SQL AS AN ABAPER

- Columns are called Fields , rows are called Records
- Fetching unique values
  `SELECT DISTINCT lv_var FROM lt_time;`
- For conditions in a SQL query we use 'WHERE' keyword
  `SELECT * FROM fruits WHERE fruit_name = 'APPLE';`
  Various ways in which we can use `'AND' 'OR' 'BETWEEN'` operators.
  `SELECT * FROM fruits WHERE name = 'APPLE' OR name = 'ORANGE' AND year BETWEEN 2000 AND 2003`
  Using `( )` parenthesis for conditions which are related to one scenario , for example in the previous query it is going to fetch even those records which have only name as apple whereas our `AND` operator will only be working for the `orange` statement.
  To make it work for combination of apple and year or orange and year, we need to use parenthesis.
  `SELECT * FROM fruits WHERE (name = 'APPLE' OR name = 'ORANGE') AND year BETWEEN 2000 AND 2003`
  Negation is done by using `NOT` before the name of the variable in the conditional segment of the query
  `SELECT * FROM fruits WHERE NOT fruit_name = 'APPLE';`
- Boolean
  For Boolean `1 OR X is True` and `0 OR space is FALSE`
  Let us take an example of the below table:

	A	B	C
1	Name	Attendence	Eligible
2	Harsh	78	1
3	harry	9	FALSE
4	happy	82	TRUE
5	sonu	2	0
6	monu	99	1

  We will be using the Boolean True to fetch the 1 entries
  Example -
  `select * from test_boolean WHERE eligible_ = TRUE or eligible_ = 'TRUE';`

Output -

Name_	Attendence_	Eligible_
Harsh	78	1
happy	82	TRUE
monu	99	1

- `IN` keyword
  This keyword can be used when there are multiple values that can be assigned to the condition part of that variable with the `OR` logical operator , so to simplify the query we use `IN` key work
  For example -
  Long Query:
  `SELECT * FROM it_final WHERE name = 'Harsh' OR name = 'Shreya' OR name = 'Steven';`
  Short Query:
  `SELECT * FROM it_final WHERE name IN ( 'Harsh', 'Shreya', 'Steven' );`
- `BETWEEN` Keyword
  This keyword can be used to replace the following scenario
  `SELECT employees FROM lt_final WHERE age >= 45 AND age <=56;`
  Here we are basically Taking the age range to 56, for this replacement of operators `>` and `<` we can use between keyword like this -
  `SELECT employees FROM lt_final WHERE age BETWEEN 45 AND 56`
  Fun thing is , it involves the numbers as `=` .
- `LIKE` keyword
  This is very useful for pattern searches where we know if the string or value should contain what character and at what place
  For ex -
  `SELECT * FROM lt_final WHERE name LIKE 'K%'`
  This is going to fetch all the entries where name starts with K
  `SELECT * FROM lt_final WHERE name LIKE '%K%';`
  This is going to capture all that contains K
  `SELECT * FROM lt_final WHERE name LIKE '%__K%;`
  This is going to fetch the entries which have K in the third character / index
  `SELECT * FROM lt_final WHERE name LIKE '%K';`
  This is going to fetch all the names which end with K
- `Aliases` Keyword
  Makes one more column in the data base copying the data from the real database table or columns

```
SELECT NAME as D_starters, hint FROM demo WHERE Name LIKE 'D%';
```

D_starters	Hint
Donate (ERC20: ETH or USDC)	0xCcc227E5615D4FADd758228Bab12ceb465D4ED18
Donate (BTC)	bc1q25zqmgll2fz0tyduusyfageh3jh7htcwjt2rdk
DELETE	DELETE FROM table_name

- Handling Dates

  For dates , in SQL it is stated in this format 'YYYY-MM-DD'

  Examples -

  ```
  SELECT id as game FROM games WHERE NOT date BETWEEN '2022-12-21' AND '2023-03-20'
  ORDER BY date desc;
  ```

  Output -

games

	id	date
1	5	2022-11-20T22:00:00.000Z
2	6	2022-01-11T22:00:00.000Z
3	7	2023-01-01T22:00:00.000Z
4	8	2023-02-12T22:00:00.000Z
5	9	2022-10-28T21:00:00.000Z
6	10	2022-08-31T21:00:00.000Z
7	11	2022-12-21T22:00:00.000Z
8	12	2022-12-19T22:00:00.000Z
9	13	2022-10-31T22:00:00.000Z
10	14	2023-03-19T22:00:00.000Z

  JULIANDAYS

  This is a function to be used whenever we have to calculate the number of days between -

  For example -

  ```
  SELECT id WHERE JULIANDAY( start ) - JULIANDAY( end ) > 1;
  ```

- Aggregations
  Whenever we have to use the aggregations in SQL it is mandatory to use the GROUP By function.
  Example of aggregations are -
  ABAP

```
8
9   REPORT  z_test_har1.
10  TYPES: BEGIN OF ty_ekpo,
11         ebeln TYPE ekpo-ebeln,
12         menge_sum TYPE ekpo-menge,
13         END OF ty_ekpo.
14  DATA: lt_ekpo TYPE STANDARD TABLE OF ty_ekpo,
15        ls_ekpo TYPE ty_ekpo.
16  SELECT ebeln SUM( menge ) AS menge_sum
17     FROM ekpo INTO CORRESPONDING FIELDS OF TABLE lt_ekpo
18     WHERE ebeln = '0002009250'
19    GROUP BY ekpo~ebeln.
20  IF sy-subrc = 0.
21    LOOP AT lt_ekpo INTO ls_ekpo.
22      WRITE:/ ls_ekpo-ebeln,ls_ekpo-menge_sum.
23    ENDLOOP.
24  ENDIF.
```

`SELECT MAX( line )+( SELECT MIN( line ) FROM lt_final )` FROM lt_final WHERE area = 120;`

- JOINS

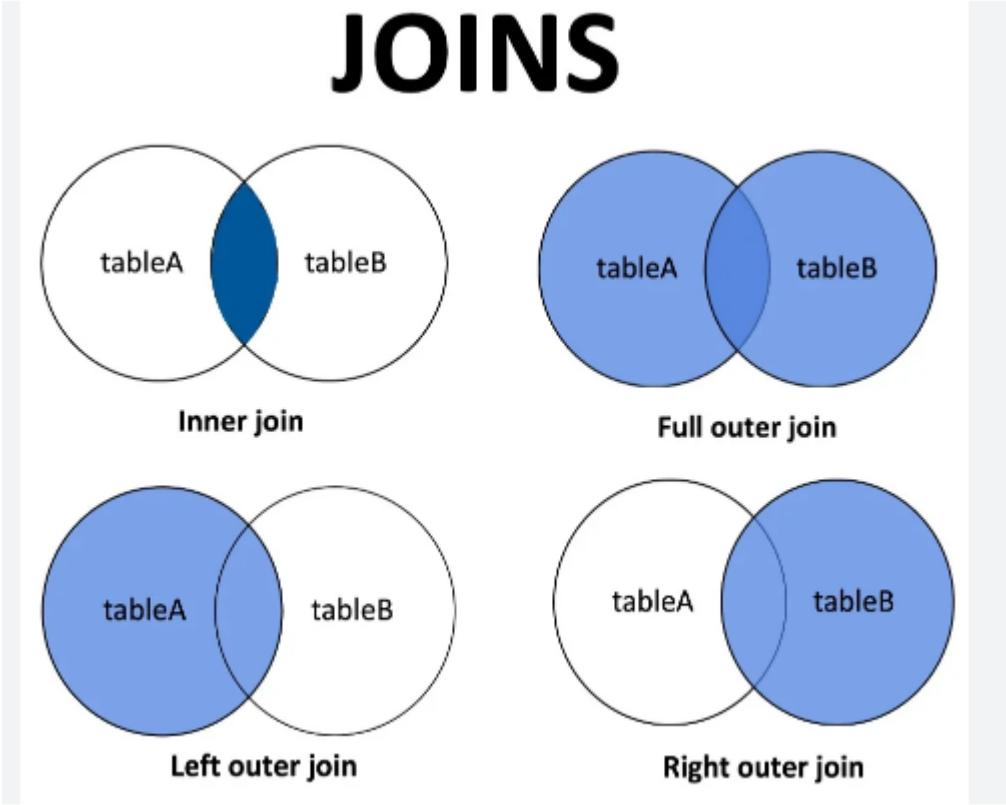


Table id*name:*

SQLite

```
1 SELECT * FROM id_name;
```

id_	name	marks_
1	harsh	1122
2	padma	12456
3	vaibhav	45536
4	vishal	18838
5	saurabh	134
6	adarsh	456
7	saransh	2378
14	Harsh	16
14	Harsh	16

Table id name2:

```
1 SELECT * FROM id_name2;
```

id_	name	class	marks_
1	harsh	11	1122
2	padma	12	12456
3	vaibhav	15	45536
4	vishal	15	18838
5	saurabh	16	134
6	adarsh	8	456
7	saransh	11	2378

`SELECT a.id_, b.class FROM id_name AS a INNER JOIN id_name2 as b ON a.id_ = b.id_;`

Inner Join

We executed inner join on the basis of the id , and we can see on those id's which are matching have come

```
1 SELECT a.id_, a.name AS first_name , b.name AS second_name, b.class FROM id_name AS a INNER JOIN id_name2 AS b ON a.id_ = b.id_;
```

id_	first_name	second_name	class
1	harsh	harsh	11
2	padma	padma	12
3	vaibhav	vaibhav	15
4	vishal	vishal	15
5	saurabh	saurabh	16
6	adarsh	adarsh	8
7	saransh	saransh	11

Left outer join -

How i would take it is that let us say , you have the foundations ready for the data like i need data around shipments , and they are lets say in my A table which is very important and cannot be compromised that any of the data is lost , now there is a B table from which I need data about BOL . I just need the data for my shipments and if the B table doesnt have then just leave blank for that column.

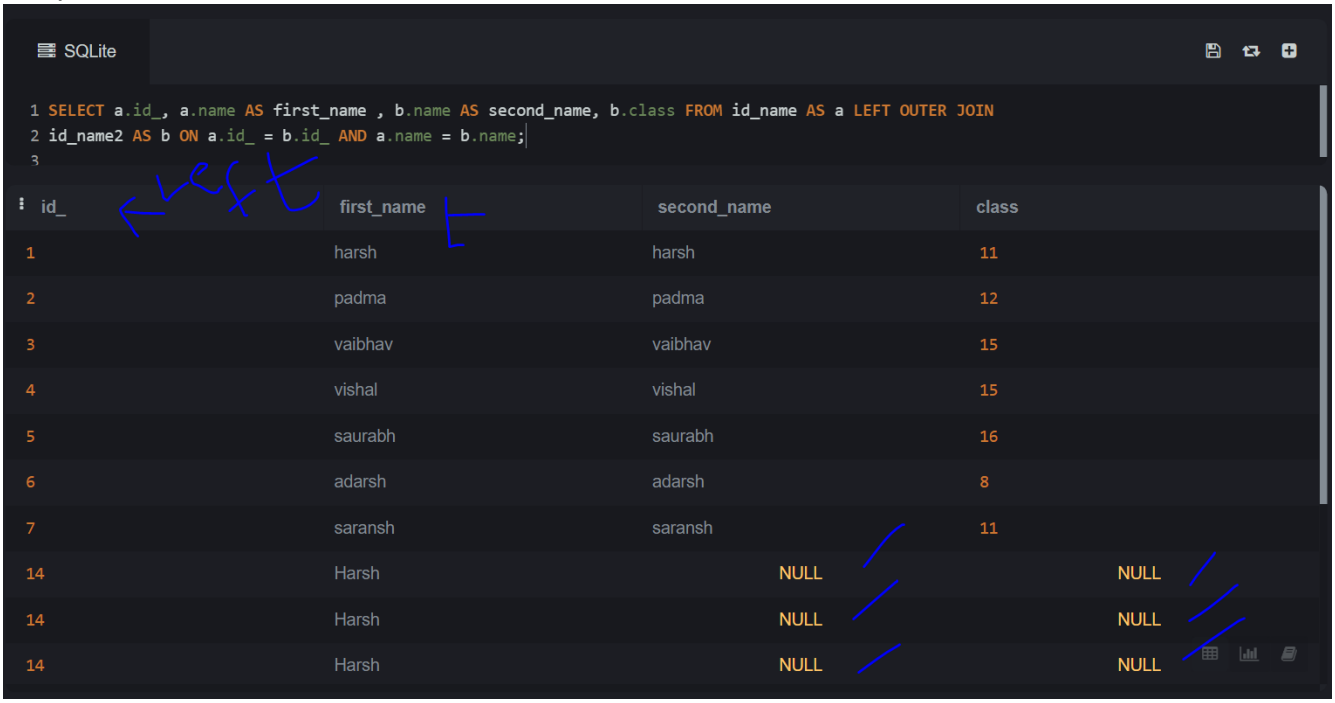
Here we will be using Left outer join and if our Right table was primary , i would have used right outer join.

Whichever direction is given n the name of the outer join , means that all the data of that table or direction will come.

One example is -

```
SELECT a.id_, a.name as first_name , b.name as second_name, b.class FROM id_name AS a
LEFT OUTER JOIN id_name2 as b ON a.id_ = b.id_ AND a.name = b.name;
```

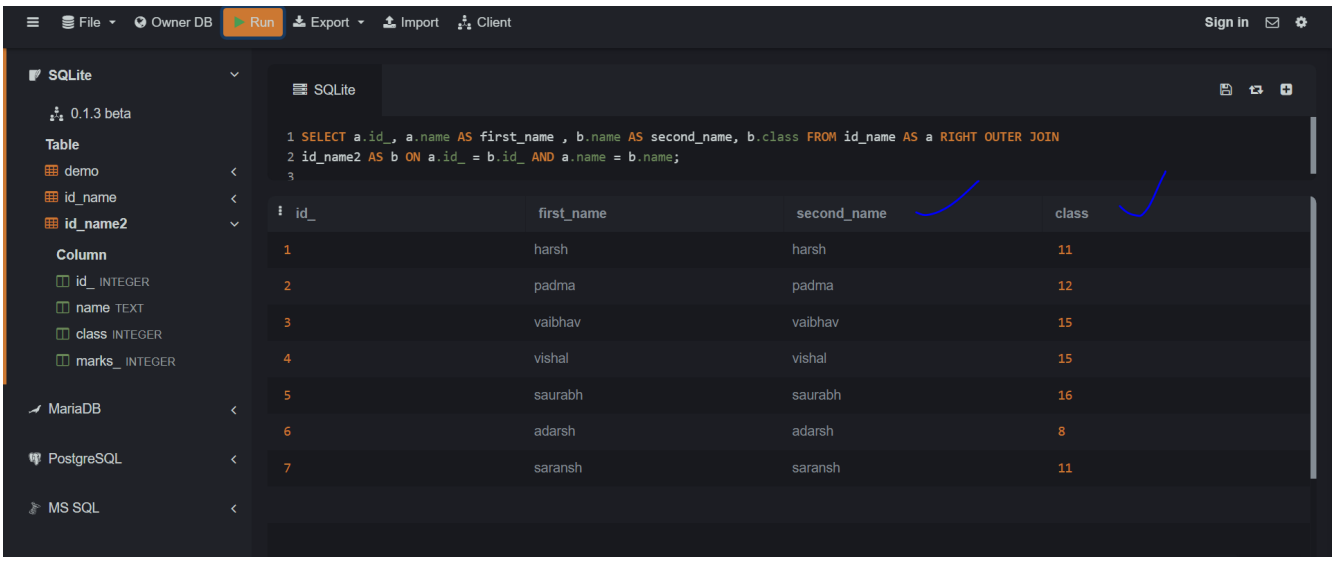
Output -



When giving preference to Right table , the left excessive data wont come as it does not fit the requirement of primary table A.

Example -

```
SELECT a.id_, a.name as first_name , b.name as second_name, b.class FROM id_name AS a
RIGHT OUTER JOIN id_name2 as b ON a.id_ = b.id_ AND a.name = b.name;
```



- Window Functions In SQL

  These window functions are used to make subsets of the rows or records which are present in

your particular selection. The function where the criteria of these windows are defined is the `OVER()` function

Types of window functions -

- Aggregate window functions
- Value window functions
- Ranking window functions

  A window is basically a set of rows or observations in a table or result set. In a table you may have more than one window depending on how you specify the query – you will learn about this shortly. A window is defined using the `OVER()` clause in SQL.

  Now OVER keyword has 2 clauses or functions which are to be used to declare the scope of the window.

  If we use OVER() after any function then it becomes a window function , but if we don't specify any criteria in over( ) by using its clause then the whole result set becomes the window.

  `PARTITION BY ()`

  Example -

  ```
  SELECT * , MAX(Marks) OVER(PARTITION BY(Subject)) as maximum_across_department
  FROM students_data;
  ```

  Another clause is -

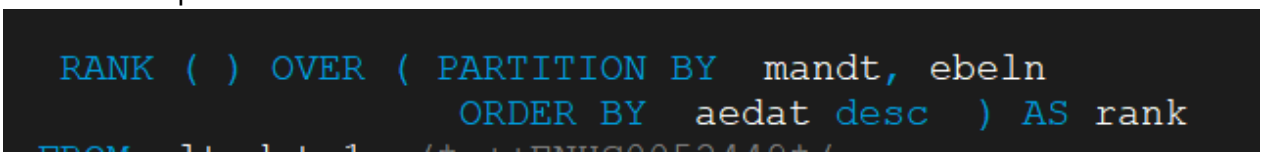
  `ORDER BY() , widely used for RANK functions`

  Example -

  ```
  SELECT *, RANK() OVER(PARTITION BY(Subjects) ORDER BY(marks)) as Topper FROM
  students_data;
  ```

  Rank is determined by the field present in ORDER BY clause.

  AMDP example of the same is -

  Reference -

  https://www.freecodecamp.org/news/window-functions-in-sql/==