# JOINS

Joining (or) Merging one or more tables is called joins.
Joins are used to fetch the data from multiple tables.

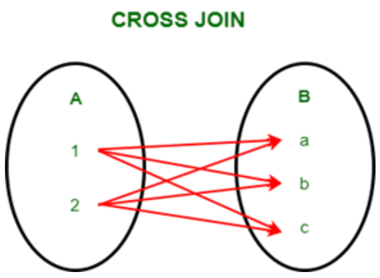
Joins are of 4 types:

1) Cross joins/Cartesian Joins
2)Equi joins/ Inner joins
3)Outer joins
   i. Left Outer Join.
   ii. Right Outer Join.
   iii. Full Outer Join.
4) Self joins
5) Natural joins

## CROSS JOINS/CARTESIAN JOINS:

Cross Join is also called a Cartesian Join as it performs cross product of records of two or more joined tables.
In this type of joins, we will be able to add one (or) two tables.
In this type of joins, each and every record of table1 is going to match with each and every record of table2.

SYNTAX:



SELECT column_name(s)
FROM table1
CROSS JOIN table2;

```
Table: employee
+-------+-------+------------+
| empId | name  | dept       |
+-------+-------+------------+
|     1 | Clark | Sales      |
|     2 | Dave  | Accounting |
|     3 | Ava   | Sales      |
+-------+-------+------------+
```

Table: employee1

```
+--------+-------+-------+
| empId1 | name1 | dept1 |
+--------+-------+-------+
|     11 | PSPK  | JOHNY |
|     22 | SSMB  | SAAHO |
|     33 | AARC  | BADRI |
+--------+-------+-------+
```

```
SELECT * FROM EMPLOYEE
CROSS JOIN EMPLOYEE1;
```

Output:

```
+-------+-------+------------+--------+-------+-------+
| empId | name  | dept       | empId1 | name1 | dept1 |
+-------+-------+------------+--------+-------+-------+
|     3 | Ava   | Sales      |     11 | PSPK  | JOHNY |
|     2 | Dave  | Accounting |     11 | PSPK  | JOHNY |
|     1 | Clark | Sales      |     11 | PSPK  | JOHNY |
|     3 | Ava   | Sales      |     22 | SSMB  | SAAHO |
|     2 | Dave  | Accounting |     22 | SSMB  | SAAHO |
|     1 | Clark | Sales      |     22 | SSMB  | SAAHO |
|     3 | Ava   | Sales      |     33 | AARC  | BADRI |
|     2 | Dave  | Accounting |     33 | AARC  | BADRI |
|     1 | Clark | Sales      |     33 | AARC  | BADRI |
+-------+-------+------------+--------+-------+-------+
```

Examples:

Write a SQL query to combine each row of the salesman table with each row of the customer table.

```
SELECT * FROM SALESMAN
CROSS JOIN CUSTOMER;
```

Output:

```
+-------------+------------+----------+------------+--------------+----------------+-------------+-------+--------------+
| SALESMAN_ID | name       | CITY     | COMMISION  | CUSTOMER_ID  | CUST_NAME      | CITY        | GRADE | SALESMAN_ID  |
+-------------+------------+----------+------------+--------------+----------------+-------------+-------+--------------+
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3002 | NICK RIMANDO   | NEWYORK     |   100 |         5001 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3002 | NICK RIMANDO   | NEWYORK     |   100 |         5001 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3002 | NICK RIMANDO   | NEWYORK     |   100 |         5001 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3002 | NICK RIMANDO   | NEWYORK     |   100 |         5001 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3002 | NICK RIMANDO   | NEWYORK     |   100 |         5001 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3002 | NICK RIMANDO   | NEWYORK     |   100 |         5001 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3007 | BRAD DAVIS     | NEWYORK     |   200 |         5001 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3007 | BRAD DAVIS     | NEWYORK     |   200 |         5001 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3007 | BRAD DAVIS     | NEWYORK     |   200 |         5001 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3007 | BRAD DAVIS     | NEWYORK     |   200 |         5001 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3007 | BRAD DAVIS     | NEWYORK     |   200 |         5001 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3007 | BRAD DAVIS     | NEWYORK     |   200 |         5001 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3005 | GRAHAM ZUSI    | CALIFORNIA  |   200 |         5002 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3005 | GRAHAM ZUSI    | CALIFORNIA  |   200 |         5002 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3005 | GRAHAM ZUSI    | CALIFORNIA  |   200 |         5002 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3005 | GRAHAM ZUSI    | CALIFORNIA  |   200 |         5002 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3005 | GRAHAM ZUSI    | CALIFORNIA  |   200 |         5002 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3005 | GRAHAM ZUSI    | CALIFORNIA  |   200 |         5002 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3008 | JULIAN GREEN   | LONDON      |   300 |         5002 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3008 | JULIAN GREEN   | LONDON      |   300 |         5002 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3008 | JULIAN GREEN   | LONDON      |   300 |         5002 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3008 | JULIAN GREEN   | LONDON      |   300 |         5002 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3008 | JULIAN GREEN   | LONDON      |   300 |         5002 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3008 | JULIAN GREEN   | LONDON      |   300 |         5002 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3004 | FABIAN JOHNSON | PARIS       |   300 |         5006 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3004 | FABIAN JOHNSON | PARIS       |   300 |         5006 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3004 | FABIAN JOHNSON | PARIS       |   300 |         5006 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3004 | FABIAN JOHNSON | PARIS       |   300 |         5006 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3004 | FABIAN JOHNSON | PARIS       |   300 |         5006 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3004 | FABIAN JOHNSON | PARIS       |   300 |         5006 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3009 | GEOFF CAMERON  | BERLIN      |   100 |         5003 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3009 | GEOFF CAMERON  | BERLIN      |   100 |         5003 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3009 | GEOFF CAMERON  | BERLIN      |   100 |         5003 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3009 | GEOFF CAMERON  | BERLIN      |   100 |         5003 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3009 | GEOFF CAMERON  | BERLIN      |   100 |         5003 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3009 | GEOFF CAMERON  | BERLIN      |   100 |         5003 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3003 | JOSY ATLIODOR  | MOSCOW      |   200 |         5007 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3003 | JOSY ATLIODOR  | MOSCOW      |   200 |         5007 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3003 | JOSY ATLIODOR  | MOSCOW      |   200 |         5007 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3003 | JOSY ATLIODOR  | MOSCOW      |   200 |         5007 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3003 | JOSY ATLIODOR  | MOSCOW      |   200 |         5007 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3003 | JOSY ATLIODOR  | MOSCOW      |   200 |         5007 |
|        5003 | LAUSEN HEN | SAN JOSE |       0.12 |         3001 | BRAD GUZAN     | LONDON      |  NULL |         5005 |
|        5007 | PAUL ADAM  | ROME     |       0.13 |         3001 | BRAD GUZAN     | LONDON      |  NULL |         5005 |
|        5006 | MC LYON    | PARIS    |       0.14 |         3001 | BRAD GUZAN     | LONDON      |  NULL |         5005 |
|        5005 | PIT ALEX   | LONDON   |       0.11 |         3001 | BRAD GUZAN     | LONDON      |  NULL |         5005 |
|        5002 | NAIL KNITE | PARIS    |       0.13 |         3001 | BRAD GUZAN     | LONDON      |  NULL |         5005 |
|        5001 | JAMES HOOG | NEWYORK  |       0.15 |         3001 | BRAD GUZAN     | LONDON      |  NULL |         5005 |
+-------------+------------+----------+------------+--------------+----------------+-------------+-------+--------------+
```

## INNER JOINS/EQUI JOINS

- ❖ Whenever we use equijoins, we always get the matched records.
- ❖ In Inner joins, there should be a common column exists between the tables.

- ❖ In inner joins, proper condition should be provided and we use equal(=) operator.

SYNTAX:

**SELECT table1.column1,table1.column2,table2.column1,....**
**FROM table1**
**INNER JOIN table2**
**ON table1.matching_column = table2.matching_column;**

EXAMPLES:

From the following tables write a SQL query to find the salesperson and customer who reside in the same city. Return Salesman, cust_name and city.

```
SELECT SALES.name, CUST.cust_name, CUST.city
FROM salesman SALES
JOIN customer CUST
ON SALES.city = CUST.city;
```

Output:

```
+------------+----------------+---------+
| name       | cust_name      | city    |
+------------+----------------+---------+
| JAMES HOOG | NICK RIMANDO   | NEWYORK |
| JAMES HOOG | BRAD DAVIS     | NEWYORK |
| PIT ALEX   | JULIAN GREEN   | LONDON  |
| MC LYON    | FABIAN JOHNSON | PARIS   |
| NAIL KNITE | FABIAN JOHNSON | PARIS   |
| PIT ALEX   | BRAD GUZAN     | LONDON  |
+------------+----------------+---------+
```

## OUTERJOINS

To get the matched and unmatched records from both the tables,we will use outer joins.
Outer joins are of 3 types. 1) Left Outer Joins
2) Right Outer Joins
3) Full outer Joins

**LEFT OUTER JOINS:**

In this type of joins we will be able to get all the records of left side table and only matched records from the right-side table.

SYNTAX:

**SELECT table1.column1,table1.column2,table2.column1,....**
**FROM table1**
**LEFT OUTER JOIN table2**
**ON table1.matching_column = table2.matching_column;**

EXAMPLES:

From the following tables write a SQL query to find those customers with a grade less than 300. Return cust_name, customer city, grade, Salesman, salesmancity. The result should be ordered by ascending customer_id.

-- (Selecting specific columns from the 'customer' and 'salesman' tables)

```
SELECT CUST.cust_name, CUST.city, CUST.grade,
       SALES.name , SALES.city
FROM customer CUST
```

-- ( Performing a left outer join based on the salesman_id, including unmatched rows from 'customer')

```
LEFT OUTER JOIN salesman SALES
ON CUST.salesman_id = SALES.salesman_id
```

--( Filtering the results based on the condition that 'grade' is less than 300 )

```
WHERE CUST.grade < 300
```

--( Sorting the result set by customer_id in ascending order )

```
ORDER BY CUST.customer_id;
```

Output:

```
+----------------+------------+-------+------------+----------+
| cust_name      | city       | grade | name       | city     |
+----------------+------------+-------+------------+----------+
| NICK RIMANDO   | NEWYORK    |   100 | JAMES HOOG | NEWYORK  |
| JOSY ATLIODOR  | MOSCOW     |   200 | PAUL ADAM  | ROME     |
| GRAHAM ZUSI    | CALIFORNIA |   200 | NAIL KNITE | PARIS    |
| BRAD DAVIS     | NEWYORK    |   200 | JAMES HOOG | NEWYORK  |
| GEOFF CAMERON  | BERLIN     |   100 | LAUSEN HEN | SAN JOSE |
+----------------+------------+-------+------------+----------+
```

## <u>RIGHT OUTER JOIN</u>

In this type of joins we will be able to get all the records of Right side table and only matched records from the Left side table.

<u>SYNTAX:</u>

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT OUTER JOIN table2
ON table1.matching_column = table2.matching_column;
```

EXAMPLES:

Write a SQL statement to generate a list in ascending order of salespersons who work either for one or more customers or have not yet joined any of the customers.

-- (Selecting specific columns and renaming them for clarity)

```
SELECT CUST.cust_name, CUST.city, CUST.grade,
       SALES.name , SALES.city
```

-- (Specifying the tables to retrieve data from ('customer' as 'CUST' and 'salesman' as 'SALES'))

```
FROM customer CUST
```

-- (Performing a right outer join based on the salesman_id, including unmatched rows from 'salesman')

```
RIGHT OUTER JOIN salesman SALES
ON SALES.salesman_id = CUST.salesman_id
```

--( Sorting the result set by salesman_id in ascending order)

```
ORDER BY SALES.salesman_id;
```

Output:

```
+----------------+------------+-------+------------+----------+
| cust_name      | city       | grade | name       | city     |
+----------------+------------+-------+------------+----------+
| BRAD DAVIS     | NEWYORK    |   200 | JAMES HOOG | NEWYORK  |
| NICK RIMANDO   | NEWYORK    |   100 | JAMES HOOG | NEWYORK  |
| JULIAN GREEN   | LONDON     |   300 | NAIL KNITE | PARIS    |
| GRAHAM ZUSI    | CALIFORNIA |   200 | NAIL KNITE | PARIS    |
| GEOFF CAMERON  | BERLIN     |   100 | LAUSEN HEN | SAN JOSE |
| BRAD GUZAN     | LONDON     |  NULL | PIT ALEX   | LONDON   |
| FABIAN JOHNSON | PARIS      |   300 | MC LYON    | PARIS    |
| JOSY ATLIODOR  | MOSCOW     |   200 | PAUL ADAM  | ROME     |
+----------------+------------+-------+------------+----------+
```

## FULL OUTER JOINS:

In this type of joins,we will get the Matched and Unmatched records from both the tables.
SQL full outer join is used to combine the result of both left and right outer join and returns all rows (don't care its matched or unmatched) from the both participating tables.

SYNTAX:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
FULL OUTER JOIN table2
ON table1.matching_column = table2.matching_column;
```

## SELF-JOINS:

Joining the table by itself is called self joins.
Without aliasing the names, we cannot achieve self joins.
It basically allows us to combine the rows from the same table based on some specific conditions.

SYNTAX:

**SELECT column_name(s)**
**FROM table1 T1, table1 T2**
**WHERE condition;**

T1 and T2 are different table aliases for the same table.

(OR)

**SELECT * FROM TABLE1 T1,TABLE1 T2**
**WHERE T1.COMMONCOLUMN=T2.COMMONCOLUMN;**

TABLE: EMPLOYEE

```
+-------+-------+-----+
| EMPNO | ENAME | MGR |
+-------+-------+-----+
|     1 | A     | 3   |
|     2 | B     | 2   |
|     3 | C     | 4   |
|     4 | D     | 1   |
+-------+-------+-----+
```

EXAMPLES:

```
SELECT * FROM EMPLOYEE E , EMPLOYEE I
WHERE E.MGR=I.EMPNO;
```

Output:

```
+-------+-------+-----+-------+-------+-----+
| EMPNO | ENAME | MGR | EMPNO | ENAME | MGR |
+-------+-------+-----+-------+-------+-----+
|     1 | A     | 3   |     3 | C     | 4   |
|     2 | B     | 2   |     2 | B     | 2   |
|     3 | C     | 4   |     4 | D     | 1   |
|     4 | D     | 1   |     1 | A     | 3   |
+-------+-------+-----+-------+-------+-----+
```

```
SELECT * FROM EMPLOYEE E , EMPLOYEE I
WHERE E.MGR=I.MGR;
```

OUTPUT:

```
+-------+-------+-----+-------+-------+-----+
| EMPNO | ENAME | MGR | EMPNO | ENAME | MGR |
+-------+-------+-----+-------+-------+-----+
|     1 | A     | 3   |     1 | A     | 3   |
|     2 | B     | 2   |     2 | B     | 2   |
|     3 | C     | 4   |     3 | C     | 4   |
|     4 | D     | 1   |     4 | D     | 1   |
+-------+-------+-----+-------+-------+-----+
```

## NATURAL JOINS

Natural join is an SQL join operation that creates a join on the base of the common columns in the tables.
To perform natural join there must be one common attribute(Column) between two tables.
 Natural join will retrieve from multiple relations. It works in three steps.

SYNTAX:

**SELECT * FROM TABLE1**
**NATURAL JOIN  TABLE2;**

Write a SQL statement to join the tables salesman, customer and orders so that the same column of each table appears once and only the relational rows are returned.

```
SELECT *
FROM orders
NATURAL JOIN customer
NATURAL JOIN salesman;
```

Output:

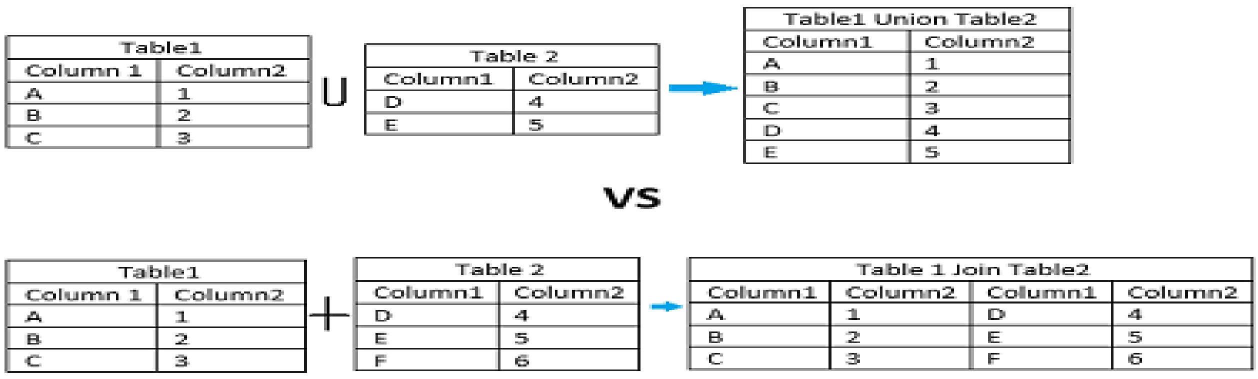
```
+-------------+---------+-------------+--------+-----------+------------+--------------+-------+------------+-----------+
| SALESMAN_ID | CITY    | CUSTOMER_ID | ORD_NO | PURCH_AMT | ORD_DATE   | CUST_NAME    | GRADE | name       | COMMISION |
+-------------+---------+-------------+--------+-----------+------------+--------------+-------+------------+-----------+
|        5001 | NEWYORK |        3002 |  70008 |      5760 | 2012-09-10 | NICK RIMANDO |   100 | JAMES HOOG |      0.15 |
|        5001 | NEWYORK |        3002 |  70002 |     65.26 | 2012-10-05 | NICK RIMANDO |   100 | JAMES HOOG |      0.15 |
|        5001 | NEWYORK |        3007 |  70005 |    2400.6 | 2012-07-27 | BRAD DAVIS   |   200 | JAMES HOOG |      0.15 |
|        5005 | LONDON  |        3001 |  70009 |    270.65 | 2012-09-10 | BRAD GUZAN   |  NULL | PIT ALEX   |      0.11 |
+-------------+---------+-------------+--------+-----------+------------+--------------+-------+------------+-----------+
```

| **NATURAL JOIN** | **INNER JOIN** |
|---|---|
| Natural Join joins two tables based on same attribute name and datatypes. | Inner Join joins two table on the basis of the column which is explicitly specified in the ON clause. |
| In Natural Join, The resulting table will contain all the attributes of both the tables but keep only one copy of each common column | In Inner Join, The resulting table will contain all the attribute of both the tables including duplicate columns also |
| In Natural Join, If there is no condition specifies then it returns the rows based on the common column | In Inner Join, only those records will return which exists in both the tables |
| SYNTAX: SELECT * <br> FROM table1 NATURAL JOIN table2; | SYNTAX: SELECT * <br> FROM table1 INNER JOIN table2 ON table1.Column_Name= table2.Column_Name; |

## UNION

The union operator is used to combine the result-set of two or more selected statements.

Each SELECT statement within UNION must have the same number of columns.
The columns must also have similar datatypes.
The columns in each SELECT statement must also be in the same order.



| Table1 | |
|---|---|
| Column 1 | Column2 |
| A | 1 |
| B | 2 |
| C | 3 |

U

| Table 2 | |
|---|---|
| Column1 | Column2 |
| D | 4 |
| E | 5 |

| Table1 Union Table2 | |
|---|---|
| Column1 | Column2 |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |
| E | 5 |

VS

| Table1 | |
|---|---|
| Column 1 | Column2 |
| A | 1 |
| B | 2 |
| C | 3 |

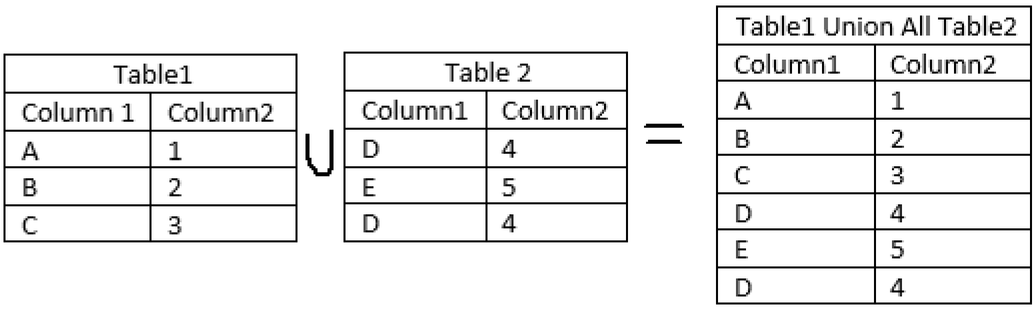
+

| Table 2 | |
|---|---|
| Column1 | Column2 |
| D | 4 |
| E | 5 |
| F | 6 |

| Table 1 Join Table2 | | | |
|---|---|---|---|
| Column1 | Column2 | Column1 | Column2 |
| A | 1 | D | 4 |
| B | 2 | E | 5 |
| C | 3 | F | 6 |

**UNION ALL**

The UNION ALL operator combines two or more results from multiple SELECT queries and returns all records into a single result set.
It does not remove the duplicate rows from the output of the SELECT statements. i.e.. it allows duplicate values also.



| Table1 | |
|---|---|
| Column 1 | Column2 |
| A | 1 |
| B | 2 |
| C | 3 |

U

| Table 2 | |
|---|---|
| Column1 | Column2 |
| D | 4 |
| E | 5 |
| D | 4 |

=

| Table1 Union All Table2 | |
|---|---|
| Column1 | Column2 |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |
| E | 5 |
| D | 4 |

EXAMPLES:

```
select city,name,SALESMAN_ID from SALESMAN
union
select city,CUST_NAME,SALESMAN_ID from CUSTOMER;
```

Output:

```
+------------+----------------+-------------+
| city       | name           | SALESMAN_ID |
+------------+----------------+-------------+
| NEWYORK    | JAMES HOOG     |        5001 |
| PARIS      | NAIL KNITE     |        5002 |
| LONDON     | PIT ALEX       |        5005 |
| PARIS      | MC LYON        |        5006 |
| ROME       | PAUL ADAM      |        5007 |
| SAN JOSE   | LAUSEN HEN     |        5003 |
| NEWYORK    | NICK RIMANDO   |        5001 |
| NEWYORK    | BRAD DAVIS     |        5001 |
| CALIFORNIA | GRAHAM ZUSI    |        5002 |
| LONDON     | JULIAN GREEN   |        5002 |
| PARIS      | FABIAN JOHNSON |        5006 |
| BERLIN     | GEOFF CAMERON  |        5003 |
| MOSCOW     | JOSY ATLIODOR  |        5007 |
| LONDON     | BRAD GUZAN     |        5005 |
```

| | | |
|---|---|---|

Ex2: From the following tables, write a SQL query to find distinct salespeople and their cities. Return salesperson ID and city.

```
SELECT salesman_id, city
FROM customer
UNION
(SELECT salesman_id, city
FROM salesman)
```

Output:

```
+-------------+------------+
| salesman_id | city       |
+-------------+------------+
|        5001 | NEWYORK    |
|        5002 | CALIFORNIA |
|        5002 | LONDON     |
|        5006 | PARIS      |
|        5003 | BERLIN     |
|        5007 | MOSCOW     |
|        5005 | LONDON     |
|        5002 | PARIS      |
|        5007 | ROME       |
|        5003 | SAN JOSE   |
+-------------+------------+
```

If we use UNION ALL we will get the duplicate values also.

```
SELECT salesman_id, city
FROM customer
UNION ALL
(SELECT salesman_id, city
FROM salesman)
```

Output:

```
+-------------+------------+
| salesman_id | city       |
+-------------+------------+
|        5001 | NEWYORK    |
|        5001 | NEWYORK    |
|        5002 | CALIFORNIA |
|        5002 | LONDON     |
|        5006 | PARIS      |
|        5003 | BERLIN     |
|        5007 | MOSCOW     |
|        5005 | LONDON     |
|        5001 | NEWYORK    |
|        5002 | PARIS      |
|        5005 | LONDON     |
|        5006 | PARIS      |
|        5007 | ROME       |
|        5003 | SAN JOSE   |
```

| | |
|---|---|

## <u>DIFFERENCES BETWEEN UNION AND JOIN</u>

** join is used to combine rows from two or more tables based on a common column.
**UNION is used to combine the result sets of two or more SELECT statements into a single result set, removing duplicates by default.
   ** the number and order of columns in all SELECT statements must be the same.
Join

Table—employee

| ID | name | salary |
|---|---|---|
| 1 | John | 2000 |
| 2 | Lisa | 3000 |
| 3 | david | 4000 |

table--department

| id | Name |
|---|---|
| 1 | It |
| 2 | finance |

```
SELECT Employee.name,Departments.Name
FROM Employee
JOIN Departments ON
Employees.id=department.id;
```

Output:

| Name | Name |
|---|---|
| John | It |
| lisa | finance |

## UNION

Table-1

| id | name |
|---|---|
| 1 | john |
| 2 | emma |

Table_2

| id | name |
|---|---|
| 1 | john |
| 3 | olivia |

```
SELECT * FROM TABLE1
UNION
SELECT * FROM TABLE2;
```

OUTPUT:

| ID | NAME |
|---|---|
| 1 | John |
| 2 | Emma |
| 3 | olivia |