>
<td>4</td>
<td>7</td>
<td>5000000</td>
<td>5</td>
</tr>
</tbody>
</table>
```
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.

GROUP BY country_id

HAVING

FROM city

GROUP BY country_id;

SELECT country_id,

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 name FROM city
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. It cannot be run independently from the outer query.

This query finds cities with the same rating as Paris:

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

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 course at LearnSQL.com, and check out our other SQL courses.