Lab 05: Working with JOINs

Objective(s):

1. Introduction of JOIN
2. INNER JON
3. LEFT JOIN
4. RIGHT JOIN
5. CROSS JOIN
6. STRAIGHT JOIN
7. NATURAL JOIN
8. EQUI JOIN
9. NON EQUI JOIN
10. SELF JOIN

1: Introduction of JOIN

A relational database consists of multiple related tables linking together using common columns which are known as foreign key columns. Because of this, data in each table is incomplete from the business perspective.

By using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL should use data from one table to select the rows in another table.

A join condition defines the way two tables are related in a query by:

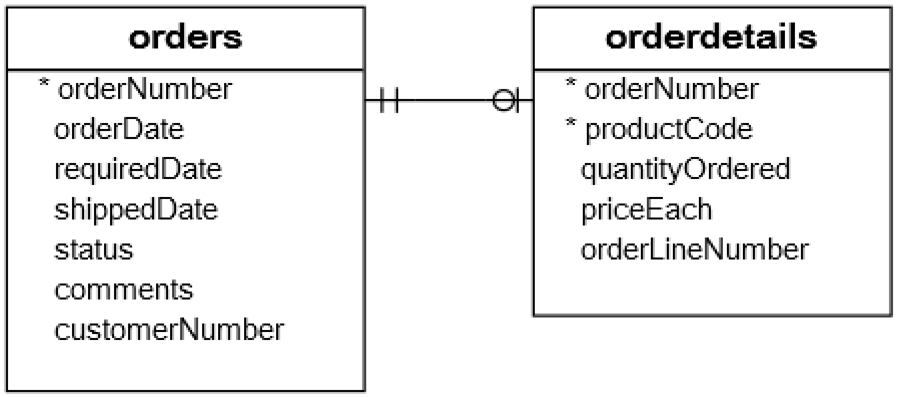
- Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.
- Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns.

There are various forms of the JOIN clause. These will include:

1. INNER JOIN
2. OUTER JOIN (both LEFT and RIGHT)
3. FULL OUTER JOIN
4. CROSS JOIN

A JOIN does just what it sounds like—it puts the information from two tables together into one result set. We can think of a result set as being a “virtual” table. It has both columns and rows, and the columns have data types. How exactly does a JOIN put the information from two tables into a single result set? Well, that depends on how you tell it to put the data together—that’s why there are four different kinds of JOINs. The thing that all JOINs have in common is that they match one record up with one or more other records to make a record that is a superset created by the combined columns of both records.

For example we have the orders and orderdetails tables that are linked using the orderNumber column:

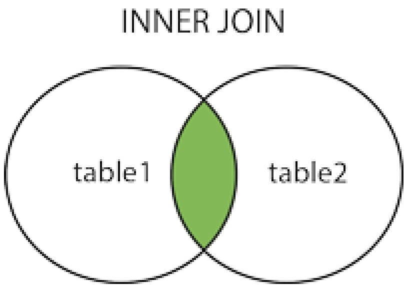


To get complete orders' information, you need to query data from both orders and orderdetails tables. That's why joins come into the play.

A join is a method of linking data between one (self-join) or more tables based on values of the common column between the tables.

2: INNER JOIN/JOIN

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



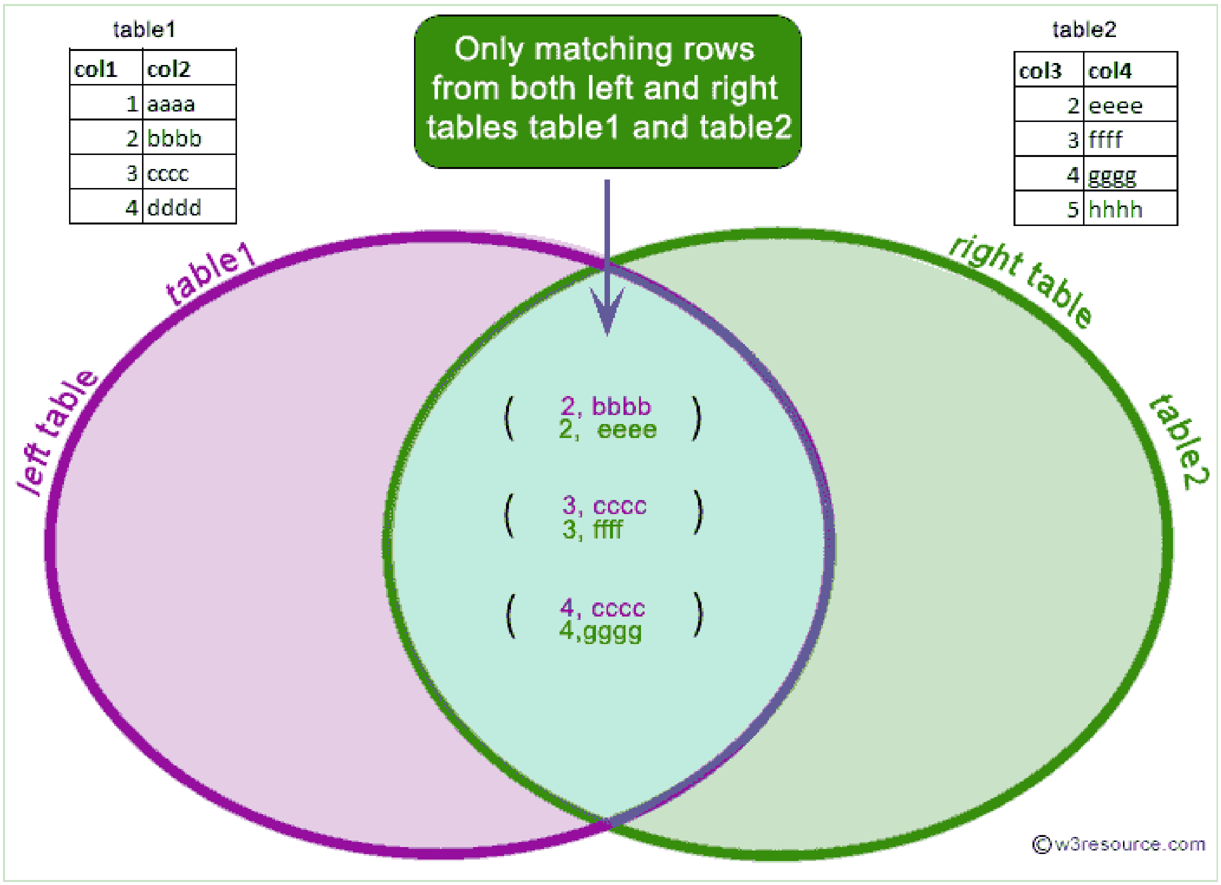
Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

If the join condition uses the equal operator (=) and the column names in both tables used for matching are the same, you can use the USING clause instead:

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
USING(column_name);
```

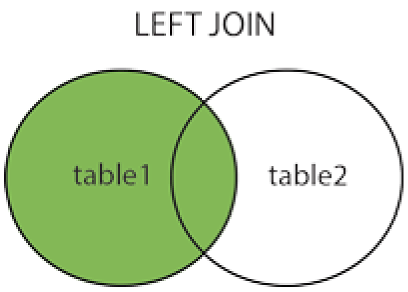


col1	col2
1	aaaa
2	bbbb
3	cccc
4	dddd

col3	col4
2	eeee
3	ffff
4	gggg
5	hhhh

3: LEFT JOIN

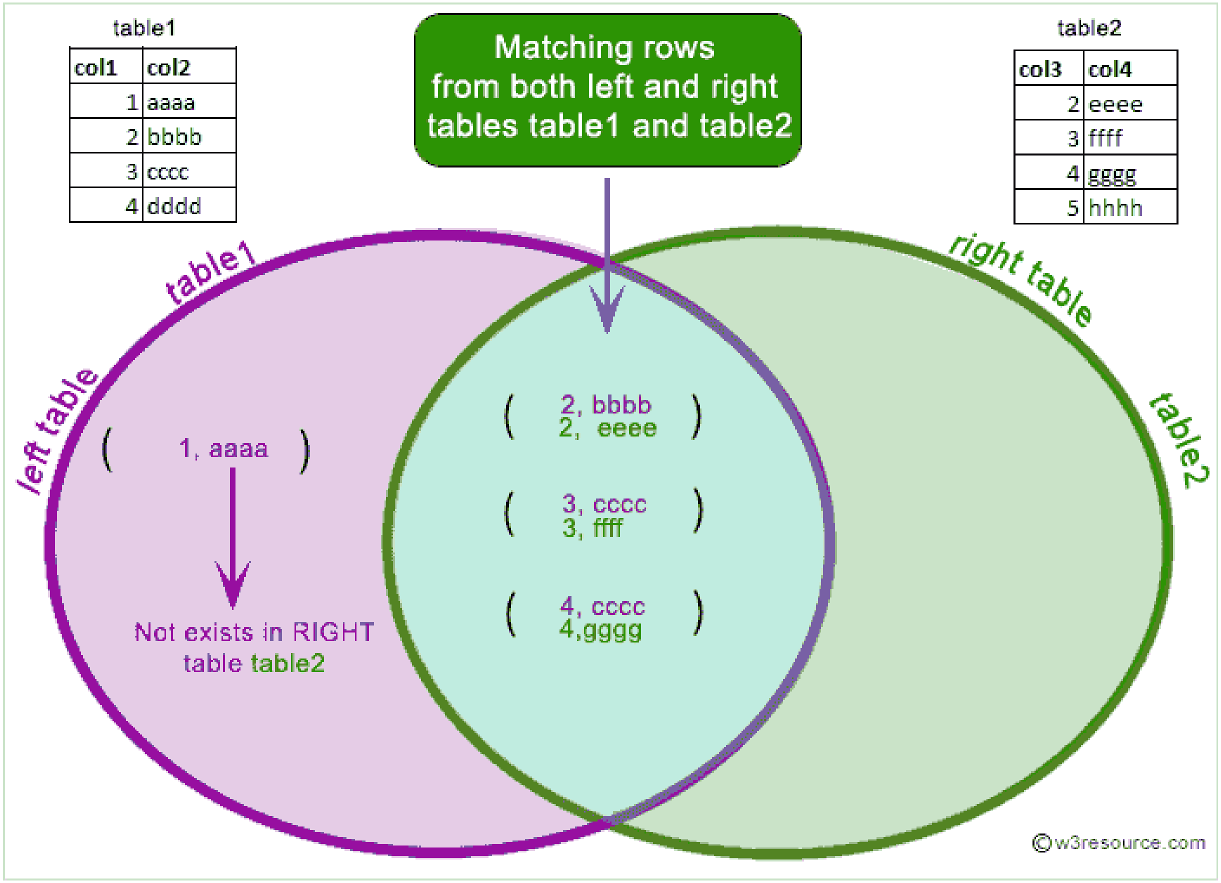
The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.



Syntax:

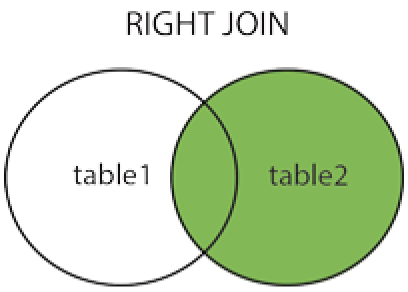
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



4: RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

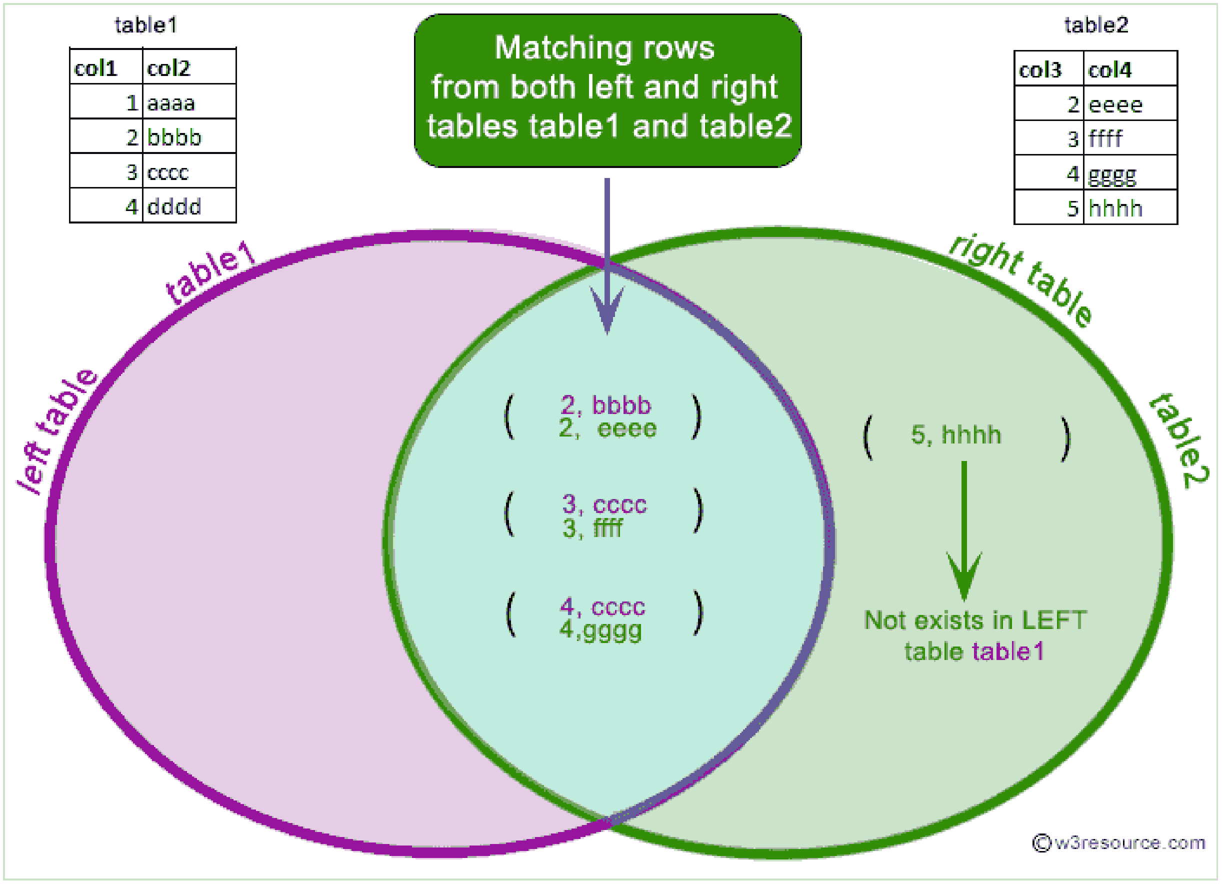


Syntax:

```
SELECT column_name(s)
FROM table1
```

```
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



5: CROSS JOIN

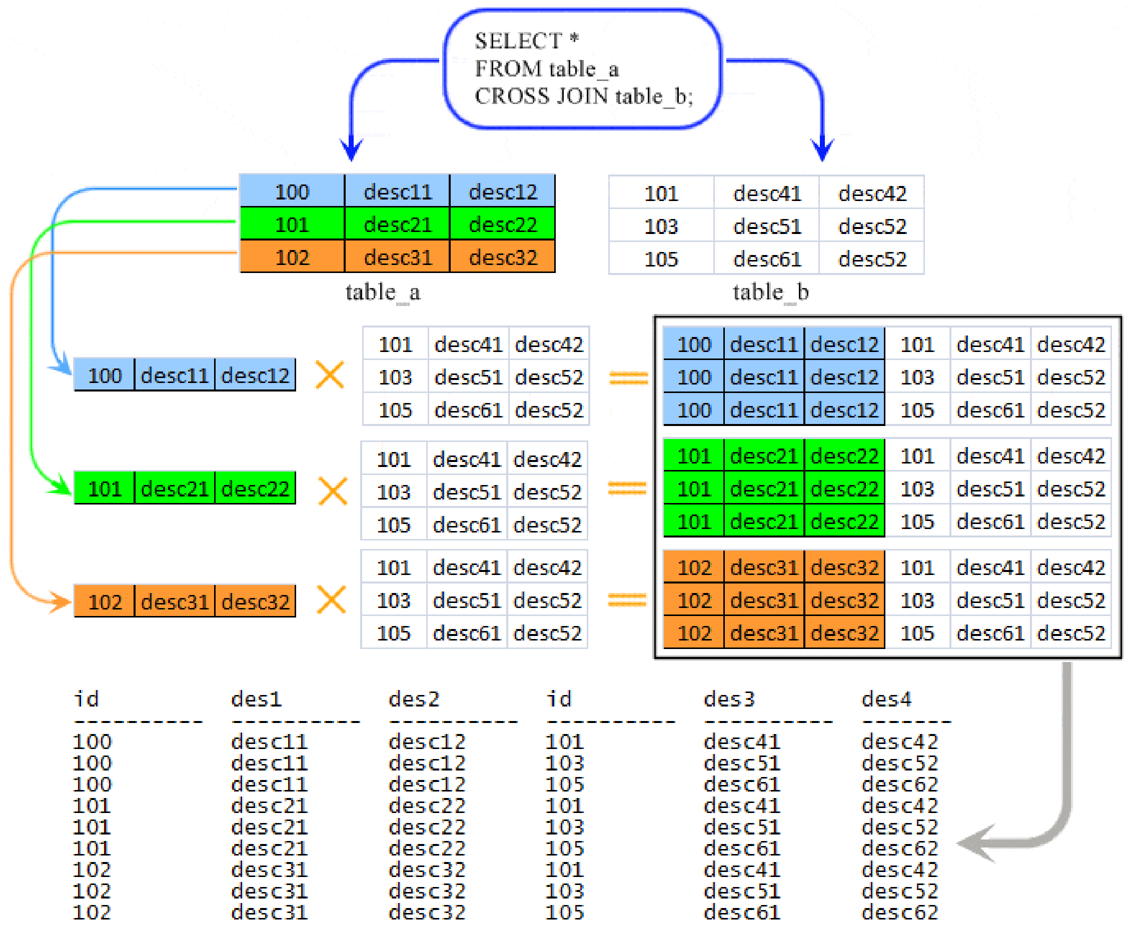
In MySQL, the CROSS JOIN produced a result set which is the product of rows of two associated tables when no WHERE clause is used with CROSS JOIN.

In this join, the result set appeared by multiplying each row of the first table with all rows in the second table if no condition introduced with CROSS JOIN.

This kind of result is called as Cartesian Product.

In MySQL, the CROSS JOIN behaves like JOIN and INNER JOIN without using any condition.

In standard SQL the difference between INNER JOIN and CROSS JOIN is ON clause can be used with INNER JOIN on the other hand ON clause can't be used with CROSS JOIN.



```
SELECT *
FROM table_a
CROSS JOIN table_b;
```

table_a

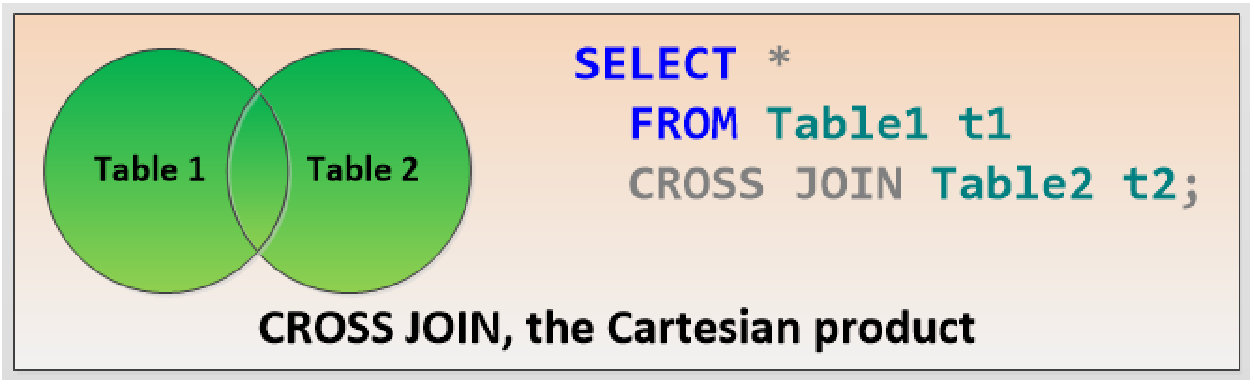
100	desc11	desc12
101	desc21	desc22
102	desc31	desc32

table_b

101	desc41	desc42
103	desc51	desc52
105	desc61	desc52

100	desc11	desc12	101	desc41	desc42
100	desc11	desc12	103	desc51	desc52
100	desc11	desc12	105	desc61	desc52
101	desc21	desc22	101	desc41	desc42
101	desc21	desc22	103	desc51	desc52
101	desc21	desc22	105	desc61	desc52
102	desc31	desc32	101	desc41	desc42
102	desc31	desc32	103	desc51	desc52
102	desc31	desc32	105	desc61	desc52

```
id          des1        des2        id          des3        des4
----------  ----------  ----------  ----------  ----------  -------
100         desc11      desc12      101         desc41      desc42
100         desc11      desc12      103         desc51      desc52
100         desc11      desc12      105         desc61      desc62
101         desc21      desc22      101         desc41      desc42
101         desc21      desc22      103         desc51      desc52
101         desc21      desc22      105         desc61      desc62
102         desc31      desc32      101         desc41      desc42
102         desc31      desc32      103         desc51      desc52
102         desc31      desc32      105         desc61      desc62
```



```
SELECT *
  FROM Table1 t1
  CROSS JOIN Table2 t2;
```

CROSS JOIN, the Cartesian product

Syntax:

```
SELECT table1.col1, table2.col1, table1.col2, table2.col3, …
FROM table1
CROSS JOIN table2;
```

6: STRAIGHT JOIN

In MySQL an STRAIGHT_JOIN scans and combines matching rows (if specified any condition) which are stored in associated tables otherwise it behaves like an INNER JOIN or JOIN of without any condition.

"STRAIGHT_JOIN is similar to JOIN, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order."

In the following example no condition have been specified, so, this join has combines each row of left table with all rows in right table.

Syntax:

```
SELECT table1.col1, table2.col1, table1.col2, table2.col3, …
FROM table1
STRAIGHT JOIN table2;
```

7: NATURAL JOIN

In MySQL, the NATURAL JOIN is such a join that performs the same task as an INNER JOIN, in which the ON or USING clause refers to all columns that the tables to be joined have in common.

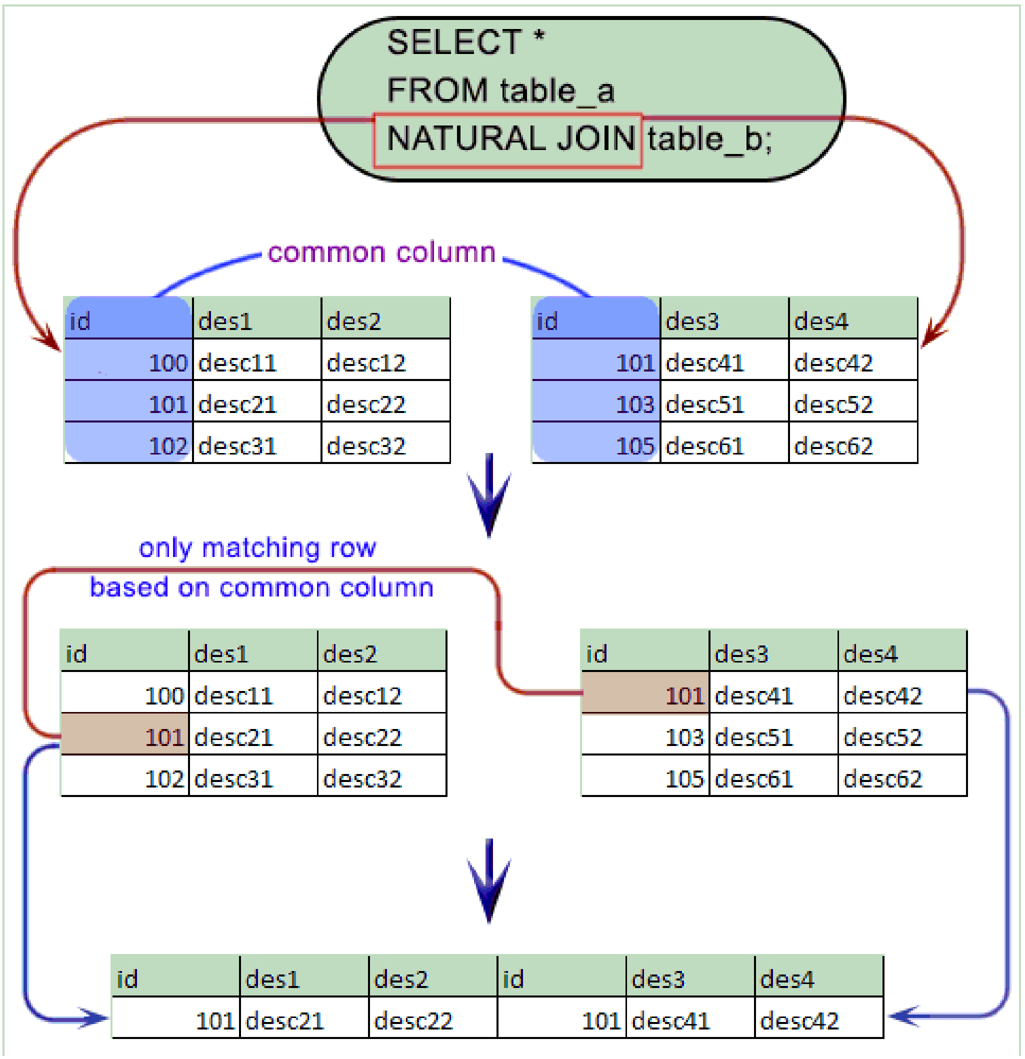
The MySQL NATURAL JOIN is a type of EQUI JOIN and is structured in such a way that, columns with the same name of associate tables will appear once only.

Natural Join: Guidelines:

- The associated tables have one or more pairs of identically named columns.
- The columns must be the same data type.
- Don’t use ON clause in a NATURAL JOIN.

Syntax:

```
SELECT table1.col1, table2.col1, table1.col2, table2.col3, …
FROM table1
NATURAL JOIN table2;
```



8: EQUI JOIN

SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables. An equal sign (=) is used as comparison operator in the where clause to refer equality.

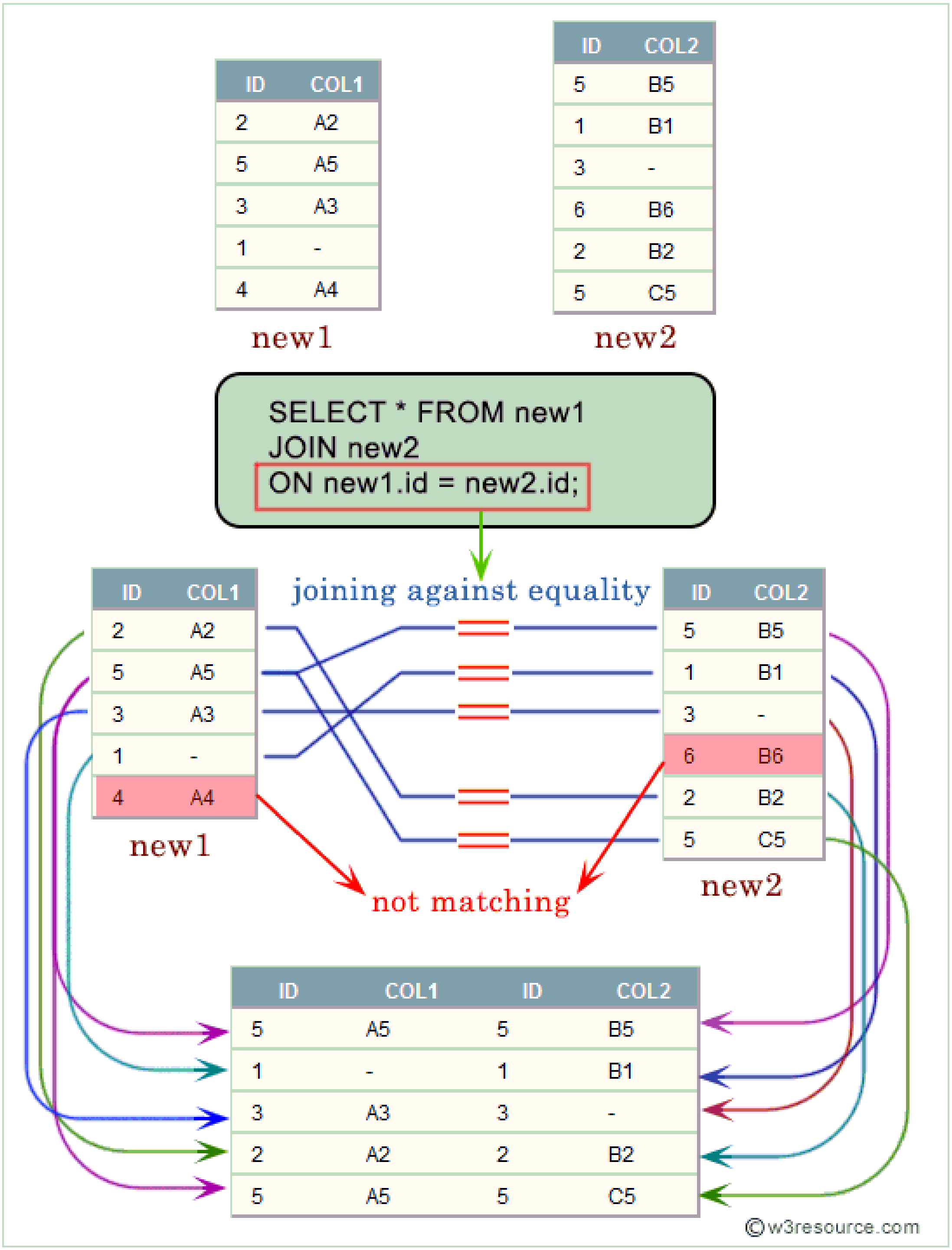
You may also perform EQUI JOIN by using JOIN/INNER JOIN keyword followed by ON keyword and then specifying names of the columns along with their associated tables to check equality.

Syntax:

```
SELECT column_list
FROM table1
JOIN table2 ON join_conditions;
```

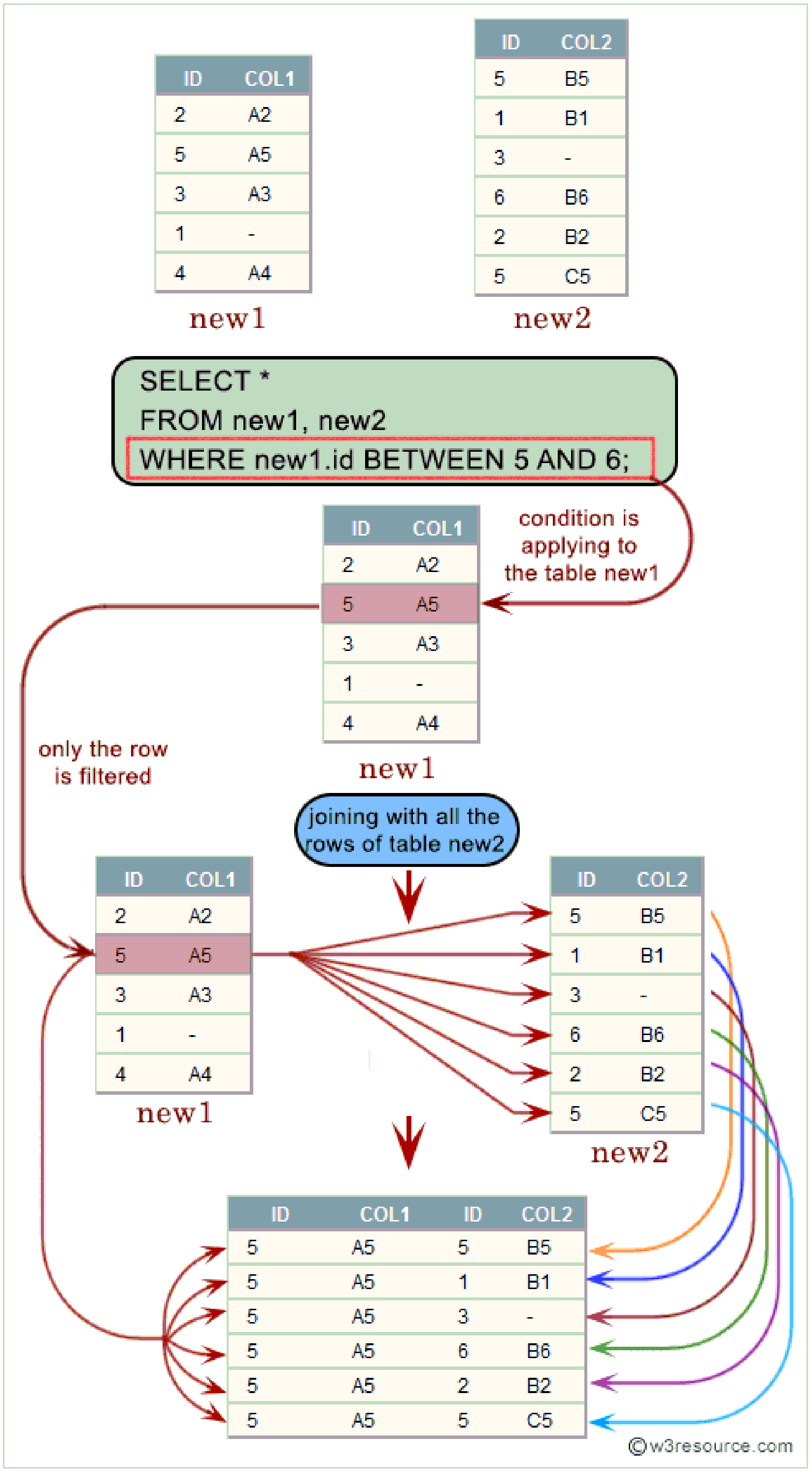
OR

```sql
SELECT column_list
FROM table1, table2
WHERE table1.col_name = table2.col_name;
```



9: NON EQUI JOIN

The SQL NON EQUI JOIN uses comparison operator instead of the equal sign like >, <, >=, <= along with conditions.



new1
new2
condition is applying to the table new1
only the row is filtered
joining with all the rows of table new2
©w3resource.com

new1

ID	COL1
2	A2
5	A5
3	A3
1	-
4	A4

new2

ID	COL2
5	B5
1	B1
3	-
6	B6
2	B2
5	C5

```
SELECT *
FROM new1, new2
WHERE new1.id BETWEEN 5 AND 6;
```

ID	COL1	ID	COL2
5	A5	5	B5
5	A5	1	B1
5	A5	3	-
5	A5	6	B6
5	A5	2	B2
5	A5	5	C5

Syntax:

```
SELECT *
FROM table1, table2
WHERE table1.column [> |  < |  >= | <= ] table2.column;
```

10: SELF JOIN

A self-join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

The self-join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

The syntax of the command for joining a table to itself is almost same as that for joining two different tables. To distinguish the column names from one another, aliases for the actual the table name are used, since both the tables have the same name. Table name aliases are defined in the FROM clause of the SELECT statement.

Syntax:

```
SELECT a.col_name, b.col_name, …
FROM table1 a, table2 b
WHERE a.common_field = b.common_field;
```

Lab Task(s):

Exercise

1. Write a query in SQL to display the first name, last name, department number, and department name for each employee. (**Sample tables:** employees & departments)
2. Write a query to find the name (first_name, last_name), job, department ID and name of the department who works in London. (**Sample tables:** employees , locations & departments)
3. Write a query in SQL to display the first and last name, department, city, and state province for each employee. (**Sample tables:** employees , locations & departments)
4. Write a query to find the employee id, name (last_name) along with their manager_id and name (last_name). (**Sample tables:** employees)
5. Write a query to find the name (first_name, last_name) and hire date of the employees who was hired after 'Jones'. (**Sample tables:** employees)
6. Write a query to get the department name and number of employees in the department. (**Sample tables:** employees & departments)
7. Write a query to display the department ID and name and first name of manager. (**Sample tables:** employees & departments)

8. Write a query to display the department name, manager name, and city. (**Sample tables:** employees , locations & departments)
9. Write a query to display the job history that were done by any employee who is currently drawing more than 10000 of salary. (**Sample tables:** employees & job_history)
10. Write a query to display the first name, last name, hire date, salary of the manager for all managers whose experience is more than 15 years. (**Sample tables:** employees & departments)
11. Write a query in SQL to display the name of the department, average salary and number of employees working in that department who got commission. (**Sample tables:** employees & departments)
12. Write a query in SQL to display the name of the country, city, and the departments which are running there. (**Sample tables:** countries , locations & departments)
13. Write a query in SQL to display department name and the full name (first and last name) of the manager. (**Sample tables:** employees & departments)
14. Write a query in SQL to display the details of jobs which was done by any of the employees who is presently earning a salary on and above 12000. (**Sample tables:** employees & job_history)
15. Write a query in SQL to display the full name (first and last name), and salary of those employees who working in any department located in London. (**Sample tables:** employees , locations & departments)
16. Write a query to display job title, employee name, and the difference between salary of the employee and minimum salary for the job. (**Sample tables:** employees & jobs)
17. Write a query to display the job title and average salary of employees. (**Sample tables:** employees & jobs)
18. Write a query to find the employee ID, job title, number of days between ending date and starting date for all jobs in department 90 from job history. (**Sample tables:** jobs & job_history)

END