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.

CONTENTS

QUERYING SINGLE TABLE ................................................................. 3
COLUMN & TABLE ALIASES .................................................................. 3
FILTERING THE OUTPUT ........................................................................ 4
COMPARISON OPERATORS ................................................................... 4
TEXT OPERATORS ................................................................................ 4
OTHER OPERATORS .............................................................................. 5
QUERYING MULTIPLE TABLES .............................................................. 6
INNER JOIN ......................................................................................... 6
LEFT JOIN ............................................................................................ 6
RIGHT JOIN .......................................................................................... 7
FULL JOIN ............................................................................................. 7
CROSS JOIN ........................................................................................ 8
NATURAL JOIN ..................................................................................... 8
AGGREGATION AND GROUPING .......................................................... 9
SUBQUERIES ....................................................................................... 11
CORRELATED SUBQUERIES ............................................................... 12
SET OPERATIONS: UNION, INTERSECT, EXCEPT ................................. 13
### COUNTRY

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>population</th>
<th>area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>France</td>
<td>66600000</td>
<td>640680</td>
</tr>
<tr>
<td>2</td>
<td>Germany</td>
<td>80700000</td>
<td>357000</td>
</tr>
</tbody>
</table>

### CITY

<table>
<thead>
<tr>
<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>2243000</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td>3460000</td>
<td>3</td>
</tr>
</tbody>
</table>

This Cheat Sheet was prepared by LearnSQL.com as part of its SQL training program. Check out other SQL Cheat Sheets.

LearnSQL.com is owned by Vertabelo SA | vertabelo.com | CC BY-NC-ND Vertabelo SA
**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;
```

**ALIASES**

**COLUMNS**

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

**TABLES**

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

COMPARISON OPERATORS

Fetch names of cities that have a rating above 3:
```
SELECT name
FROM city
WHERE rating > 3;
```

Fetch names of cities that are neither Berlin nor Madrid:
```
SELECT name
FROM city
WHERE name != 'Berlin'
  AND name != 'Madrid';
```

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';
```

Fetch names of cities that start with any letter followed by 'ublin' (like Dublin in Ireland or Lublin in Poland):
```
SELECT name
FROM city
WHERE name LIKE '_ublin';
```
OTHER OPERATORS

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

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

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);
```
QUERYING MULTIPLE TABLES

**INNER JOIN**

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

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

<table>
<thead>
<tr>
<th>CITY</th>
<th>COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
</tr>
<tr>
<td>3</td>
<td>Warsaw</td>
</tr>
</tbody>
</table>

**LEFT JOIN**

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.

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

<table>
<thead>
<tr>
<th>CITY</th>
<th>COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
</tr>
<tr>
<td>3</td>
<td>Warsaw</td>
</tr>
</tbody>
</table>
RIGHT JOIN

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;
```

<table>
<thead>
<tr>
<th>CITY</th>
<th>COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

FULL JOIN

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 [OUTER] JOIN country
  ON city.country_id = country.id;
```

<table>
<thead>
<tr>
<th>CITY</th>
<th>COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
</tr>
<tr>
<td>3</td>
<td>Warsaw</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

This Cheat Sheet was prepared by LearnSQL.com as part of its SQL training program. Check out other SQL Cheat Sheets.

LearnSQL.com is owned by Vertabelo SA | vertabelo.com | CC BY-NC-ND Vertabelo SA
**CROSS JOIN**

CROSS JOIN returns all possible combinations of rows from both tables. There are two syntaxes available.

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

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

<table>
<thead>
<tr>
<th>CITY</th>
<th>COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
</tr>
</tbody>
</table>

**NATURAL JOIN**

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

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

<table>
<thead>
<tr>
<th>CITY</th>
<th>COUNTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>6</td>
<td>San Marino</td>
</tr>
<tr>
<td>7</td>
<td>Vatican City</td>
</tr>
<tr>
<td>5</td>
<td>Greece</td>
</tr>
<tr>
<td>10</td>
<td>Monaco</td>
</tr>
</tbody>
</table>

NATURAL JOIN used these columns to match rows: `city.id, city.name, country.id, country.name`.

NATURAL JOIN is very rarely used in practice.
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>country_id</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>id 1</td>
<td>Paris</td>
<td>1</td>
</tr>
<tr>
<td>101 Marseille</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>102 Lyon</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2 Berlin</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>103 Hamburg</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>104 Munich</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3 Warsaw</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>105 Cracow</td>
<td>4</td>
<td></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:
```sql
SELECT COUNT(*)
FROM city;
```

Find out the number of cities with non-null ratings:
```sql
SELECT COUNT(rating)
FROM city;
```

Find out the number of distinctive country values:
```sql
SELECT COUNT(DISTINCT country_id)
FROM city;
```

Find out the smallest and the greatest country populations:
```sql
SELECT MIN(population), MAX(population)
FROM country;
```

Find out the total population of cities in respective countries:
```sql
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:
```sql
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. 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.

<table>
<thead>
<tr>
<th>CYCLING</th>
<th>SKATING</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>YK</td>
</tr>
<tr>
<td>2</td>
<td>ZG</td>
</tr>
<tr>
<td>3</td>
<td>WT</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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 / UNION ALL
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:

```sql
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:

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