SQL and Databases

Table of Contents

[1.Introduction to SQL 3](#_Toc65346202)

[1.1. Selecting Columns 3](#_Toc65346203)

[1.2. Filtering Results 4](#_Toc65346204)

[1.3. Aggregate Functions 5](#_Toc65346205)

[1.4. Sorting (ORDER BY) 6](#_Toc65346206)

[1.4.1. Sorting single column 6](#_Toc65346207)

[1.4.2. Sorting multiple columns 7](#_Toc65346208)

[1.5. Grouping (GROUP BY) 7](#_Toc65346209)

[2.Joining data in SQL 7](#_Toc65346210)

[2.1. Introduction to joins 7](#_Toc65346211)

[2.1.1. INNER JOIN 7](#_Toc65346212)

[2.1.2. SELF JOIN 7](#_Toc65346213)

[2.1.3. OUTER JOIN 7](#_Toc65346214)

[2.1.4. CROSS JOIN 7](#_Toc65346215)

[2.1.5. SEMI JOIN 7](#_Toc65346216)

[2.1.6. ANTI JOIN 7](#_Toc65346217)

[2.2. Set theory clauses 7](#_Toc65346218)

[I.UNION 7](#_Toc65346219)

[II.UNION ALL 7](#_Toc65346220)

[III.INTERSECT 7](#_Toc65346221)

[IV.EXCEPT 7](#_Toc65346222)

[V.SEMI JOINS 7](#_Toc65346223)

[2.3. Subqueries 7](#_Toc65346224)

[2.3.1. Subqueries inside **WHERE** clause 7](#_Toc65346225)

[2.3.2. Subqueries inside **SELECT** clause 7](#_Toc65346226)

[2.3.3. Subqueries inside **FROM** clause 7](#_Toc65346227)

# 1.Introduction to SQL

SQL, which stands for Structured Query Language, is a language for interacting with data stored in a relational database.

A relational database is a type of database that stores and provides access to data points that are related to one another. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

  A query is a request for data from a database table (or combination of tables).

## Selecting Columns

The ***SELECT***keyword allows to select data from database.

|  |  |
| --- | --- |
| *Basic Queries For Selecting Columns* | |
| Selecting ***Single*** column | SELECT **col**  FROM table name; |
| Selecting ***Multiple*** columns | SELECT **col1, col2**  FROM table name; |
| Selecting ***All***columns | SELECT **\***  FROM table name; |
| Selecting All columns with ***LIMIT***  (limit the number of rows returned) | SELECT **\***  FROM table name  **LIMIT** 10; |
| Select all the ***Unique*** values from a column | SELECT **DISTINCT** col  FROM table name; |
| ***Count*** number of rows in ***Single*** column | SELECT **COUNT(col)**  FROM table name; |
| ***Count*** number of rows in ***Multiple*** columns. | SELECT **COUNT(\*)**  FROM table name; |
| ***Count*** the number of **Distinct** values in a column | SELECT **COUNT(DISTINCT col)**  FROM table name; |

## Filtering Results

The ***WHERE***keyword allows to filter based on both text and numeric values in a table. here are a few different comparison operators you can use:

* = equal
* <> not equal
* < less than
* > greater than
* <= less than or equal to
* >= greater than or equal to

Note: The WHERE clause always comes after the FROM statement!

|  |  |
| --- | --- |
| *Basic Queries For Filtering rows* | |
| ***WHERE*** | |
| Simple filtering of numeric values - Details | SELECT \*  FROM table name  **WHERE** col >=< value; |
| Simple filtering of text - Details  Note: Remember to use ISO date format ('YYYY-MM-DD') | SELECT \*  FROM table name  **WHERE** col = ‘text’; |
| ***WHERE AND*** *- combining multiple conditions with the AND keyword.* | |
| Simple filtering of numeric values and text | SELECT \*  FROM table name  **WHERE** col1 = ‘text’  **AND** col2 >=< value or ‘text’; |
| ***WHERE AND OR*** *- combining multiple conditions with the AND OR keyword.* | |
| Simple filtering of numeric values and text | SELECT \*  FROM table name  **WHERE** (col1 = ‘text’ **OR** col1 = ‘text2’)  **AND** (col2 >=< val1 OR col2 >=< val2); |
| ***WHERE IN*** *- IN operator allows to specify multiple values in  WHERE clause, making it easier and quicker to specify multiple OR conditions* | |
| Simple filtering of numeric values and text | SELECT \*  FROM table name  WHERE col1 **IN** (val1, vl2, val3)  AND col2 **IN** (‘text1’, ‘text2’)  AND col3 >=< value or ‘text’; |
| Ex: All films except those released in 2015 | SELECT \*  FROM table name  WHERE col1 <> year; |
| ***WHERE AND OR BETWEEN*** *- BETWEEN keyword used for filtering values within a specified range* | |
| Simple filtering of numeric values and text | SELECT \*  FROM table name  WHERE (col1 **BETWEEN** val1 AND val2)  AND col2 >=< value or ‘text’; |
| ***WHERE IS NULL*** *- check for NULL values using the expression IS NULL* | |
| Simple filtering Nulls | SELECT \*  FROM table name  WHERE col **IS NULL**; |
| ***WHERE LIKE*** *- LIKE operator can be used in a WHERE clause to search for records that match a pattern in a* *column* | |
| ***% wildcard or placeholder will match a*zero, one or many*character*** | |
| Search for patterns in a column using **%** as a wild card or a placeholder | SELECT \*  FROM table name  WHERE col **LIKE** ‘pattern**%**’; |
| ***\_ wildcard or placeholder will match a single character*** | |
| Search for patterns in a column using **\_** as a wild card or a placeholder | SELECT \*  FROM table name  WHERE col **LIKE** ‘\_pattern%’; |
| ***WHERE NOT LIKE*** *– NOT LIKE operator can be used in a WHERE clause to search for records that doesn’t match a pattern in a* *column* | |
| ***% wildcard or placeholder will match a*zero, one or many*character*** | |
| Search for patterns in a column using **%** as a wild card or a placeholder | SELECT \*  FROM table name  WHERE col **NOT** **LIKE** ‘pattern**%**’; |
| ***\_ wildcard or placeholder will match a single character*** | |
| Search for patterns in a column using **\_** as a wild card or a placeholder | SELECT \*  FROM table name  WHERE col **NOT** **LIKE** ‘\_pattern%’; |

**Note: Important: in PostgreSQL (the version of SQL we're using), you must use single quotes with WHERE.**

## Aggregate Functions

|  |  |
| --- | --- |
| *Basic Queries For Aggregate Functions* | |
| ***SUM*** | |
| SUM returns the result of adding up the numeric values in a column | SELECT **SUM**(col)  FROM table name; |
| ***AVG*** | |
| AVG returns the Average value of the numeric values in a column | SELECT **AVG**(col)  FROM table name; |
| ***MIN*** | |
| MIN returns the least value of the numeric values in a column | SELECT **MIN**(col)  FROM table name; |
| ***MAX*** | |
| MAX returns the largest value of the numeric values in a column | SELECT **MAX**(col)  FROM table name; |

|  |  |
| --- | --- |
| *Combining Aggregate functions with WHERE* | |
| ***WHERE SUM*** | |
| Calculate the SUM of a column for a specific condition | SELECT **SUM**(col)  FROM table name  **WHERE** col2 >=< value; |
| ***WHERE AVG*** | |
| Calculate the AVG of a column for a specific condition | SELECT **AVG**(col)  FROM table name  **WHERE** col2 >=< value; |
| ***WHERE MIN*** | |
| Calculate the MIN of a column for a specific condition | SELECT **MIN**(col)  FROM table name  **WHERE** col2 >=< value; |
| ***WHERE MAX*** | |
| Calculate the MAX of a column for a specific condition | SELECT **MAX**(col) **AS** maximum %**AS Aliasing**  FROM table name  **WHERE** col2 >=< value; |

To filter based on the result of an aggregate function, you need another way! That's where the HAVING clause comes in. For example,

SELECT \*

FROM table name

GROUP BY col1

HAVING COUNT(col2) >=< value;

## Sorting (ORDER BY)

### Sorting single column

ORDER BY keyword is used to sort results in ascending or descending order according to the values of one or more columns. By default, ORDER BY will sort in ascending order. If you want to sort the results in descending order, you can use the DESC keyword.

SELECT \*

FROM table name

**ORDER BY** col;

SELECT \*

FROM table name

**ORDER BY** col **DESC**;

Note: ORDER BY sorts a column of text values by default Alphabetically (A-Z)

### 1.4.2. Sorting multiple columns

ORDER BY can also be used to sort on multiple columns. It will sort by the first column specified, then sort by the next, then the next, and so on.

SELECT \*

FROM table name

**ORDER BY** col1, col2;

SELECT \*

FROM table name

**ORDER BY** col1, col2 **DESC**;

Note: ORDER BY sorts a column of text values by default Alphabetically (A-Z)

## 1.5. Grouping (GROUP BY)

GROUP BY allows you to group a result by one or more columns. Commonly, GROUP BY is used with aggregate functions like COUNT() or MAX().

Note: GROUP BY always goes after the FROM clause!

A word of warning: SQL will return an error if you try to SELECT a field that is not in your GROUP BY clause without using it to calculate some kind of value about the entire group.

SELECT \*

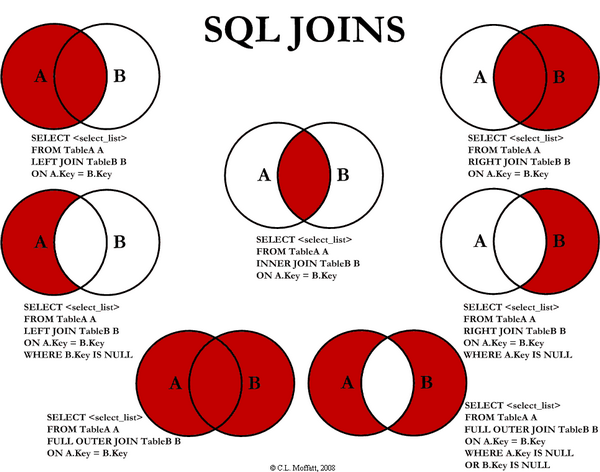
FROM table name

**GROUP BY** col

**ORDER BY** col1 **DESC**;

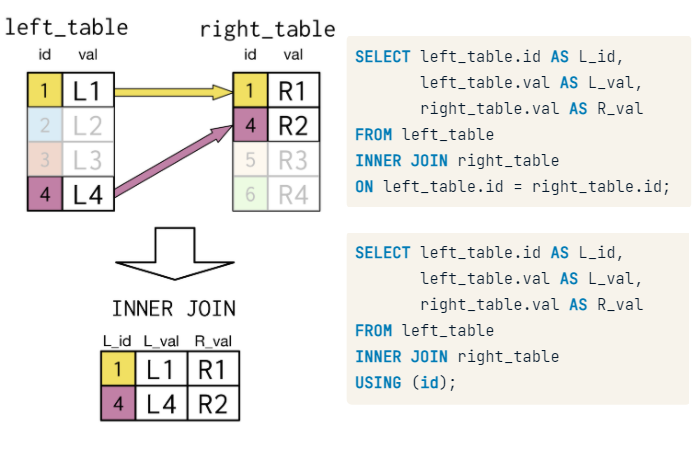
# 2.Joining data in SQL

## 2.1. Introduction to joins



### 2.1.1. INNER JOIN

The INNER JOIN (INTERSECTION) keyword selects records that have matching values in both tables.



#### I. SINGLE INNER JOIN

SELECT \*

FROM left\_table

**INNER JOIN** right\_table

**ON** left\_table.id = right\_table.id;

#### II. MULTIPLE INNER JOIN

SELECT \*

FROM left\_table

**INNER JOIN** right\_table

**ON** left\_table.id = right\_table.id

**INNER JOIN** another\_table

**ON** left\_table.id = another\_table.id;

#### III. Via USING – COMMON FIELD NAMES

SELECT \*

FROM left\_table

**INNER JOIN** right\_table

**USING** (id);

#### IV. CASE WHEN - THEN

SELECT country\_code, size,

  CASE WHEN size > 50000000 THEN 'large'

       WHEN size > 1000000 THEN 'medium'

       ELSE 'small' END

       AS popsize\_group

INTO pop\_plus

FROM populations

WHERE year = 2015;

#### V. SPECIAL CASE SELF JOIN

SELECT \*

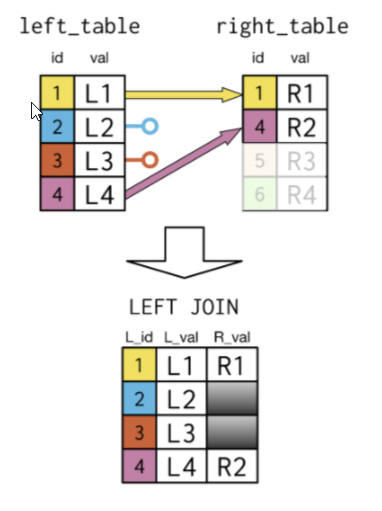
FROM table as t1

**INNER JOIN** table as t2

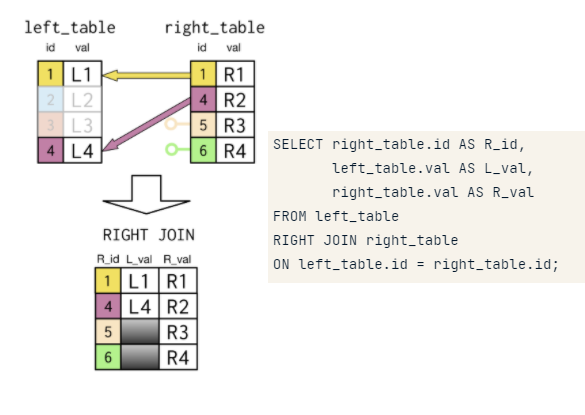
**USING** (id);

### 2.1.2. OUTER JOIN

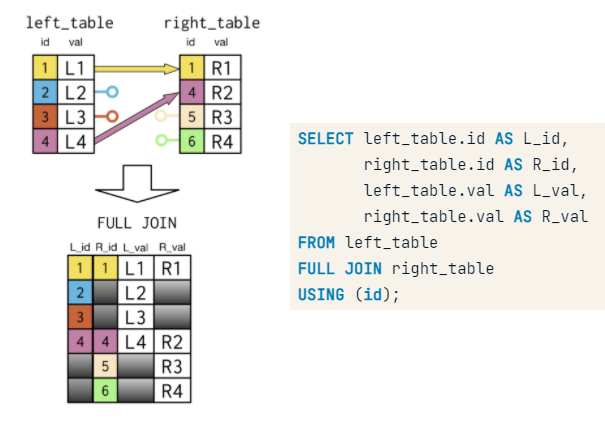
#### I. LEFT JOIN



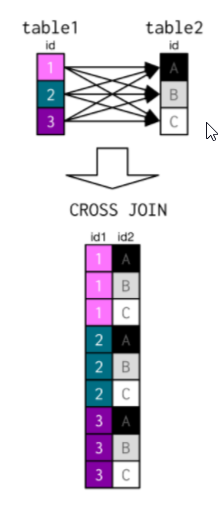
#### II. RIGHT JOIN



#### III. FULL JOIN



### 2.1.3. CROSS JOIN



### 2.1.4. Semi join

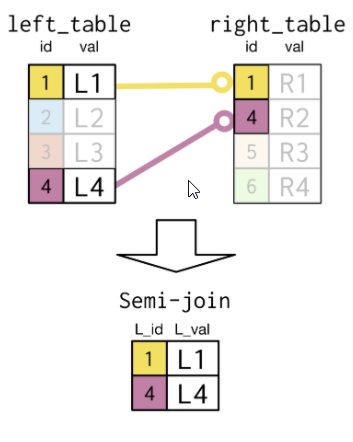
SELECT col1

FROM table1

**WHERE col2 IN**

(SELECT col2

FROM table2;)



**SEMI-JOIN Equivalent INNER JOIN**

SELECT DISTINCT name

FROM languages

**WHERE code IN**

(SELECT code

FROM countries

WHERE region = 'Middle East')

ORDER BY name;

SELECT DISTINCT languages.name AS language

FROM languages

**INNER JOIN** countries

**ON** languages.code = countries.code

**WHERE** region = 'Middle East'

ORDER BY language;

### 2.1.6. Anti join

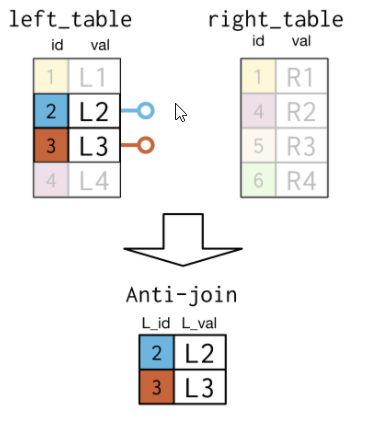
SELECT col1

FROM table1

**WHERE col2 NOT IN**

(SELECT col2

FROM table2;)



## 2.2. Set theory clauses

### I.UNION

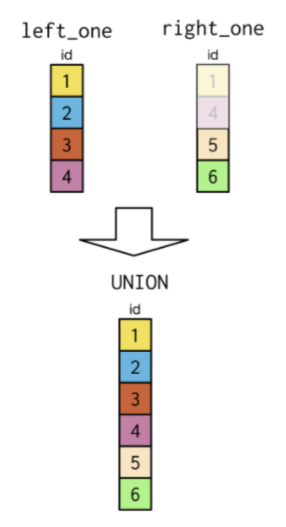
SELECT col

FROM table1

**UNION**

SELECT \*

FROM table2;



### II.UNION ALL

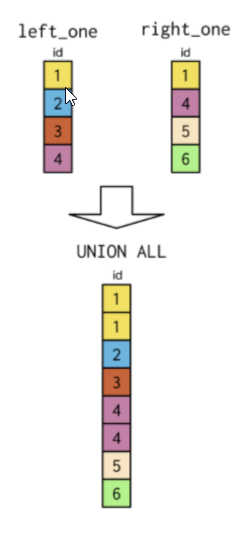
SELECT \*

FROM table1

**UNION ALL**

SELECT \*

FROM table2;



### III.INTERSECT

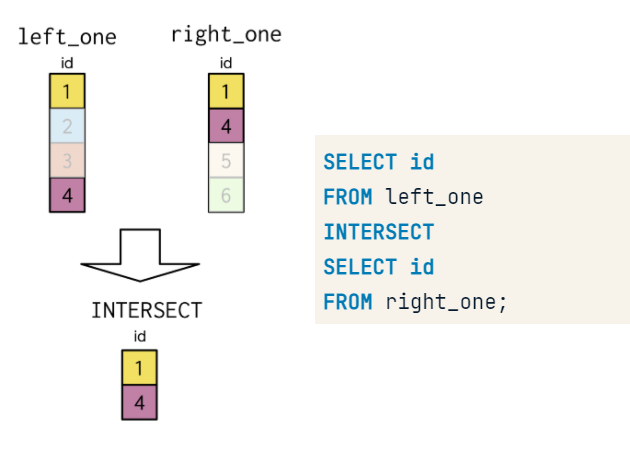
SELECT \*

FROM table1

**INTERSECT**

SELECT \*

FROM table2;



### IV.EXCEPT

SELECT \*

FROM table1

**EXCEPT**

SELECT \*

FROM table2;

### V.SEMI JOINS

## 2.3. Subqueries

### 2.3.1. Subqueries inside **WHERE** clause

### 2.3.2. Subqueries inside **SELECT** clause

SELECT countries.name AS country, COUNT(\*) AS cities\_num

  FROM cities

    INNER JOIN countries

    ON countries.code = cities.country\_code

GROUP BY country

ORDER BY cities\_num DESC, country

LIMIT 9;

SELECT countries.name AS country,

  (SELECT COUNT(\*)

   FROM cities

   WHERE countries.code = cities.country\_code) AS cities\_num

FROM countries

ORDER BY cities\_num DESC, country

LIMIT 9;

### 2.3.3. Subqueries inside **FROM** clause