**SQL Basics Cheat Sheet**

**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';
  ```

**TEXT OPERATORS**

- Fetch names of cities 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';
  ```

**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;
  ```

- 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);
  ```

**QUERYING MULTIPLE TABLES**

**INNER JOIN**

- 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**

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

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

**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;
  ```

**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;
  ```

**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;
  ```

**ALIASES**

- SELECT name AS city_name
- FROM city;

**COLUMNS**

- SELECT name, country_id
- FROM city;

**TABLES**

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

**Try out the interactive [SQL Basics](http://LearnSQL.com) course at LearnSQL.com, and check out our other SQL courses.**
### 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>Paris</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Marseille</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>Lyon</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Berlin</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>Hamburg</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Munich</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Warsaw</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>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 `=, <, <=, >, >=.`

This query finds cities with the same rating as Paris:

```sql
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:

```sql
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:

```sql
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:

```sql
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>id</th>
<th>name</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YK</td>
<td>DE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ZG</td>
<td>DE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>WT</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SKATING</th>
<th>id</th>
<th>name</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>YK</td>
<td>DE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>DF</td>
<td>DE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>AK</td>
<td>PL</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<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:

```sql
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 / EXCEPT ALL
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.