SQL COMPLETE
CHEATSHEET

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

COUNTRY

id	name	population	area
1	France	66600000	640680
2	Germany	80700000	357030
...	...	...	...

CITY

id	name	country_id	population	rating
1	Paris	1	2243090	5
2	Berlin	2	3460000	3
...	...	...	...	...

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:

```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);
```

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.

CITY

id	name	country_id
1	Paris	1
101	Marseille	1
102	Lyon	1
2	Berlin	2
103	Hamburg	2
104	Munich	2
3	Warsaw	4
105	Cracow	4

$\rightarrow$

CITY

country_id	count
1	3
2	3
4	2

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;
```

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.
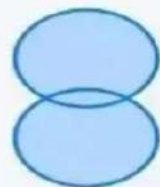
CYCLING

id	name	country
1	YK	DE
2	ZG	DE
3	WT	PL
...	...	...

SKATING

id	name	country
1	YK	DE
2	DF	DE
3	AK	PL
...	...	...

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:

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

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;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

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.

```
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

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;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

CROSS JOIN

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

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

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

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

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.

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

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

NATURAL JOIN

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

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

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

NATURAL JOIN used these columns to match rows:
city.id, city.name, country.id, country.name
NATURAL JOIN is very rarely used in practice.

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
);
```