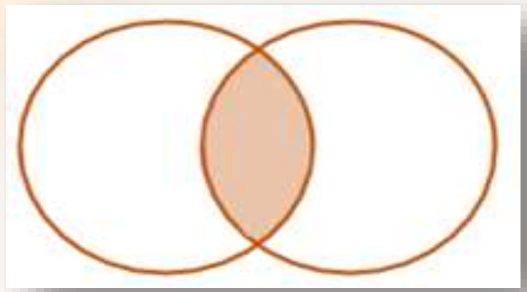
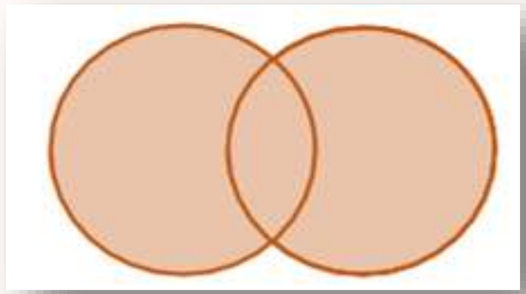
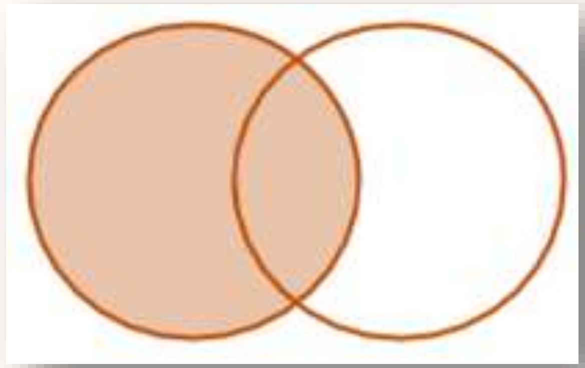
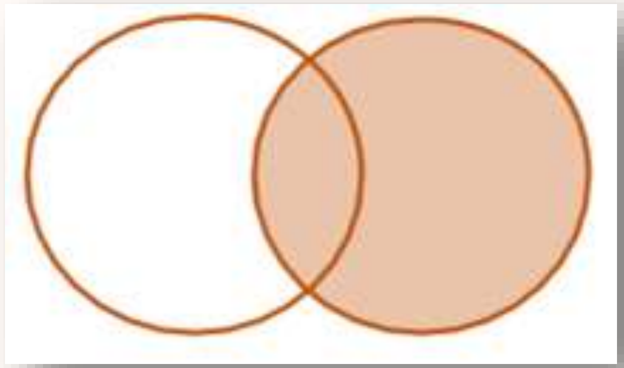





MASTERING JOINS IN MYSQL

Presented by Gagandeep Kaur Bhatti | Data Analyst

❑ *Agenda*

- **What are SQL Joins?**
  Used to combine data from multiple tables based on related columns.

- **When to Use SQL Joins**
  When data is stored across different tables and needs to be analyzed together.

- **Ways to Combine Data in SQL**
  a. **Row-wise:** Using **SET operations** (APPEND)
  b. **Column-wise:** Using **JOINS**

- **Combining Data Using Joins**
  a. INNER JOIN
  b. LEFT JOIN
  c. RIGHT JOIN
  d. FULL JOIN
  e. SELF JOIN
  f. CROSS JOIN

- **Most Frequently Used Joins**
  INNER JOIN and LEFT JOIN

- **Practice Dataset**
  CodeBasics **Movies dataset** used to practice joins and multiple joins.

Understanding SQL JOIN

SQL JOIN is used to combine data from two or more tables based on a related column between them. It helps us get meaningful information by bringing data together from different tables.

For example:
If one table has customer details and another has order details, we can use a JOIN to see which customer placed which order.

When to Use SQL JOINs

We use SQL JOINs when we need to fetch related data from multiple tables.

For example:

- One table has student information, and another has marks.
- If we want to know each student's marks, we need to JOIN both tables using a common column like StudentID.

Ways to Combine Data in SQL

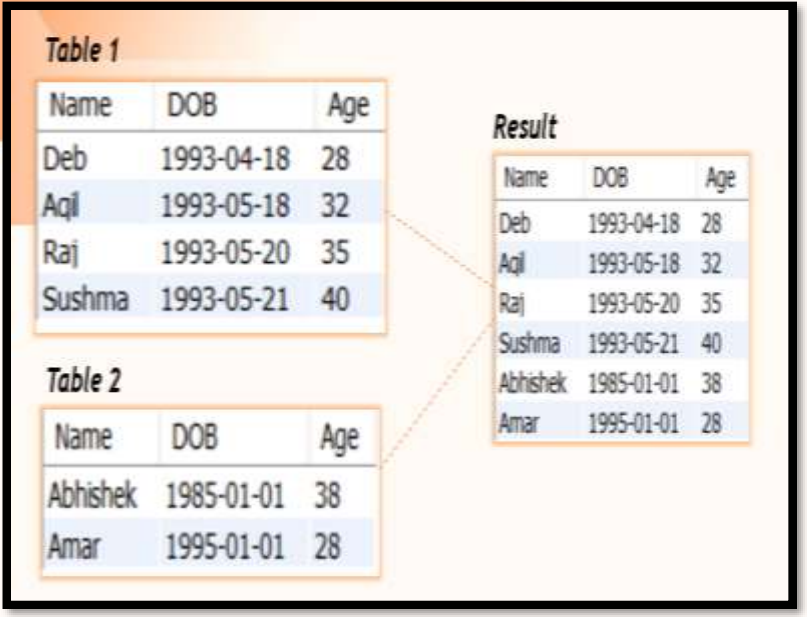


Table 1

Name	DOB	Age
Deb	1993-04-18	28
Aqil	1993-05-18	32
Raj	1993-05-20	35
Sushma	1993-05-21	40

Table 2

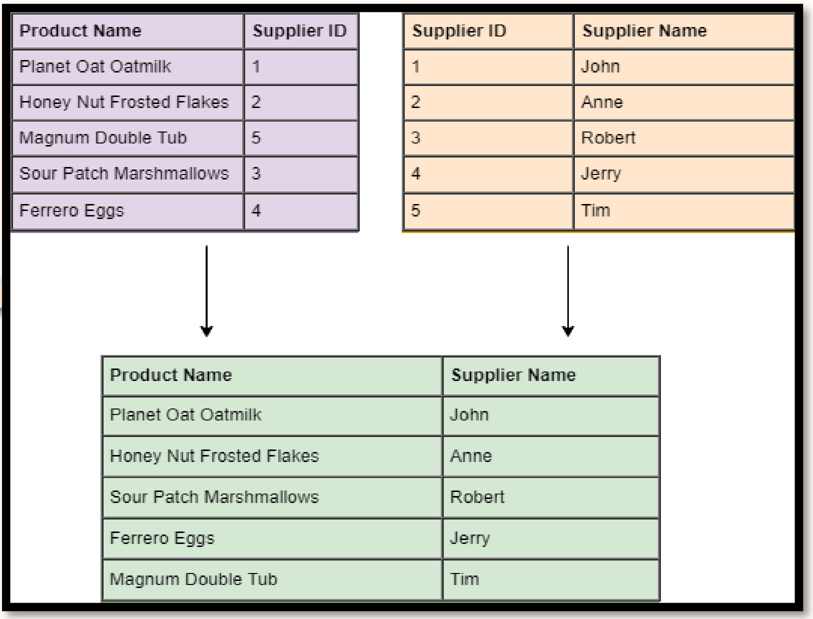
Name	DOB	Age
Abhishek	1985-01-01	38
Amar	1995-01-01	28

Result

Name	DOB	Age
Deb	1993-04-18	28
Aqil	1993-05-18	32
Raj	1993-05-20	35
Sushma	1993-05-21	40
Abhishek	1985-01-01	38
Amar	1995-01-01	28

Row-Wise / Append

Column Wise / Joins



Product Name	Supplier ID
Planet Oat Oatmilk	1
Honey Nut Frosted Flakes	2
Magnum Double Tub	5
Sour Patch Marshmallows	3
Ferrero Eggs	4

Supplier ID	Supplier Name
1	John
2	Anne
3	Robert
4	Jerry
5	Tim

Product Name	Supplier Name
Planet Oat Oatmilk	John
Honey Nut Frosted Flakes	Anne
Sour Patch Marshmallows	Robert
Ferrero Eggs	Jerry
Magnum Double Tub	Tim

APPEND

JOIN

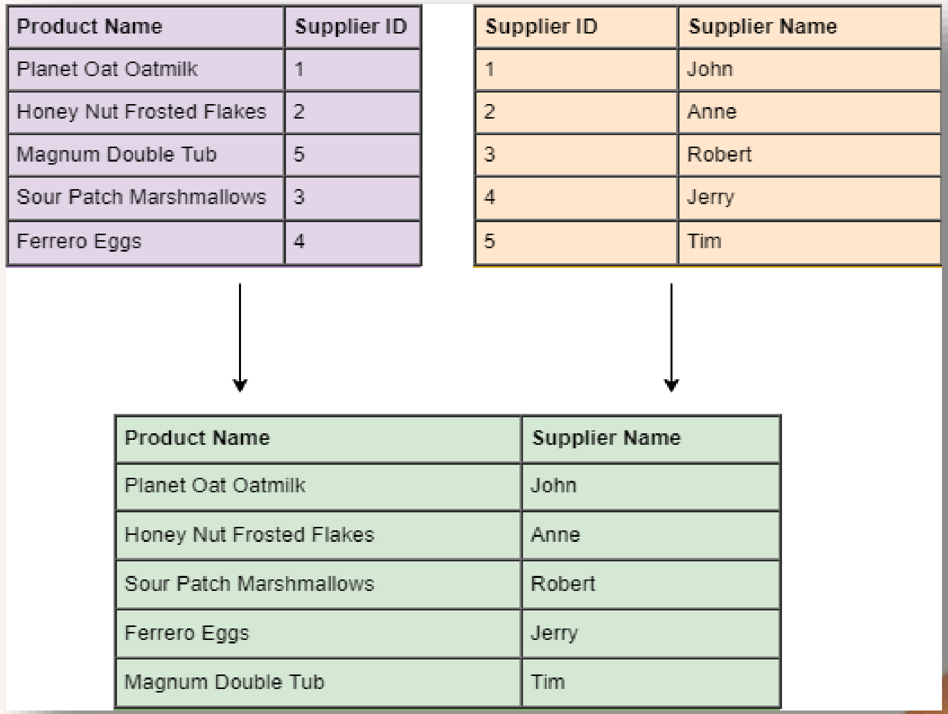
NOTE:- In this section, we will focus only on column-wise data combination using SQL Joins.

Combining Data Using Joins

A SQL join combines rows from **two or more tables based on a related column**, allowing you to retrieve connected or complementary data in a single result.

JOIN → Horizontal

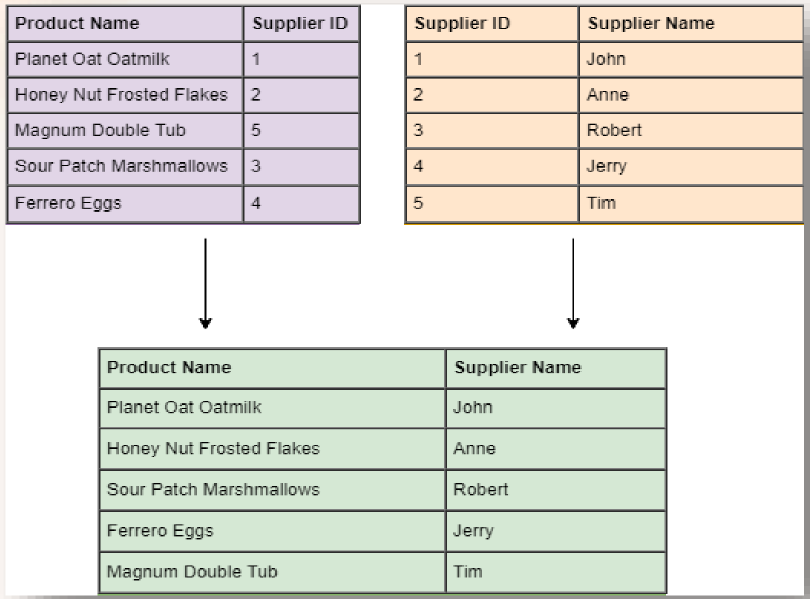
- Needs at least **one common column**
- Column names can differ, but **values must match**
- Often used when two tables store **linked information**



Product Name	Supplier ID
Planet Oat Oatmilk	1
Honey Nut Frosted Flakes	2
Magnum Double Tub	5
Sour Patch Marshmallows	3
Ferrero Eggs	4

Supplier ID	Supplier Name
1	John
2	Anne
3	Robert
4	Jerry
5	Tim

Product Name	Supplier Name
Planet Oat Oatmilk	John
Honey Nut Frosted Flakes	Anne
Sour Patch Marshmallows	Robert
Ferrero Eggs	Jerry
Magnum Double Tub	Tim



Product Name	Supplier ID
Planet Oat Oatmilk	1
Honey Nut Frosted Flakes	2
Magnum Double Tub	5
Sour Patch Marshmallows	3
Ferrero Eggs	4

Supplier ID	Supplier Name
1	John
2	Anne
3	Robert
4	Jerry
5	Tim

Product Name	Supplier Name
Planet Oat Oatmilk	John
Honey Nut Frosted Flakes	Anne
Sour Patch Marshmallows	Robert
Ferrero Eggs	Jerry
Magnum Double Tub	Tim

1. ***Works by Matching Columns (Keys or Common Fields)***
   Column-wise joins combine **columns from different tables** based on a common key (e.g., Order ID, Seller). This allows you to bring in **extra information** across tables.

2. ***Doesn't Require Same Columns or Order***
   Unlike set operators (UNION, INTERSECT), **tables can have different columns.** Only join key must be common, like Order ID or Seller.

3. ***Columns Can Be Selected as Needed***
   You don't need to include all columns — just pick what's needed:
   ✓ Order ID, Seller

Column-wise / Joins (Join Types)

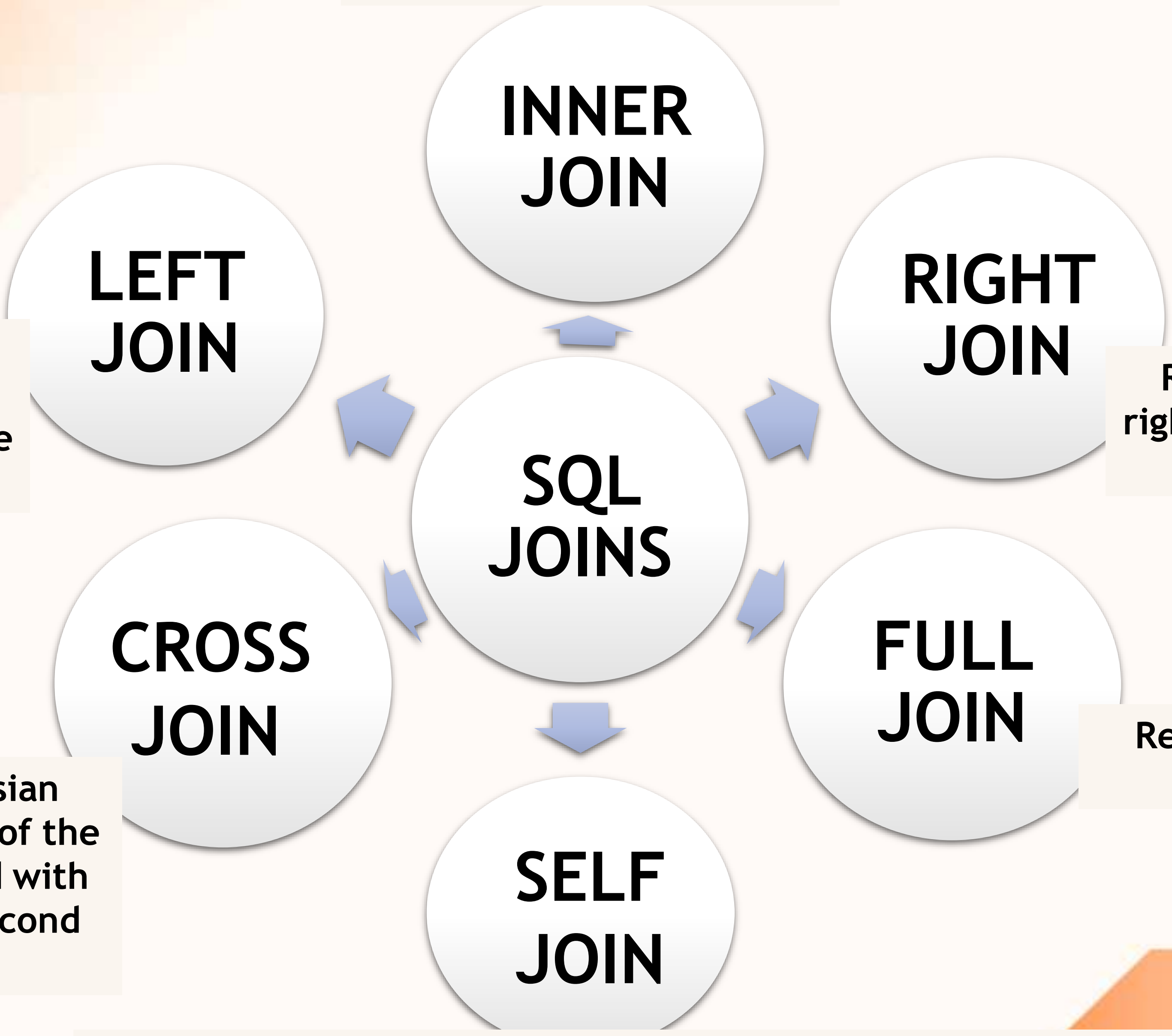
Column-wise joining means adding columns from one table to another by matching a common key.

This is done using Join Types:

Join Types:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN
- CROSS JOIN
- SELF JOIN

FULL JOIN / FULL OUTER JOIN are **not directly supported in MySQL**.
It can be achieved using alternative approaches such as **JOINs, UNION, NOT EXISTS, or subqueries.**



Returns only the matching rows from both tables

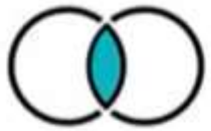
Returns all rows from the left table and matching rows from the right table.

Returns all rows from the right table and matching rows from the left table.

Returns all rows from both tables.

Returns the Cartesian product - every row of the first table combined with every row of the second table

Returns all rows from the same table by joining it to itself using aliases based on a related column.



Inner join *(Matching Records Only)*

INNER JOIN returns **only the rows with matching values in both tables** based on the specified condition.

Syntax :-

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

Syntax :-

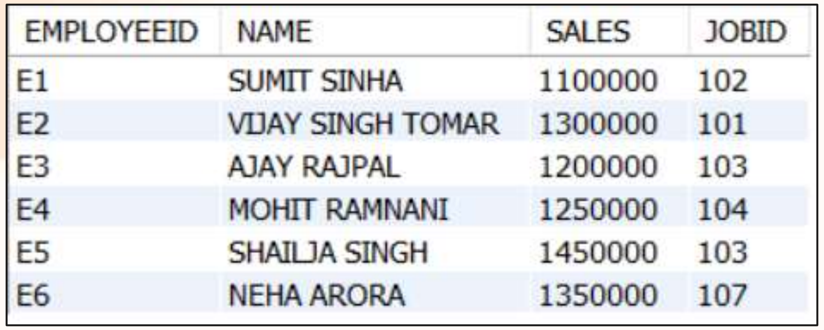
```
SELECT table1.column1,table2.column2
FROM table1
INNER JOIN table2
USING(common_column);
```

Use ON when:
- The **column names are different** in the two tables.
- You need more flexibility in the join condition.

Use USING when:
- **The column name is the same in both tables.**
- **You want a cleaner, simpler join syntax.**

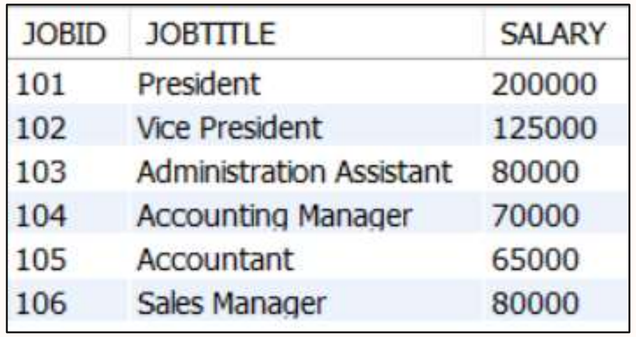
❖ These are our EMPLOYEE and JOB tables, which we will use to perform INNER JOIN.

EMPLOYEE TABLE



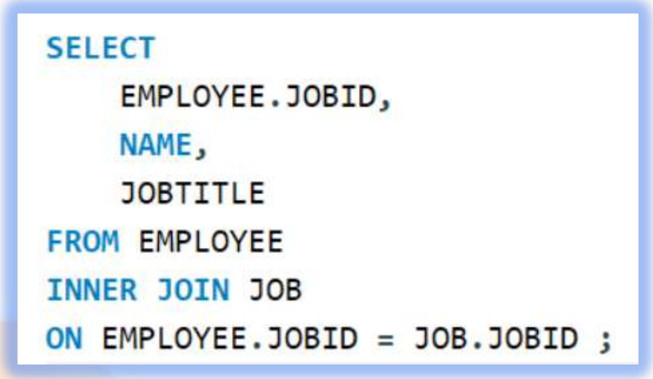
EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOB TABLE

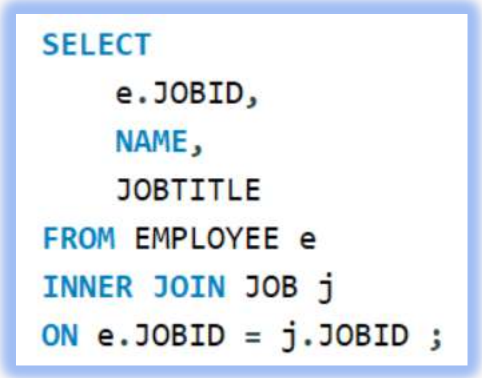


JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

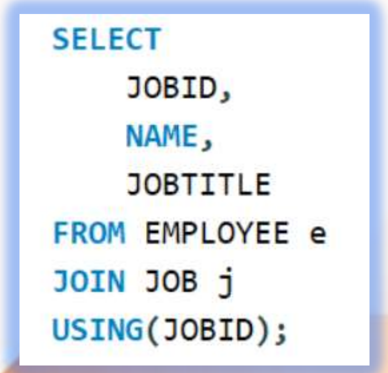
-- Write an SQL query using **INNER JOIN** to display the JOBID, NAME (from the EMPLOYEE table), and JOBTITLE (from the JOB table) by combining data from both tables where the JOBID matches.



```
SELECT
    EMPLOYEE.JOBID,
    NAME,
    JOBTITLE
FROM EMPLOYEE
INNER JOIN JOB
ON EMPLOYEE.JOBID = JOB.JOBID ;
```

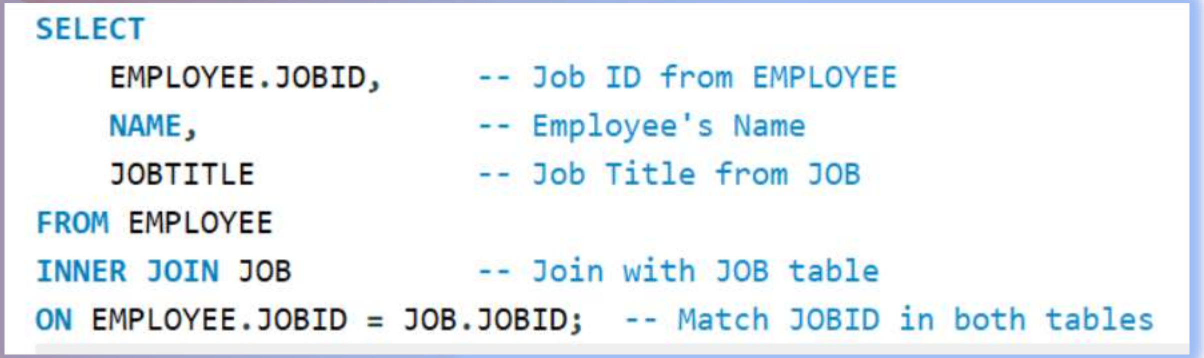


```
SELECT
    e.JOBID,
    NAME,
    JOBTITLE
FROM EMPLOYEE e
INNER JOIN JOB j
ON e.JOBID = j.JOBID ;
```



```
SELECT
    JOBID,
    NAME,
    JOBTITLE
FROM EMPLOYEE e
JOIN JOB j
USING(JOBID);
```

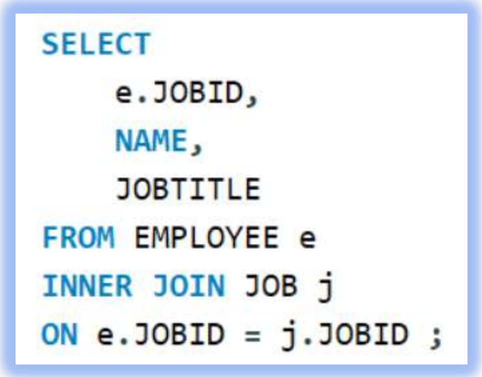
All of these queries will produce the same result.



```sql
SELECT
    EMPLOYEE.JOBID,      -- Job ID from EMPLOYEE
    NAME,                -- Employee's Name
    JOBTITLE             -- Job Title from JOB
FROM EMPLOYEE
INNER JOIN JOB           -- Join with JOB table
ON EMPLOYEE.JOBID = JOB.JOBID;  -- Match JOBID in both tables
```

Taking Full Name like <u>EMPLOYEE.JOBID</u>

- Use **table name** with column (EMPLOYEE.JOBID) when:
  - You're using **multiple tables**.
  - To **avoid confusion** if column names are the same.
- It's called **fully qualified name**.



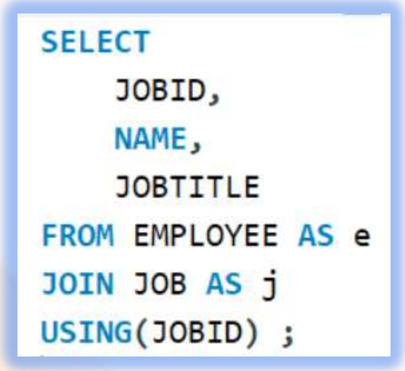
```sql
SELECT
    e.JOBID,
    NAME,
    JOBTITLE
FROM EMPLOYEE e
INNER JOIN JOB j
ON e.JOBID = j.JOBID ;
```

<u>Using Alias (AS) or Not</u>

- AS is **optional**:
  - EMPLOYEE AS e (explicit alias)
  - EMPLOYEE e (implicit alias)
- Both are correct. Just a **style choice**.

<u>INNER JOIN vs No Keyword (Default Join)</u>

- Whether you write INNER JOIN or simply JOIN, the result will be the same — both perform an inner join by default.

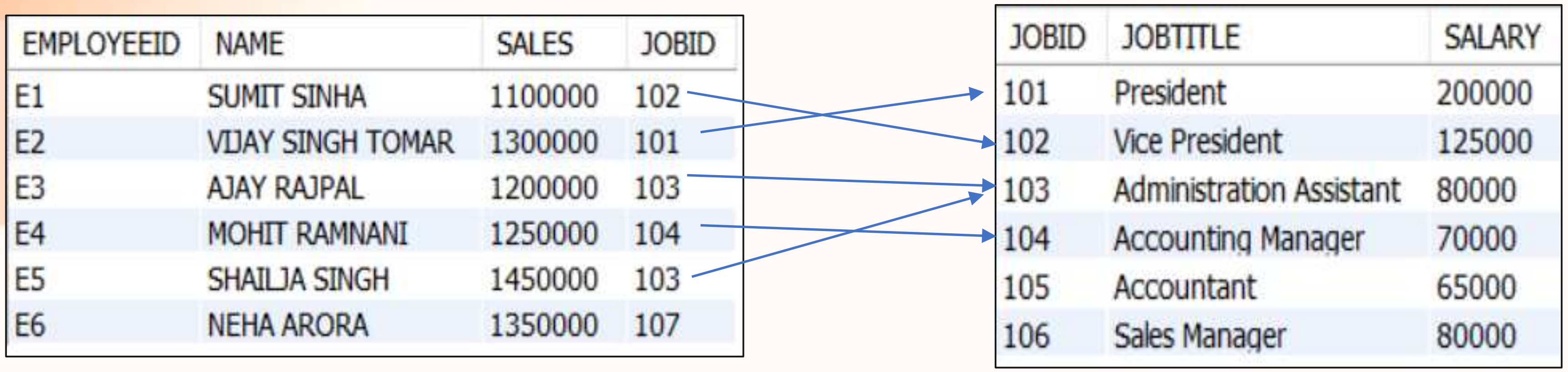


```sql
SELECT
    JOBID,
    NAME,
    JOBTITLE
FROM EMPLOYEE AS e
JOIN JOB AS j
USING(JOBID) ;
```

<u>Difference Between ON and USING</u>

- ON is used when the column names in both tables are **different** or when you want more control in join conditions.
- USING is used when the column name is **the same** in both tables. It makes the query shorter and cleaner.

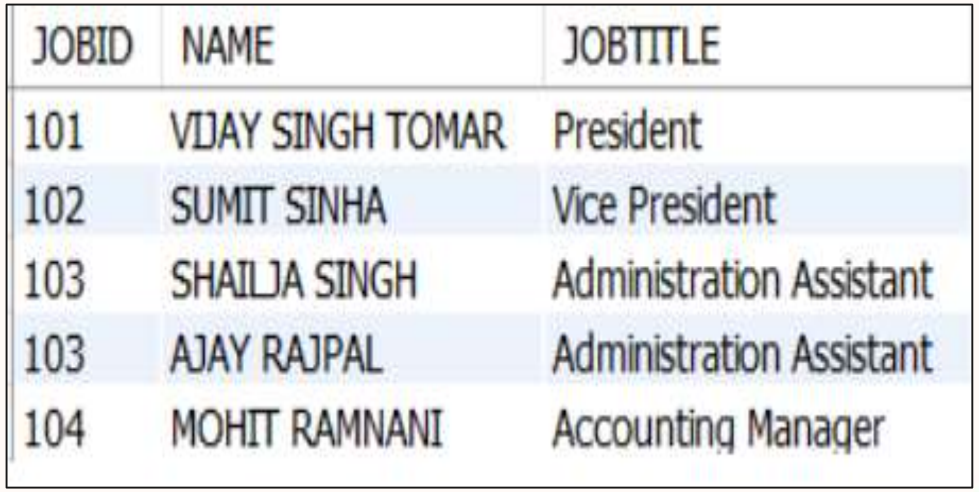
It returns only the rows where the values match in both tables.



EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

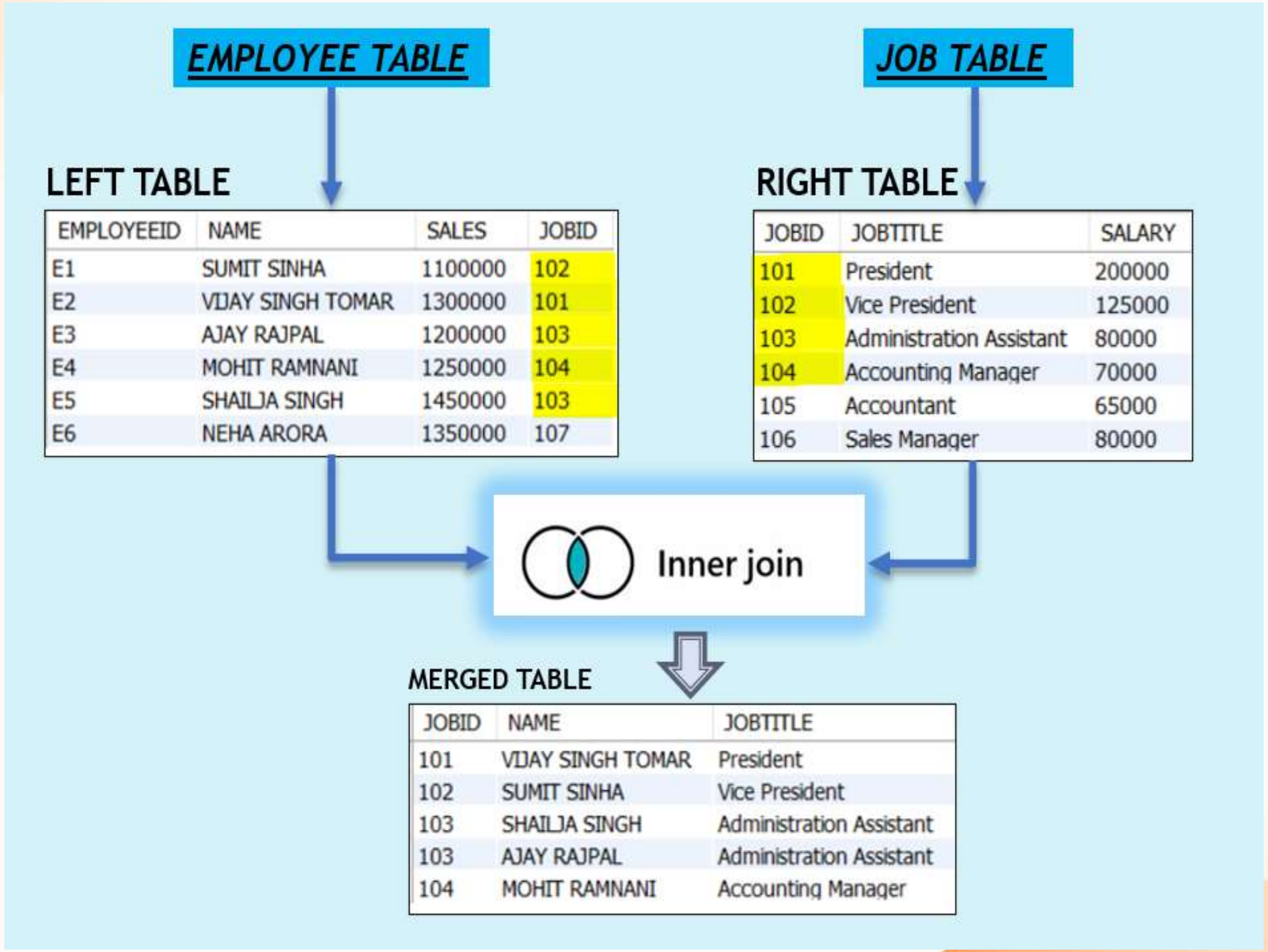
JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

After performing the INNER JOIN, we get the following result.



JOBID	NAME	JOBTITLE
101	VIJAY SINGH TOMAR	President
102	SUMIT SINHA	Vice President
103	SHAILJA SINGH	Administration Assistant
103	AJAY RAJPAL	Administration Assistant
104	MOHIT RAMNANI	Accounting Manager

RESULT :-



EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

JOBID	NAME	JOBTITLE
101	VIJAY SINGH TOMAR	President
102	SUMIT SINHA	Vice President
103	SHAILJA SINGH	Administration Assistant
103	AJAY RAJPAL	Administration Assistant
104	MOHIT RAMNANI	Accounting Manager

Left Join OR Left outer join

(All Left Table Records & Matching Right Table Records)

LEFT JOIN returns all records from the **left table**, and the **matched records** from the **right table**.
Both **LEFT JOIN** and **LEFT OUTER JOIN** mean the same thing.
OUTER is optional and commonly omitted.

Syntax :-

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

Syntax :-

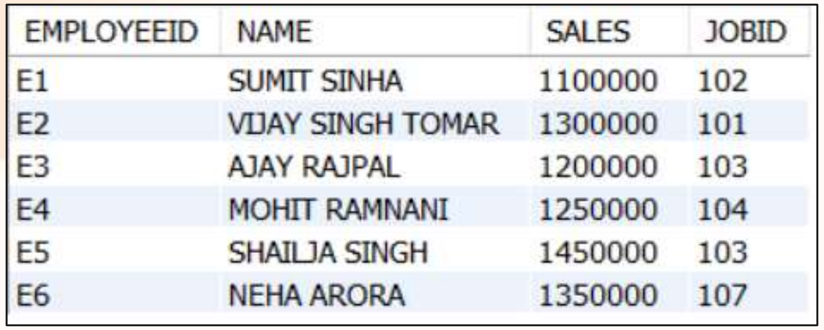
```
SELECT table1.column1, table2.column2
FROM table1
LEFT JOIN table2
USING(common_column);
```

Use ON when:
- The **column names are different** in the two tables.
- You need more flexibility in the join condition.

Use USING when:
- **The column name is the same in both tables.**
- **You want a cleaner, simpler join syntax.**

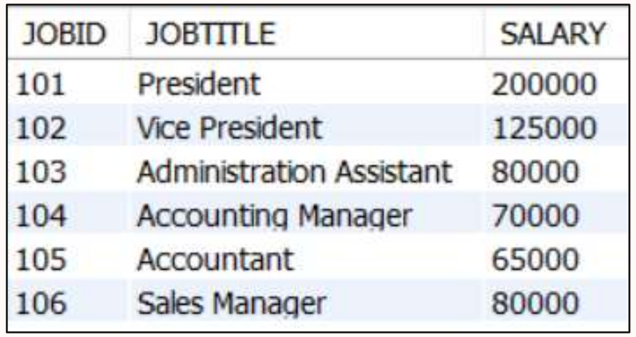
❖ These are our EMPLOYEE and JOB tables, which we will use to perform LEFTJOIN.

EMPLOYEE TABLE



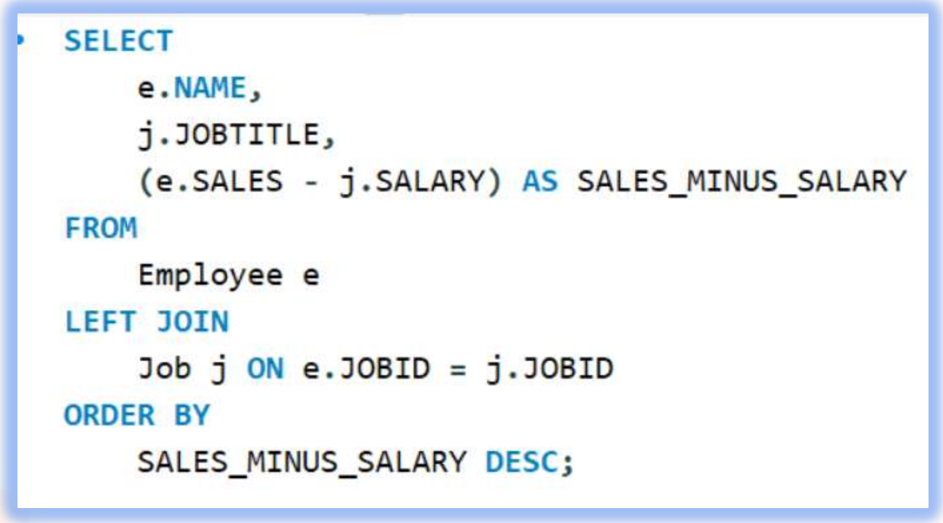
EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOB TABLE

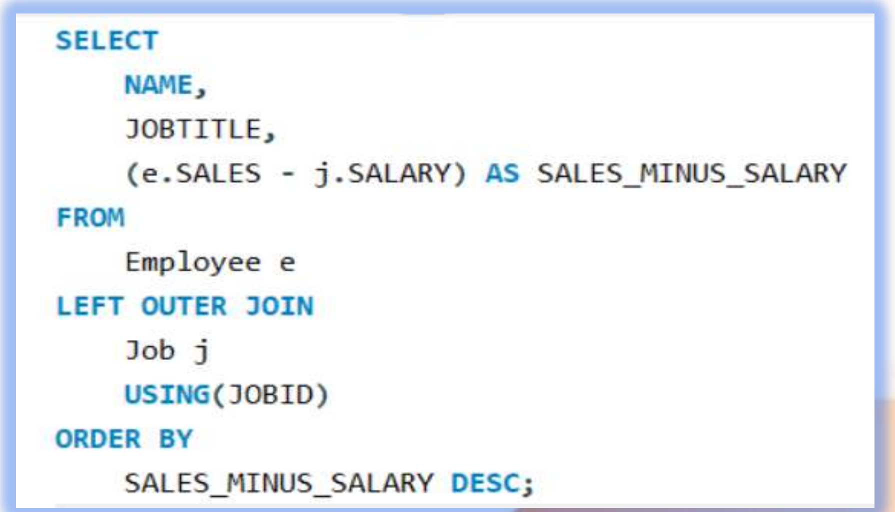


JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

-- List each employee's name, job title, and the difference between their sales and salary.
Sort the result by this difference in descending order.



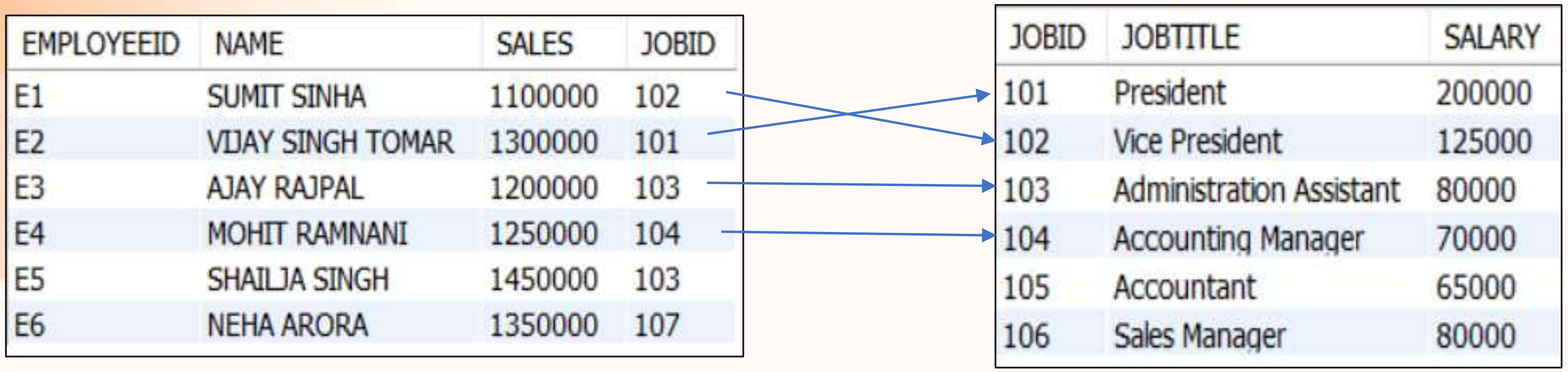
```
SELECT
    e.NAME,
    j.JOBTITLE,
    (e.SALES - j.SALARY) AS SALES_MINUS_SALARY
FROM
    Employee e
LEFT JOIN
    Job j ON e.JOBID = j.JOBID
ORDER BY
    SALES_MINUS_SALARY DESC;
```



```
SELECT
    NAME,
    JOBTITLE,
    (e.SALES - j.SALARY) AS SALES_MINUS_SALARY
FROM
    Employee e
LEFT OUTER JOIN
    Job j
    USING(JOBID)
ORDER BY
    SALES_MINUS_SALARY DESC;
```

Both of these queries will produce the same result.

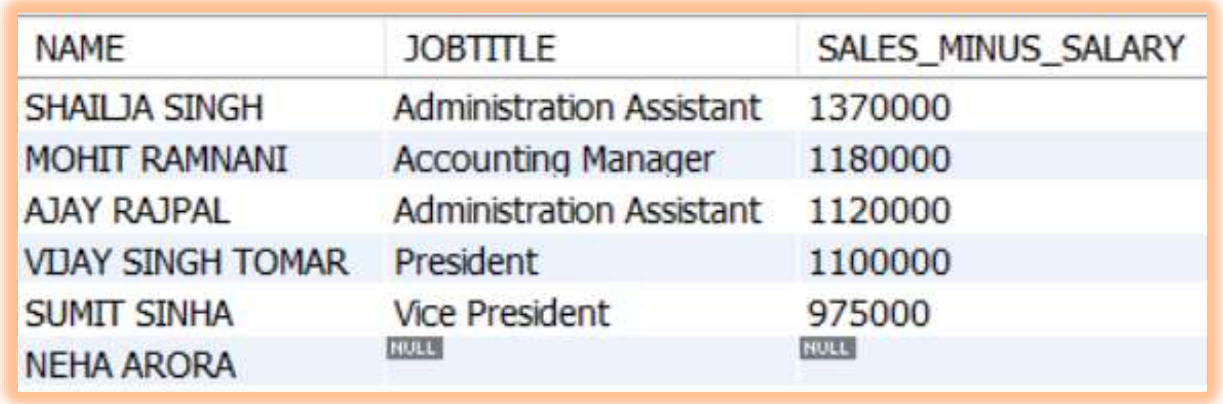
It returns all rows from the left table and the matching rows from the right table.



EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

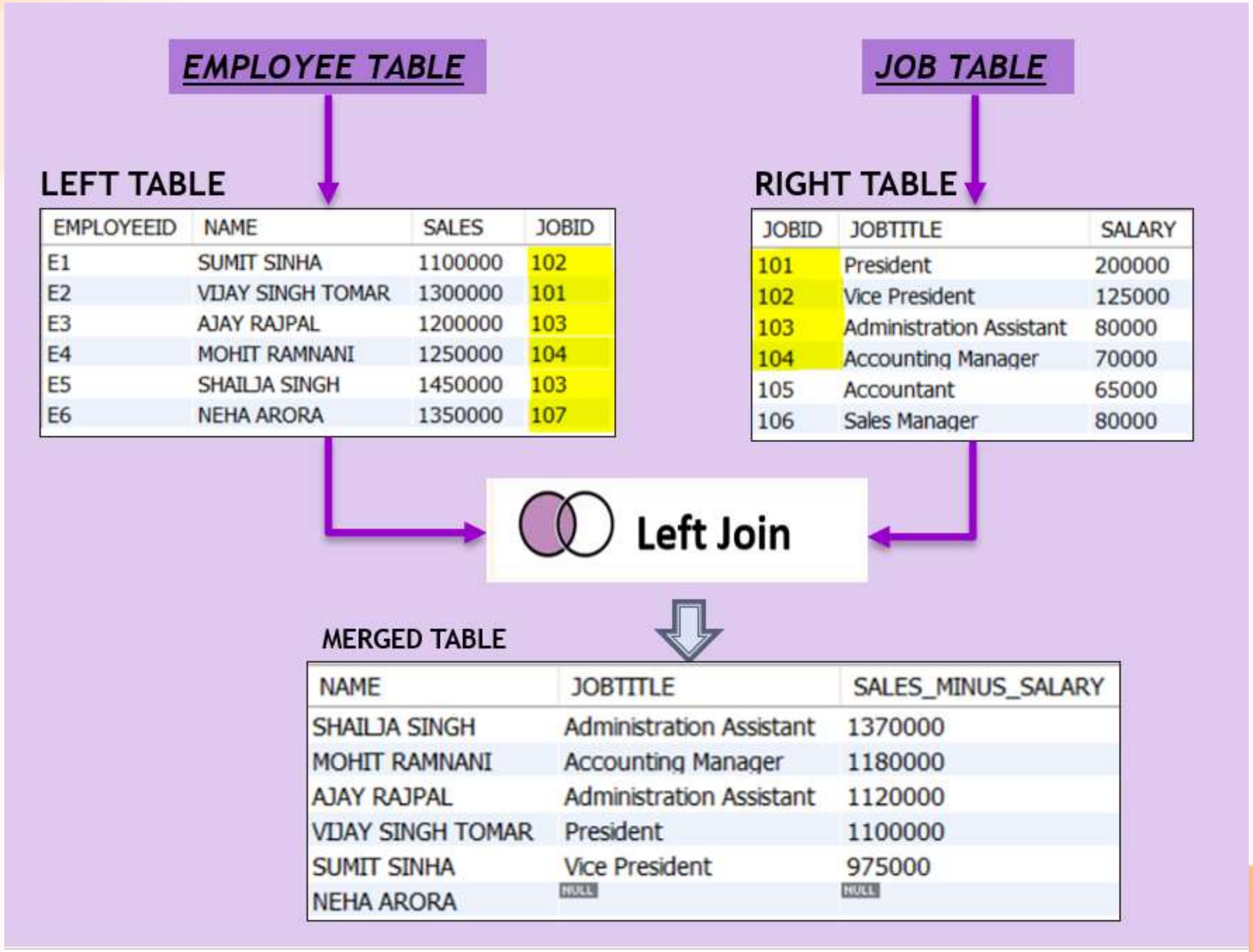
After performing the LEFT JOIN, we get the following result.



NAME	JOBTITLE	SALES_MINUS_SALARY
SHAILJA SINGH	Administration Assistant	1370000
MOHIT RAMNANI	Accounting Manager	1180000
AJAY RAJPAL	Administration Assistant	1120000
VIJAY SINGH TOMAR	President	1100000
SUMIT SINHA	Vice President	975000
NEHA ARORA	NULL	NULL

NEHA ARORA gets NULL values in the result because her JOBID (107) doesn't exist in the Jobs table.
Since we are using a **LEFT JOIN**, all rows from the Employees table are included, even if there's no matching row in the Jobs table. When there's no match, the columns from the right table (Jobs) show NULL, which is why her JOBTITLE and SALES_MINUS_SALARY are NULL.

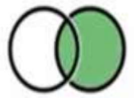
RESULT :-



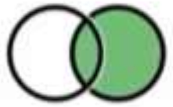
EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

NAME	JOBTITLE	SALES_MINUS_SALARY
SHAILJA SINGH	Administration Assistant	1370000
MOHIT RAMNANI	Accounting Manager	1180000
AJAY RAJPAL	Administration Assistant	1120000
VIJAY SINGH TOMAR	President	1100000
SUMIT SINHA	Vice President	975000
NEHA ARORA	NULL	NULL


Right Join

OR


Right outer join

(All Right Table Records & Matching Left Table Records)

A RIGHT JOIN returns all rows from the right table, and the matching rows from the left table. If there is no match on the left side, you'll see NULL values appear for the left table's columns. Note: RIGHT OUTER JOIN and RIGHT JOIN mean the same thing — "OUTER" is optional.

Syntax :-

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

Syntax :-

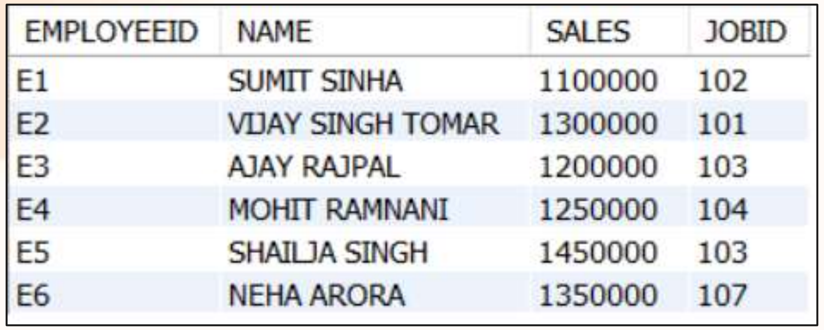
```
SELECT table1.column1, table2.column2
FROM table1
RIGHT JOIN table2
USING(common_column);
```

Use ON when:
- The **column names are different** in the two tables.
- You need more flexibility in the join condition.

Use USING when:
- **The column name is the same in both tables.**
- **You want a cleaner, simpler join syntax.**

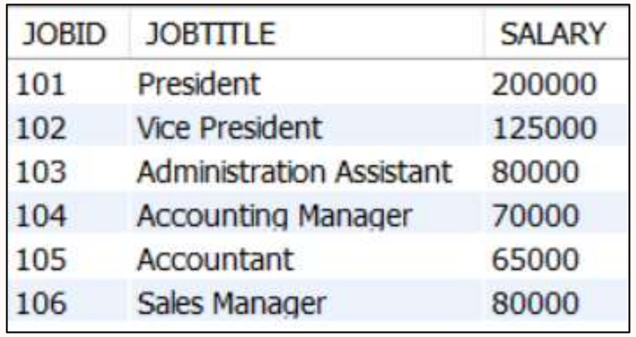
❖ These are our EMPLOYEE and JOB tables, which we will use to perform RIGHT JOIN .

EMPLOYEE TABLE



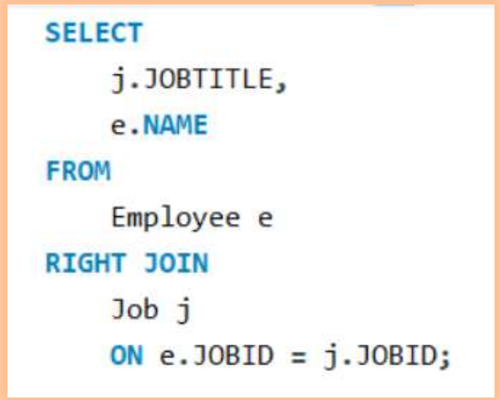
EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOB TABLE

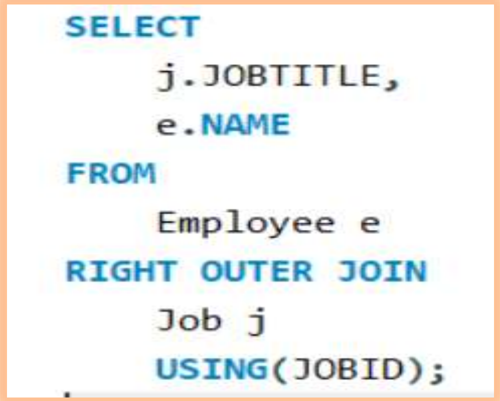


JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

-- Show all job titles and the names of employees assigned to each job.



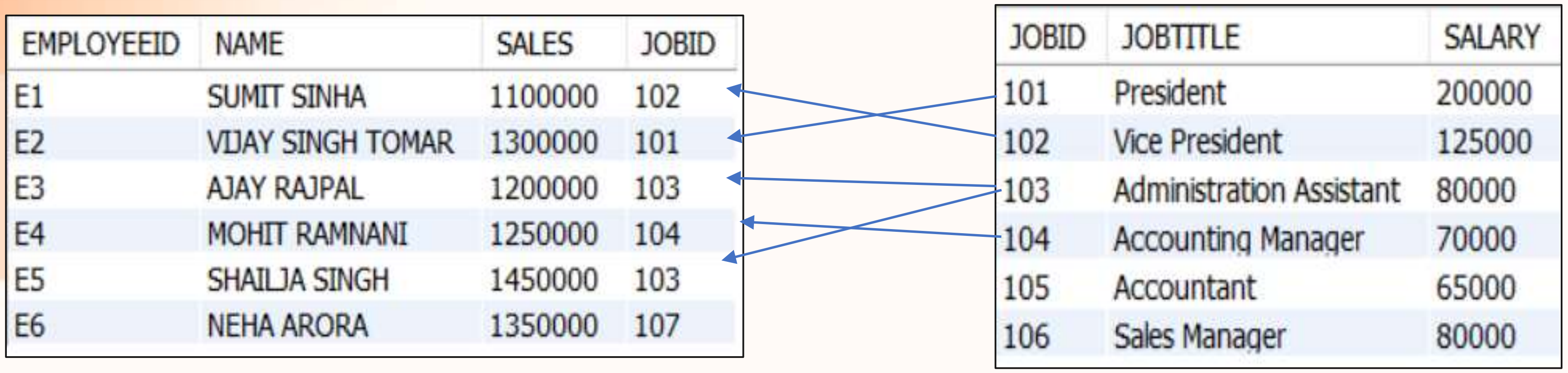
```
SELECT
    j.JOBTITLE,
    e.NAME
FROM
    Employee e
RIGHT JOIN
    Job j
    ON e.JOBID = j.JOBID;
```



```
SELECT
    j.JOBTITLE,
    e.NAME
FROM
    Employee e
RIGHT OUTER JOIN
    Job j
    USING(JOBID);
```

Both of these queries will produce the same result.

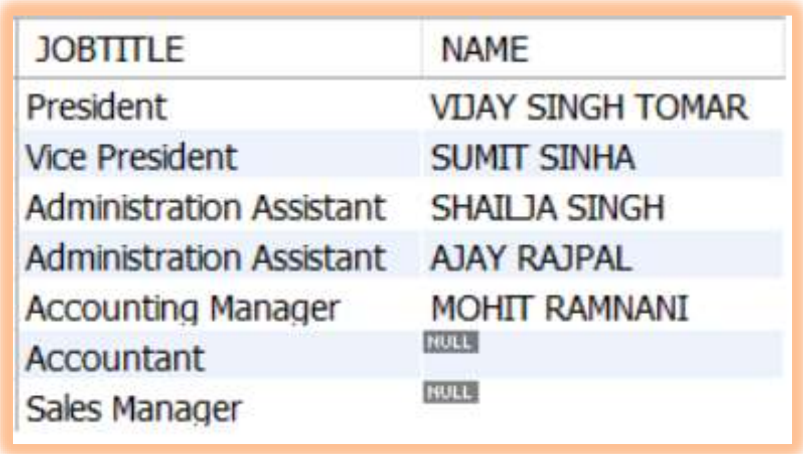
It returns all rows from the right table and the matching rows from the left table.



EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

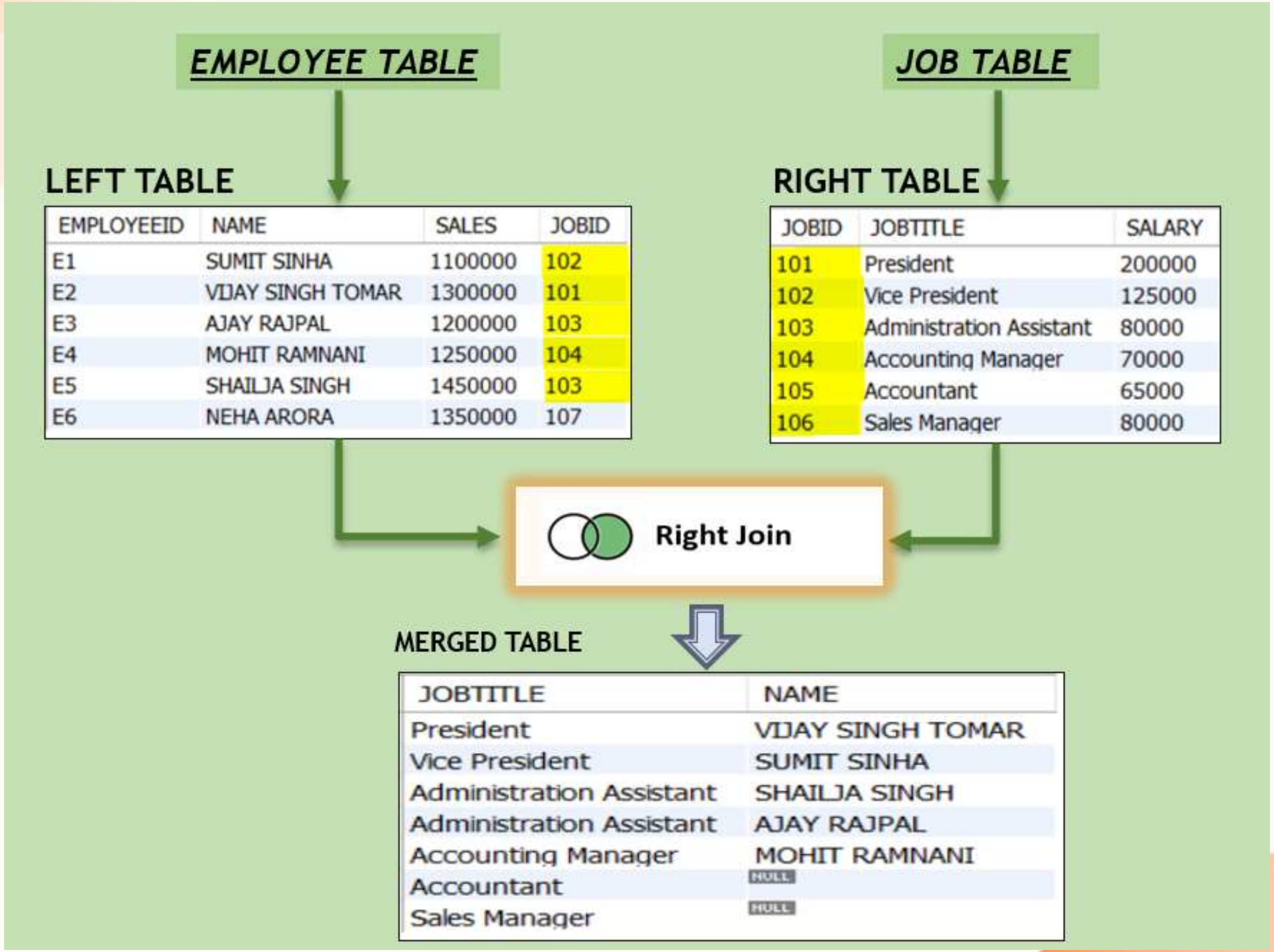
After performing the RIGHT JOIN, we get the following result.



JOBTITLE	NAME
President	VIJAY SINGH TOMAR
Vice President	SUMIT SINHA
Administration Assistant	SHAILJA SINGH
Administration Assistant	AJAY RAJPAL
Accounting Manager	MOHIT RAMNANI
Accountant	NULL
Sales Manager	NULL

In the output, NULL appears in the **NAME** column for the jobs "Accountant" and "Sales Manager" because **no employee is assigned** to these job titles. Since we are using a **RIGHT JOIN**, all jobs from the Jobs table are included, even if there's no matching employee in the Employees table. When there's no match, the employee details are shown as NULL.

RESULT :-



LEFT TABLE

EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

RIGHT TABLE

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

MERGED TABLE

JOBTITLE	NAME
President	VIJAY SINGH TOMAR
Vice President	SUMIT SINHA
Administration Assistant	SHAILJA SINGH
Administration Assistant	AJAY RAJPAL
Accounting Manager	MOHIT RAMNANI
Accountant	NULL
Sales Manager	NULL

Full Join

O
R

Full outer join

A **FULL JOIN** (or **FULL OUTER JOIN**) returns **all rows from both tables:**

- When there is a **match**, it combines the data.
- When there is **no match**, it still includes the row, filling in NULL for the missing side.

It’s like combining **LEFT JOIN + RIGHT JOIN**.

Syntax :-

```
SELECT columns
FROM table1
FULL JOIN table2
ON table1.common_column = table2.common_column;
```

This works in databases like PostgreSQL, SQL Server, and Oracle — not in MySQL

FULL JOIN Syntax in MySQL

Syntax :-

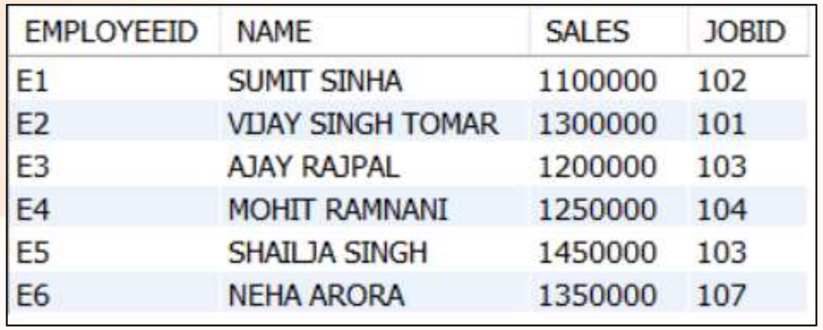
```
SELECT
    *
FROM table1 t1
LEFT JOIN table2 t2
    ON t1.common_column = t2.common_column

UNION

SELECT
    *
FROM table1 t1
RIGHT JOIN table2 t2
    ON t1.common_column = t2.common_column;
```

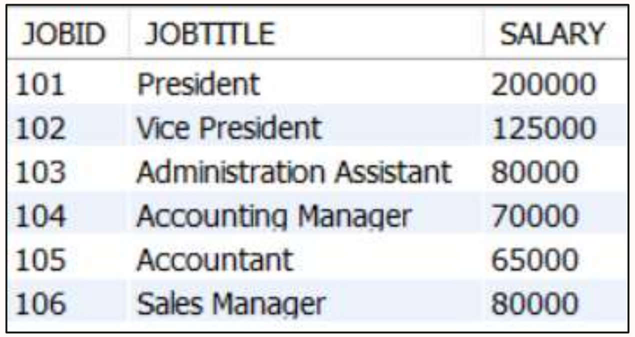
❖ These are our EMPLOYEE and JOB tables, which we will use to perform FULL JOIN.

EMPLOYEE TABLE



EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

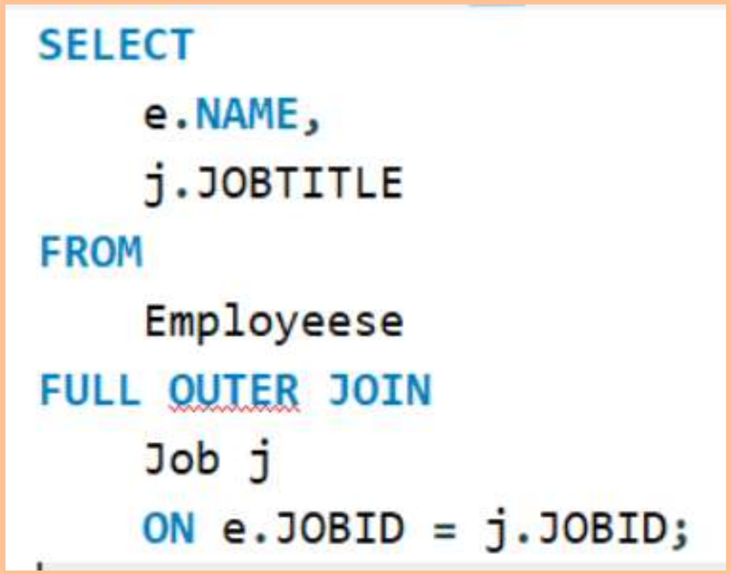
JOB TABLE



JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

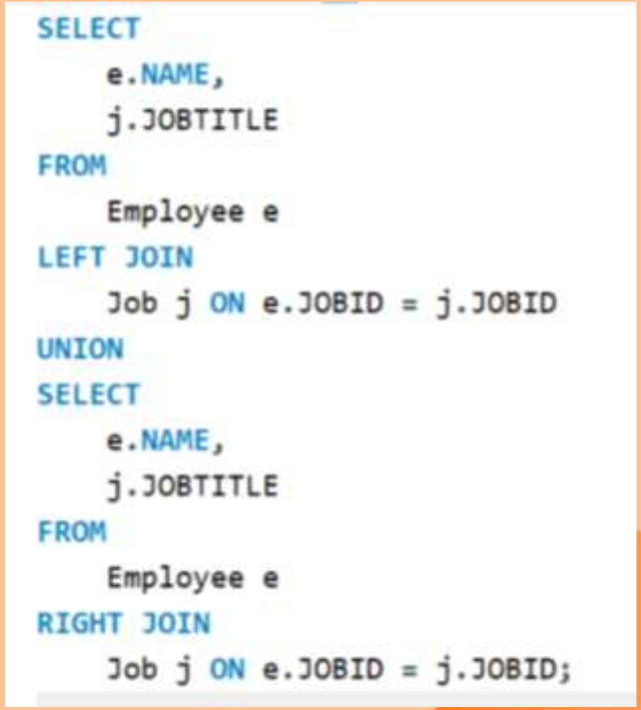
-- List all employees with their job titles , including those employees who don’t have a matching job , and jobs that don’t have a matching employee.

FULL JOIN in PostgreSQL, SQL Server, and Oracle



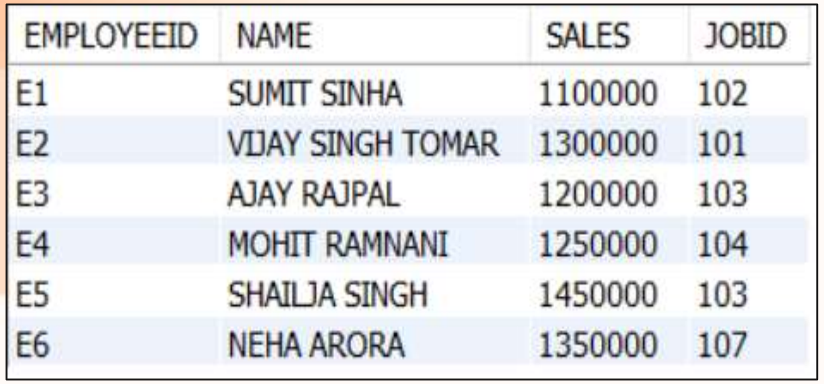
```
SELECT
    e.NAME,
    j.JOBTITLE
FROM
    Employeese
FULL OUTER JOIN
    Job j
    ON e.JOBID = j.JOBID;
```

FULL JOIN in MySQL

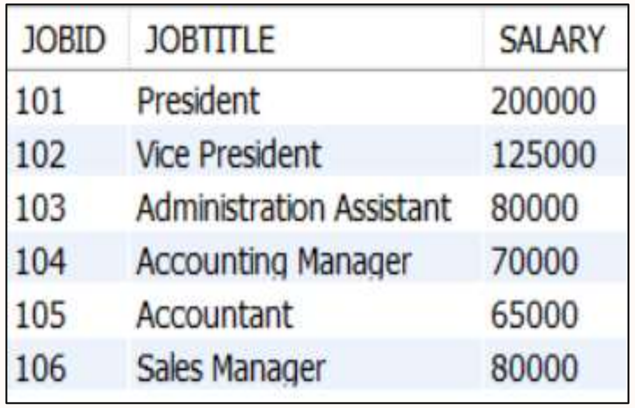


```
SELECT
    e.NAME,
    j.JOBTITLE
FROM
    Employee e
LEFT JOIN
    Job j ON e.JOBID = j.JOBID
UNION
SELECT
    e.NAME,
    j.JOBTITLE
FROM
    Employee e
RIGHT JOIN
    Job j ON e.JOBID = j.JOBID;
```

It returns all rows from the right table and from the left table.

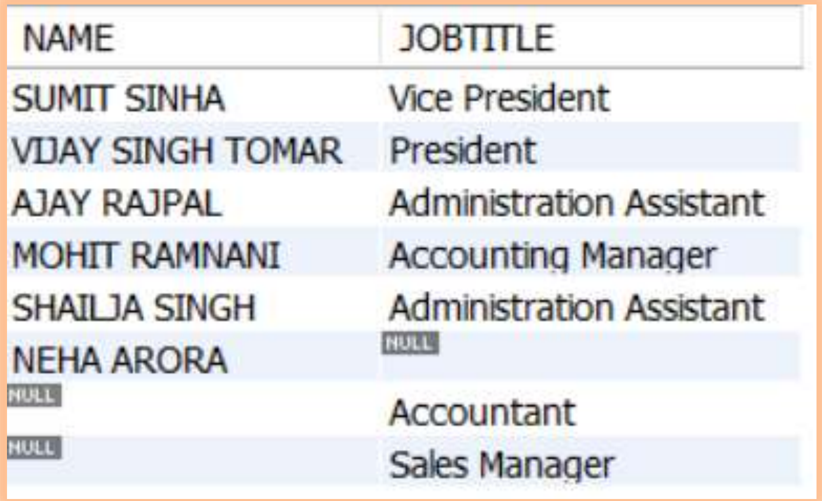


EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107



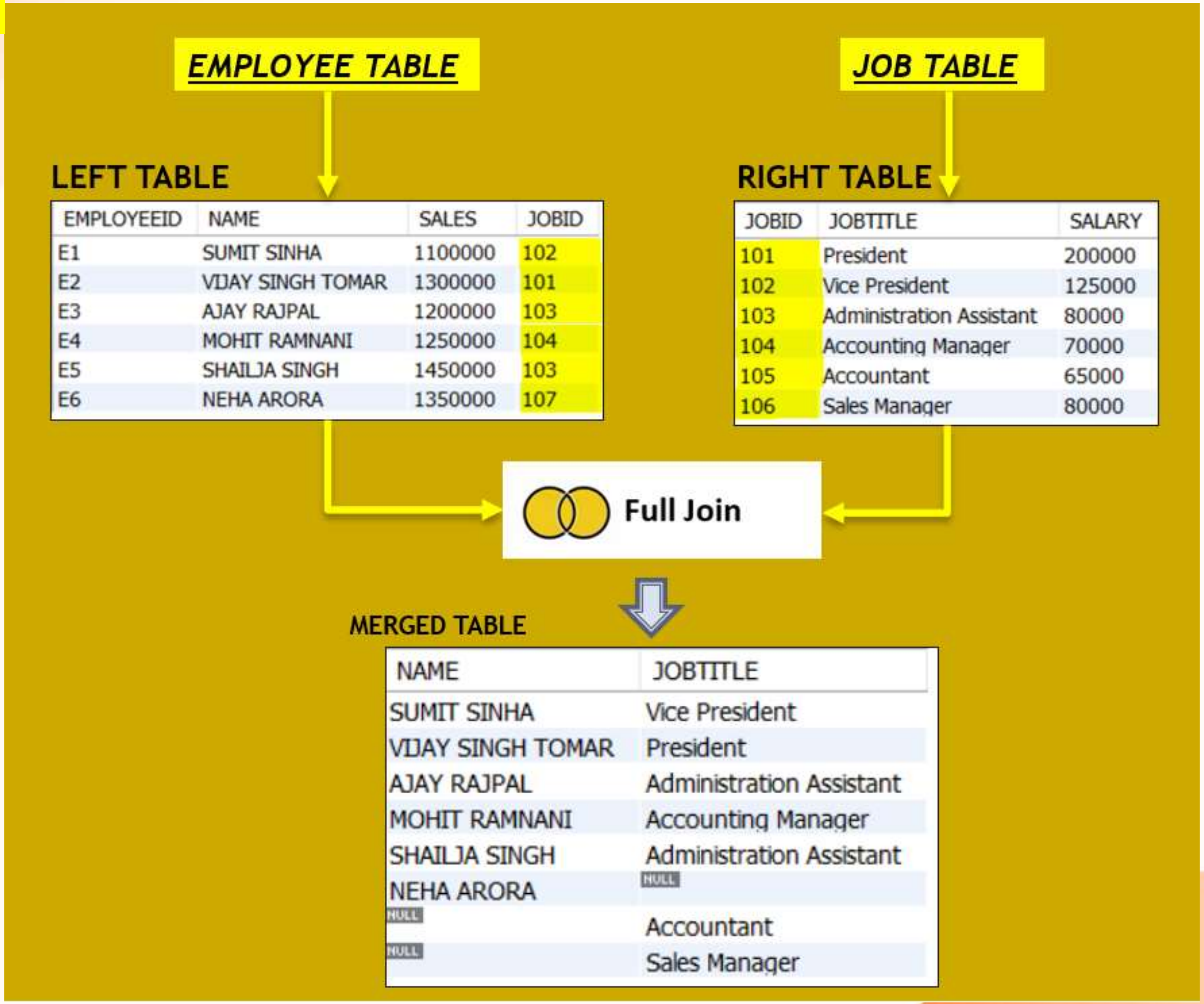
JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

<u>After performing the FULL JOIN, we get the following result.</u>



NAME	JOBTITLE
SUMIT SINHA	Vice President
VIJAY SINGH TOMAR	President
AJAY RAJPAL	Administration Assistant
MOHIT RAMNANI	Accounting Manager
SHAILJA SINGH	Administration Assistant
NEHA ARORA	NULL
NULL	Accountant
NULL	Sales Manager

RESULT :-



EMPLOYEEID	NAME	SALES	JOBID
E1	SUMIT SINHA	1100000	102
E2	VIJAY SINGH TOMAR	1300000	101
E3	AJAY RAJPAL	1200000	103
E4	MOHIT RAMNANI	1250000	104
E5	SHAILJA SINGH	1450000	103
E6	NEHA ARORA	1350000	107

JOBID	JOBTITLE	SALARY
101	President	200000
102	Vice President	125000
103	Administration Assistant	80000
104	Accounting Manager	70000
105	Accountant	65000
106	Sales Manager	80000

NAME	JOBTITLE
SUMIT SINHA	Vice President
VIJAY SINGH TOMAR	President
AJAY RAJPAL	Administration Assistant
MOHIT RAMNANI	Accounting Manager
SHAILJA SINGH	Administration Assistant
NEHA ARORA	NULL
NULL	Accountant
NULL	Sales Manager

SELF JOIN

A **SELF JOIN** is a join where a table is joined with itself.
It is used to compare or relate rows within the same table by using **table aliases**.

Syntax :-

```
SELECT a.column_name, b.column_name
FROM table_name a
JOIN table_name b
 ON a.common_column = b.common_column;
```

Aliases (a, b) are **mandatory** to differentiate the same table.

MySQL supports SELF JOIN

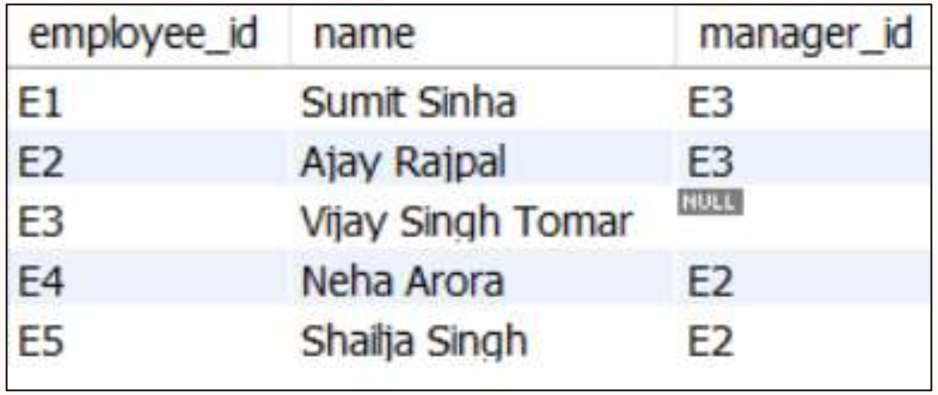
- SELF JOIN is **not a separate JOIN type**
- It is implemented using:
  - INNER JOIN
  - LEFT JOIN

When to Use SELF JOIN

- When a table has a **hierarchical relationship** (employee-manager)
- When comparing **rows within the same table**
- When finding:
  - Employees with the same job
  - Higher/lower values within the same table
  - Parent-child relationships

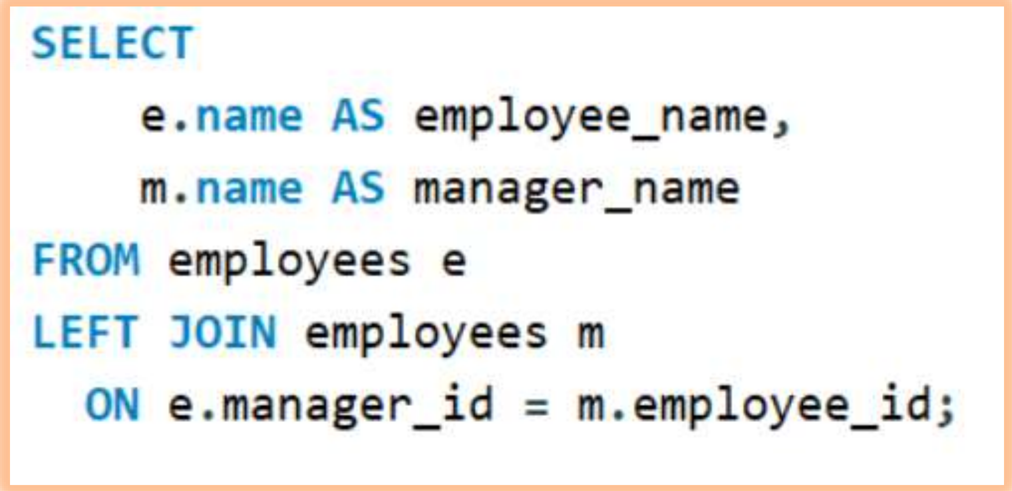
❖ **This EMPLOYEES table will be used to perform a SELF JOIN using INNER JOIN or LEFT JOIN.**

EMPLOYEES TABLE



employee_id	name	manager_id
E1	Sumit Sinha	E3
E2	Ajay Rajpal	E3
E3	Vijay Singh Tomar	NULL
E4	Neha Arora	E2
E5	Shailja Singh	E2

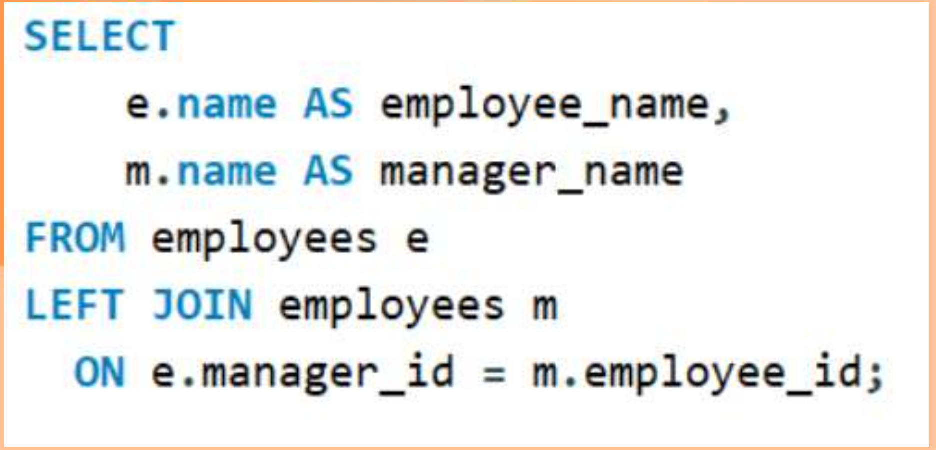
-- Write a SQL query to list all employees along with the name of their manager. If an employee does not have a manager, still include the employee in the result



```
SELECT
    e.name AS employee_name,
    m.name AS manager_name
FROM employees e
LEFT JOIN employees m
  ON e.manager_id = m.employee_id;
```

- Each row shows an **employee and their manager**
- LEFT JOIN ensures **all employees appear**
- Employees without a manager show **NULL**

Behind-the-Scenes of Employee-Manager Self-Join Using LEFT JOIN



```
SELECT
    e.name AS employee_name,
    m.name AS manager_name
FROM employees e
LEFT JOIN employees m
  ON e.manager_id = m.employee_id;
```

- employees e → represents the **employee table**
- employees m → represents the **manager table** (same table, different alias)

Even though it's the same table, SQL treats them as two separate tables in the join operation.

For each row in e, we look for m.employee_id = e.manager_id

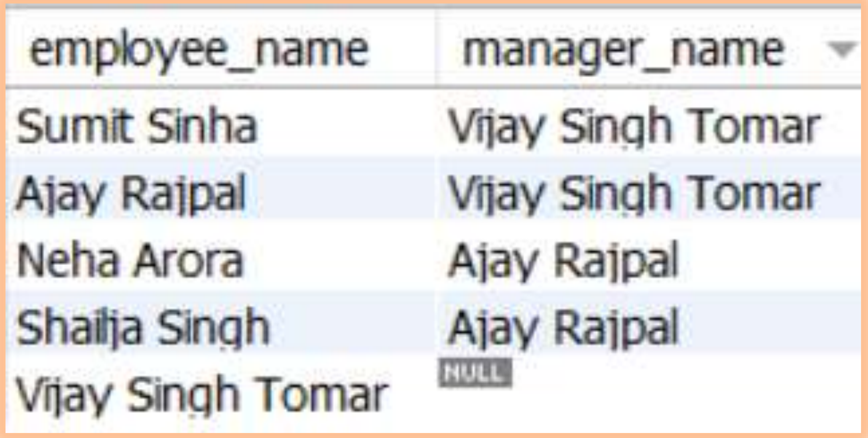
e.employee_id	e.name	e.manager_id	m.employee_id	m.name
E1	Sumit Sinha	E3	E3	Vijay Singh Tomar
E2	Ajay Rajpal	E3	E3	Vijay Singh Tomar
E3	Vijay Singh Tomar	NULL	NULL	NULL
E4	Neha Arora	E2	E2	Ajay Rajpal
E5	Shailja Singh	E2	E2	Ajay Rajpal

e.employee_id	e.name	e.manager_id	m.employee_id	m.name
E1	Sumit Sinha	E3	E3	Vijay Singh Tomar
E2	Ajay Rajpal	E3	E3	Vijay Singh Tomar
E3	Vijay Singh Tomar	NULL	NULL	NULL
E4	Neha Arora	E2	E2	Ajay Rajpal
E5	Shailja Singh	E2	E2	Ajay Rajpal

After matching, these names will appear as the **manager names** for the employees:

- **Vijay Singh Tomar → Manager of Sumit Sinha & Ajay Rajpal**
- **Ajay Rajpal → Manager of Neha Arora & Shailja Singh**

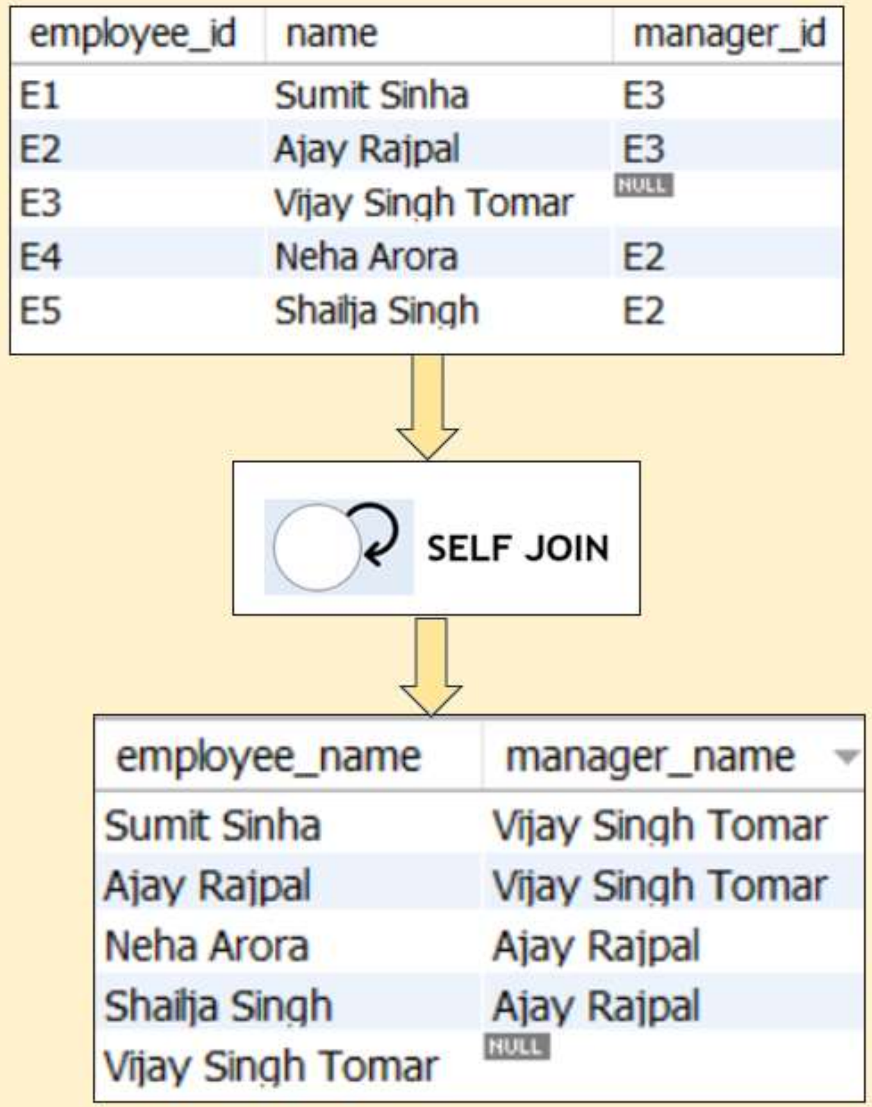
After performing the SELF JOIN, we get the following result.



employee_name	manager_name
Sumit Sinha	Vijay Singh Tomar
Ajay Rajpal	Vijay Singh Tomar
Neha Arora	Ajay Rajpal
Shailja Singh	Ajay Rajpal
Vijay Singh Tomar	NULL

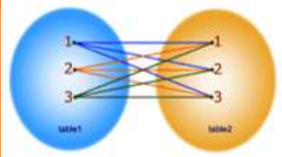
RESULT :-

EMPLOYEES TABLE



employee_id	name	manager_id
E1	Sumit Sinha	E3
E2	Ajay Rajpal	E3
E3	Vijay Singh Tomar	NULL
E4	Neha Arora	E2
E5	Shailja Singh	E2

employee_name	manager_name
Sumit Sinha	Vijay Singh Tomar
Ajay Rajpal	Vijay Singh Tomar
Neha Arora	Ajay Rajpal
Shailja Singh	Ajay Rajpal
Vijay Singh Tomar	NULL



CROSS JOIN

A **CROSS JOIN** returns **all possible combinations** of rows from two tables.

- If Table A has m rows and Table B has n rows, the result will have m × n rows.
- It does **not require any condition** to join.

Syntax :-

```
SELECT *
FROM table1
CROSS JOIN table2;
```

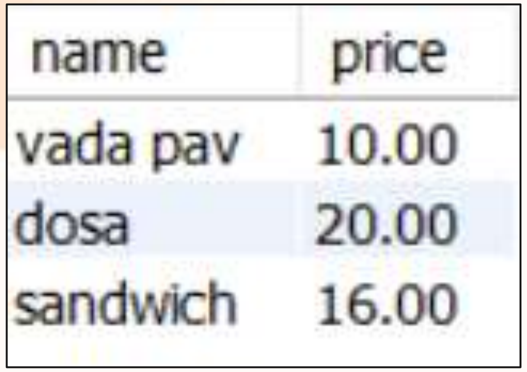
OR

Syntax :-

```
SELECT *
FROM table1, table2;
```

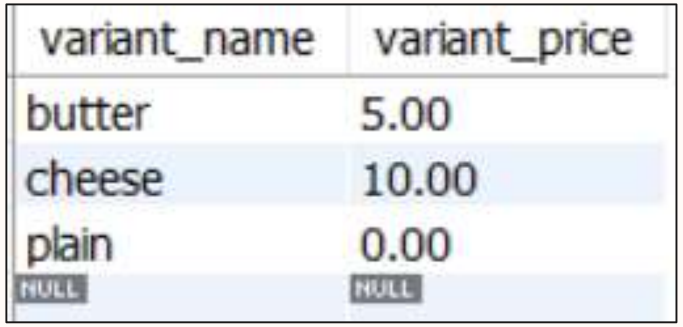
❖ These are our ITEMS and VARIANTS tables, which we will use to perform CROSS JOIN.

ITEMS TABLE



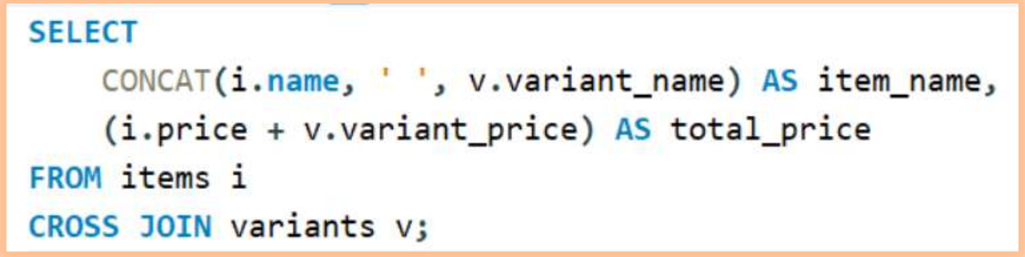
name	price
vada pav	10.00
dosa	20.00
sandwich	16.00

VARIANTS TABLE

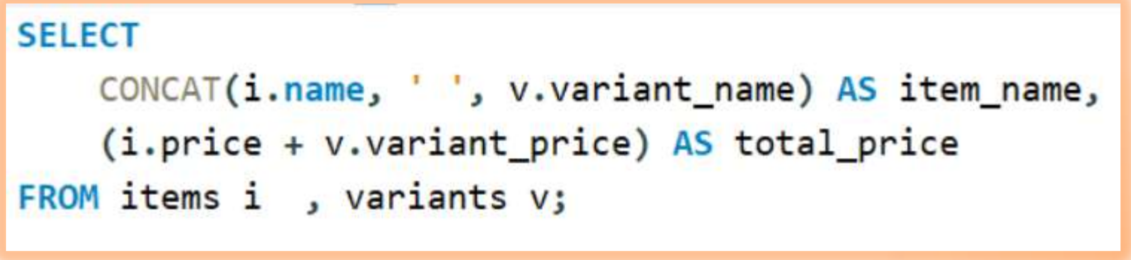


variant_name	variant_price
butter	5.00
cheese	10.00
plain	0.00
NULL	NULL

-- Write a SQL query to list all possible combinations of items and variants.



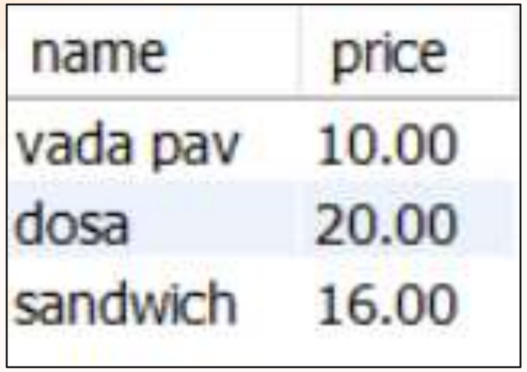
```
SELECT
    CONCAT(i.name, ' ', v.variant_name) AS item_name,
    (i.price + v.variant_price) AS total_price
FROM items i
CROSS JOIN variants v;
```



```
SELECT
    CONCAT(i.name, ' ', v.variant_name) AS item_name,
    (i.price + v.variant_price) AS total_price
FROM items i  , variants v;
```

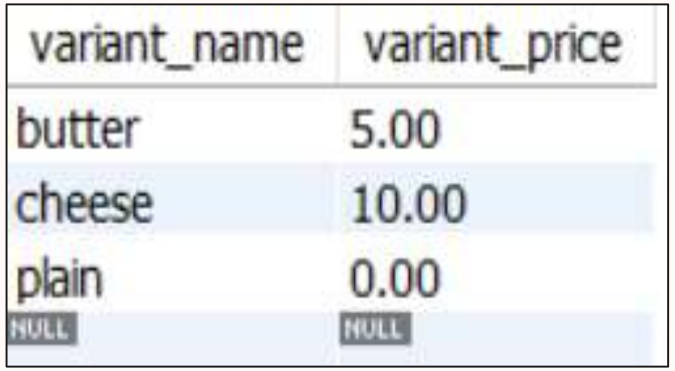
Both of these queries will produce the same result.

It returns all possible combinations of rows from the first table and the second table



name	price
vada pav	10.00
dosa	20.00
sandwich	16.00

×



variant_name	variant_price
butter	5.00
cheese	10.00
plain	0.00
NULL	NULL

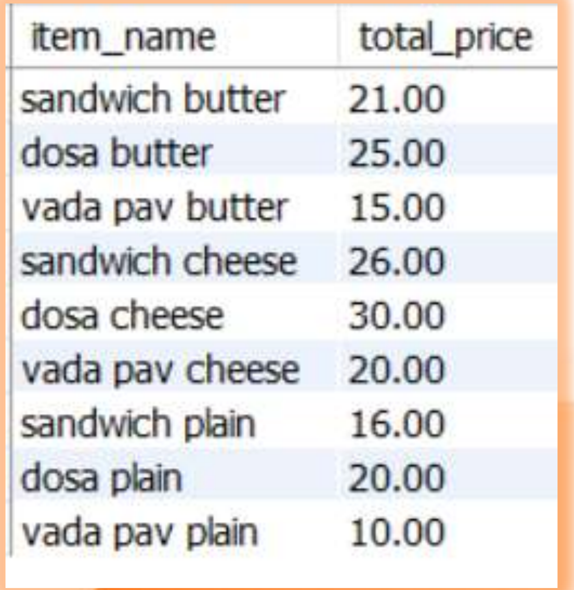
Table1 has 3 rows and **Table2 has 3 rows**, a **CROSS JOIN** will return:

$$3 \times 3 = 9 \; rows$$

Each row from Table1 is paired with **every row** from Table2.

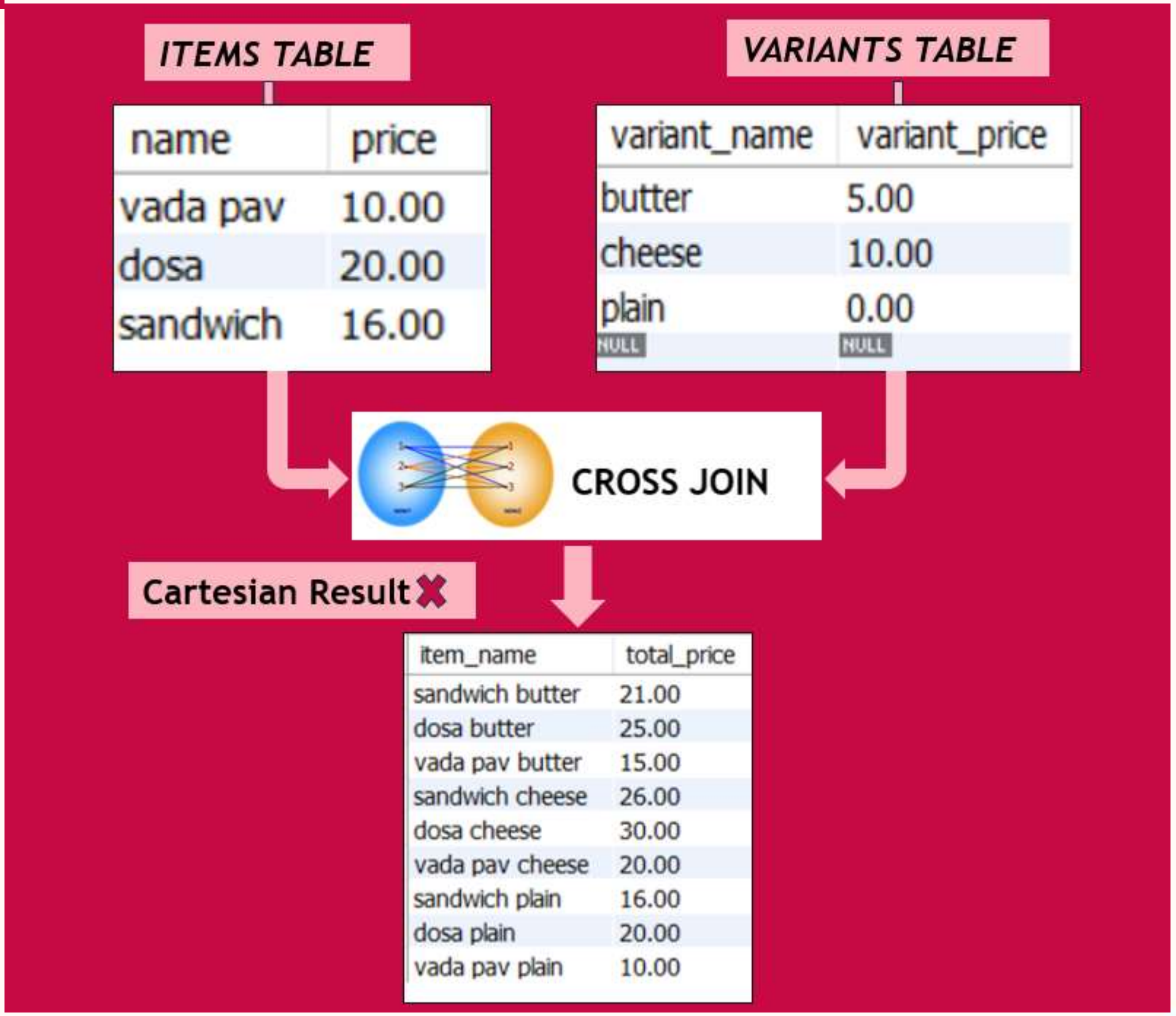
This is exactly the **Cartesian product** principle, combining every row from the first table with every row from the second table.

<ins>**After performing the CROSS JOIN, we get the following result.**</ins>



item_name	total_price
sandwich butter	21.00
dosa butter	25.00
vada pav butter	15.00
sandwich cheese	26.00
dosa cheese	30.00
vada pav cheese	20.00
sandwich plain	16.00
dosa plain	20.00
vada pav plain	10.00

RESULT :-



name	price
vada pav	10.00
dosa	20.00
sandwich	16.00

variant_name	variant_price
butter	5.00
cheese	10.00
plain	0.00
NULL	NULL

item_name	total_price
sandwich butter	21.00
dosa butter	25.00
vada pav butter	15.00
sandwich cheese	26.00
dosa cheese	30.00
vada pav cheese	20.00
sandwich plain	16.00
dosa plain	20.00
vada pav plain	10.00

ANTI JOINS

❖ **What Are Anti Joins?**
Anti Joins are used to return **rows that do NOT have matching records** in the other table.

TYPES OF ANTI JOIN

- LEFT ANTI JOIN
- RIGHT ANTI JOIN
- FULL ANTI JOIN

- ***LEFT ANTI JOIN***
  Returns rows **from the left table** that have **no match** in the right table.

Syntax :-

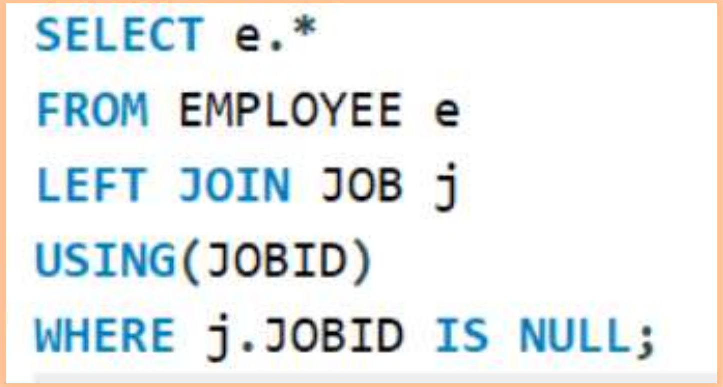
```
SELECT t1.*
FROM table1 t1
LEFT JOIN table2 t2
ON t1.id = t2.id
WHERE t2.id IS NULL;
```

Syntax :-

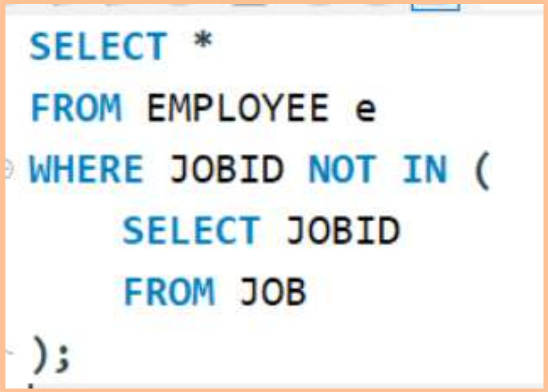
```
SELECT *
FROM table1
WHERE id NOT IN (
    SELECT id
    FROM table2
);
```

Syntax :-

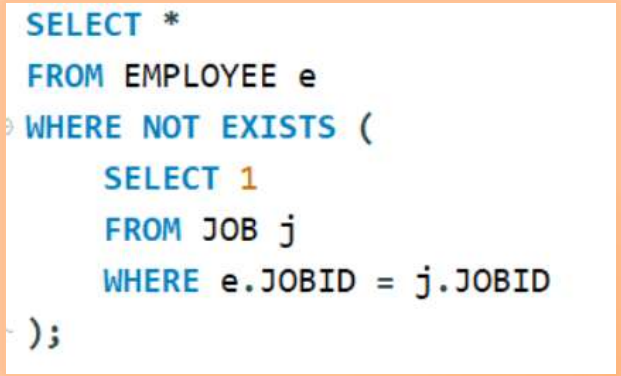
```
SELECT *
FROM table1
WHERE NOT EXISTS (
    SELECT 1
    FROM table2
    WHERE table1.id = table2.id
);
```



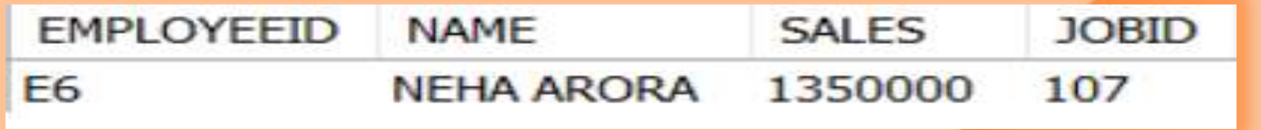
```
SELECT e.*
FROM EMPLOYEE e
LEFT JOIN JOB j
USING(JOBID)
WHERE j.JOBID IS NULL;
```



```
SELECT *
FROM EMPLOYEE e
WHERE JOBID NOT IN (
    SELECT JOBID
    FROM JOB
);
```



```
SELECT *
FROM EMPLOYEE e
WHERE NOT EXISTS (
    SELECT 1
    FROM JOB j
    WHERE e.JOBID = j.JOBID
);
```



EMPLOYEEID	NAME	SALES	JOBID
E6	NEHA ARORA	1350000	107

- ***RIGHT ANTI JOIN***

  Returns rows **from the right table** that have **no match** in the left table.

Syntax :-

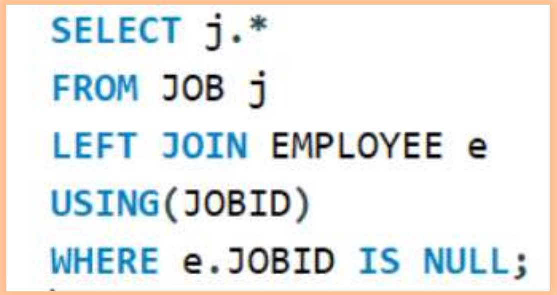
```
SELECT t2.*
FROM table2 t2
LEFT JOIN table1 t1 ON t1.id = t2.id
WHERE t1.id IS NULL;
```

Syntax :-

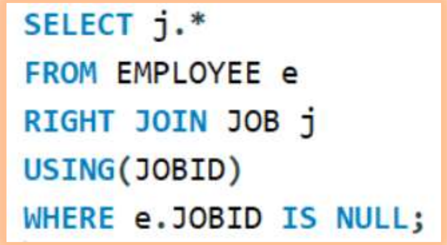
```
SELECT *
FROM table2
WHERE id NOT IN (
    SELECT id
    FROM table1
);
```

Syntax :-

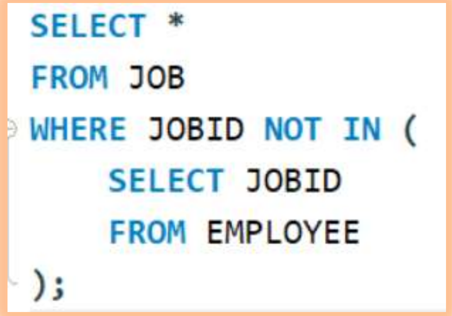
```
SELECT *
FROM table2
WHERE NOT EXISTS (
    SELECT 1
    FROM table1
    WHERE table1.id = table2.id
);
```



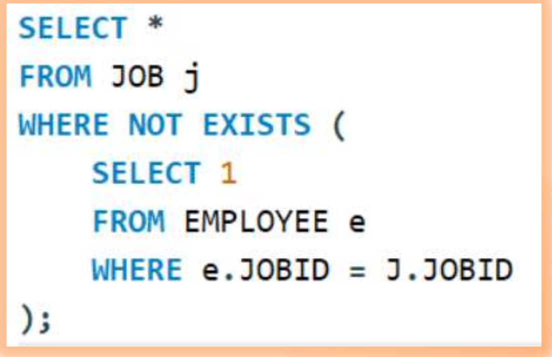
```
SELECT j.*
FROM JOB j
LEFT JOIN EMPLOYEE e
USING(JOBID)
WHERE e.JOBID IS NULL;
```



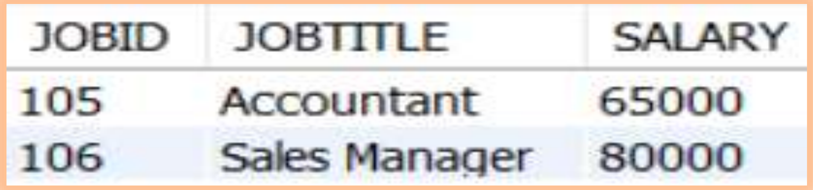
```
SELECT j.*
FROM EMPLOYEE e
RIGHT JOIN JOB j
USING(JOBID)
WHERE e.JOBID IS NULL;
```



```
SELECT *
FROM JOB
WHERE JOBID NOT IN (
    SELECT JOBID
    FROM EMPLOYEE
);
```



```
SELECT *
FROM JOB j
WHERE NOT EXISTS (
    SELECT 1
    FROM EMPLOYEE e
    WHERE e.JOBID = J.JOBID
);
```



JOBID	JOBTITLE	SALARY
105	Accountant	65000
106	Sales Manager	80000

- ***FULL ANTI JOIN***

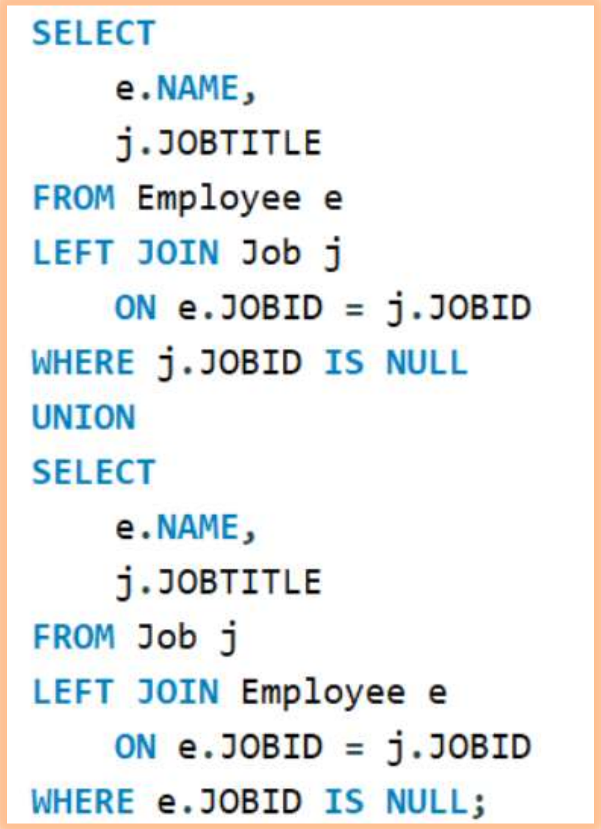
  Returns all rows from both tables that **don’t have a match** in the either table.

Syntax :-

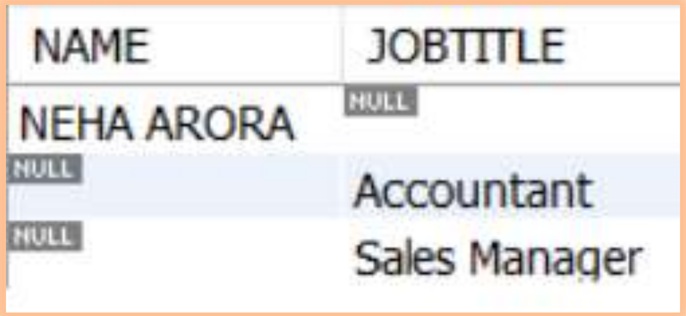
```
-- Unmatched from table1
SELECT t1.*
FROM table1 t1
LEFT JOIN table2 t2
ON t1.id = t2.id
WHERE t2.id IS NULL

UNION

-- Unmatched from table2
SELECT t2.*
FROM table2 t2
LEFT JOIN table1 t1
ON t1.id = t2.id
WHERE t1.id IS NULL;
```



```
SELECT
    e.NAME,
    j.JOBTITLE
FROM Employee e
LEFT JOIN Job j
    ON e.JOBID = j.JOBID
WHERE j.JOBID IS NULL
UNION
SELECT
    e.NAME,
    j.JOBTITLE
FROM Job j
LEFT JOIN Employee e
    ON e.JOBID = j.JOBID
WHERE e.JOBID IS NULL;
```



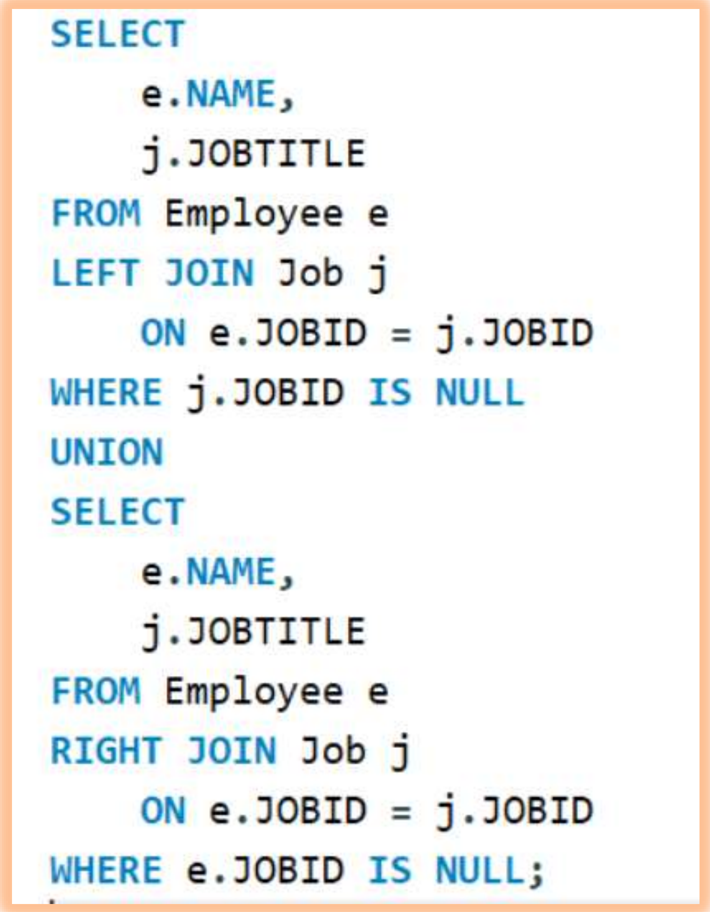
NAME	JOBTITLE
NEHA ARORA	NULL
NULL	Accountant
NULL	Sales Manager

Syntax :-

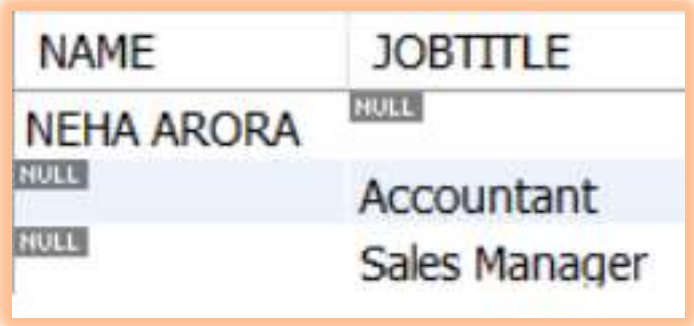
```
-- Unmatched from table1
SELECT t1.*
FROM table1 t1
LEFT JOIN table2 t2
ON t1.id = t2.id
WHERE t2.id IS NULL

UNION

-- Unmatched from table2
SELECT t2.*
FROM table1 t1
RIGHT JOIN table2 t2
ON t1.id = t2.id
WHERE t1.id IS NULL;
```



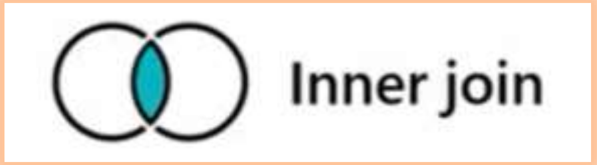
```
SELECT
    e.NAME,
    j.JOBTITLE
FROM Employee e
LEFT JOIN Job j
    ON e.JOBID = j.JOBID
WHERE j.JOBID IS NULL
UNION
SELECT
    e.NAME,
    j.JOBTITLE
FROM Employee e
RIGHT JOIN Job j
    ON e.JOBID = j.JOBID
WHERE e.JOBID IS NULL;
```

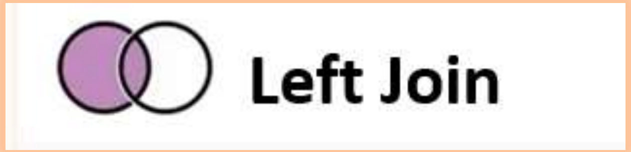


NAME	JOBTITLE
NEHA ARORA	NULL
NULL	Accountant
NULL	Sales Manager

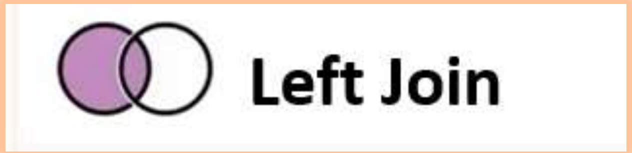
Frequently Used SQL Joins

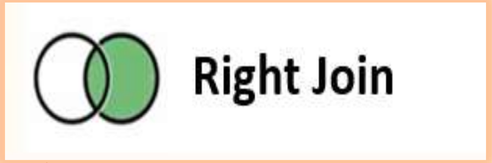
Mainly used joins are **INNER JOIN** and **LEFT JOIN**.



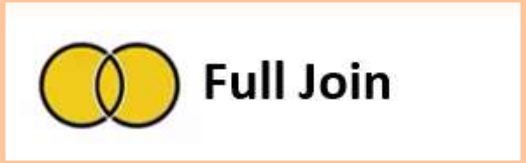


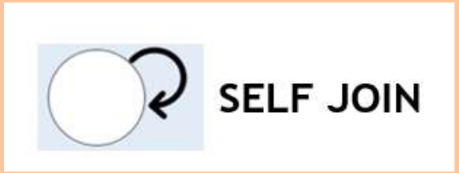
A **LEFT JOIN** can replace a **RIGHT JOIN**, but you must ensure the **table order in the FROM and JOIN clauses is correct.**

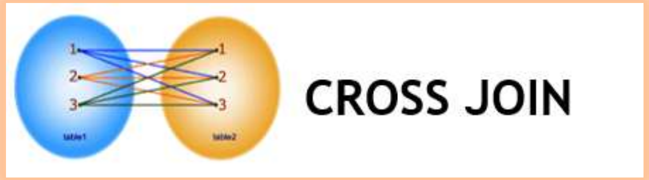




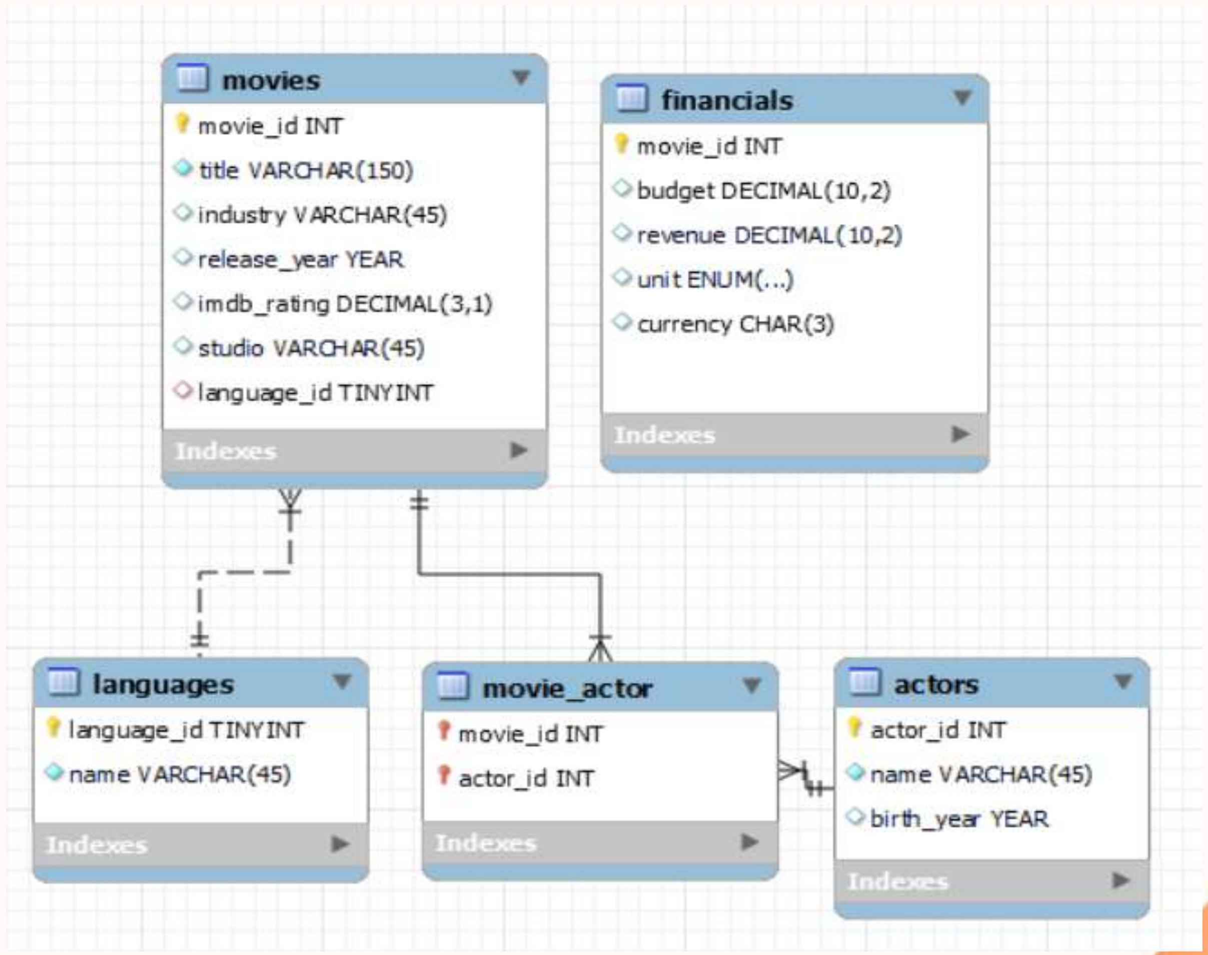
Other joins are used based on specific requirements.



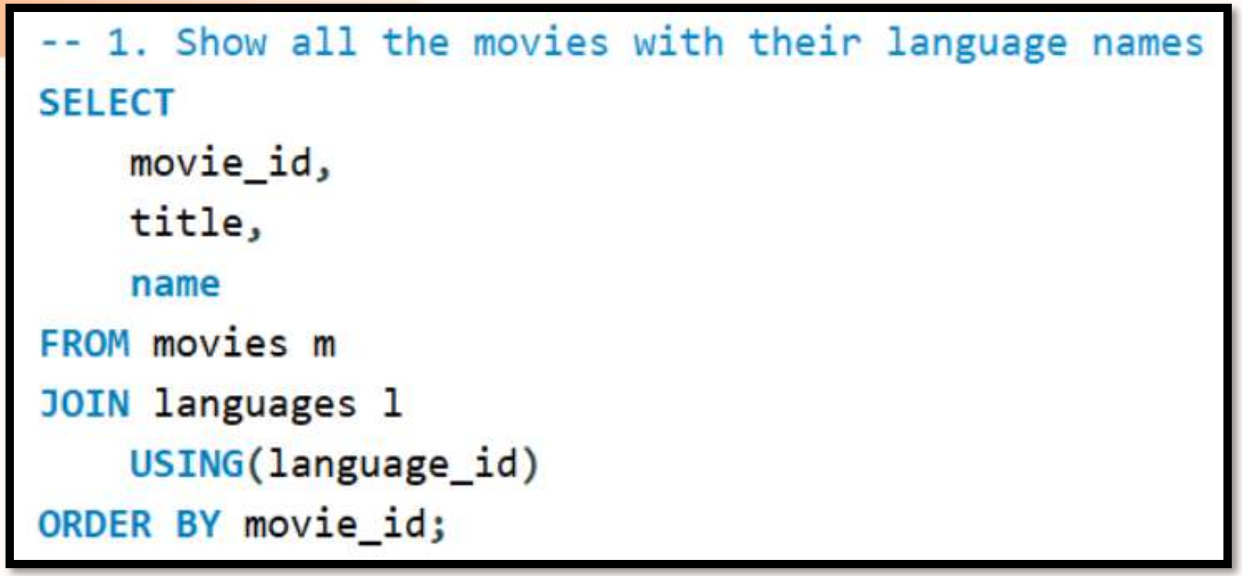




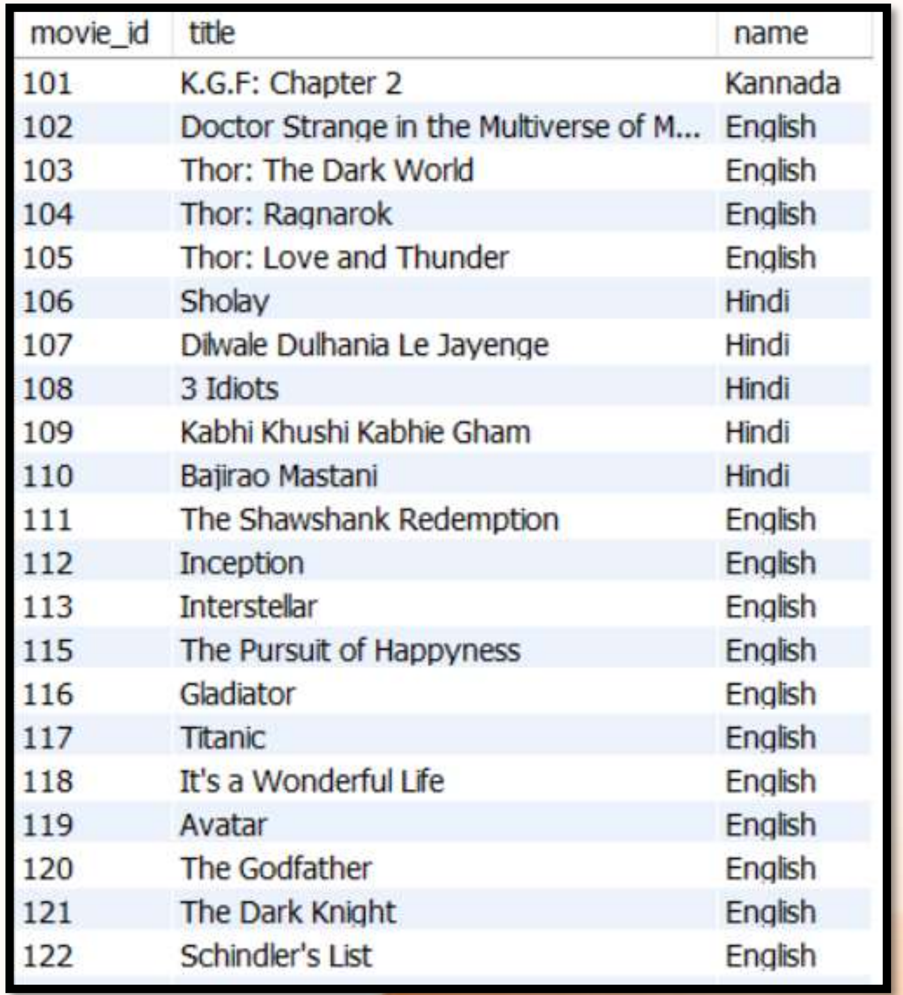
This is the **Movies dataset** from **CodeBasics**, used for SQL practice.
A similar dataset can also be downloaded from **Kaggle**.
Using this dataset, we will practice and understand different **SQL JOIN operations**.



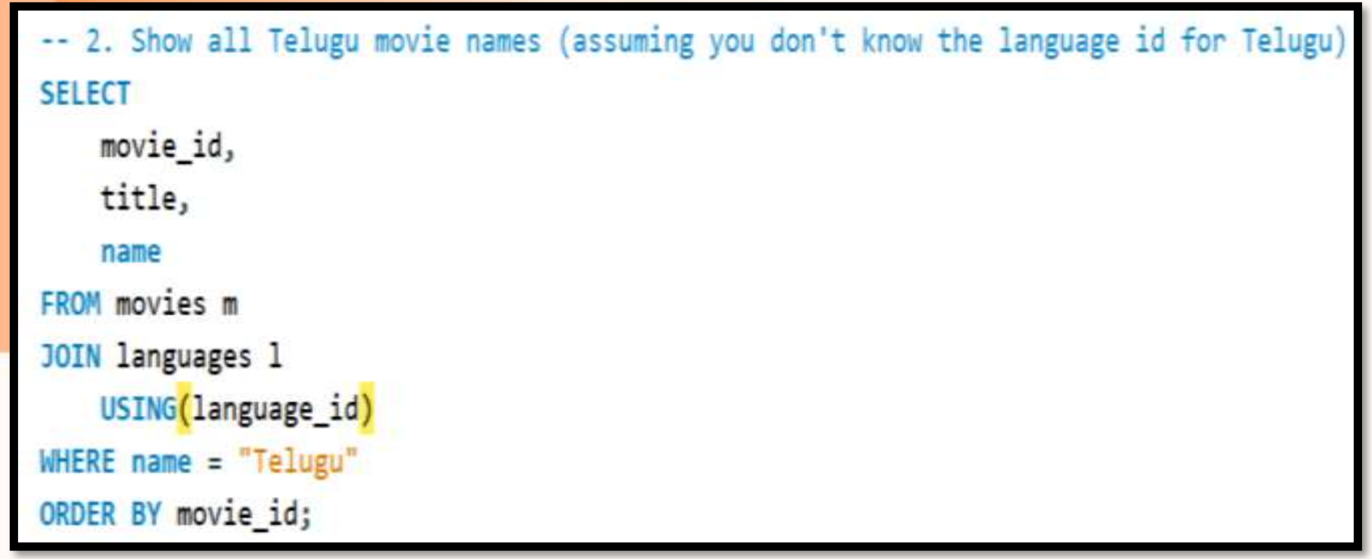
- **Using INNER JOIN along with WHERE, GROUP BY, ORDER BY, and aggregate functions for practice and analysis.**



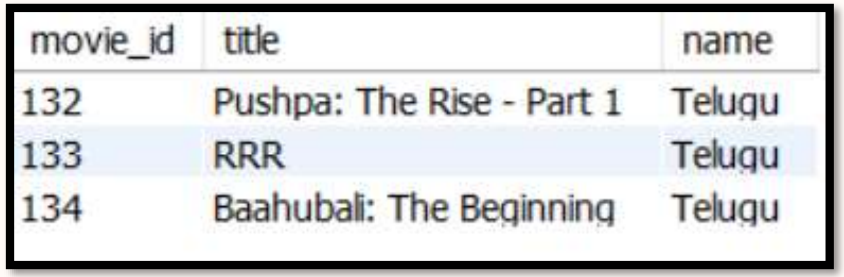
```
-- 1. Show all the movies with their language names
SELECT
    movie_id,
    title,
    name
FROM movies m
JOIN languages l
    USING(language_id)
ORDER BY movie_id;
```



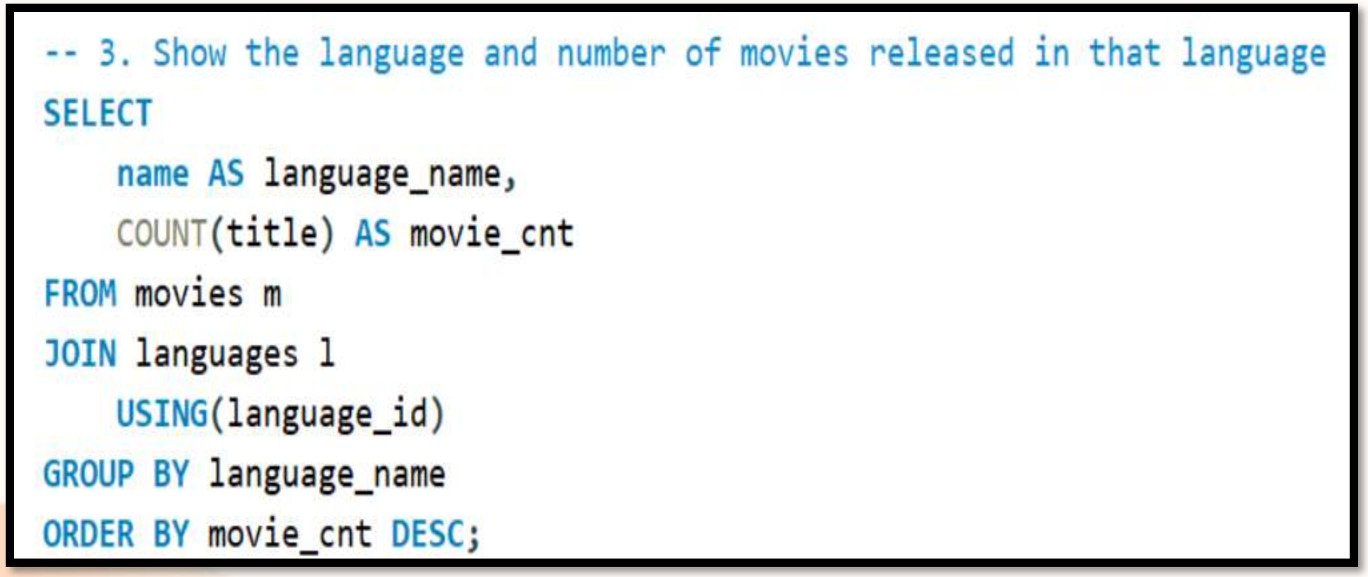
movie_id	title	name
101	K.G.F: Chapter 2	Kannada
102	Doctor Strange in the Multiverse of M...	English
103	Thor: The Dark World	English
104	Thor: Ragnarok	English
105	Thor: Love and Thunder	English
106	Sholay	Hindi
107	Dilwale Dulhania Le Jayenge	Hindi
108	3 Idiots	Hindi
109	Kabhi Khushi Kabhie Gham	Hindi
110	Bajirao Mastani	Hindi
111	The Shawshank Redemption	English
112	Inception	English
113	Interstellar	English
115	The Pursuit of Happyness	English
116	Gladiator	English
117	Titanic	English
118	It's a Wonderful Life	English
119	Avatar	English
120	The Godfather	English
121	The Dark Knight	English
122	Schindler's List	English



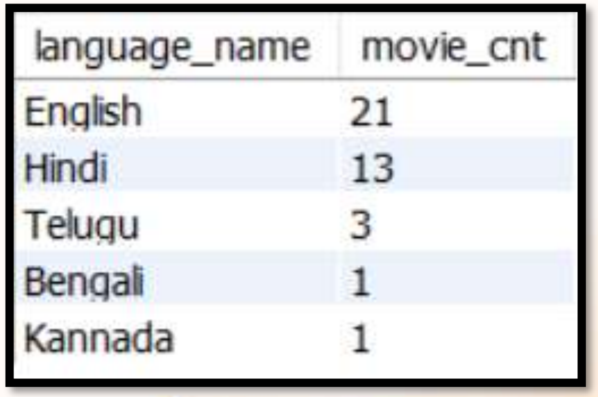
```sql
-- 2. Show all Telugu movie names (assuming you don't know the language id for Telugu)
SELECT
    movie_id,
    title,
    name
FROM movies m
JOIN languages l
    USING(language_id)
WHERE name = "Telugu"
ORDER BY movie_id;
```



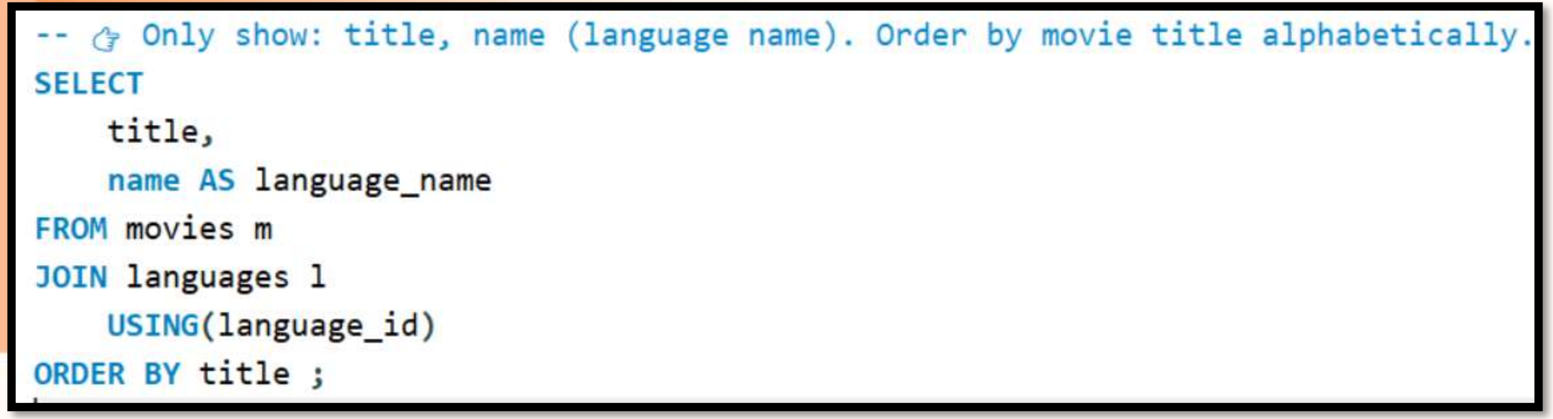
movie_id	title	name
132	Pushpa: The Rise - Part 1	Telugu
133	RRR	Telugu
134	Baahubali: The Beginning	Telugu



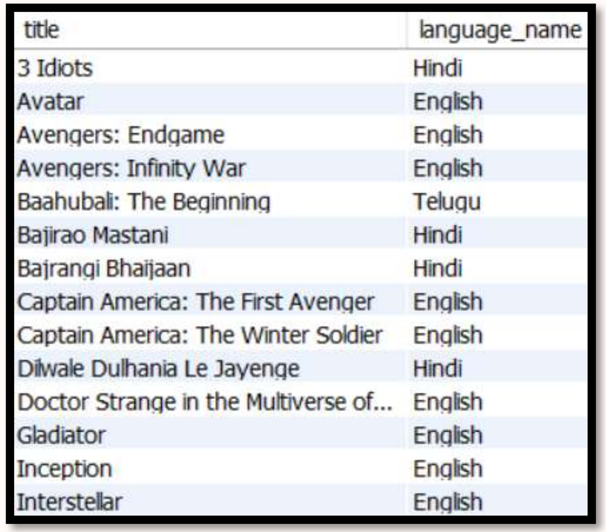
```sql
-- 3. Show the language and number of movies released in that language
SELECT
    name AS language_name,
    COUNT(title) AS movie_cnt
FROM movies m
JOIN languages l
    USING(language_id)
GROUP BY language_name
ORDER BY movie_cnt DESC;
```



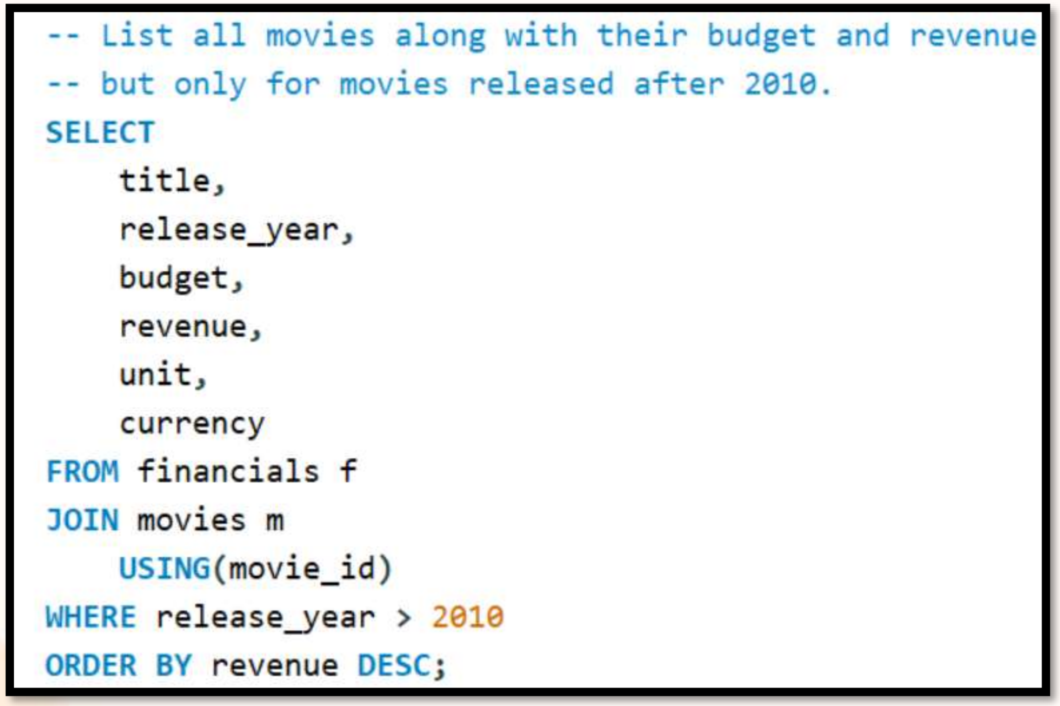
language_name	movie_cnt
English	21
Hindi	13
Telugu	3
Bengali	1
Kannada	1



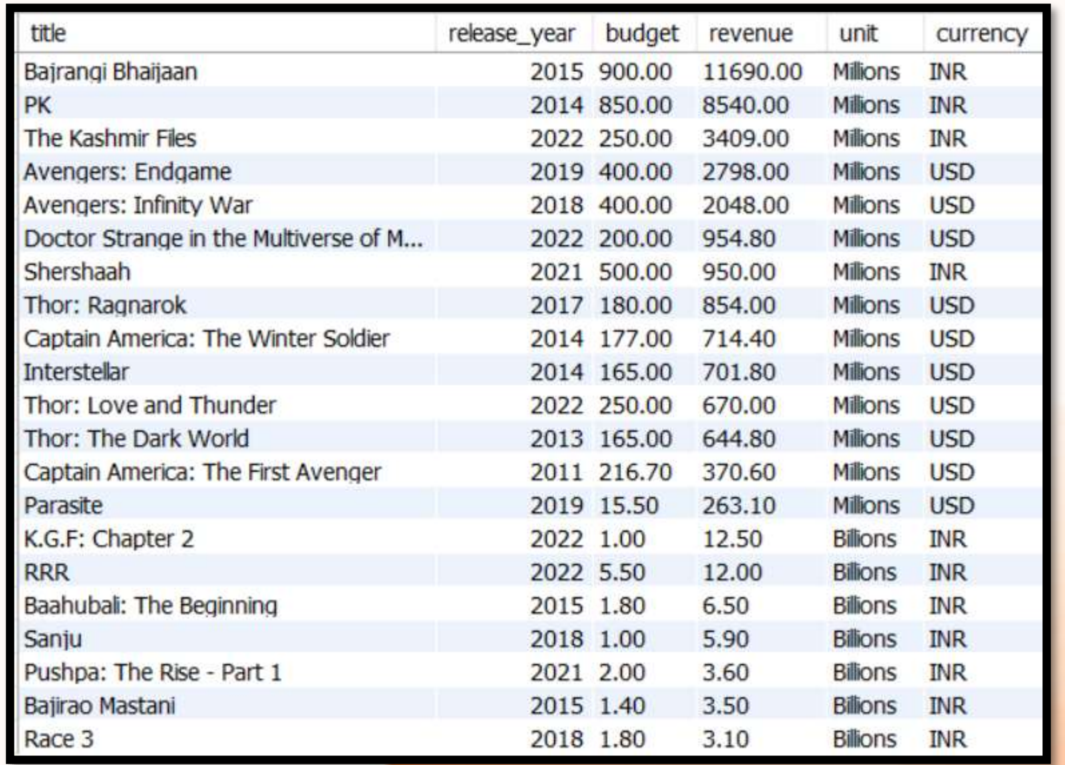
```sql
-- 👉 Only show: title, name (language name). Order by movie title alphabetically.
SELECT
    title,
    name AS language_name
FROM movies m
JOIN languages l
    USING(language_id)
ORDER BY title ;
```



title	language_name
3 Idiots	Hindi
Avatar	English
Avengers: Endgame	English
Avengers: Infinity War	English
Baahubali: The Beginning	Telugu
Bajirao Mastani	Hindi
Bajrangi Bhaijaan	Hindi
Captain America: The First Avenger	English
Captain America: The Winter Soldier	English
Dilwale Dulhania Le Jayenge	Hindi
Doctor Strange in the Multiverse of...	English
Gladiator	English
Inception	English
Interstellar	English

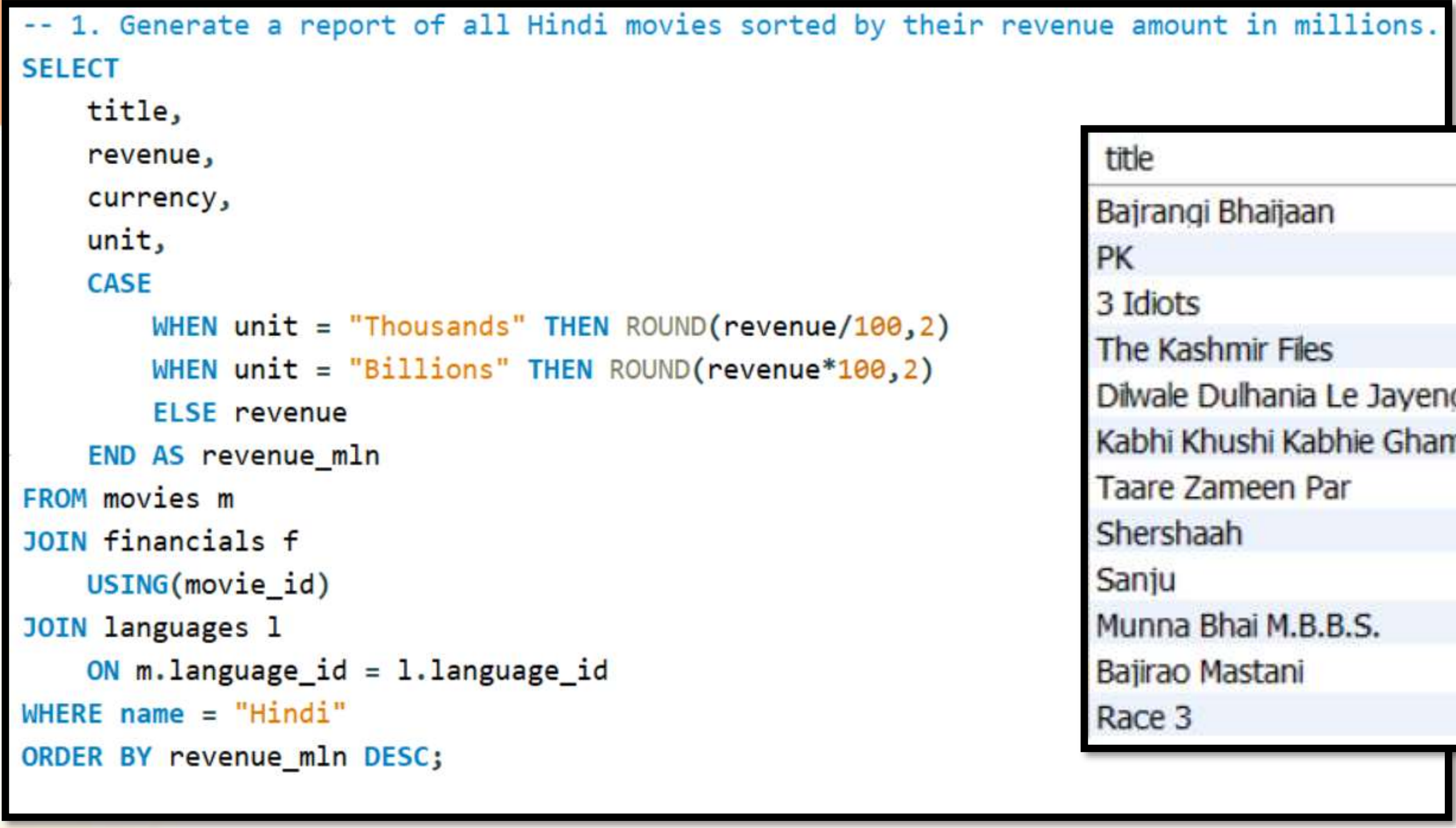


```sql
-- List all movies along with their budget and revenue
-- but only for movies released after 2010.
SELECT
    title,
    release_year,
    budget,
    revenue,
    unit,
    currency
FROM financials f
JOIN movies m
    USING(movie_id)
WHERE release_year > 2010
ORDER BY revenue DESC;
```

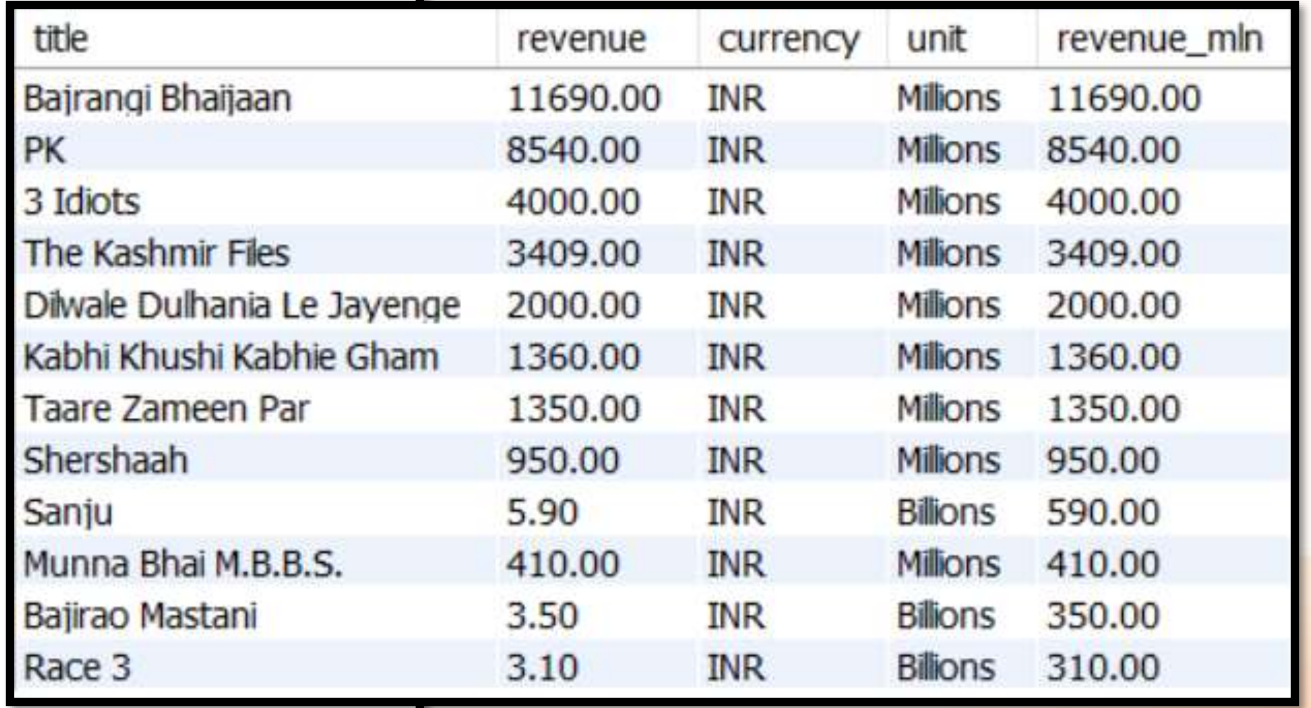


title	release_year	budget	revenue	unit	currency
Bajrangi Bhaijaan	2015	900.00	11690.00	Millions	INR
PK	2014	850.00	8540.00	Millions	INR
The Kashmir Files	2022	250.00	3409.00	Millions	INR
Avengers: Endgame	2019	400.00	2798.00	Millions	USD
Avengers: Infinity War	2018	400.00	2048.00	Millions	USD
Doctor Strange in the Multiverse of M...	2022	200.00	954.80	Millions	USD
Shershaah	2021	500.00	950.00	Millions	INR
Thor: Ragnarok	2017	180.00	854.00	Millions	USD
Captain America: The Winter Soldier	2014	177.00	714.40	Millions	USD
Interstellar	2014	165.00	701.80	Millions	USD
Thor: Love and Thunder	2022	250.00	670.00	Millions	USD
Thor: The Dark World	2013	165.00	644.80	Millions	USD
Captain America: The First Avenger	2011	216.70	370.60	Millions	USD
Parasite	2019	15.50	263.10	Millions	USD
K.G.F: Chapter 2	2022	1.00	12.50	Billions	INR
RRR	2022	5.50	12.00	Billions	INR
Baahubali: The Beginning	2015	1.80	6.50	Billions	INR
Sanju	2018	1.00	5.90	Billions	INR
Pushpa: The Rise - Part 1	2021	2.00	3.60	Billions	INR
Bajirao Mastani	2015	1.40	3.50	Billions	INR
Race 3	2018	1.80	3.10	Billions	INR

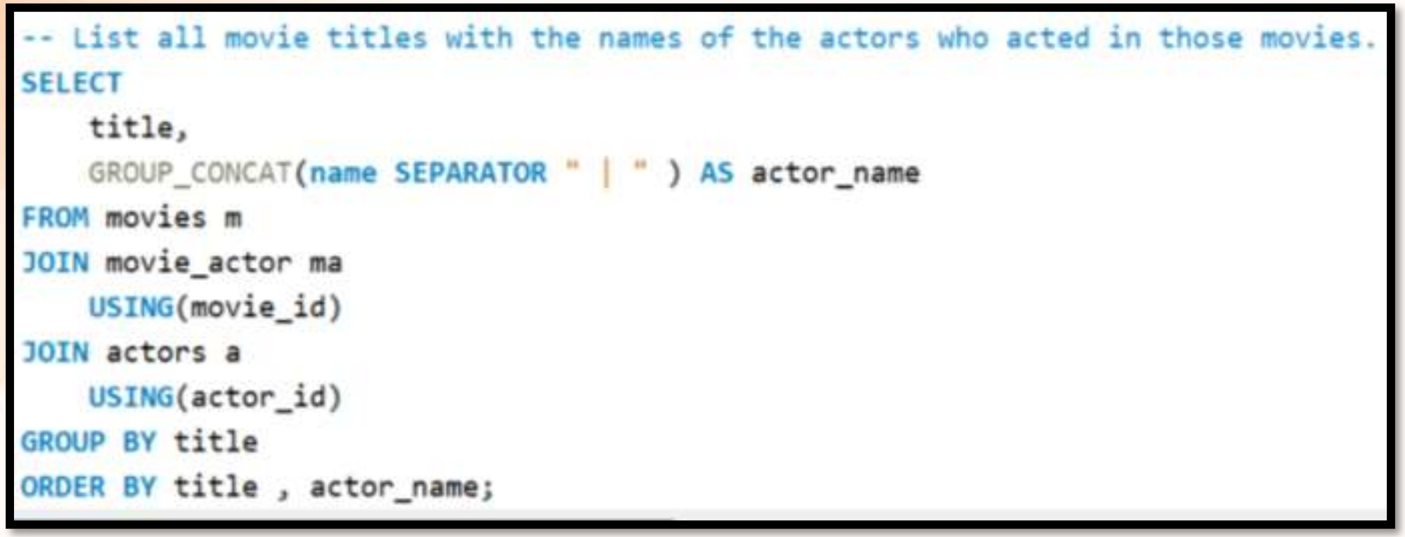
- **Using multiple joins with aggregate functions, HAVING, and ORDER BY for analysis.**



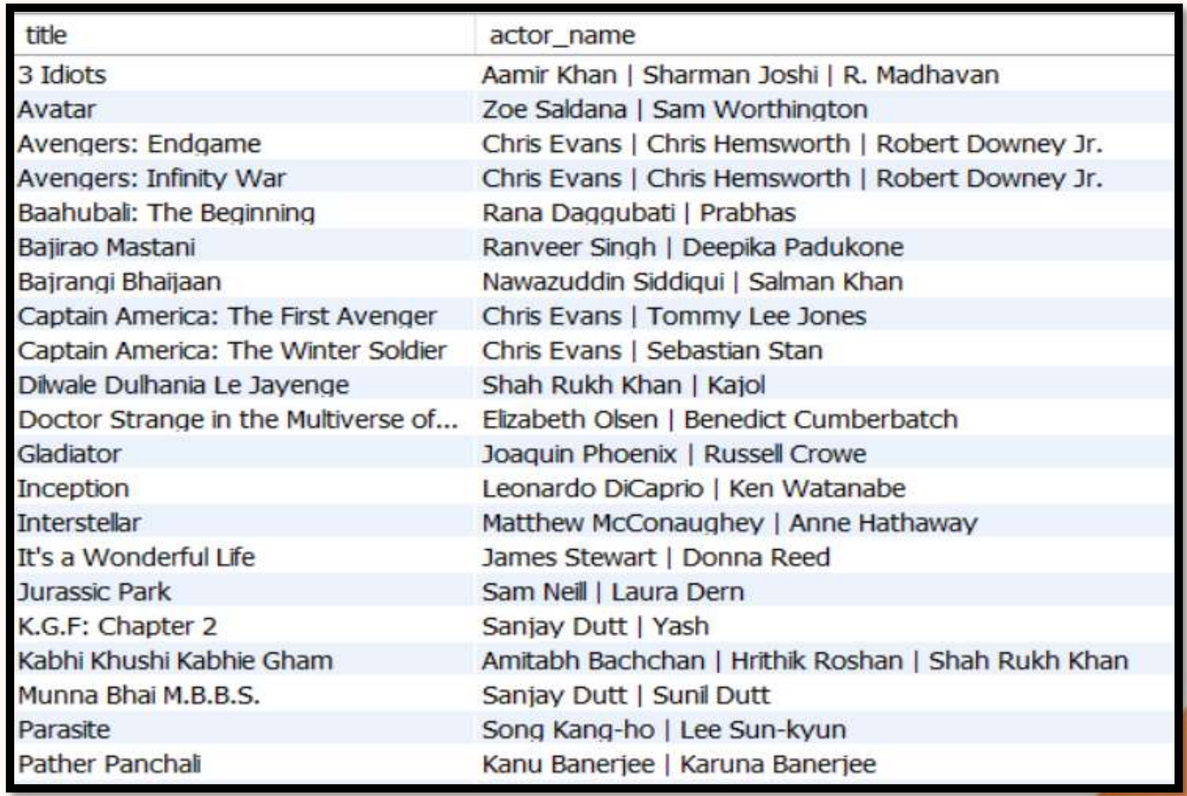
```sql
-- 1. Generate a report of all Hindi movies sorted by their revenue amount in millions.
SELECT
    title,
    revenue,
    currency,
    unit,
    CASE
        WHEN unit = "Thousands" THEN ROUND(revenue/100,2)
        WHEN unit = "Billions" THEN ROUND(revenue*100,2)
        ELSE revenue
    END AS revenue_mln
FROM movies m
JOIN financials f
    USING(movie_id)
JOIN languages l
    ON m.language_id = l.language_id
WHERE name = "Hindi"
ORDER BY revenue_mln DESC;
```



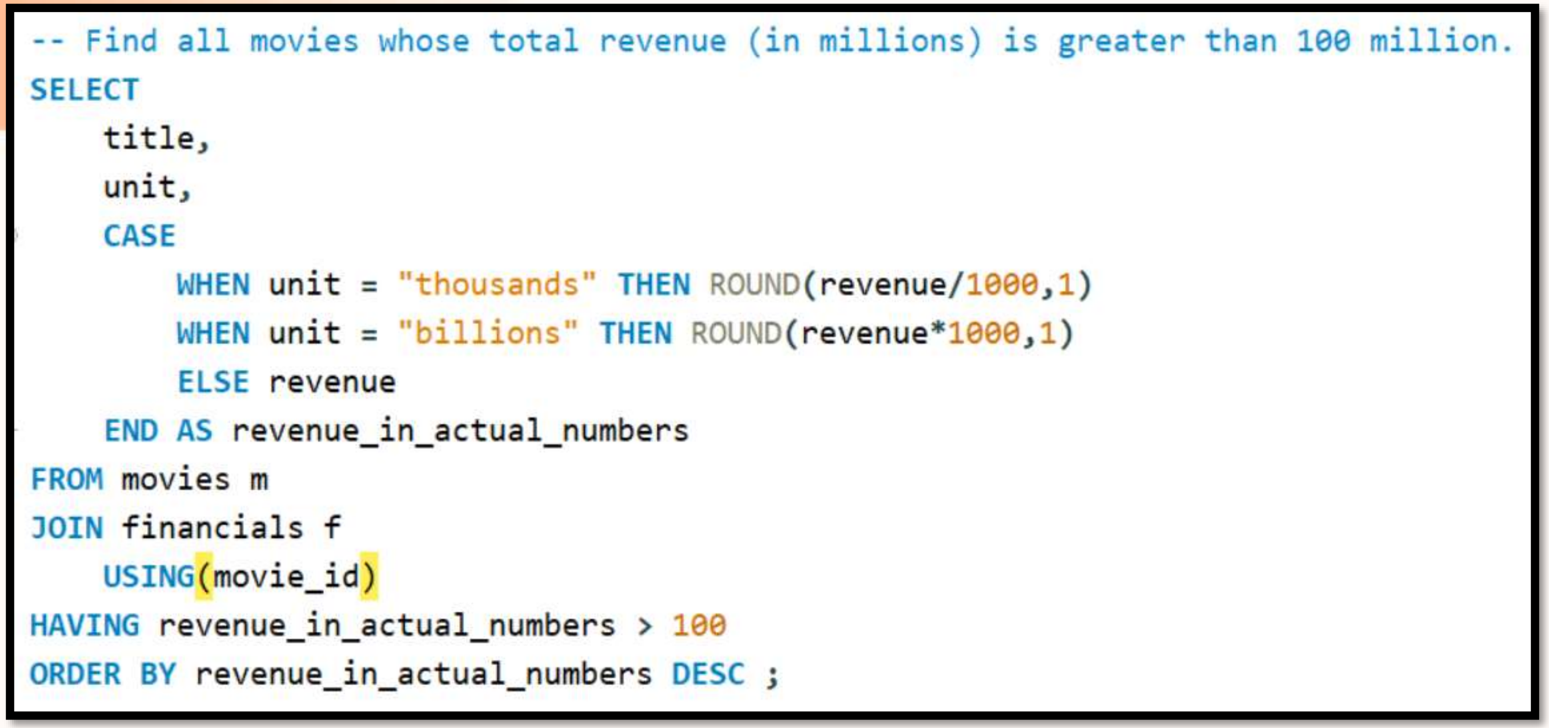
title	revenue	currency	unit	revenue_mln
Bajrangi Bhaijaan	11690.00	INR	Millions	11690.00
PK	8540.00	INR	Millions	8540.00
3 Idiots	4000.00	INR	Millions	4000.00
The Kashmir Files	3409.00	INR	Millions	3409.00
Dilwale Dulhania Le Jayenge	2000.00	INR	Millions	2000.00
Kabhi Khushi Kabhie Gham	1360.00	INR	Millions	1360.00
Taare Zameen Par	1350.00	INR	Millions	1350.00
Shershaah	950.00	INR	Millions	950.00
Sanju	5.90	INR	Billions	590.00
Munna Bhai M.B.B.S.	410.00	INR	Millions	410.00
Bajirao Mastani	3.50	INR	Billions	350.00
Race 3	3.10	INR	Billions	310.00



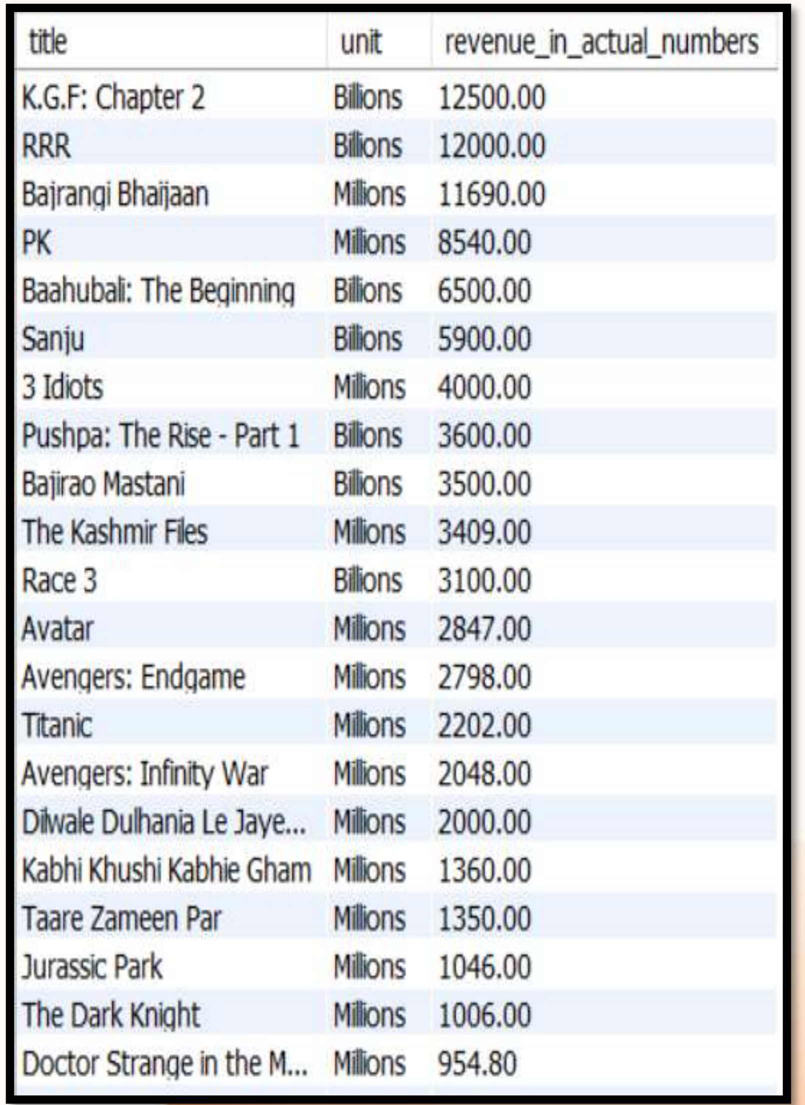
```sql
-- List all movie titles with the names of the actors who acted in those movies.
SELECT
    title,
    GROUP_CONCAT(name SEPARATOR " | " ) AS actor_name
FROM movies m
JOIN movie_actor ma
    USING(movie_id)
JOIN actors a
    USING(actor_id)
GROUP BY title
ORDER BY title , actor_name;
```



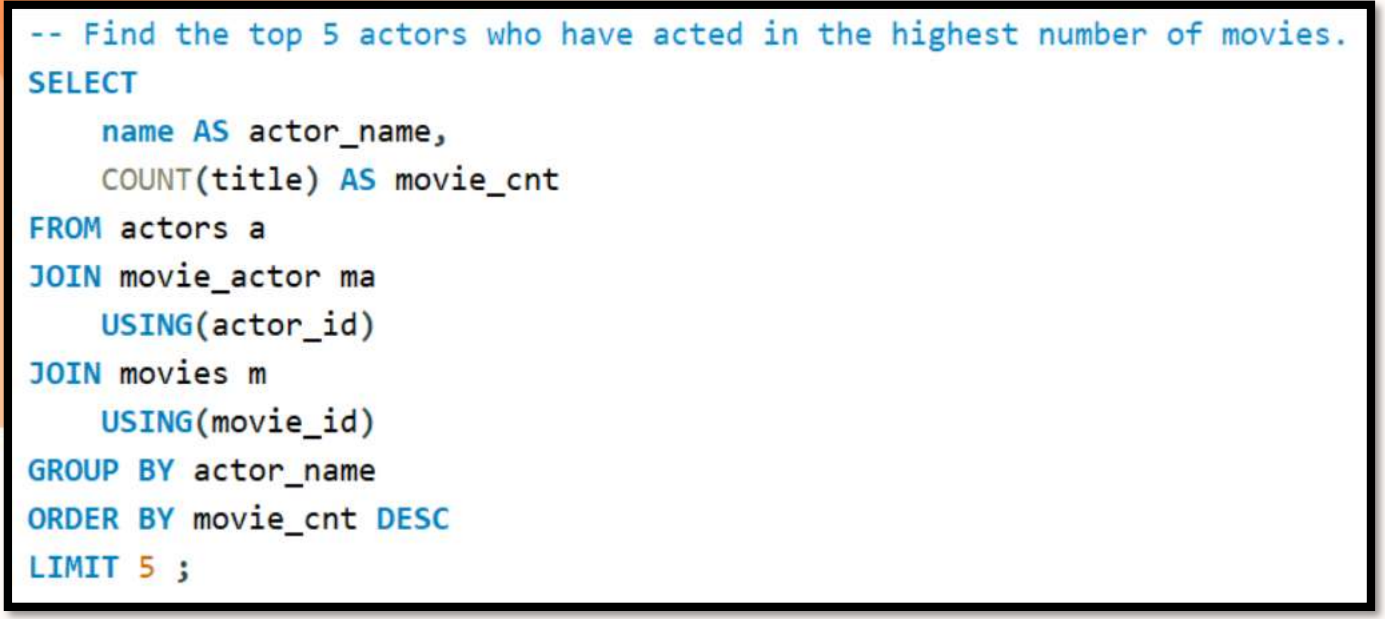
title	actor_name
3 Idiots	Aamir Khan | Sharman Joshi | R. Madhavan
Avatar	Zoe Saldana | Sam Worthington
Avengers: Endgame	Chris Evans | Chris Hemsworth | Robert Downey Jr.
Avengers: Infinity War	Chris Evans | Chris Hemsworth | Robert Downey Jr.
Baahubali: The Beginning	Rana Daggubati | Prabhas
Bajirao Mastani	Ranveer Singh | Deepika Padukone
Bajrangi Bhaijaan	Nawazuddin Siddiqui | Salman Khan
Captain America: The First Avenger	Chris Evans | Tommy Lee Jones
Captain America: The Winter Soldier	Chris Evans | Sebastian Stan
Dilwale Dulhania Le Jayenge	Shah Rukh Khan | Kajol
Doctor Strange in the Multiverse of...	Elizabeth Olsen | Benedict Cumberbatch
Gladiator	Joaquin Phoenix | Russell Crowe
Inception	Leonardo DiCaprio | Ken Watanabe
Interstellar	Matthew McConaughey | Anne Hathaway
It's a Wonderful Life	James Stewart | Donna Reed
Jurassic Park	Sam Neill | Laura Dern
K.G.F: Chapter 2	Sanjay Dutt | Yash
Kabhi Khushi Kabhie Gham	Amitabh Bachchan | Hrithik Roshan | Shah Rukh Khan
Munna Bhai M.B.B.S.	Sanjay Dutt | Sunil Dutt
Parasite	Song Kang-ho | Lee Sun-kyun
Pather Panchali	Kanu Banerjee | Karuna Banerjee



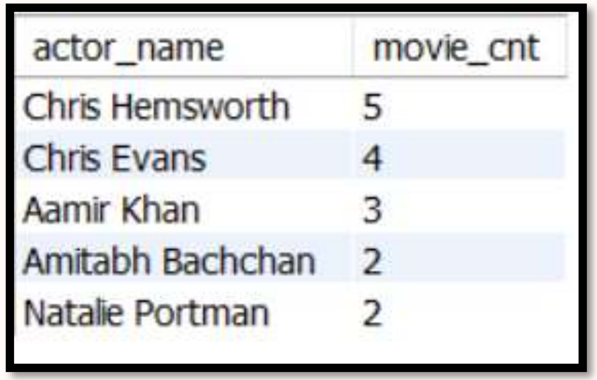
```
-- Find all movies whose total revenue (in millions) is greater than 100 million.
SELECT
    title,
    unit,
    CASE
        WHEN unit = "thousands" THEN ROUND(revenue/1000,1)
        WHEN unit = "billions" THEN ROUND(revenue*1000,1)
        ELSE revenue
    END AS revenue_in_actual_numbers
FROM movies m
JOIN financials f
    USING(movie_id)
HAVING revenue_in_actual_numbers > 100
ORDER BY revenue_in_actual_numbers DESC ;
```



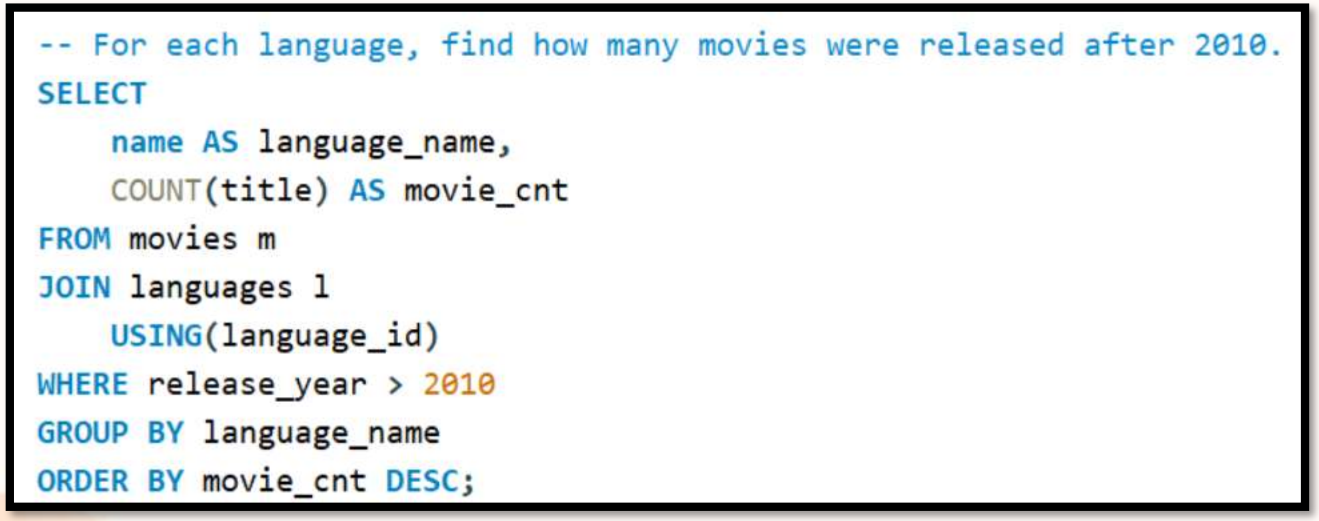
title	unit	revenue_in_actual_numbers
K.G.F: Chapter 2	Billions	12500.00
RRR	Billions	12000.00
Bajrangi Bhaijaan	Millions	11690.00
PK	Millions	8540.00
Baahubali: The Beginning	Billions	6500.00
Sanju	Billions	5900.00
3 Idiots	Millions	4000.00
Pushpa: The Rise - Part 1	Billions	3600.00
Bajirao Mastani	Billions	3500.00
The Kashmir Files	Millions	3409.00
Race 3	Billions	3100.00
Avatar	Millions	2847.00
Avengers: Endgame	Millions	2798.00
Titanic	Millions	2202.00
Avengers: Infinity War	Millions	2048.00
Dilwale Dulhania Le Jaye...	Millions	2000.00
Kabhi Khushi Kabhie Gham	Millions	1360.00
Taare Zameen Par	Millions	1350.00
Jurassic Park	Millions	1046.00
The Dark Knight	Millions	1006.00
Doctor Strange in the M...	Millions	954.80



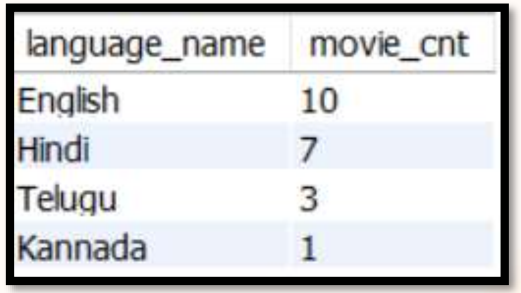
```sql
-- Find the top 5 actors who have acted in the highest number of movies.
SELECT
    name AS actor_name,
    COUNT(title) AS movie_cnt
FROM actors a
JOIN movie_actor ma
    USING(actor_id)
JOIN movies m
    USING(movie_id)
GROUP BY actor_name
ORDER BY movie_cnt DESC
LIMIT 5 ;
```



actor_name	movie_cnt
Chris Hemsworth	5
Chris Evans	4
Aamir Khan	3
Amitabh Bachchan	2
Natalie Portman	2



```sql
-- For each language, find how many movies were released after 2010.
SELECT
    name AS language_name,
    COUNT(title) AS movie_cnt
FROM movies m
JOIN languages l
    USING(language_id)
WHERE release_year > 2010
GROUP BY language_name
ORDER BY movie_cnt DESC;
```



language_name	movie_cnt
English	10
Hindi	7
Telugu	3
Kannada	1

Key Takeaways

- How tables connect using primary & foreign keys
- Writing efficient queries with joins and aggregations
- Understanding data relationships through SQL
- Practiced **INNER, LEFT, and multiple joins**

THANK YOU