Contents

1. *SQL Join*
2. *SQL Join Types*
   - *2.1. INNER JOIN*
   - *2.2. LEFT JOIN*
   - *2.3. RIGHT JOIN*
   - *2.4. SELF JOIN*
   - *2.5. CARTESIAN JOIN*

1. SQL Join

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables, (a) customers table is as follows:

ID	NAME	AGE	ADDRESS
1	Ali	22	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
6	Kamil	41	Misan

(b) Another table is orders as follows:

ID	DATE	AMOUNT	CUSTOMER_ID
1	2010-10-10 00:00:00	200	1
2	2010-10-10 00:00:00	300	1
3	2010-10-10 00:00:00	100	2
4	2010-12-21 00:00:00	100	3
5	2010-12-21 00:00:00	200	4
6	2010-12-21 00:00:00	100	5
7	2010-12-21 00:00:00	200	6
8	2010-12-21 00:00:00	300	6

Now, let us join these two tables in our SELECT statement as follows:

```
SELECT ID, NAME, AGE, AMOUNT
FROM customers,orders
WHERE customers.ID=orders.CUSTOMER_ID;
```

ID	NAME	AGE	AMOUNT
1	Ali	22	200
1	Ali	22	300
2	Ahmed	30	100
3	Samer	32	100
4	Hamed	32	200
5	Ali	23	100
6	Kamil	41	200
6	Kamil	41	300

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

2. SQL Join Types

There are different types of joins available in SQL:

- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

2.1 INNER JOIN

The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an EQUIJOIN. The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

The basic syntax of **INNER JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_filed = table2.common_field;
```

`

Example: Consider the following two tables, (a) customers table is as follows:

ID	NAME	AGE	ADDRESS
1	Ali	22	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
6	Kamil	41	Misan

(b) Another table is orders as follows:

Ord_ID	DATE	AMOUNT	CUSTOMER_ID
1	2010-10-10 00:00:00	200	1
2	2010-10-10 00:00:00	300	1
3	2010-10-10 00:00:00	100	2
4	2010-12-21 00:00:00	100	2
5	2010-12-21 00:00:00	200	4
6	2010-12-21 00:00:00	100	5
7	2010-12-21 00:00:00	200	5
8	2010-12-21 00:00:00	300	6

Now, let us join these two tables using INNER JOIN as follows:

```
SELECT ID, NAME, AMOUNT, DATE FROM customers
INNER JOIN orders ON customers.ID = orders.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ali	200	2010-10-10 00:00:00
1	Ali	300	2010-10-10 00:00:00
2	Ahmed	100	2010-10-10 00:00:00
2	Ahmed	100	2010-12-21 00:00:00
4	Hamed	200	2010-12-21 00:00:00
5	Ali	100	2010-12-21 00:00:00
5	Ali	200	2010-12-21 00:00:00
6	Kamil	300	2010-12-21 00:00:00

2.2 LEFT JOIN

The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax:

The basic syntax of **LEFT JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
LEFT JOIN table2
ON table1.common_filed = table2.common_field;
```

Example: Consider the following two tables, (a) customers table is as follows:

ID	NAME	AGE	ADDRESS
1	Ali	22	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
6	Kamil	41	Misan

(b) Another table is orders as follows:

Ord_ID	DATE	AMOUNT	CUSTOMER_ID
1	2010-10-10 00:00:00	200	1
2	2010-10-10 00:00:00	300	1
3	2010-10-10 00:00:00	100	2
4	2010-12-21 00:00:00	100	2
5	2010-12-21 00:00:00	200	4
6	2010-12-21 00:00:00	100	5
7	2010-12-21 00:00:00	200	5
8	2010-12-21 00:00:00	300	6

Now, let us join these two tables using LEFT JOIN as follows:

```
SELECT ID, NAME, AMOUNT, DATE FROM customers
LEFT JOIN orders ON customers.ID = orders.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ali	200	2010-10-10 00:00:00
1	Ali	300	2010-10-10 00:00:00
2	Ahmed	100	2010-10-10 00:00:00
2	Ahmed	100	2010-12-21 00:00:00
3	Samer	NULL	NULL
4	Hamed	200	2010-12-21 00:00:00
5	Ali	100	2010-12-21 00:00:00
5	Ali	200	2010-12-21 00:00:00
6	Kamil	300	2010-12-21 00:00:00

2.3 RIGHT JOIN

The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result, but with NULL in each column from left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax:

The basic syntax of **RIGHT JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
RIGHT JOIN table2
ON table1.common_filed = table2.common_field;
```

Example: Consider the following two tables, (a) customers table is as follows:

ID	NAME	AGE	ADDRESS
1	Ali	22	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
6	Kamil	41	Misan

(b) Another table is orders as follows:

Ord_ID	DATE	AMOUNT	CUSTOMER_ID
1	2010-10-10 00:00:00	200	1
2	2010-10-10 00:00:00	300	1
3	2010-10-10 00:00:00	100	2
4	2010-12-21 00:00:00	100	2
5	2010-12-21 00:00:00	200	4
6	2010-12-21 00:00:00	100	5
7	2010-12-21 00:00:00	200	5
8	2010-12-21 00:00:00	300	6

Now, let us join these two tables using RIGHT JOIN as follows:

```
SELECT ID, NAME, AMOUNT, DATE
FROM customers RIGHT JOIN orders ON customers.ID = orders.CUSTOMER_ID;
```

ID	NAME	AMOUNT	DATE
1	Ali	200	2010-10-10 00:00:00
1	Ali	300	2010-10-10 00:00:00
2	Ahmed	100	2010-10-10 00:00:00
2	Ahmed	100	2010-12-21 00:00:00
4	Hamed	200	2010-12-21 00:00:00
5	Ali	100	2010-12-21 00:00:00
5	Ali	200	2010-12-21 00:00:00
6	Kamil	300	2010-12-21 00:00:00

2.4 SELF JOIN

The SQL **SELF JOIN** is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax:

The basic syntax of **SELF JOIN** is as follows:

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

Example: Consider the following two tables, (a) customers table is as follows:

ID	NAME	AGE	ADDRESS
1	Ali	22	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
6	Kamil	41	Misan

Now, let us join the table using SELF JOIN as follows:

```
SELECT a.ID, a.NAME, a.AGE, a.ADDRESS  from customers a, customers b
WHERE a.AGE< b.AGE order by ID;
```

ID	NAME	AGE	ADDRESS
1	Sameer	22	Basrah
1	Sameer	22	Basrah
1	Sameer	22	Basrah
1	Sameer	22	Basrah
1	Sameer	22	Basrah
1	Sameer	22	Basrah
2	Ahmed	30	Basrah
2	Ahmed	30	Basrah
2	Ahmed	30	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
5	Ali	23	Basrah

2.5 CARTESIAN JOIN

The **CARTESIAN JOIN** or **CROSS JOIN** returns the cartesian product of the sets of records from the two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.

Syntax:

The basic syntax of **INNER JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM  table1, table2 [, table3 ]
```

Example: Consider the following two tables, (a) customers table is as follows:

ID	NAME	AGE	ADDRESS
1	Ali	22	Basrah
2	Ahmed	30	Basrah
3	Samer	32	Basrah
4	Hamed	32	Misan
5	Ali	23	Basrah
6	Kamil	41	Misan

(b) Another table is orders as follows:

Ord_ID	DATE	AMOUNT	CUSTOMER_ID
1	2010-10-10 00:00:00	200	1
2	2010-10-10 00:00:00	300	1
3	2010-10-10 00:00:00	100	2
4	2010-12-21 00:00:00	100	2
5	2010-12-21 00:00:00	200	4
6	2010-12-21 00:00:00	100	5
7	2010-12-21 00:00:00	200	5
8	2010-12-21 00:00:00	300	6

Now, let us join these two tables using CARTESIAN JOIN as follows:

```
SELECT ID, NAME, AMOUNT, DATE FROM customers, orders;
```

ID	NAME	AMOUNT	DATE
1	Ali	200	2010-10-10 00:00:00
2	Ahmed	200	2010-10-10 00:00:00
3	Samer	200	2010-10-10 00:00:00
4	Hamed	200	2010-10-10 00:00:00
5	Ali	200	2010-10-10 00:00:00
6	Kamil	200	2010-10-10 00:00:00
1	Ali	300	2010-10-10 00:00:00
2	Ahmed	300	2010-10-10 00:00:00
3	Samer	300	2010-10-10 00:00:00
4	Hamed	300	2010-10-10 00:00:00
5	Ali	300	2010-10-10 00:00:00
6	Kamil	300	2010-10-10 00:00:00
1	Ali	100	2010-10-10 00:00:00
2	Ahmed	100	2010-10-10 00:00:00
3	Samer	100	2010-10-10 00:00:00
4	Hamed	100	2010-10-10 00:00:00
5	Ali	100	2010-10-10 00:00:00
6	Kamil	100	2010-10-10 00:00:00
1	Ali	100	2010-12-21 00:00:00
2	Ahmed	100	2010-12-21 00:00:00