Subqueries, Operators, and Derived Tables in SQL

Learning Objectives

By the end of this lesson, you will be able to:

- Identify the different join operations performed on tables
- List the various type of set operations and their usage
- Create and view subqueries with different methods
- Use derived tables for complex operations
- Verify subquery output with EXISTS operator

Use Case: Project and Team Management

Use Case: Project and Team Management

Problem Statement: You work as a junior analyst at your organization. You must help the HR department create employee information, project details, and project assignment tables in one of the databases so that managers from all departments can monitor the status and progress of ongoing projects.

Objective: Build the appropriate database and table structures for storing the required manager-specific data.

Download the **EMP_RECORDS.csv**, **PROJ_RECORDS.csv**, and **PROJ_ASSIGN.csv** files from the course resources section.

Use Case: Project and Team Management

Step 1: Create a database named **PROJ_DB** with the **CREATE DATABASE** statement.

SQL Query

```
CREATE DATABASE IF NOT EXISTS PROJ_DB;
```

Use Case: Project and Team Management

Step 2: Set **PROJ_DB** as the default database in MySQL with the **USE** statement.

SQL Query

```
USE PROJ_DB;
```

Use Case: Project and Team Management

Step 3: Set **INNODB** as the default storage engine for **PROJ_DB** database in MySQL with the **SET** statement.

SQL Query

```sql
SET default_storage_engine = INNODB;
```

Use Case: Project and Team Management

The managers have provided a detailed description of the required employee table as shown below:

Column Name	Value Type
EMP_ID	A unique ID assigned to each employee while joining the organization
MANAGER_ID	EMP_ID of the reporting manager for the project
FIRST_NAME	First name of the employee
LAST_NAME	Last name of the employee
GENDER	Gender of the employee abbreviated as M (male), F (female), and O (others)
EXP	Overall work experience of the employee
ROLE	Employee job designation
CONTINENT	Location of the branch
COUNTRY	Country of the branch
DEPT	Department of the employee

Use Case: Project and Team Management

Step 4: Create the required **EMP_RECORDS** table in the **PROJ_DB** database with the **CREATE TABLE** statement as shown below:

SQL Query

```
CREATE TABLE IF NOT EXISTS PROJ_DB.EMP_RECORDS (

    EMP_ID VARCHAR(4) NOT NULL PRIMARY KEY,

    FIRST_NAME VARCHAR(100) NOT NULL,

    LAST_NAME VARCHAR(100),

    GENDER VARCHAR(1) NOT NULL,

    ROLE VARCHAR(100) NOT NULL,

    DEPT VARCHAR(100) NOT NULL,

    EXP INTEGER NOT NULL CHECK (EXP >= 0),

    COUNTRY VARCHAR(80) NOT NULL,

    CONTINENT VARCHAR(50) NOT NULL,

    MANAGER_ID VARCHAR(100),

    CONSTRAINT empid_check CHECK ( SUBSTR(EMP_ID,1,1) = 'E'),

    CONSTRAINT gender_check CHECK(GENDER in ('M', 'F', 'O'))

)  ENGINE=INNODB;
```

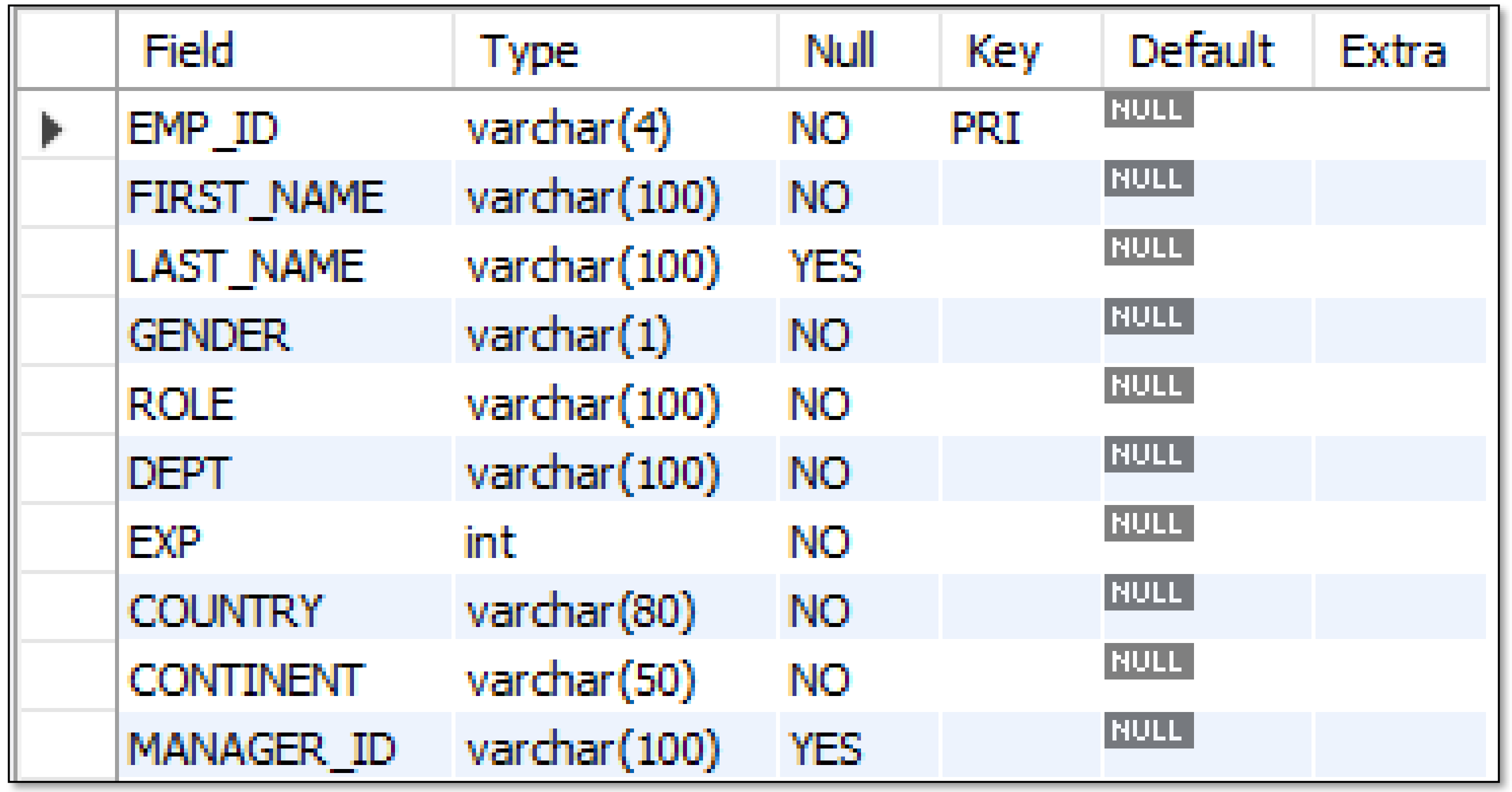
Use Case: Project and Team Management

Step 5: Analyze the structure of the **EMP_RECORDS** table with the **DESCRIBE** statement.

SQL Query

```
DESCRIBE PROJ_DB.EMP_RECORDS;
```

Output:



Field	Type	Null	Key	Default	Extra
EMP_ID	varchar(4)	NO	PRI	NULL	
FIRST_NAME	varchar(100)	NO		NULL	
LAST_NAME	varchar(100)	YES		NULL	
GENDER	varchar(1)	NO		NULL	
ROLE	varchar(100)	NO		NULL	
DEPT	varchar(100)	NO		NULL	
EXP	int	NO		NULL	
COUNTRY	varchar(80)	NO		NULL	
CONTINENT	varchar(50)	NO		NULL	
MANAGER_ID	varchar(100)	YES		NULL	

Use Case: Project and Team Management

Step 6: Insert the required data from the downloaded **EMP_RECORDS.csv** file into the **EMP_RECORDS** table as shown below:

SQL Query

```
INSERT INTO
PROJ_DB.EMP_RECORDS(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT,EXP,COUNTRY,CONT
INENT,MANAGER_ID)

VALUES

("E083","Patrick","Voltz","M","MANAGER","HEALTHCARE",15,"USA","NORTH
AMERICA","E002"),

("E052","Dianna","Wilson","F","SENIOR DATA
SCIENTIST","HEALTHCARE",6,"CANADA","NORTH AMERICA","E083"),

...

("E002","Cynthia","Brooks","F","PRESIDENT","ALL",17,"CANADA","NORTH
AMERICA","E001"),

("E001","Arthur","Black","M","CEO","ALL",20,"USA","NORTH AMERICA","E001");
```

Note: You can download the dataset directly or use the insert function to populate values in the table.

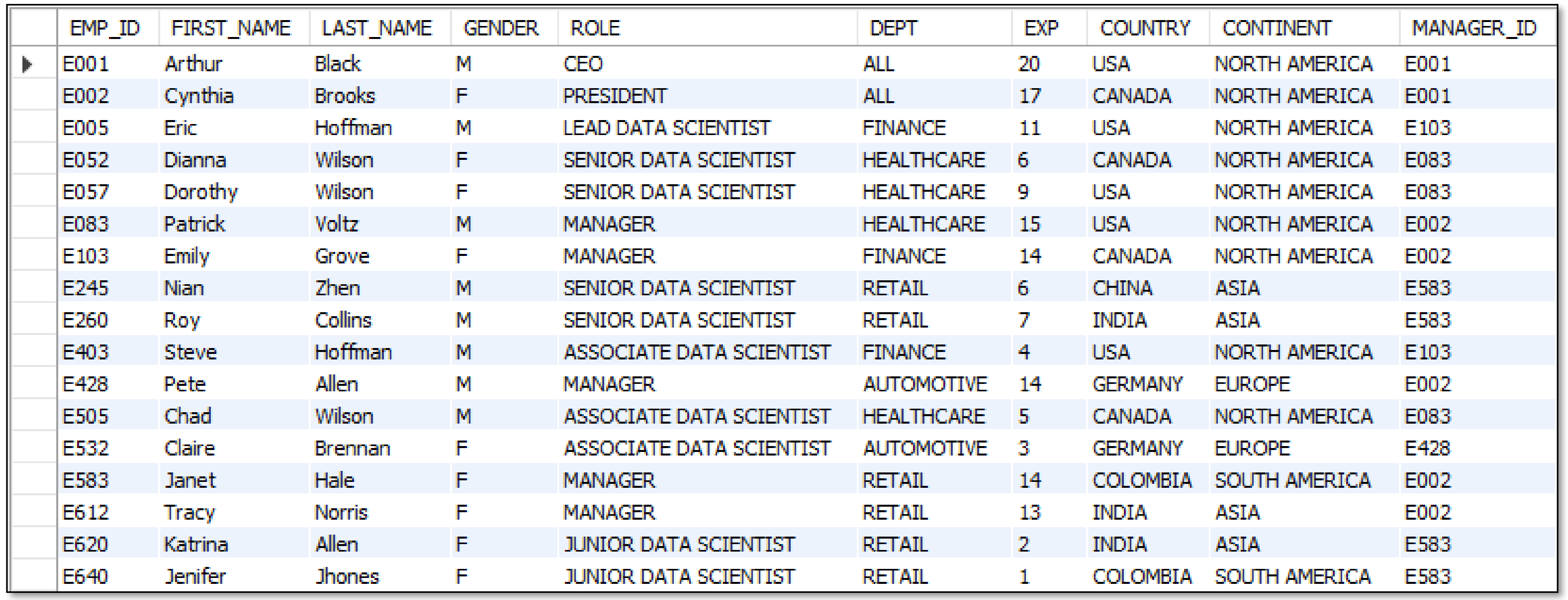
Use Case: Project and Team Management

Step 7: Analyze the data entered into the **EMP_RECORDS** table with the **SELECT** statement.

SQL Query

```
SELECT * FROM PROJ_DB.EMP_RECORDS;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	EXP	COUNTRY	CONTINENT	MANAGER_ID
E001	Arthur	Black	M	CEO	ALL	20	USA	NORTH AMERICA	E001
E002	Cynthia	Brooks	F	PRESIDENT	ALL	17	CANADA	NORTH AMERICA	E001
E005	Eric	Hoffman	M	LEAD DATA SCIENTIST	FINANCE	11	USA	NORTH AMERICA	E103
E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	6	CANADA	NORTH AMERICA	E083
E057	Dorothy	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	9	USA	NORTH AMERICA	E083
E083	Patrick	Voltz	M	MANAGER	HEALTHCARE	15	USA	NORTH AMERICA	E002
E103	Emily	Grove	F	MANAGER	FINANCE	14	CANADA	NORTH AMERICA	E002
E245	Nian	Zhen	M	SENIOR DATA SCIENTIST	RETAIL	6	CHINA	ASIA	E583
E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	7	INDIA	ASIA	E583
E403	Steve	Hoffman	M	ASSOCIATE DATA SCIENTIST	FINANCE	4	USA	NORTH AMERICA	E103
E428	Pete	Allen	M	MANAGER	AUTOMOTIVE	14	GERMANY	EUROPE	E002
E505	Chad	Wilson	M	ASSOCIATE DATA SCIENTIST	HEALTHCARE	5	CANADA	NORTH AMERICA	E083
E532	Claire	Brennan	F	ASSOCIATE DATA SCIENTIST	AUTOMOTIVE	3	GERMANY	EUROPE	E428
E583	Janet	Hale	F	MANAGER	RETAIL	14	COLOMBIA	SOUTH AMERICA	E002
E612	Tracy	Norris	F	MANAGER	RETAIL	13	INDIA	ASIA	E002
E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	2	INDIA	ASIA	E583
E640	Jenifer	Jhones	F	JUNIOR DATA SCIENTIST	RETAIL	1	COLOMBIA	SOUTH AMERICA	E583

Use Case: Project and Team Management

Similarly, the managers have provided a detailed description of the required projects table shown below:

Column Name	Value Type
PROJ_ID	A unique ID assigned to each project after planning
PROJ_NAME	A unique name assigned to each project based on the client requirements and application domain
START _DATE	The planned date within the DEV_QTR for starting the project development (Format - YYYY-MM-DD)
CLOSURE_DATE	The planned date within the DEV_QTR for finishing the project development (Format - YYYY-MM-DD)
DOMAIN	The domain of focus for the project
DEV_QTR	The quarter in the current financial year selected for end-to-end development of the project
STATUS	The development status of the project as YTS (Yet To Start), WIP (Work In Progress), DONE (Finished on time), and DELAYED (Delayed and Ongoing)

Use Case: Project and Team Management

Step 8: Create the required **PROJ_RECORDS** table in the **PROJ_DB** database with the **CREATE TABLE** statement as shown below:

SQL Query

```
CREATE TABLE IF NOT EXISTS PROJ_DB.PROJ_RECORDS (

    PROJ_ID VARCHAR(4) NOT NULL PRIMARY KEY,

    PROJ_NAME VARCHAR(200) NOT NULL,

    DOMAIN VARCHAR(100) NOT NULL,

    START_DATE DATE NOT NULL,

    CLOSURE_DATE DATE NOT NULL,

    DEV_QTR VARCHAR(2) NOT NULL,

    STATUS VARCHAR(7),

    CONSTRAINT projid_check CHECK ( SUBSTR(PROJ_ID,1,1) = 'P'),

    CONSTRAINT check_start_date CHECK (START_DATE >= '2021-04-01'),

    CONSTRAINT check_closure_date CHECK (CLOSURE_DATE <= '2022-03-30'),

    CONSTRAINT chk_qtr CHECK (DEV_QTR IN ('Q1', 'Q2', 'Q3', 'Q4')),

    CONSTRAINT chk_status CHECK (STATUS IN ('YTS', 'WIP', 'DONE', 'DELAYED'))

)  ENGINE=INNODB;
```

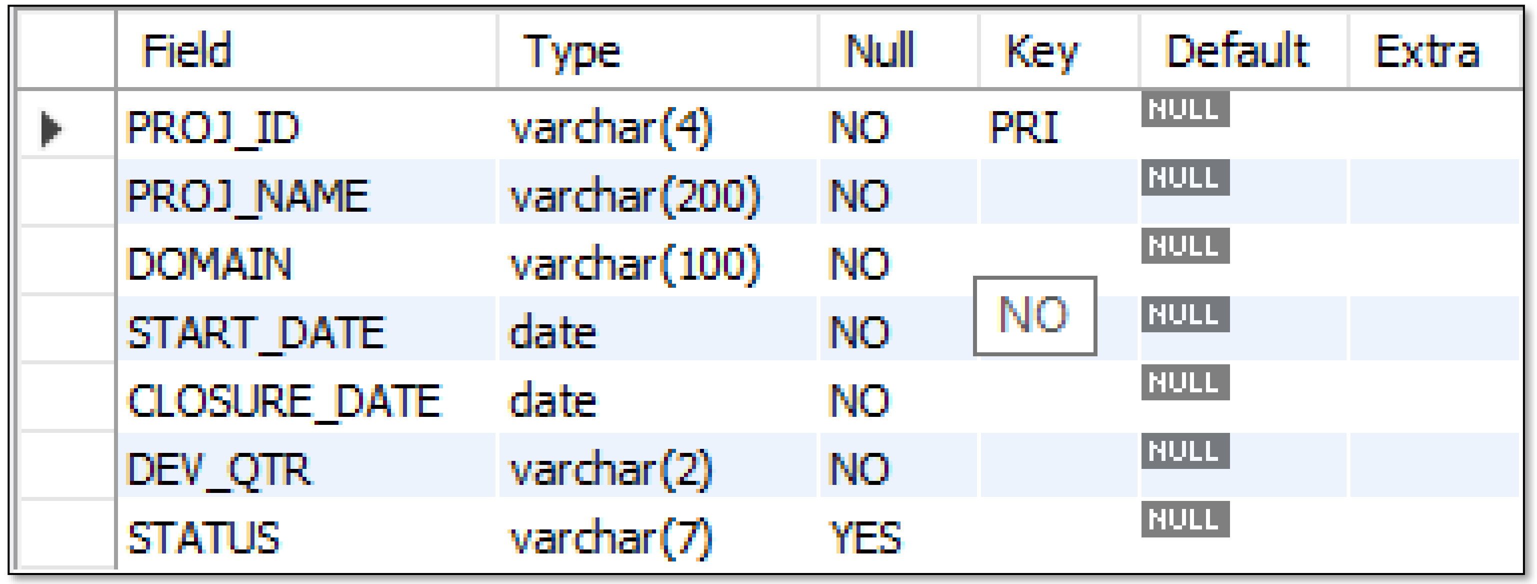
Use Case: Project and Team Management

Step 9: Analyze the structure of the **EMP_RECORDS** table with the **DESCRIBE** statement.

SQL Query

```
DESCRIBE PROJ_DB.PROJ_RECORDS;
```

Output:



Field	Type	Null	Key	Default	Extra
PROJ_ID	varchar(4)	NO	PRI	NULL	
PROJ_NAME	varchar(200)	NO		NULL	
DOMAIN	varchar(100)	NO		NULL	
START_DATE	date	NO	NO	NULL	
CLOSURE_DATE	date	NO		NULL	
DEV_QTR	varchar(2)	NO		NULL	
STATUS	varchar(7)	YES		NULL	

Use Case: Project and Team Management

Step 10: Insert the required data from the downloaded **PROJ_RECORDS.csv** file into the **PROJ_RECORDS** table as shown below:

SQL Query

```
INSERT INTO

PROJ_DB.PROJ_RECORDS(PROJ_ID,PROJ_NAME,DOMAIN,START_DATE,CLOSURE_DATE,DEV_QTR,STA
TUS)

VALUES

("P103","Drug Discovery","HEALTHCARE","2021-04-06","2021-06-20","Q1","DONE"),

("P105","Fraud Detection","FINANCE","2021-04-11","2021-06-25","Q1","DONE"),

...

...

("P109","Market Basket Analysis","RETAIL","2021-04-12","2021-06-
21","Q1","DELAYED");
```

Note: You can download the dataset directly or use the insert function to populate values in the table.

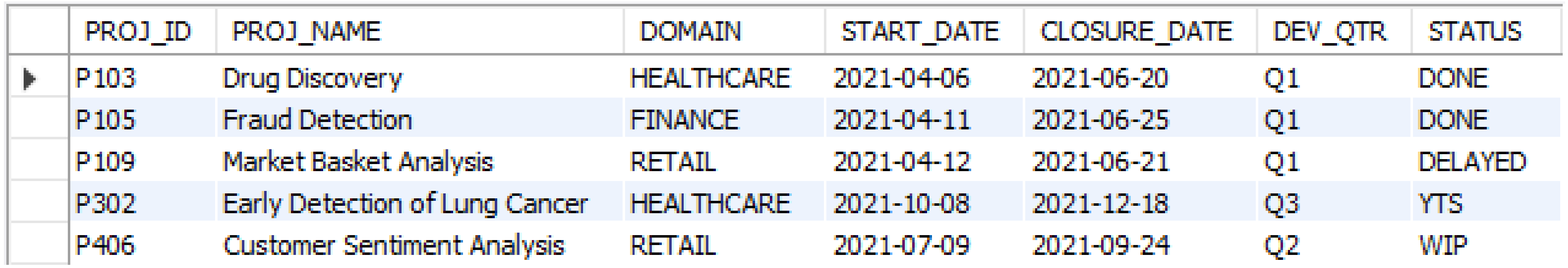
Use Case: Project and Team Management

Step 11: Analyze the data entered into the **PROJ_RECORDS** table with the **SELECT** statement.

SQL Query

```
SELECT * FROM PROJ_DB.PROJ_RECORDS;
```

Output:



	PROJ_ID	PROJ_NAME	DOMAIN	START_DATE	CLOSURE_DATE	DEV_QTR	STATUS
▶	P103	Drug Discovery	HEALTHCARE	2021-04-06	2021-06-20	Q1	DONE
	P105	Fraud Detection	FINANCE	2021-04-11	2021-06-25	Q1	DONE
	P109	Market Basket Analysis	RETAIL	2021-04-12	2021-06-21	Q1	DELAYED
	P302	Early Detection of Lung Cancer	HEALTHCARE	2021-10-08	2021-12-18	Q3	YTS
	P406	Customer Sentiment Analysis	RETAIL	2021-07-09	2021-09-24	Q2	WIP

Use Case: Project and Team Management

Also, the managers have provided a detailed description of the required project assignment table, which links each individual's employee ID to the project ID of the project to which they have been assigned.

Column Name	Value Type
EMP_ID	A unique ID assigned to each employee while joining the organization
PROJ_ID	A unique ID assigned to each project after planning

Use Case: Project and Team Management

Step 12: Create the required **PROJ_ASSIGN** table in the **PROJ_DB** database with the **CREATE TABLE** statement as shown below:

SQL Query

```
CREATE TABLE IF NOT EXISTS PROJ_DB.PROJ_ASSIGN (
    EMP_ID VARCHAR(4) NOT NULL,
    PROJ_ID VARCHAR(4) NOT NULL,
    CONSTRAINT empid_check_2 CHECK ( SUBSTR(EMP_ID,1,1) = 'E'),
    CONSTRAINT projid_check_2 CHECK ( SUBSTR(PROJ_ID,1,1) = 'P'),
    FOREIGN KEY(EMP_ID) REFERENCES PROJ_DB.EMP_RECORDS(EMP_ID),
    FOREIGN KEY(PROJ_ID) REFERENCES PROJ_DB.PROJ_RECORDS(PROJ_ID)
)  ENGINE=INNODB;
```

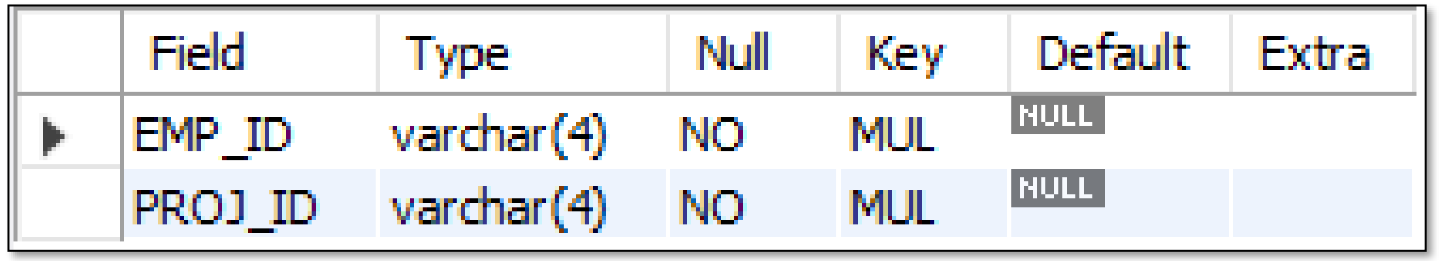
Use Case: Project and Team Management

Step 13: Analyze the structure of the **PROJ_ASSIGN** table with the **DESCRIBE** statement.

SQL Query

```
DESCRIBE PROJ_DB.PROJ_ASSIGN;
```

Output:



Field	Type	Null	Key	Default	Extra
EMP_ID	varchar(4)	NO	MUL	NULL	
PROJ_ID	varchar(4)	NO	MUL	NULL	

Use Case: Project and Team Management

Step 14: Insert the required data from the downloaded **PROJ_ASSIGN.csv** file into the **PROJ_RECORDS** table as shown below.

SQL Query

```
INSERT INTO PROJ_DB.PROJ_ASSIGN(EMP_ID,PROJ_ID)
VALUES ("E052","P103"),
       ("E505","P103"),
        ...
        ...
       ("E083","P103");
```

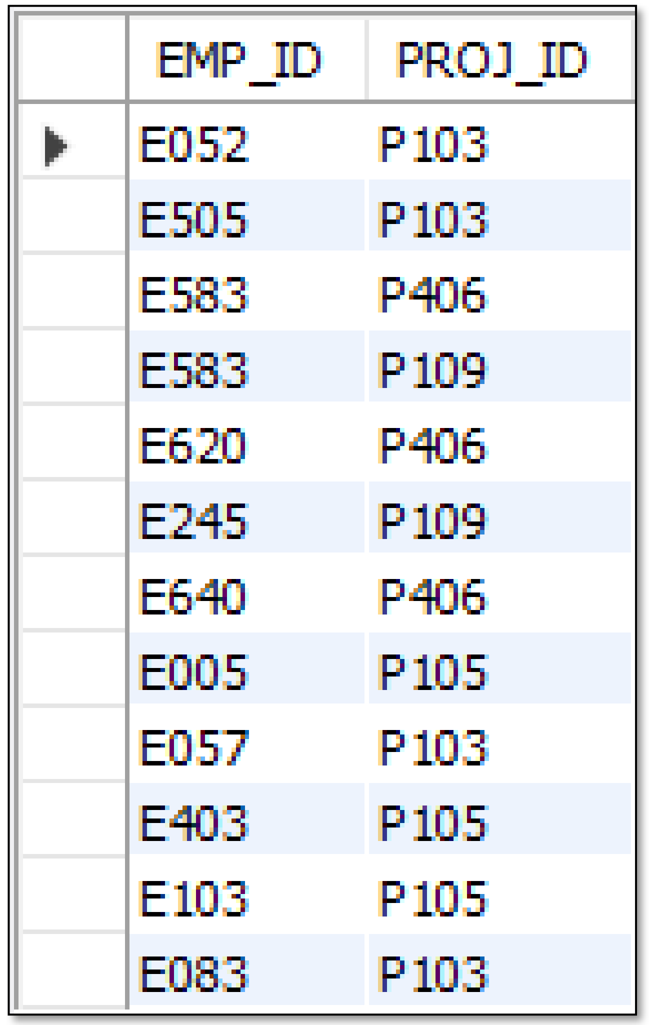
Use Case: Project and Team Management

Step 15: Analyze the data entered into the **PROJ_ASSIGN** table with the **SELECT** statement.

SQL Query

```
SELECT * FROM PROJ_DB.PROJ_ASSIGN;
```

Output:



EMP_ID	PROJ_ID
E052	P103
E505	P103
E583	P406
E583	P109
E620	P406
E245	P109
E640	P406
E005	P105
E057	P103
E403	P105
E103	P105
E083	P103

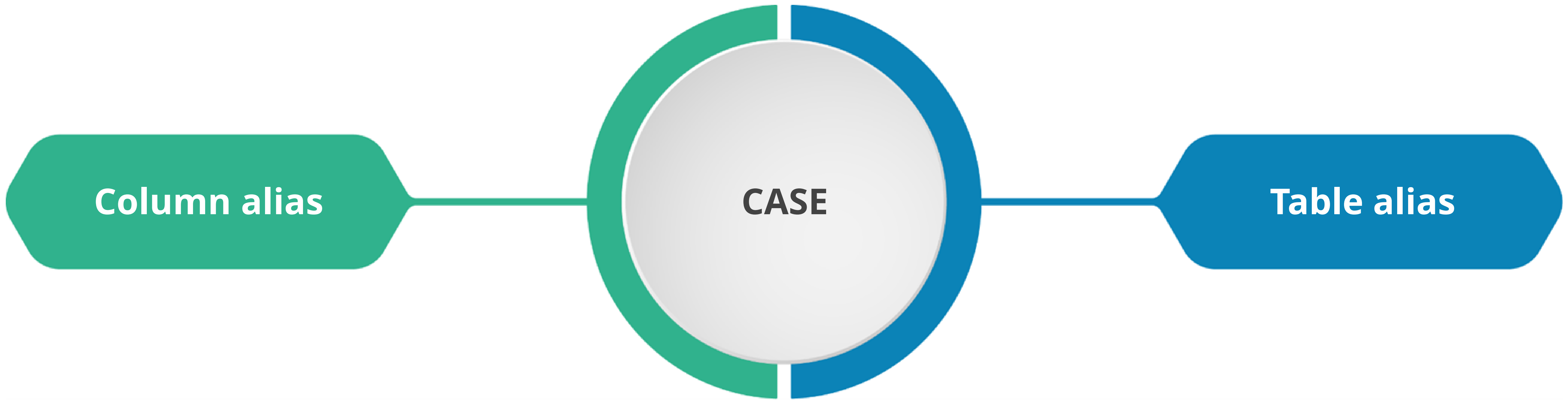
Introduction to Alias

Alias in SQL

SQL subquery

- An alias is a temporary name assigned to a table, column, or expression for the purpose of an SQL query, and it exists only for the duration of the query.
- The AS keyword is used to make an alias.
- If an alias contains space, it must be enclosed in quotation marks.
- Aliases are typically used to make column names easier to comprehend.

Types of Alias



Column Alias

A column alias is a temporary name assigned to a column having a long or descriptive technical name in order to simplify the query output.

Syntax

```
SELECT
   [column_1 | expression] AS
column_alias_name
FROM
   table_name;
```

A column alias cannot be used in the WHERE clause because the WHERE clause is evaluated before the values of columns specified in the SELECT statement during query execution in MySQL.

Column Alias

Problem Statement: Your manager expects you to identify the last and first names of all the employees from the healthcare department.

Objective: Write two MySQL queries, one omitting the column alias and the other including it to list the last and first names of all employees of the healthcare department.

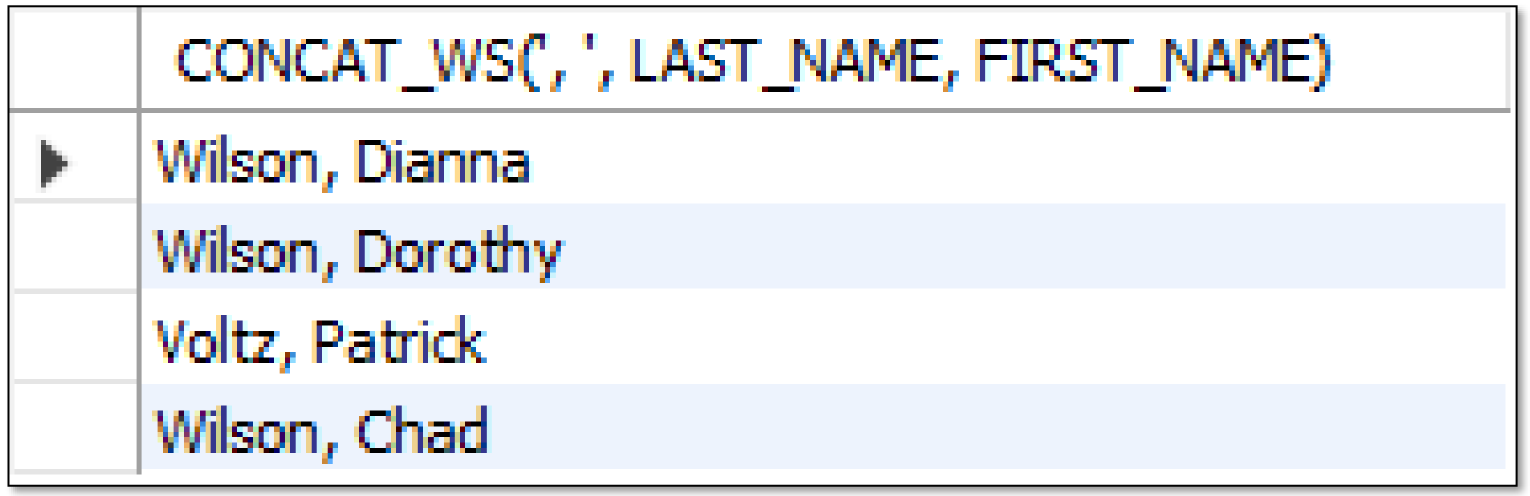
Column Alias: Example

Step 1: Use the **CONCAT_WS** function in the **SELECT** statement as shown below:

SQL Query

```
SELECT CONCAT_WS(', ', LAST_NAME, FIRST_NAME)
FROM PROJ_DB.EMP_RECORDS
WHERE DEPT = "HEALTHCARE";
```

Output:



```
CONCAT_WS(', ', LAST_NAME, FIRST_NAME)
Wilson, Dianna
Wilson, Dorothy
Voltz, Patrick
Wilson, Chad
```

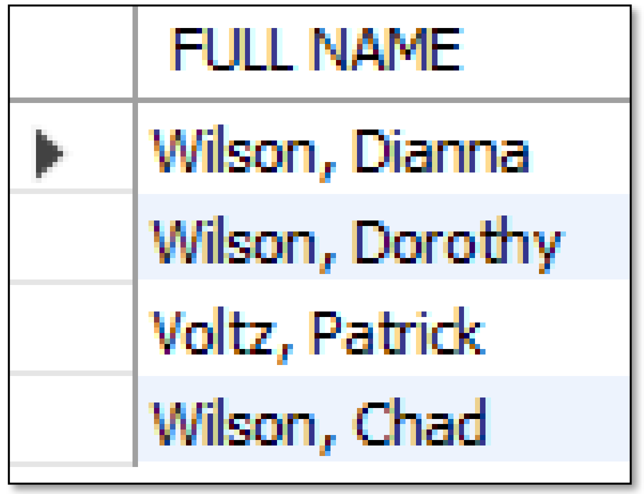
Column Alias: Example

Step 2: Use the **CONCAT_WS** function in the **SELECT** statement followed by the **AS** keyword as shown below:

SQL Query

```
SELECT CONCAT_WS(', ', LAST_NAME, FIRST_NAME) AS
`FULL NAME`

FROM PROJ_DB.EMP_RECORDS

WHERE DEPT = "HEALTHCARE";
```

Output:



```
FULL NAME
Wilson, Dianna
Wilson, Dorothy
Voltz, Patrick
Wilson, Chad
```

Table Alias

A table alias is a temporary name assigned to a table that has a descriptive technical name to simplify the query output.

Syntax

```
SELECT
    [column_1 | expression], column_2
FROM
    table_name AS table_alias_name;
```

Table aliases are preferred when there are multiple tables involved in an SQL query. It helps in collecting data and connecting the tables via field relations.

Table Alias

Problem Statement: Your manager expects you to identify the first and last names of all the employees separately from the healthcare department.

Objective: Write a MySQL queries to list the first name and last name of all employees from the healthcare department using a table alias to refer to the table.

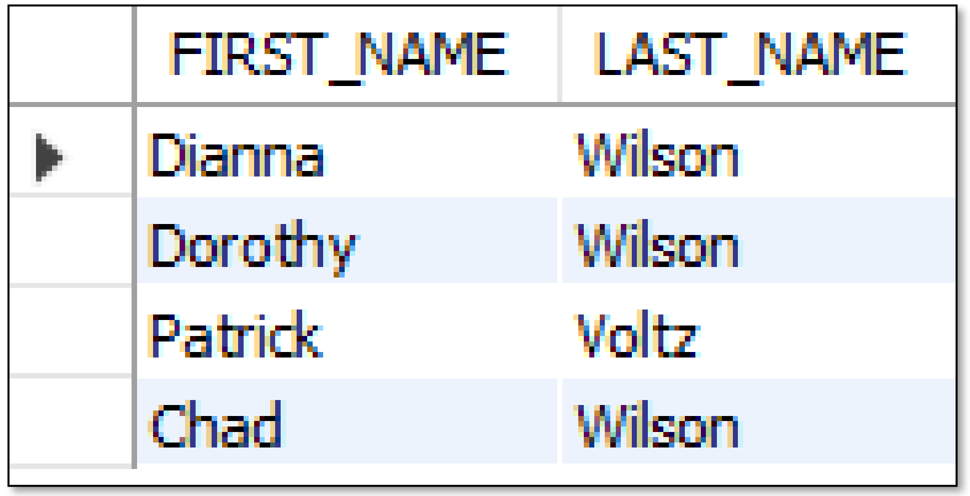
Table Alias: Example

Step 1: Use the **SELECT** statement with an alias name for the table as shown below:

SQL Query – Par 1

```
SELECT e.FIRST_NAME, e.LAST_NAME
FROM PROJ_DB.EMP_RECORDS AS e
WHERE e.DEPT = "HEALTHCARE";
```

Output:

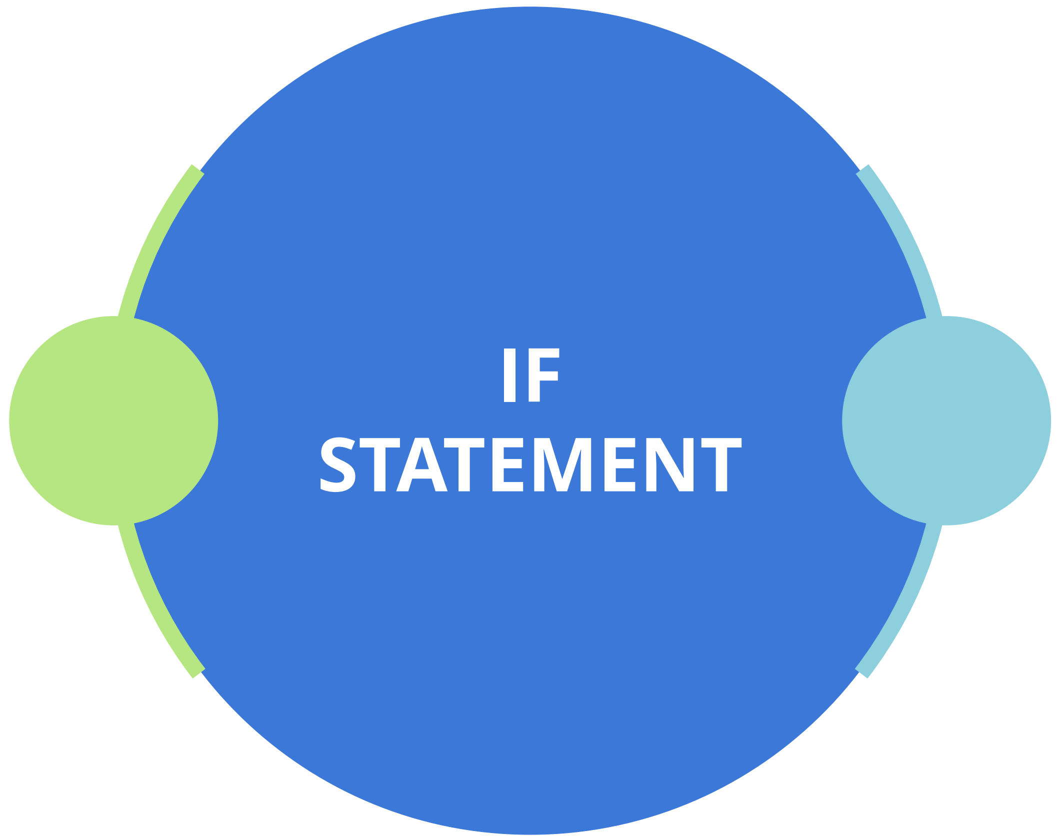


FIRST_NAME	LAST_NAME
Dianna	Wilson
Dorothy	Wilson
Patrick	Voltz
Chad	Wilson

Introduction to JOINS

JOINS in MySQL

A JOIN is a method of linking data between one (self-join) or more tables based on a related column between them.



Joins are frequently used in SELECT, UPDATE, and DELETE statements. It is also used in subqueries to join multiple tables.

Types of JOINS

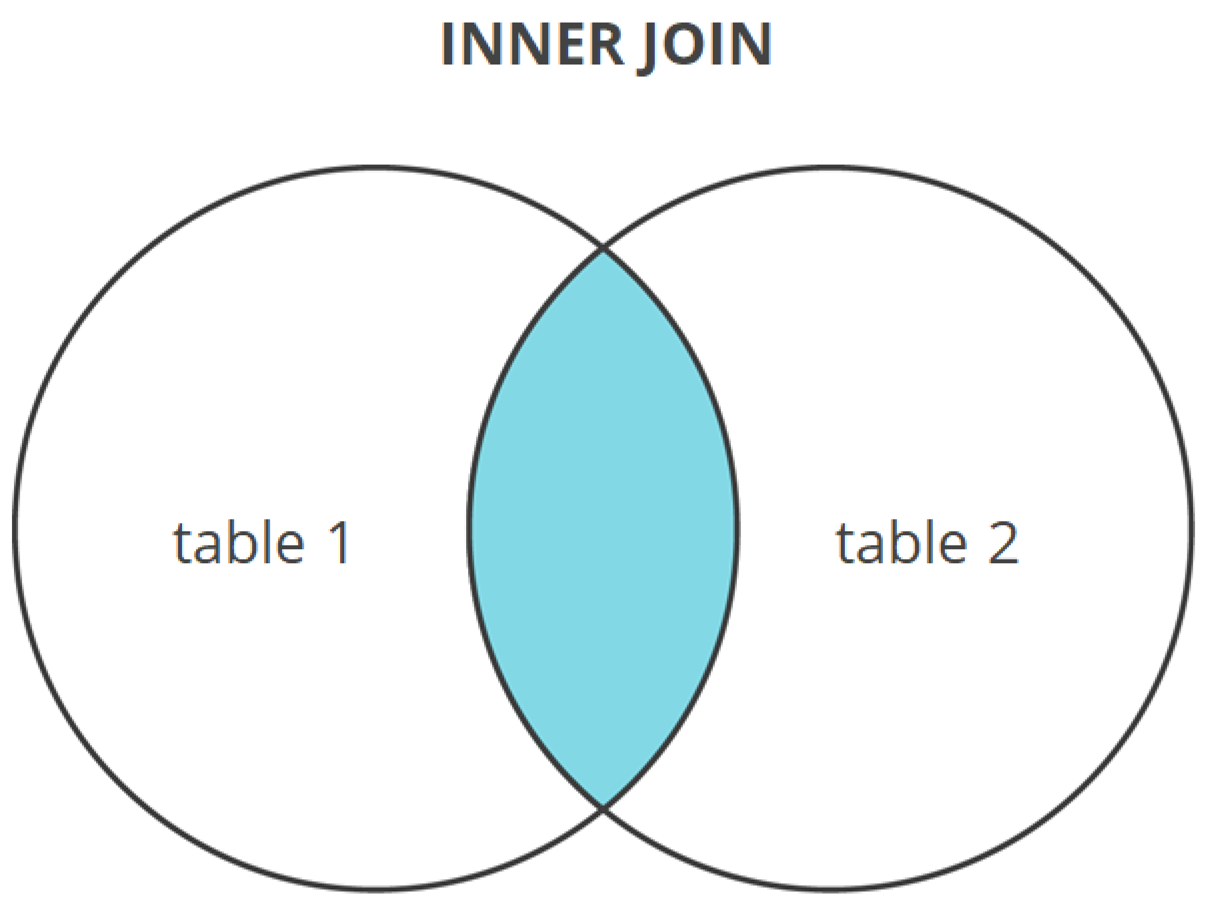
Full Outer Join

Inner Join

Right (Outer) Join

Left (Outer) Join

INNER JOIN



- The INNER JOIN clause joins two tables based on a condition which is known as a join predicate.
- It returns only the matching rows from both tables.
- The column names are enclosed in the USING clause if the JOIN condition utilizes the equal operator (=) and the column names in both tables, used for matching, are the same.

INNER JOIN

Syntax 01 - INNER JOIN

```
SELECT column_list
FROM table_1
INNER JOIN table_2 ON join_condition;
```

Syntax 02 - INNER JOIN with USING

```
SELECT column_list
FROM table_1
INNER JOIN table_2 USING (column_name);
```

INNER JOIN: Example

Problem Statement: Your manager expects you to identify employees assigned to projects.

Objective: Write an SQL query using the **INNER JOIN** clause with either **ON** or **USING** keyword to perform the inner join on the **EMP_RECORDS** and **PROJ_ASSIGN** tables in MySQL.

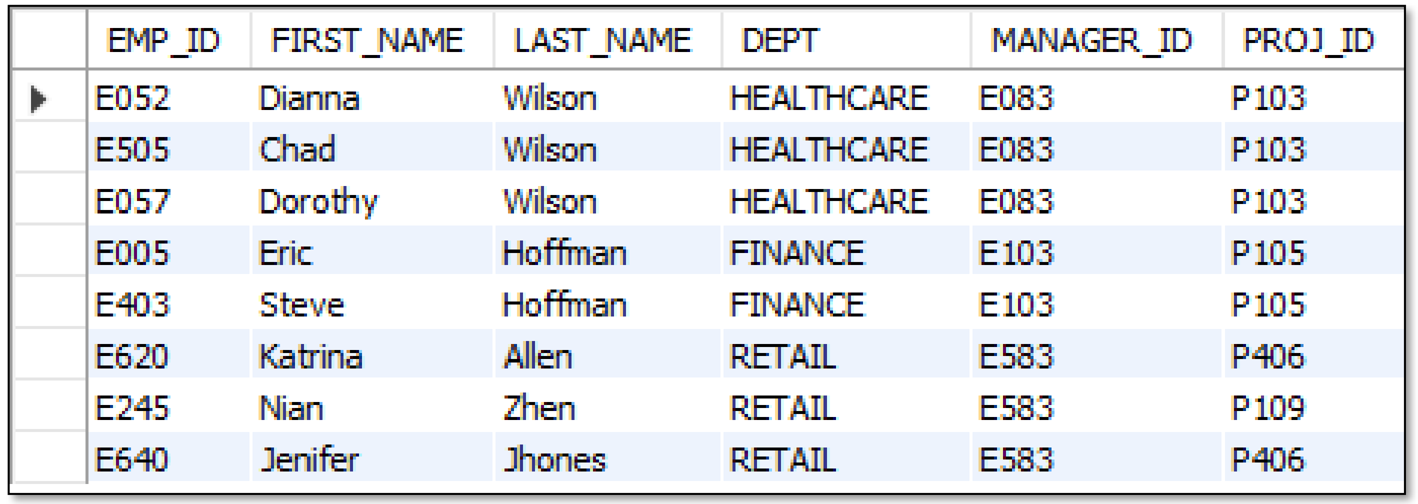
INNER JOIN: Example

Step 1: Use the **INNER JOIN** clause with the **ON** keyword in the **SELECT** statement to join **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query – INNER JOIN with ON

```
SELECT
  e.EMP_ID, e.FIRST_NAME, e.LAST_NAME,
  e.DEPT, e.MANAGER_ID,
  p.PROJ_ID
FROM
  EMP_RECORDS AS e
INNER JOIN PROJ_ASSIGN p ON e.EMP_ID = p.EMP_ID
WHERE e.ROLE NOT IN ("MANAGER", "PRESIDENT", "CEO")
ORDER BY e.MANAGER_ID;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	DEPT	MANAGER_ID	PROJ_ID
E052	Dianna	Wilson	HEALTHCARE	E083	P103
E505	Chad	Wilson	HEALTHCARE	E083	P103
E057	Dorothy	Wilson	HEALTHCARE	E083	P103
E005	Eric	Hoffman	FINANCE	E103	P105
E403	Steve	Hoffman	FINANCE	E103	P105
E620	Katrina	Allen	RETAIL	E583	P406
E245	Nian	Zhen	RETAIL	E583	P109
E640	Jenifer	Jhones	RETAIL	E583	P406

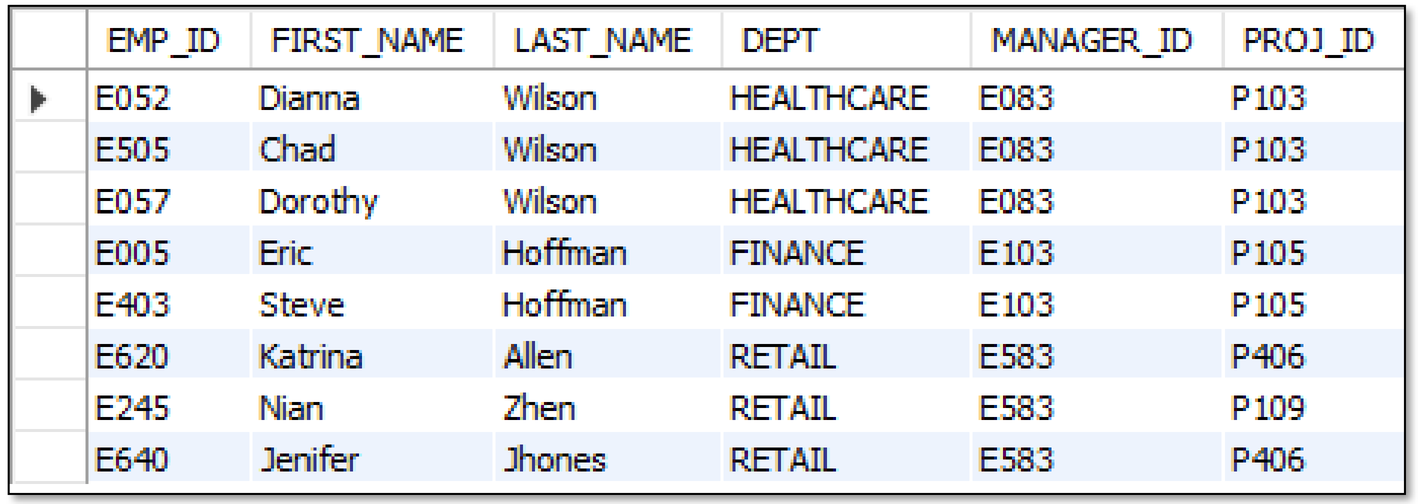
INNER JOIN: Example

Step 2: Use the **INNER JOIN** clause with the **USING** keyword in the **SELECT** statement to join **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query – INNER JOIN with USING

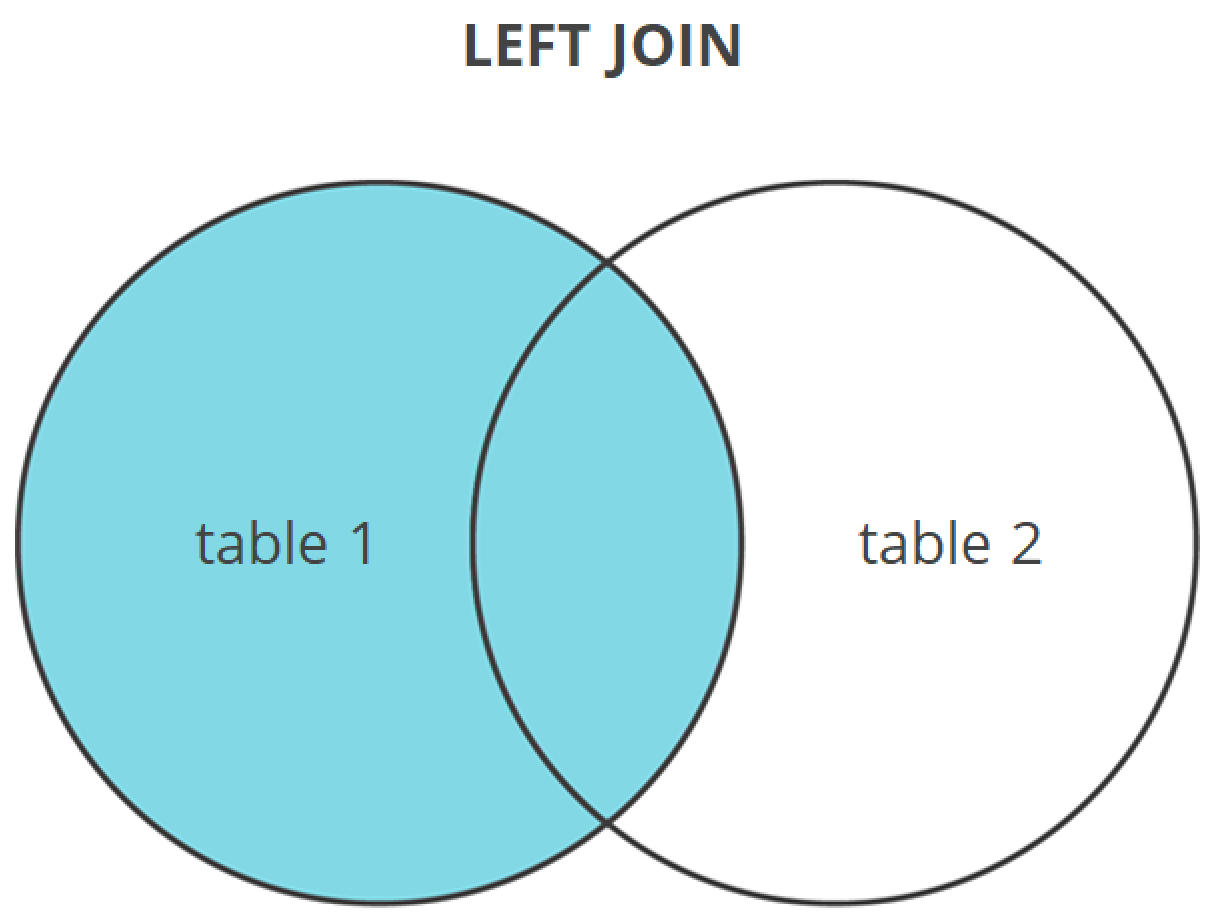
```
SELECT
  e.EMP_ID, e.FIRST_NAME, e.LAST_NAME,
  e.DEPT, e.MANAGER_ID,
  p.PROJ_ID
FROM
  EMP_RECORDS e
INNER JOIN PROJ_ASSIGN p ON(EMP_ID)
WHERE e.ROLE NOT IN ("MANAGER", "PRESIDENT", "CEO")
ORDER BY e.MANAGER_ID;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	DEPT	MANAGER_ID	PROJ_ID
▶	E052	Dianna	Wilson	HEALTHCARE	E083	P103
	E505	Chad	Wilson	HEALTHCARE	E083	P103
	E057	Dorothy	Wilson	HEALTHCARE	E083	P103
	E005	Eric	Hoffman	FINANCE	E103	P105
	E403	Steve	Hoffman	FINANCE	E103	P105
	E620	Katrina	Allen	RETAIL	E583	P406
	E245	Nian	Zhen	RETAIL	E583	P109
	E640	Jenifer	Jhones	RETAIL	E583	P406

LEFT JOIN



- Similar to INNER JOIN clause, the LEFT JOIN clause also requires a join predicate to join two tables.
- The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table 2). If there is no match on the right side, the outcome is 0 records.
- The LEFT JOIN also supports the USING clause.

LEFT JOIN

The LEFT JOIN selects all data from the left table, regardless of whether there are matching rows in the right table.

Syntax

```
SELECT column_list
FROM table_1
LEFT JOIN table_2 ON join_condition;
```

If no matching rows from the right table are found, the LEFT JOIN utilizes NULLs in the result set for columns of the row from the right table.

LEFT JOIN: Example

Problem Statement: Your manager wants the details of the ongoing projects along with the number of employees working on them.

Objective: Write an SQL query using the **LEFT JOIN** clause with either **ON** or **USING** keyword to perform the left join on the **PROJ_RECORDS** and **PROJ_ASSIGN** tables in MySQL.

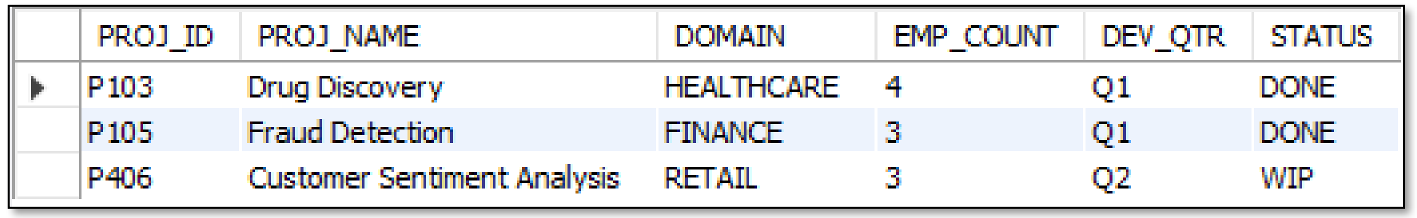
LEFT JOIN: Example

Step 1: Use the **LEFT JOIN** clause with the **ON** keyword in the **SELECT** statement to join **PROJ_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query – LEFT JOIN with ON

```
SELECT
  p.PROJ_ID, p.PROJ_NAME, p.DOMAIN,
  COUNT(DISTINCT a.EMP_ID) AS `EMP_COUNT`,
  p.DEV_QTR, p.STATUS
FROM
  PROJ_RECORDS p
LEFT JOIN PROJ_ASSIGN a ON p.PROJ_ID = a.PROJ_ID
WHERE p.STATUS IN ("DONE", "WIP")
GROUP BY p.PROJ_NAME
ORDER BY p.PROJ_ID;
```

Output:



PROJ_ID	PROJ_NAME	DOMAIN	EMP_COUNT	DEV_QTR	STATUS
P103	Drug Discovery	HEALTHCARE	4	Q1	DONE
P105	Fraud Detection	FINANCE	3	Q1	DONE
P406	Customer Sentiment Analysis	RETAIL	3	Q2	WIP

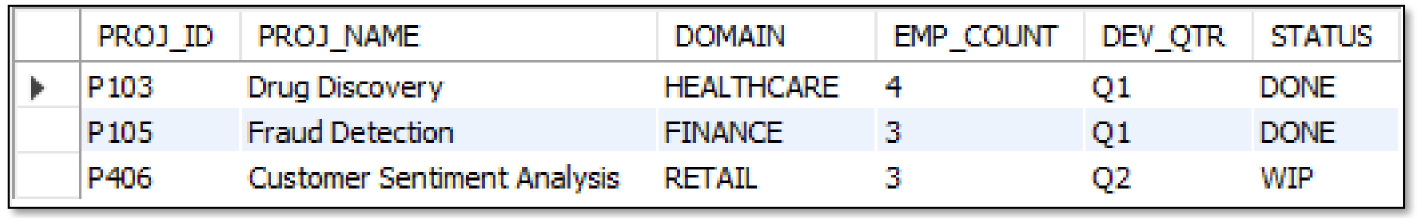
LEFT JOIN: Example

Step 2: Use the **LEFT JOIN** clause with the **USING** keyword in the **SELECT** statement to join **PROJ_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query – LEFT JOIN with USING

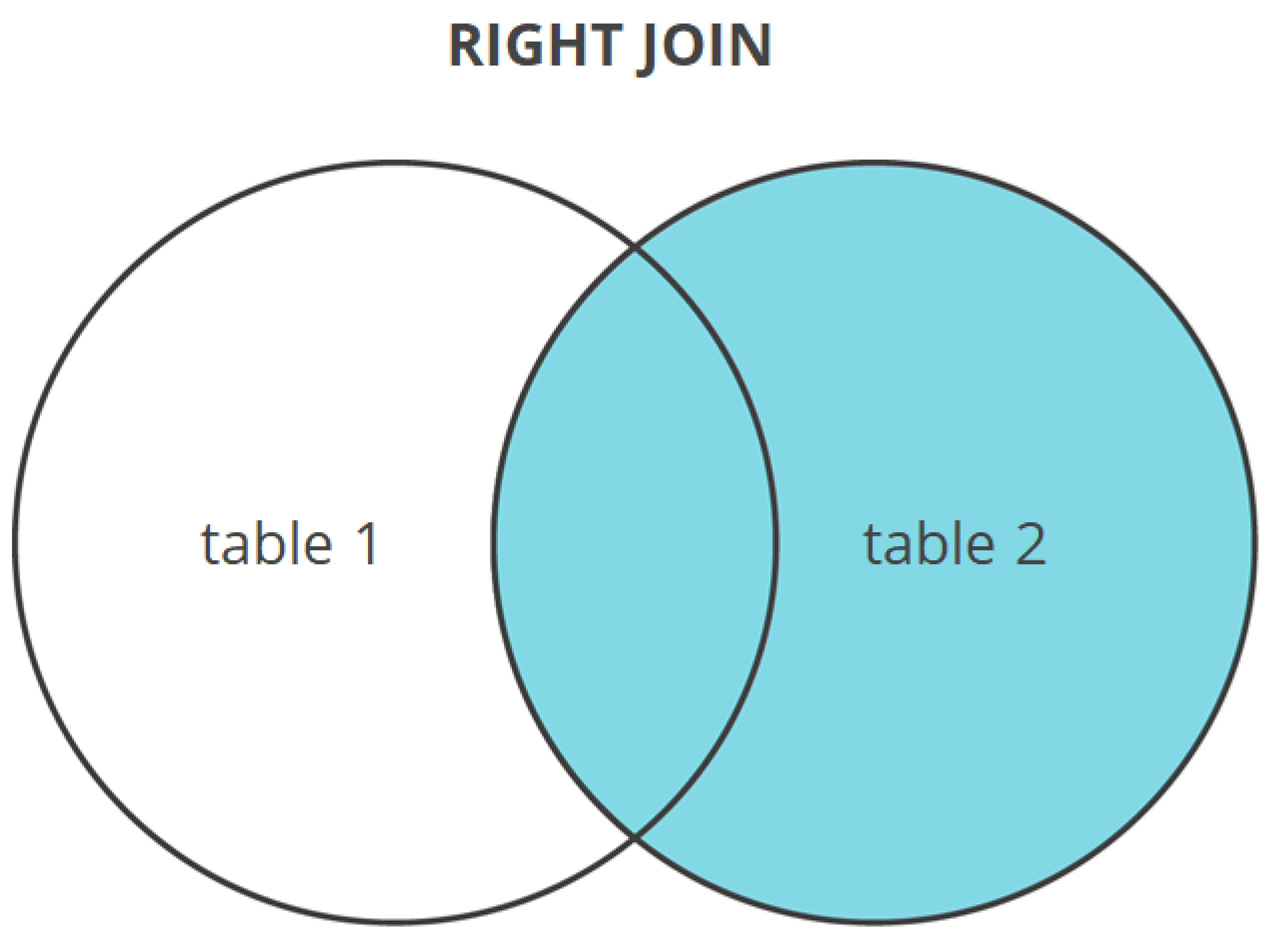
```
SELECT
  p.PROJ_ID, p.PROJ_NAME, p.DOMAIN,
  COUNT(DISTINCT a.EMP_ID) AS `EMP_COUNT`,
  p.DEV_QTR, p.STATUS
FROM
  PROJ_RECORDS p
LEFT JOIN PROJ_ASSIGN a ON(PROJ_ID)
WHERE p.STATUS IN ("DONE", "WIP")
GROUP BY p.PROJ_NAME
ORDER BY p.PROJ_ID;
```

Output:



PROJ_ID	PROJ_NAME	DOMAIN	EMP_COUNT	DEV_QTR	STATUS
P103	Drug Discovery	HEALTHCARE	4	Q1	DONE
P105	Fraud Detection	FINANCE	3	Q1	DONE
P406	Customer Sentiment Analysis	RETAIL	3	Q2	WIP

RIGHT JOIN



- The RIGHT JOIN is just the opposite of the LEFT JOIN and requires a join predicate to join two tables.
- The RIGHT JOIN keyword returns all records from the right table (table 1), and the matching records from the left table (table 2). The result is 0 records from the left side if there is no match.
- The RIGHT JOIN also supports the USING clause.

RIGHT JOIN

The RIGHT JOIN selects all data from the right table, regardless of whether there are matching rows in the left table.

Syntax

```
SELECT column_list
FROM table_1
RIGHT JOIN table_2 ON join_condition;
```

If no matching rows from the left table are found, the RIGHT JOIN utilizes NULLs in the result set for columns of the row from the left table.

RIGHT JOIN: Example

Problem Statement: Your manager wants the details of each employee along with the number of projects assigned to them.

Objective: Write an SQL query using the **RIGHT JOIN** clause with either **ON** or **USING** keyword to perform the left join on the **EMP_RECORDS** and **PROJ_ASSIGN** tables in MySQL.

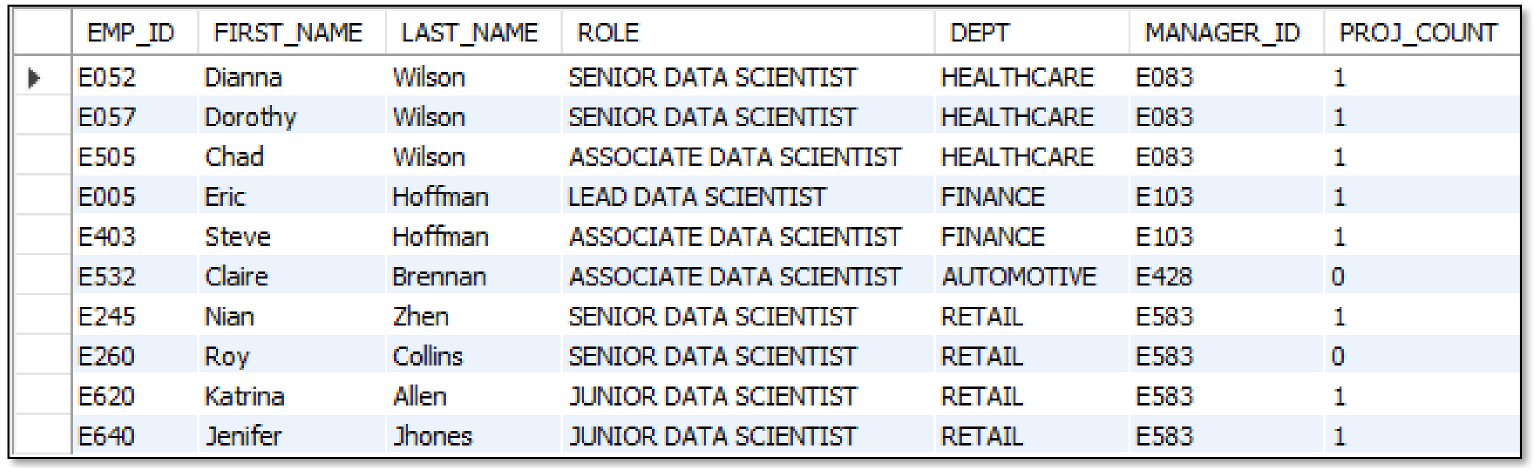
RIGHT JOIN: Example

Step 1: Use the **RIGHT JOIN** clause with the **ON** keyword in the **SELECT** statement to join **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query – LEFT JOIN with ON

```
SELECT
  e.EMP_ID, e.FIRST_NAME, e.LAST_NAME,
  e.ROLE, e.DEPT, e.MANAGER_ID,
  COUNT(DISTINCT a.PROJ_ID) AS `PROJ_COUNT`
FROM
  PROJ_ASSIGN a
RIGHT JOIN EMP_RECORDS e ON a.EMP_ID = e.EMP_ID
WHERE e.ROLE NOT IN ("MANAGER", "PRESIDENT", "CEO")
GROUP BY e.EMP_ID
ORDER BY e.MANAGER_ID;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	ROLE	DEPT	MANAGER_ID	PROJ_COUNT
E052	Dianna	Wilson	SENIOR DATA SCIENTIST	HEALTHCARE	E083	1
E057	Dorothy	Wilson	SENIOR DATA SCIENTIST	HEALTHCARE	E083	1
E505	Chad	Wilson	ASSOCIATE DATA SCIENTIST	HEALTHCARE	E083	1
E005	Eric	Hoffman	LEAD DATA SCIENTIST	FINANCE	E103	1
E403	Steve	Hoffman	ASSOCIATE DATA SCIENTIST	FINANCE	E103	1
E532	Claire	Brennan	ASSOCIATE DATA SCIENTIST	AUTOMOTIVE	E428	0
E245	Nian	Zhen	SENIOR DATA SCIENTIST	RETAIL	E583	1
E260	Roy	Collins	SENIOR DATA SCIENTIST	RETAIL	E583	0
E620	Katrina	Allen	JUNIOR DATA SCIENTIST	RETAIL	E583	1
E640	Jenifer	Jhones	JUNIOR DATA SCIENTIST	RETAIL	E583	1

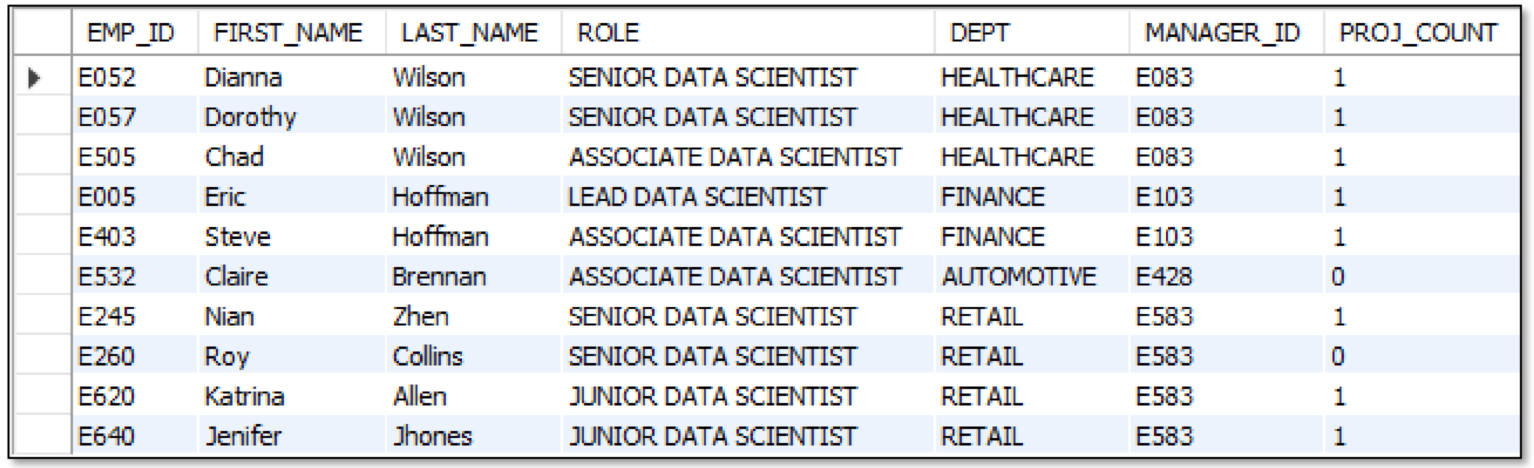
RIGHT JOIN: Example

Step 2: Use the **RIGHT JOIN** clause with the **USING** keyword in the **SELECT** statement to join **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query – LEFT JOIN with USING

```
SELECT
  e.EMP_ID, e.FIRST_NAME, e.LAST_NAME,
  e.ROLE, e.DEPT, e.MANAGER_ID,
  COUNT(DISTINCT a.PROJ_ID) AS `PROJ_COUNT`
FROM
  PROJ_ASSIGN a
RIGHT JOIN EMP_RECORDS e ON(EMP_ID)
WHERE e.ROLE NOT IN ("MANAGER", "PRESIDENT", "CEO")
GROUP BY e.EMP_ID
ORDER BY e.MANAGER_ID;
```

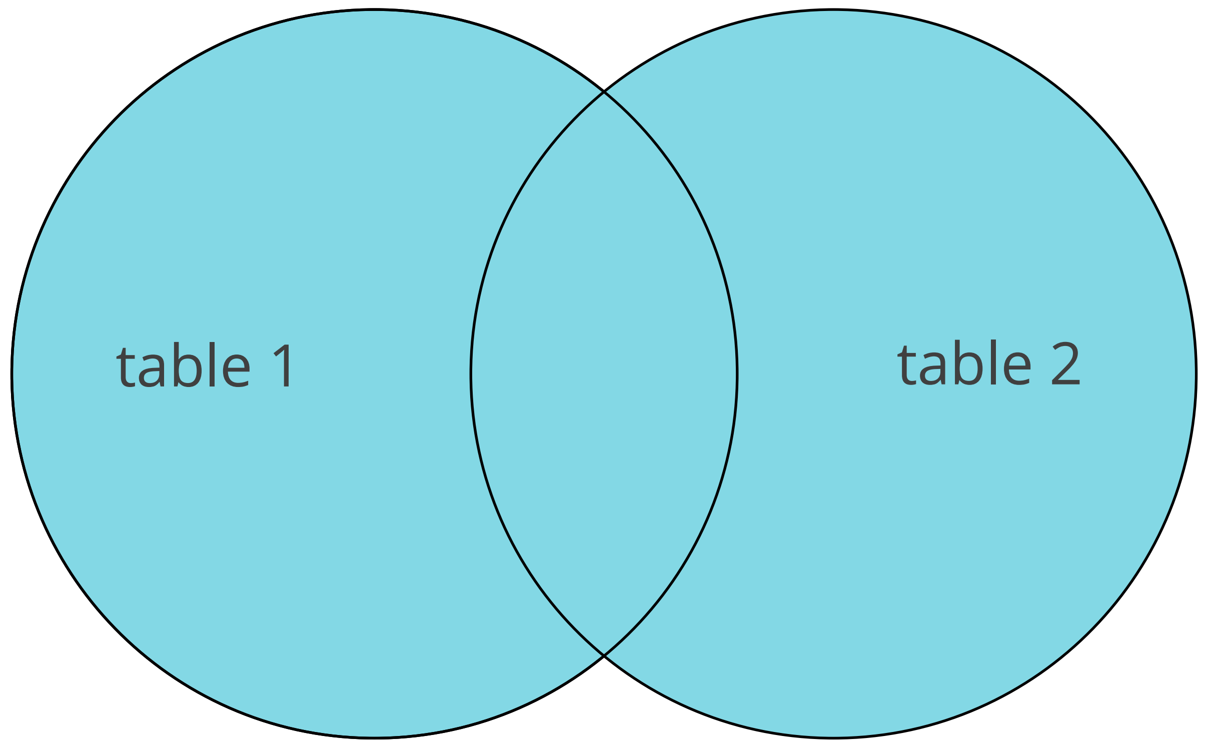
Output:



EMP_ID	FIRST_NAME	LAST_NAME	ROLE	DEPT	MANAGER_ID	PROJ_COUNT
E052	Dianna	Wilson	SENIOR DATA SCIENTIST	HEALTHCARE	E083	1
E057	Dorothy	Wilson	SENIOR DATA SCIENTIST	HEALTHCARE	E083	1
E505	Chad	Wilson	ASSOCIATE DATA SCIENTIST	HEALTHCARE	E083	1
E005	Eric	Hoffman	LEAD DATA SCIENTIST	FINANCE	E103	1
E403	Steve	Hoffman	ASSOCIATE DATA SCIENTIST	FINANCE	E103	1
E532	Claire	Brennan	ASSOCIATE DATA SCIENTIST	AUTOMOTIVE	E428	0
E245	Nian	Zhen	SENIOR DATA SCIENTIST	RETAIL	E583	1
E260	Roy	Collins	SENIOR DATA SCIENTIST	RETAIL	E583	0
E620	Katrina	Allen	JUNIOR DATA SCIENTIST	RETAIL	E583	1
E640	Jenifer	Jhones	JUNIOR DATA SCIENTIST	RETAIL	E583	1

FULL OUTER JOIN

FULL OUTER JOIN



- Merges the outcomes of both left outer join and right outer join
- Encompasses all rows from both tables, pairing rows from each table when possible
- If a row in one table has no corresponding match, the result retains that row, filling in with NULL values for columns from the other table.

FULL OUTER JOIN

Syntax

```
SELECT select_list
FROM table_1
FULL OUTER JOIN table_2;
```

FULL OUTER JOIN: Example

Problem Statement: Your manager expects you to fetch an all-encompassing list of records from both the employee and project records tables, ensuring the inclusion of every row from each table, irrespective of whether a match exists or not.

Objective: Write an SQL query using the **FULL OUTER JOIN** clause to perform the full outer join on the **EMP_RECORDS** and **PROJ_ASSIGN** tables to obtain their full information in MySQL.

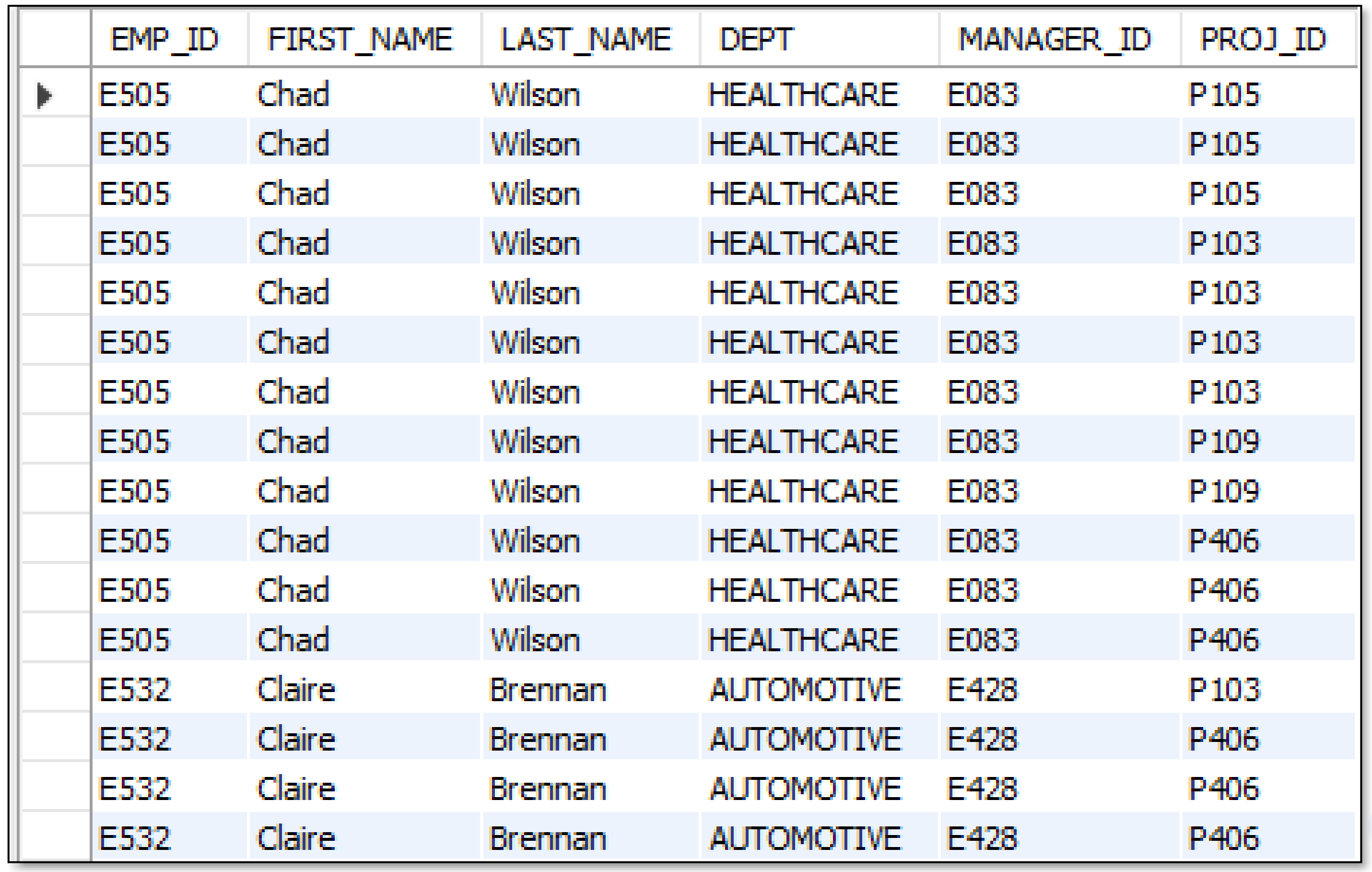
FULL OUTER JOIN: Example

Step 1: Use the **FULL OUTER JOIN** clause in the **SELECT** statement to join **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query

```
SELECT
  e.EMP_ID, e.FIRST_NAME, e.LAST_NAME,
  e.DEPT, e.MANAGER_ID,
  a.PROJ_ID
FROM
  PROJ_ASSIGN a
FULL OUTER JOIN EMP_RECORDS e
WHERE e.ROLE NOT IN ("MANAGER", "PRESIDENT", "CEO")
ORDER BY e.FIRST_NAME;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	DEPT	MANAGER_ID	PROJ_ID
E505	Chad	Wilson	HEALTHCARE	E083	P105
E505	Chad	Wilson	HEALTHCARE	E083	P105
E505	Chad	Wilson	HEALTHCARE	E083	P105
E505	Chad	Wilson	HEALTHCARE	E083	P103
E505	Chad	Wilson	HEALTHCARE	E083	P103
E505	Chad	Wilson	HEALTHCARE	E083	P103
E505	Chad	Wilson	HEALTHCARE	E083	P103
E505	Chad	Wilson	HEALTHCARE	E083	P109
E505	Chad	Wilson	HEALTHCARE	E083	P109
E505	Chad	Wilson	HEALTHCARE	E083	P406
E505	Chad	Wilson	HEALTHCARE	E083	P406
E505	Chad	Wilson	HEALTHCARE	E083	P406
E532	Claire	Brennan	AUTOMOTIVE	E428	P103
E532	Claire	Brennan	AUTOMOTIVE	E428	P406
E532	Claire	Brennan	AUTOMOTIVE	E428	P406
E532	Claire	Brennan	AUTOMOTIVE	E428	P406

SELF JOIN

SELF JOIN

- The SELF JOIN clause joins a table to itself using the INNER JOIN or LEFT JOIN.
- The SELF JOIN is often used to query hierarchical data or to compare rows within the same table.
- To perform a SELF JOIN, table aliases must be used to avoid repeating the same table name in a single query.
- MySQL throws an error if a table is referenced twice or more in a query without utilizing table aliases.

SELF JOIN: Example

Problem Statement: Your manager wants you to identify the number of employees reporting to each manager including the President and the CEO.

Objective: Write an SQL query using either **INNER JOIN** or **LEFT JOIN** clause to simulate the **SELF JOIN** clause on the **EMP_RECORDS** table in MySQL.

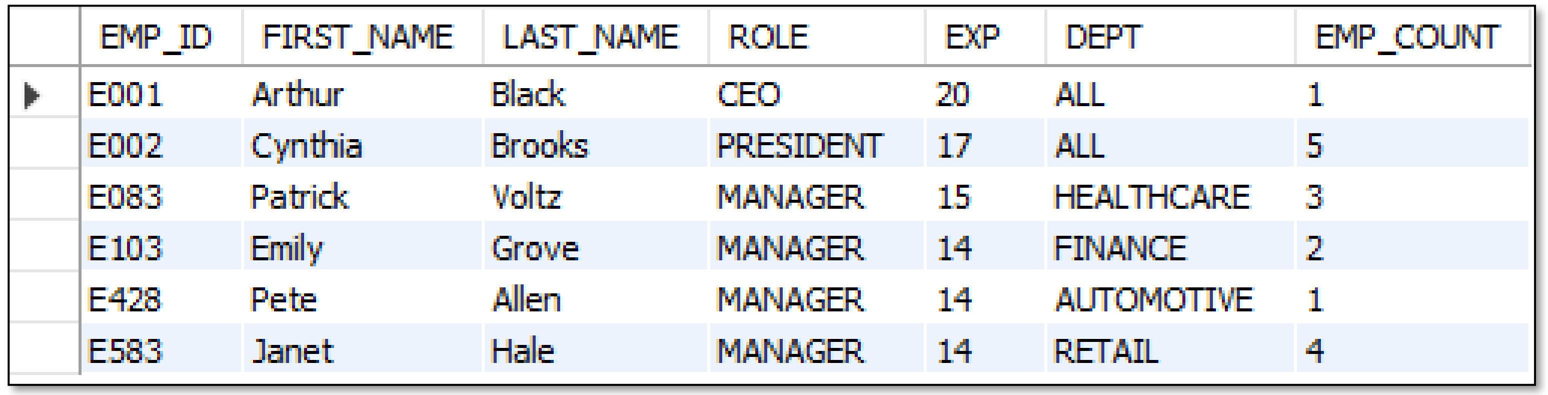
SELF JOIN: Example

Step 1: Use the **INNER JOIN** clause in the **SELECT** statement to simulate the **SELF JOIN** clause to join **EMP_RECORDS** table as shown below:

SQL Query – SELF JOIN using INNER JOIN

```
SELECT
  m.EMP_ID, m.FIRST_NAME, m.LAST_NAME, m.ROLE,
  m.EXP, m.DEPT, COUNT(e.EMP_ID) as "EMP_COUNT"
FROM
  EMP_RECORDS m
INNER JOIN EMP_RECORDS e
  ON m.EMP_ID = e.MANAGER_ID
  AND e.EMP_ID != e.MANAGER_ID
WHERE m.ROLE IN ("MANAGER", "PRESIDENT", "CEO")
GROUP BY m.EMP_ID
ORDER BY m.EMP_ID;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	ROLE	EXP	DEPT	EMP_COUNT
E001	Arthur	Black	CEO	20	ALL	1
E002	Cynthia	Brooks	PRESIDENT	17	ALL	5
E083	Patrick	Voltz	MANAGER	15	HEALTHCARE	3
E103	Emily	Grove	MANAGER	14	FINANCE	2
E428	Pete	Allen	MANAGER	14	AUTOMOTIVE	1
E583	Janet	Hale	MANAGER	14	RETAIL	4

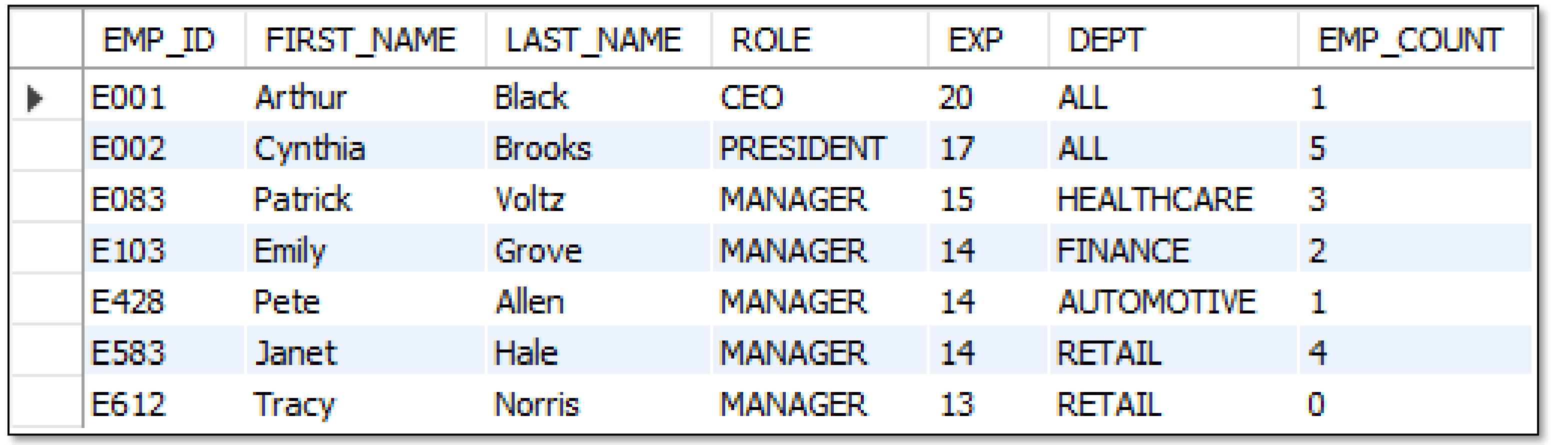
SELF JOIN: Example

Step 2: Use the **LEFT JOIN** clause in the **SELECT** statement to simulate the **SELF JOIN** clause to join **EMP_RECORDS** table as shown below:

SQL Query – SELF JOIN using LEFT JOIN

```
SELECT
  m.EMP_ID, m.FIRST_NAME, m.LAST_NAME, m.ROLE,
  m.EXP, m.DEPT, COUNT(e.EMP_ID) AS `EMP_COUNT`
FROM
  EMP_RECORDS m
LEFT JOIN EMP_RECORDS e
  ON m.EMP_ID = e.MANAGER_ID
  AND e.EMP_ID != e.MANAGER_ID
WHERE m.ROLE IN ("MANAGER", "PRESIDENT", "CEO")
GROUP BY m.EMP_ID
ORDER BY m.EMP_ID;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	ROLE	EXP	DEPT	EMP_COUNT
E001	Arthur	Black	CEO	20	ALL	1
E002	Cynthia	Brooks	PRESIDENT	17	ALL	5
E083	Patrick	Voltz	MANAGER	15	HEALTHCARE	3
E103	Emily	Grove	MANAGER	14	FINANCE	2
E428	Pete	Allen	MANAGER	14	AUTOMOTIVE	1
E583	Janet	Hale	MANAGER	14	RETAIL	4
E612	Tracy	Norris	MANAGER	13	RETAIL	0

Assisted Practice: Joins One

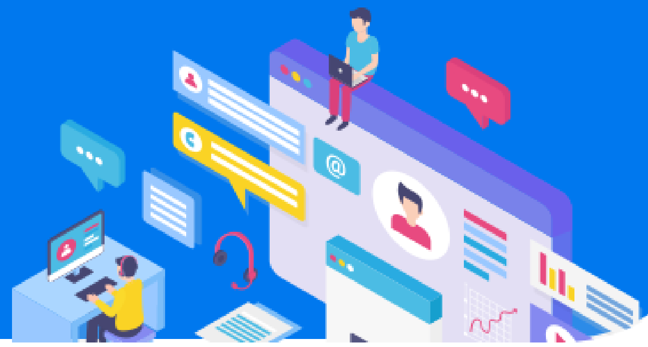
Duration: 15 mins

Problem statement: Assume that you have organized a quiz in your company and that people from different offices across the world have participated. You have been asked to analyze the quiz data to understand participation data and prize distribution across geographies.

You will be referring to the following datasets :

participant - This dataset contains information about the people who participated in the quiz and has 1000 rows and 11 columns
states - This dataset contains information about the states' names, codes, regions, and the division a participant belongs to and has 50 rows and 4 columns

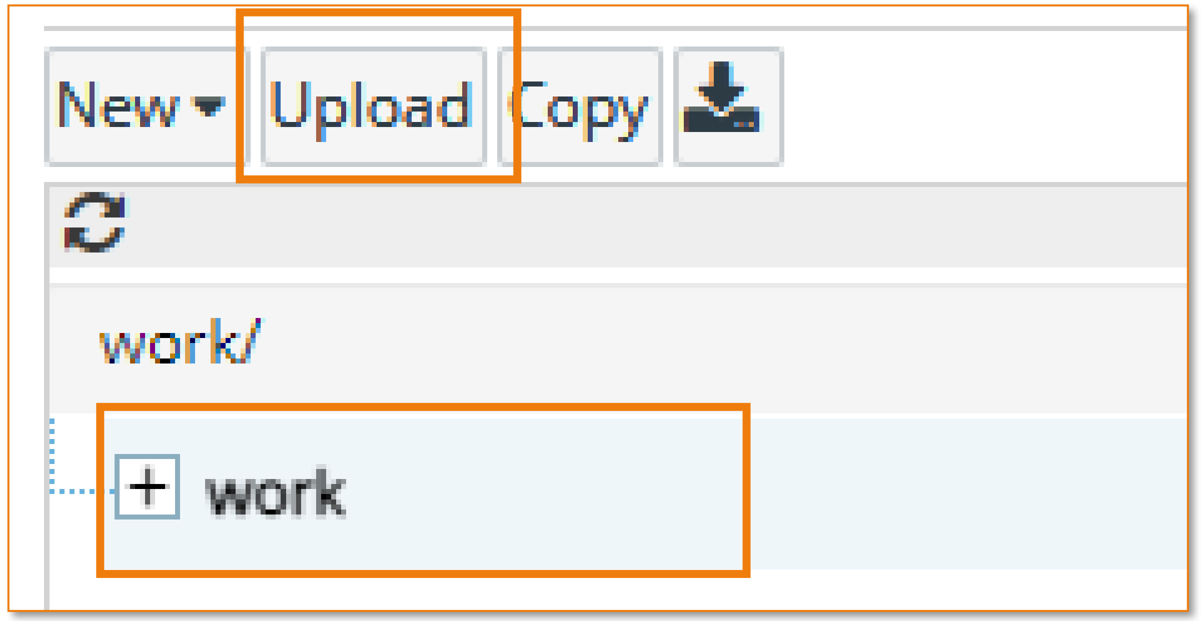
Assisted Practice: Joins One



Step 01: Upload the **participant** and **states** datasets in the lab

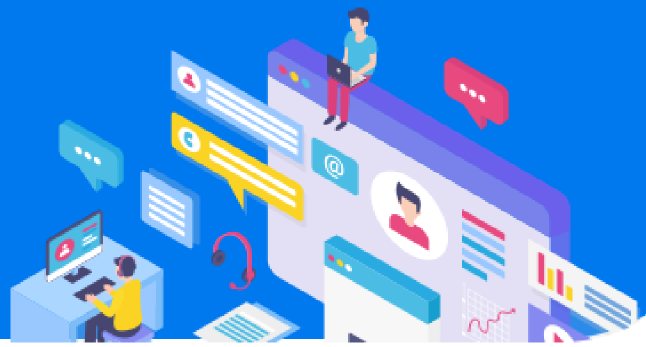
1. Download these datasets from the **Course Resources** section of the **LMS**

2. In the file explorer area in the lab, click on the **work** tab and then click on the **Upload** button to upload the datasets



ASSISTED PRACTICE

Assisted Practice: Joins One



Step 02: Import **participants.csv** and **states.csv** in the database as tables

Note: Select your current working database to perform the following:

1. Right-click on the database, select **Table Data Import,** click on **Browse**, select the .csv files, and then click on **Next**

2. Select **Create a new table** and **Drop table if exists** and then click on **Next**

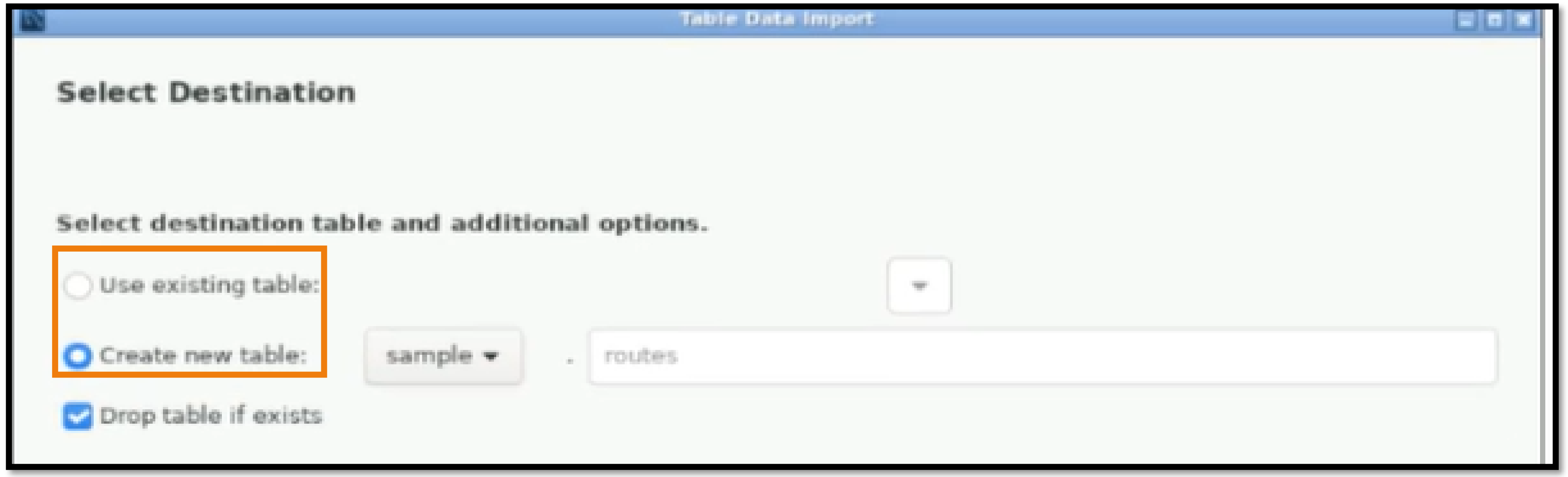


Table Data Import

Select Destination

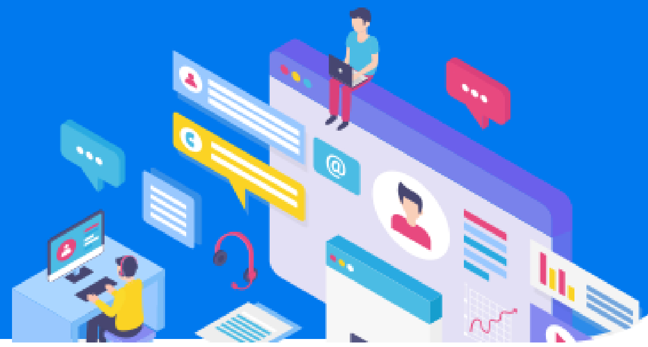
Select destination table and additional options.

Use existing table:

[✓ Create new table:] sample . routes

[✓ Drop table if exists]

Assisted Practice: Joins One



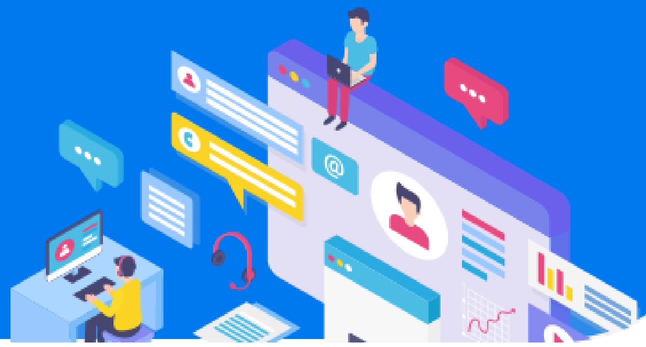
Step 03: Write a query to show the two tables (**participant** and **state**)

QUERY

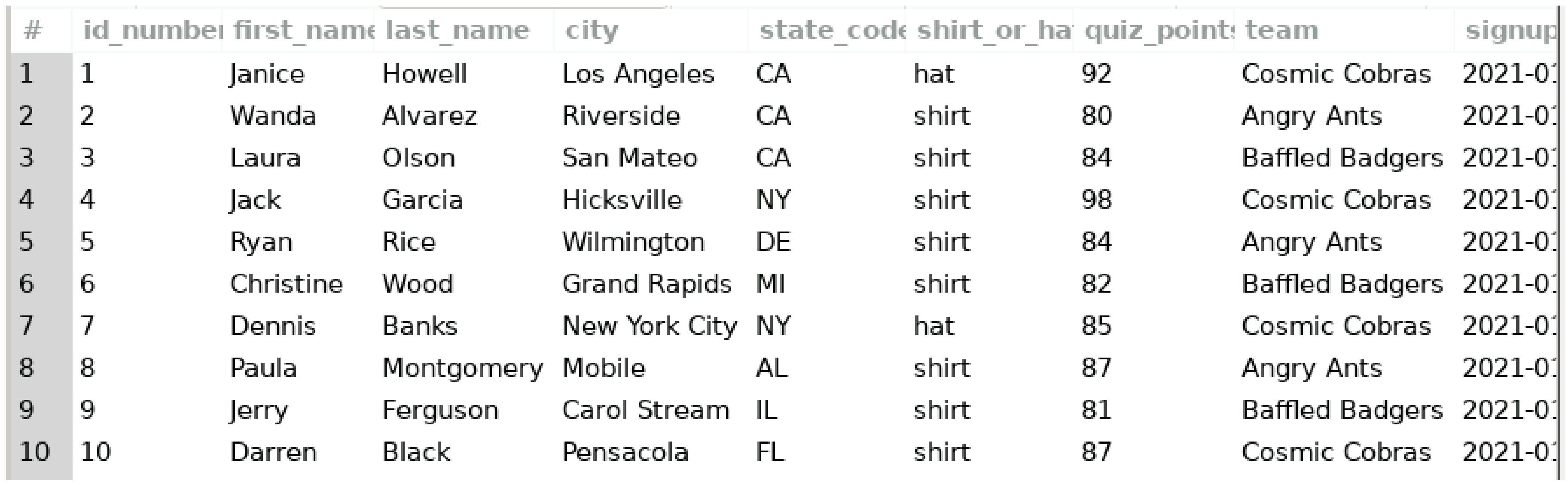
```
SELECT * FROM participant;
SELECT * FROM state;
```

ASSISTED PRACTICE

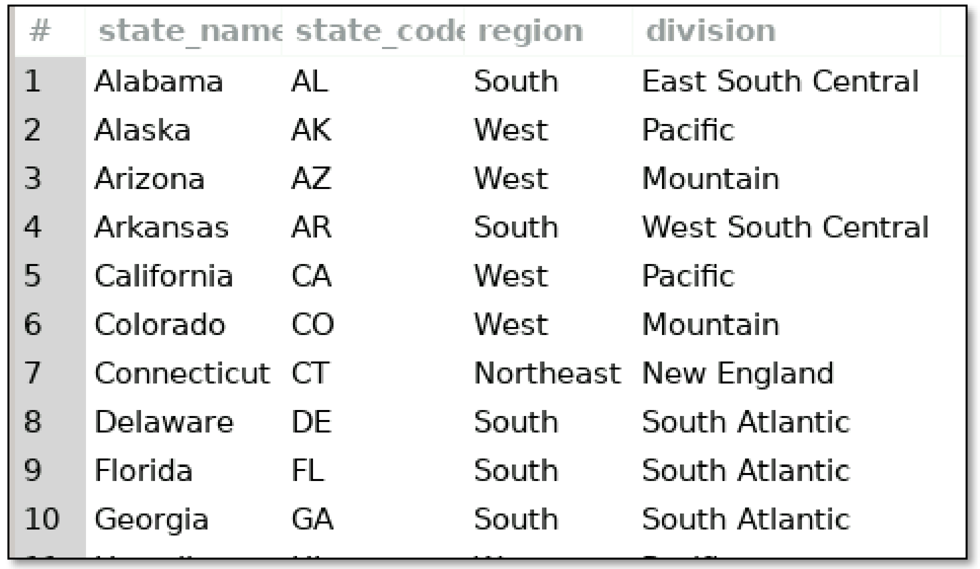
Assisted Practice: Joins One



Output:

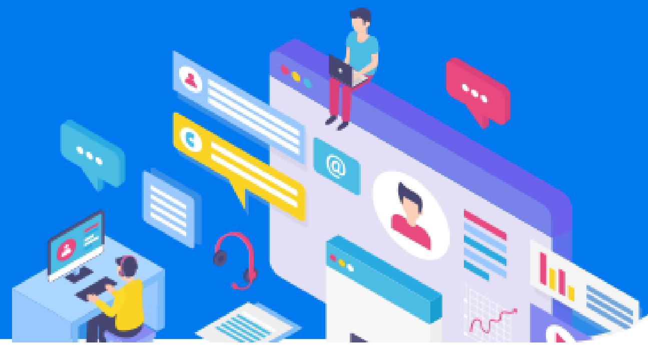


#	id_numbe	first_nam	last_name	city	state_cod	shirt_or_ha	quiz_point	team	signup
1	1	Janice	Howell	Los Angeles	CA	hat	92	Cosmic Cobras	2021-0
2	2	Wanda	Alvarez	Riverside	CA	shirt	80	Angry Ants	2021-0
3	3	Laura	Olson	San Mateo	CA	shirt	84	Baffled Badgers	2021-0
4	4	Jack	Garcia	Hicksville	NY	shirt	98	Cosmic Cobras	2021-0
5	5	Ryan	Rice	Wilmington	DE	shirt	84	Angry Ants	2021-0
6	6	Christine	Wood	Grand Rapids	MI	shirt	82	Baffled Badgers	2021-0
7	7	Dennis	Banks	New York City	NY	hat	85	Cosmic Cobras	2021-0
8	8	Paula	Montgomery	Mobile	AL	shirt	87	Angry Ants	2021-0
9	9	Jerry	Ferguson	Carol Stream	IL	shirt	81	Baffled Badgers	2021-0
10	10	Darren	Black	Pensacola	FL	shirt	87	Cosmic Cobras	2021-0



#	state_nam	state_cod	region	division
1	Alabama	AL	South	East South Central
2	Alaska	AK	West	Pacific
3	Arizona	AZ	West	Mountain
4	Arkansas	AR	South	West South Central
5	California	CA	West	Pacific
6	Colorado	CO	West	Mountain
7	Connecticut	CT	Northeast	New England
8	Delaware	DE	South	South Atlantic
9	Florida	FL	South	South Atlantic
10	Georgia	GA	South	South Atlantic

Assisted Practice: Joins One

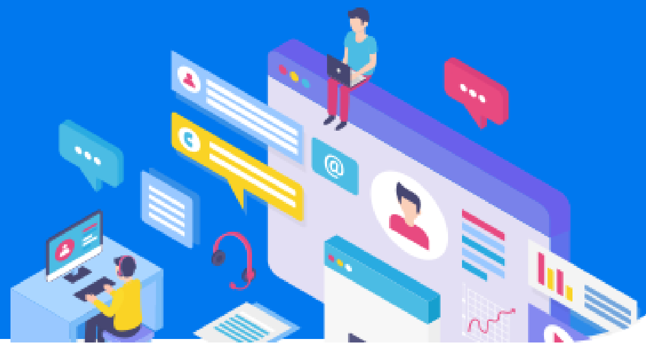


Step 04: Create a dataset containing the information on how many shirts need to be shipped to each state, filter the data for the top five states with the most shirt shipments, and ensure data has the state's name and not just the two-letter codes

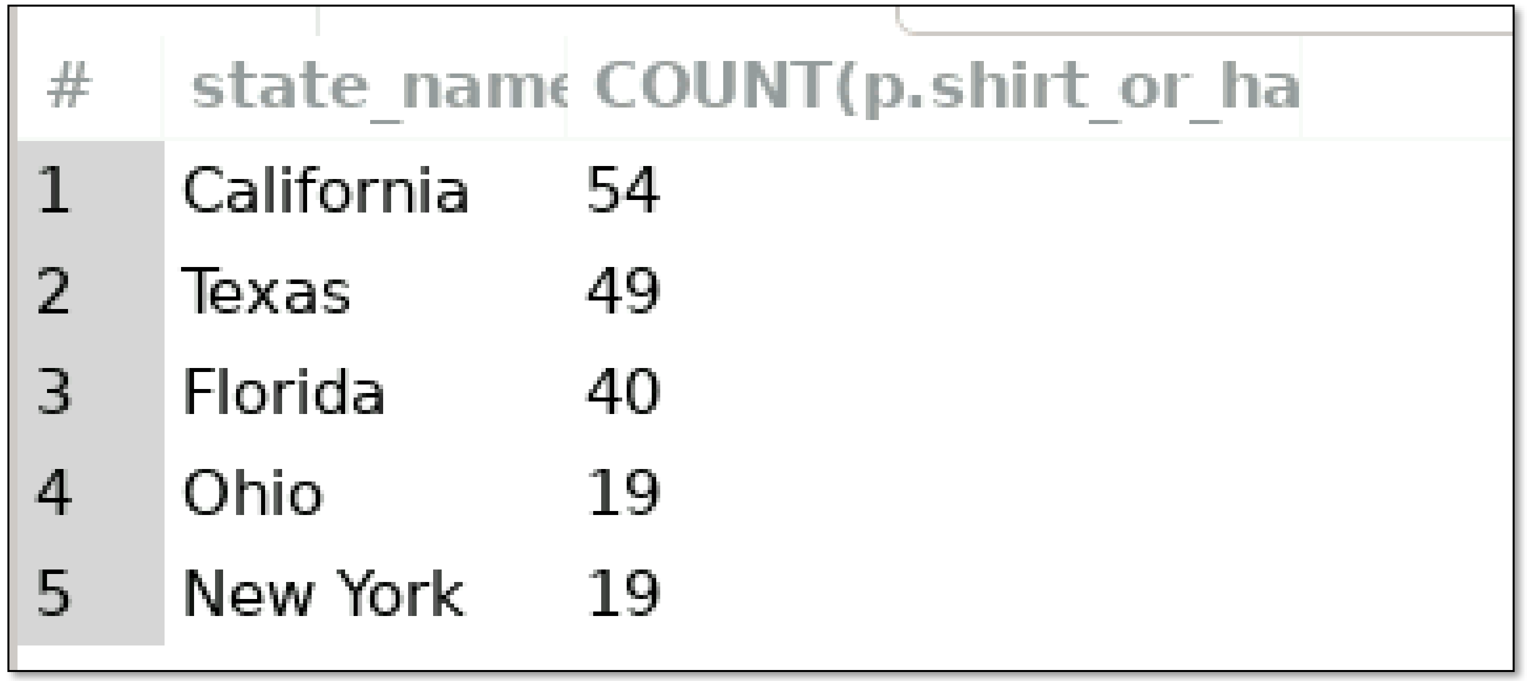
CREATE

```
SELECT s.state_name, COUNT(p.shirt_or_hat)
FROM states s JOIN participant p
ON s.state_code = p.state_code
WHERE p.shirt_or_hat='shirt'
GROUP BY s.state_name
ORDER BY COUNT(p.shirt_or_hat) DESC
LIMIT 5
```

Assisted Practice: Joins One

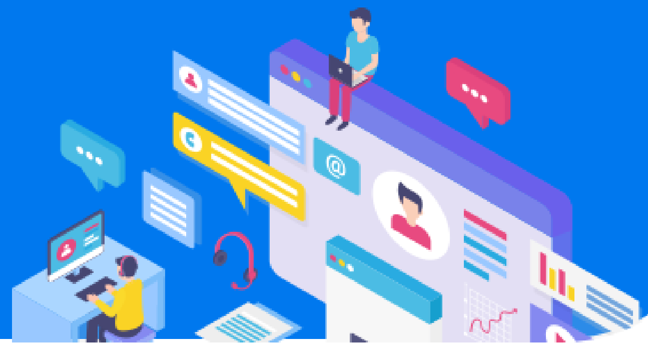


Output:



#	state_name	COUNT(p.shirt_or_ha
1	California	54
2	Texas	49
3	Florida	40
4	Ohio	19
5	New York	19

Assisted Practice: Joins One

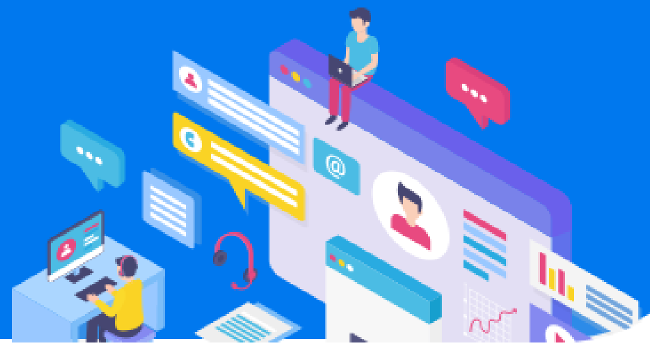


Step 05: Write a query to collect the sorted information showing how many members of each team are in each division and pull the bottom three divisions with the least members

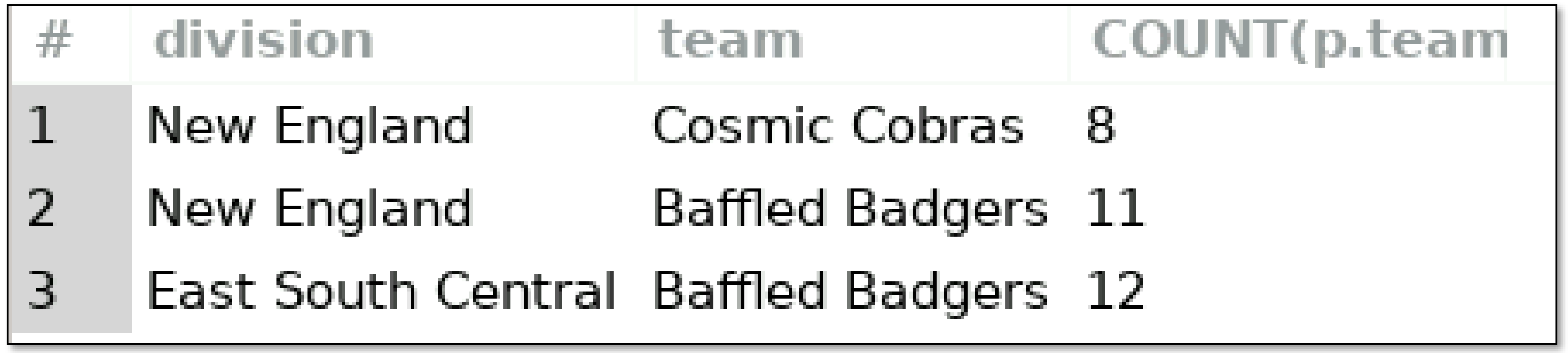
SQL Query

```
SELECT s.division, p.team, COUNT(p.team)
FROM states s JOIN participant p
ON s.state_code = p.state_code
GROUP BY s.division, p.team
ORDER BY COUNT(p.team)
LIMIT 3
```

Assisted Practice: Joins One

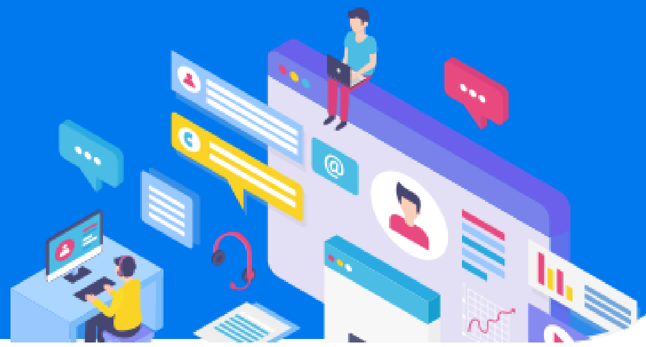


Output:



#	division	team	COUNT(p.team
1	New England	Cosmic Cobras	8
2	New England	Baffled Badgers	11
3	East South Central	Baffled Badgers	12

Assisted Practice: Joins Two

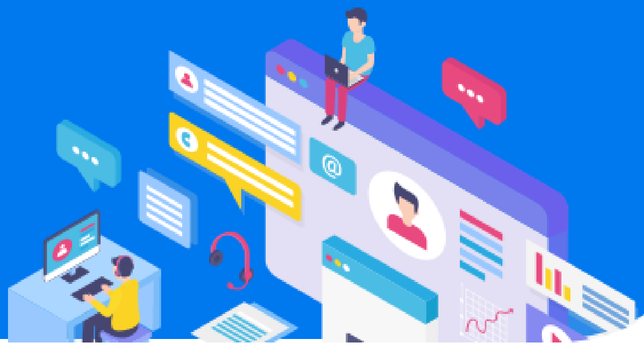


Duration: 15 mins

Problem statement: You are working as a database administrator for an insurance firm. The head of the insurance campaign planning department of your organization is looking for a dataset containing the information of customers who have claimed an amount only under the B2C category and not under the B2B category.

Objective: Use **JOIN** in MySQL to fetch the data of those specific customers who have claimed an amount only under the B2C category and share it with the head of the insurance campaign planning department

Assisted Practice: Joins Two



Duration: 15 mins

Datasets: You've been asked to create the following tables to perform the required tasks:

CUST_ID	POLICY_NO	CLAIM_DATE	CLAIM_AMT
C1	PO21	2020-03-03	1000
C2	PO22	2020-03-04	3200
C3	PO33	2020-03-05	5000
C4	PO44	2020-03-06	4500
C5	PO55	2020-03-07	1200

B2C_customer table

CUST_ID	POLICY_NO	CLAIM_DATE	CLAIM_AMT
C1	PO21	2020-03-03	1000
C2	PO22	2020-03-04	3200
C8	PO88	2020-03-07	4000

B2B_customer table

Assisted Practice: Joins Two

Duration: 15 mins

Steps to be performed:

Step 01: Create the **B2C_customers** table per the given structure

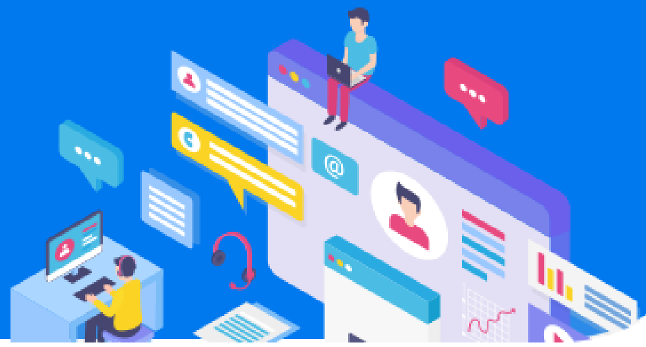
Step 02: Create the **B2B_customers** table per the given structure

Step 03: Insert records in the **B2C_customers** table

Step 04: Insert records in the **B2B_customers** table

Step 05: Fetch the data of those specific customers who have claimed an amount only under the B2C category and not under the B2B category

Assisted Practice: Joins Two

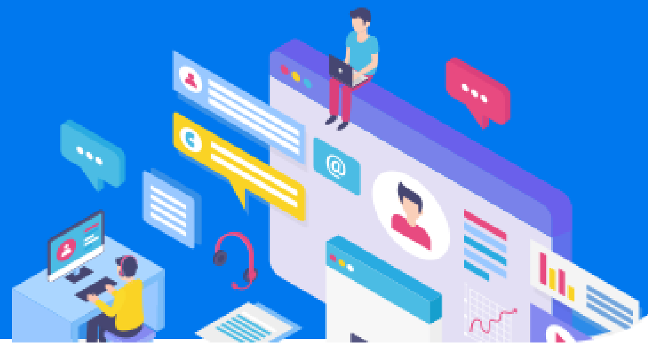


Step 01: Create the **B2C_customers** table per the given structure

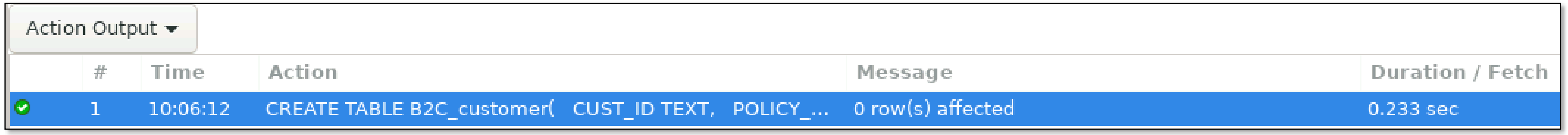
Query

```
CREATE TABLE B2C_customer(
  CUST_ID TEXT,
  POLICY_NO TEXT,
  CLAIM_DATE DATE,
  CLAIM_AMT INT
);
```

Assisted Practice: Joins Two

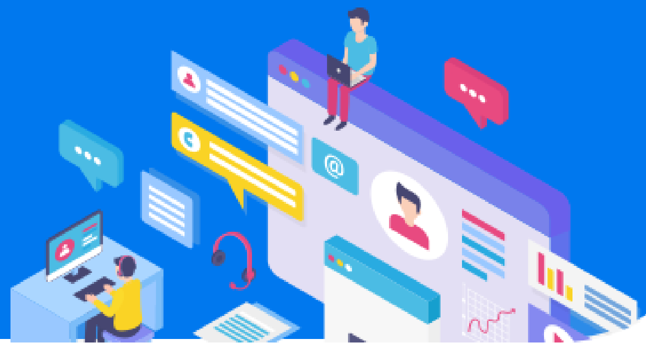


Output:



```
#   Time      Action                                              Message             Duration / Fetch
1   10:06:12  CREATE TABLE B2C_customer(  CUST_ID TEXT,  POLICY_...  0 row(s) affected   0.233 sec
```

Assisted Practice: Joins Two

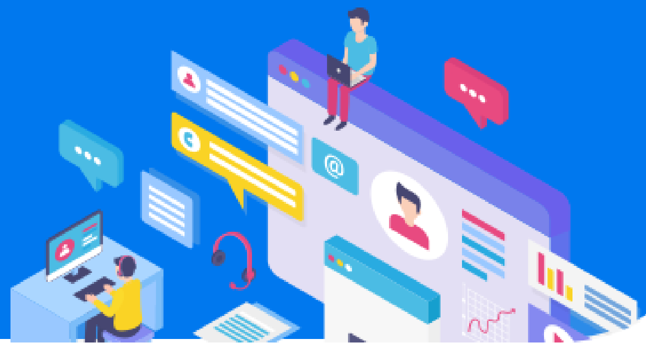


Step 02: Create the **B2B_customers** table per the given structure

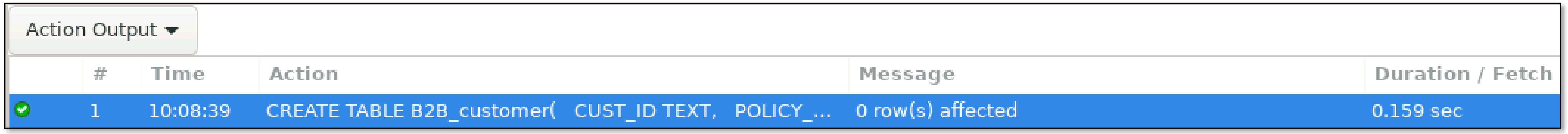
Query

```
CREATE TABLE B2B_customer(
  CUST_ID TEXT,
  POLICY_NO TEXT,
  CLAIM_DATE DATE,
  CLAIM_AMT INT
);
```

Assisted Practice: Joins Two

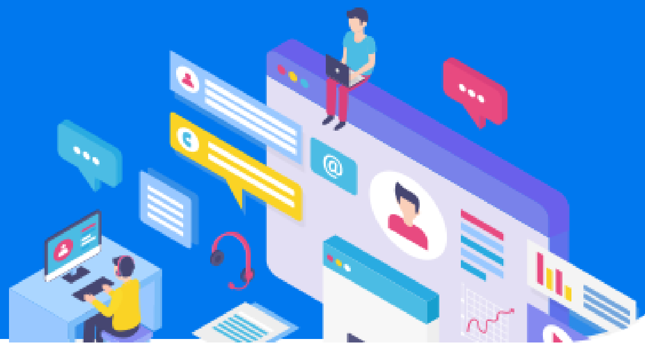


Output:



```
#   Time       Action                                              Message             Duration / Fetch
1   10:08:39   CREATE TABLE B2B_customer(   CUST_ID TEXT,   POLICY_...   0 row(s) affected   0.159 sec
```

Assisted Practice: Joins Two

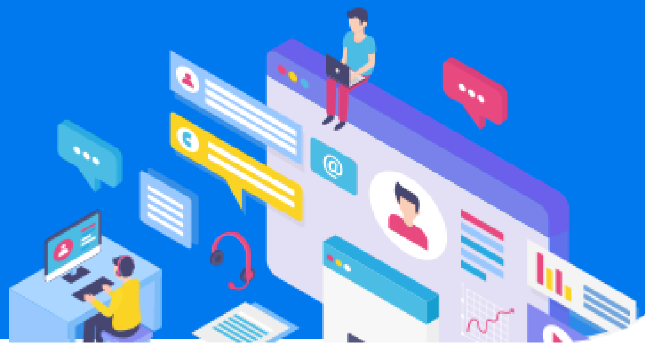


Step 03: Insert records in the **B2C_customers** table

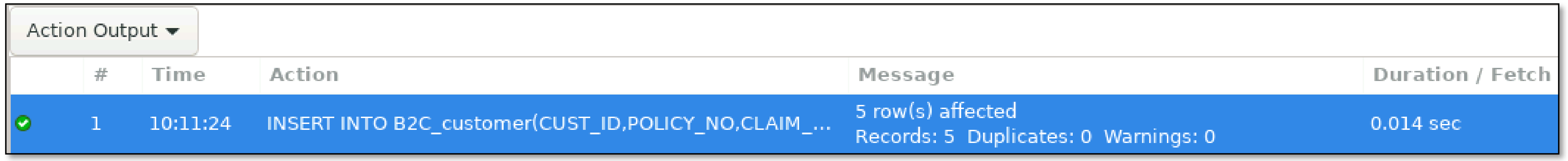
Query

```
INSERT INTO
B2C_customer(CUST_ID,POLICY_NO,CLAIM_DATE,CLAIM_AMT)
VALUES
("C","PO21","2020-03-03",1000),
("C2","PO22","2020-03-04",3200),
("C3","PO33","2020-03-05",5000),
("C4","PO44","2020-03-06",4500),
("C5","PO55","2020-03-07",1200);
```

Assisted Practice: Joins Two

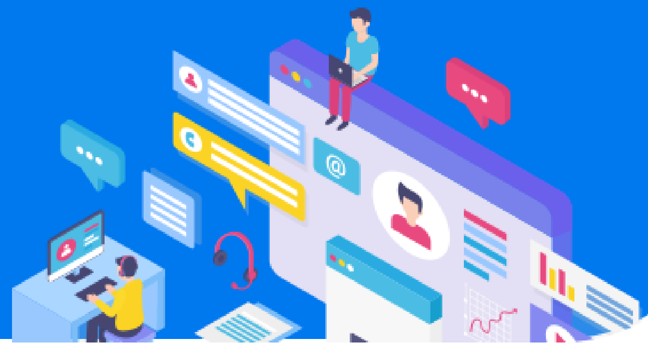


Output:



```
Action Output ▾
     #   Time       Action                                                  Message                                      Duration / Fetch
✓    1   10:11:24   INSERT INTO B2C_customer(CUST_ID,POLICY_NO,CLAIM_...    5 row(s) affected                            0.014 sec
                                                                            Records: 5  Duplicates: 0  Warnings: 0
```

Assisted Practice: Joins Two

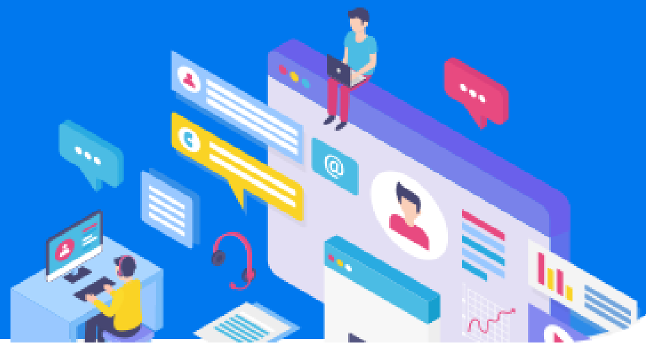


Step 04: Insert records in the **B2B_customers** table

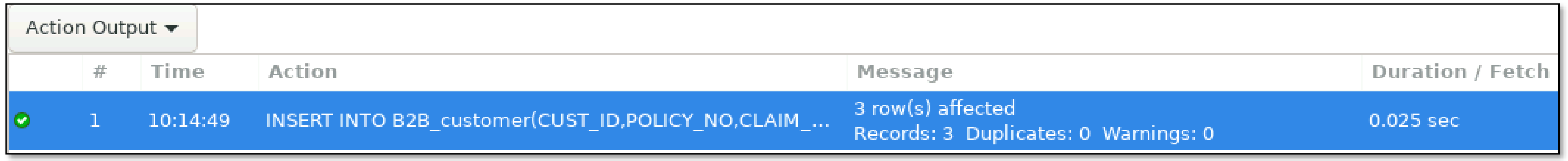
Query

```
INSERT INTO
B2B_customer(CUST_ID,POLICY_NO,CLAIM_DATE,CLAIM_AMT)
VALUES
("C","PO21","2020-03-03",1000),
("C2","PO22","2020-03-04",3200),
("C8","PO33","2020-03-05",5000);
```

Assisted Practice: Joins Two



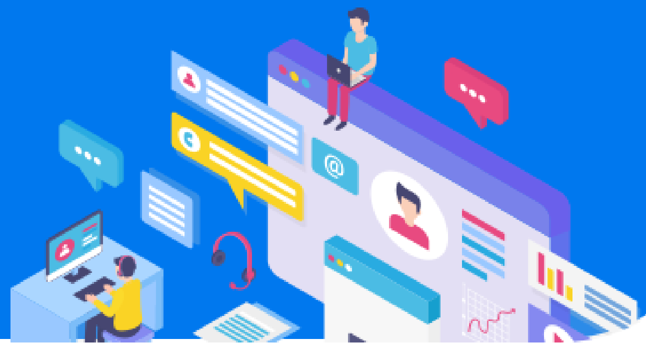
Output:



Action Output

```
#   Time      Action                                            Message                                     Duration / Fetch
1   10:14:49  INSERT INTO B2B_customer(CUST_ID,POLICY_NO,CLAIM_...  3 row(s) affected                       0.025 sec
                                                                Records: 3  Duplicates: 0  Warnings: 0
```

Assisted Practice: Joins Two

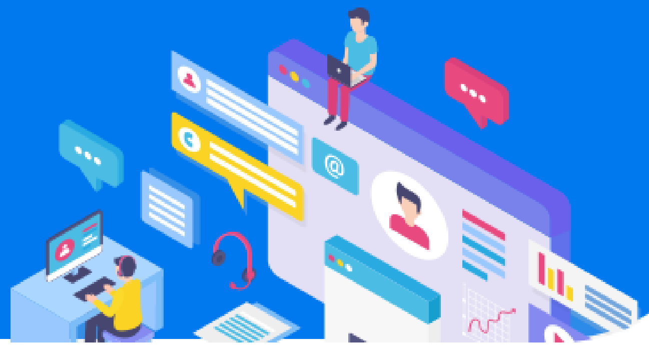


Step 05: Fetch the data of those specific customers who have claimed an amount only under the B2C category and not under the B2B category

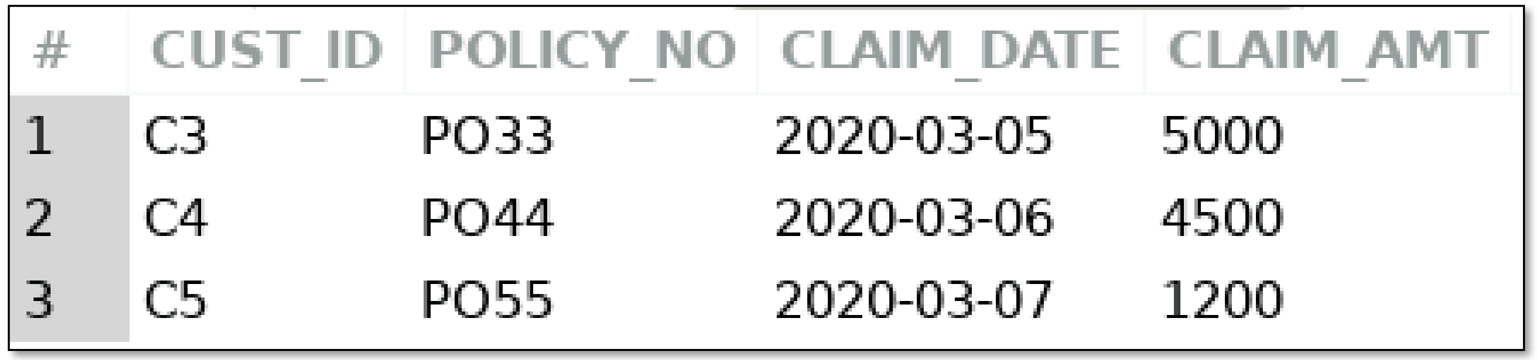
Query

```
SELECT A.* FROM B2C_customer as A
LEFT JOIN B2B_customer as B
ON A.CUST_ID=B.CUST_ID
WHERE B.CUST_ID IS NULL;
```

Assisted Practice: Joins Two



Output:



#	CUST_ID	POLICY_NO	CLAIM_DATE	CLAIM_AMT
1	C3	PO33	2020-03-05	5000
2	C4	PO44	2020-03-06	4500
3	C5	PO55	2020-03-07	1200

Operators in MySQL

SET Operators

DEFINITION

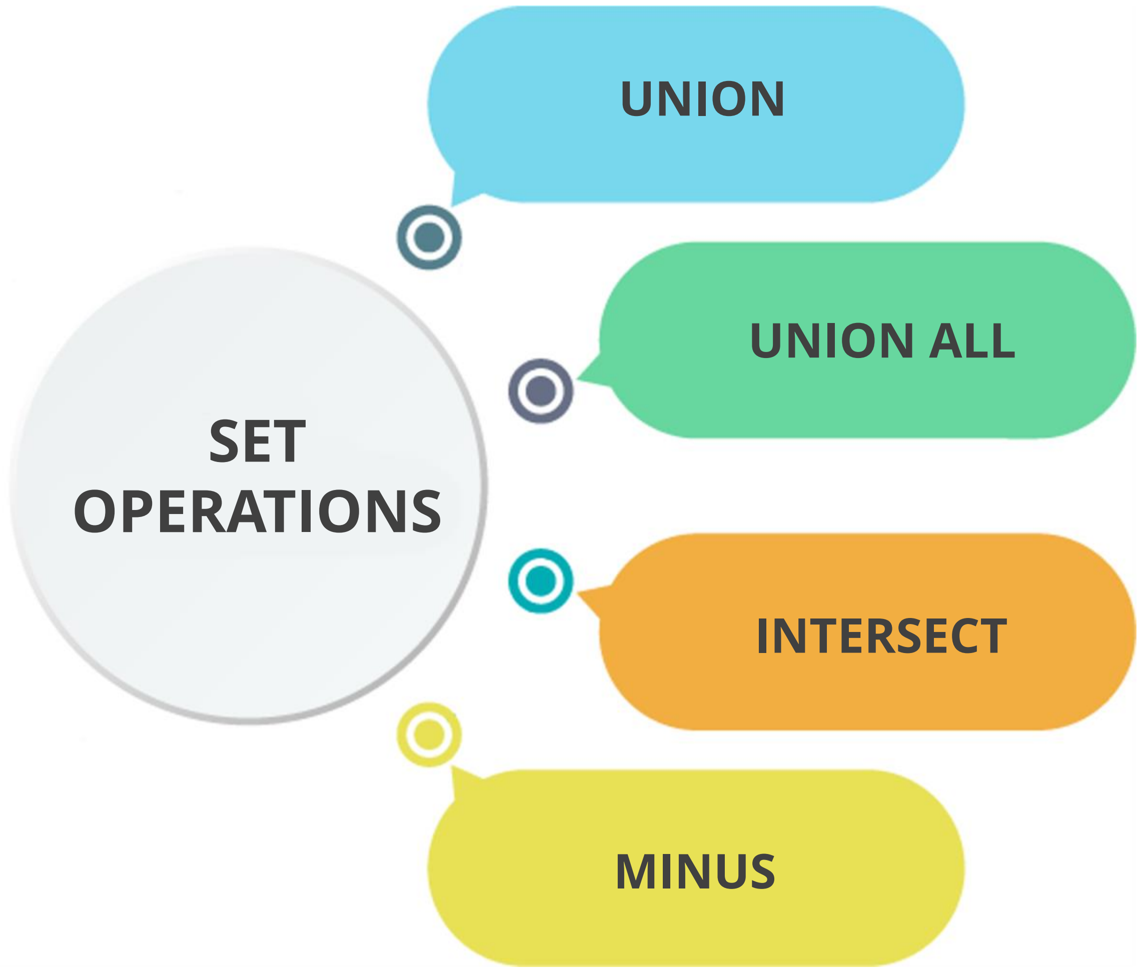
Set operators combine the results of two component queries into a single result.

Basic LOOP

FUNCTIONALITY

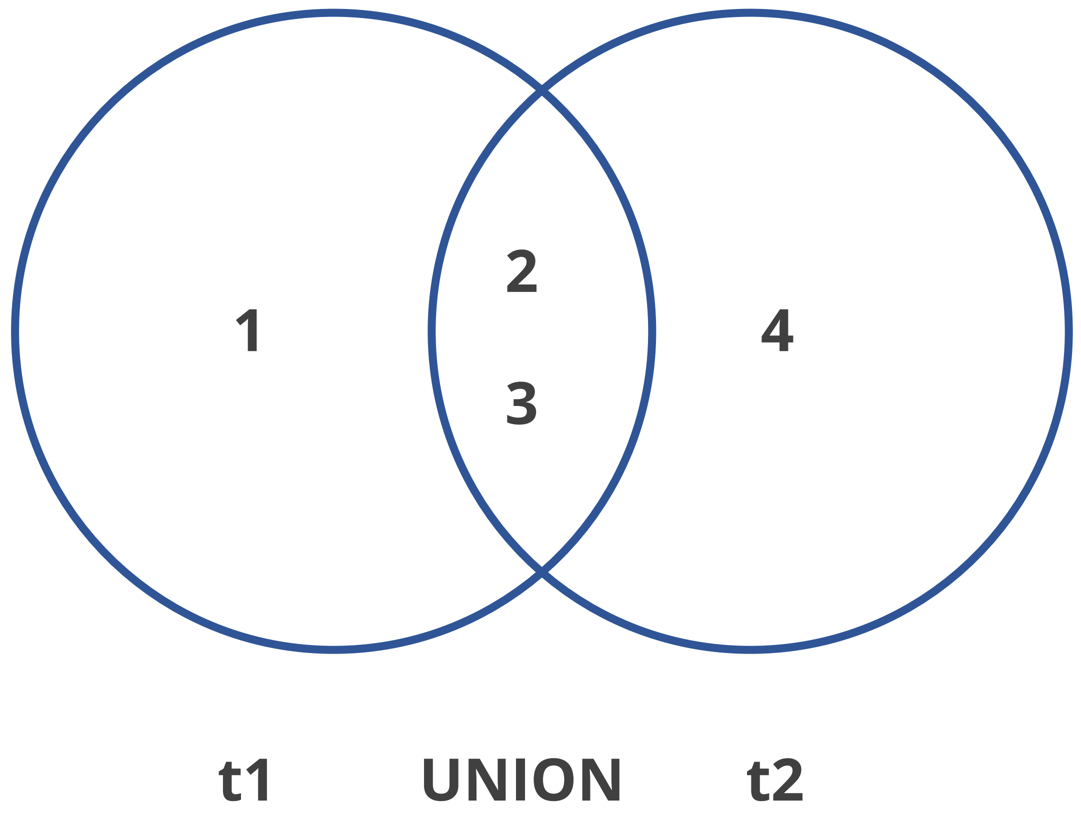
Set operators are used to get meaningful results from data stored in the table under different special conditions.

Types of SET Operators



INTERSECT and MINUS set operations are not supported by MySQL; however, they can be emulated by combining other MySQL components.

UNION Operator



- The UNION operator is used to combine two or more result sets from multiple SELECT statements into a single result set.
- By default, the UNION operator eliminates duplicate rows even if the DISTINCT operator is not explicitly provided.

Note: In this example, t1 contains 1, 2, and 3, while t2 contains 2, 3, and 4. After applying UNION, the output will be 1, 2, 3, and 4.

UNION Operator

UNION RULES

01 STEP

In all SELECT statements, the number of columns and their order must be the same.

02 STEP

The data type of columns must be compatible or same.

UNION Operator

Syntax

```
SELECT column_list
UNION [DISTINCT | ALL]
SELECT column_list
UNION [DISTINCT | ALL]
SELECT column_list
...
```

UNION Operator: Example

Problem Statement: Your manager wants you to provide the full names and departments of the employees along with the name and domain of projects as **name** and **department** from both the employee and project records tables.

Objective: Write an SQL query using the **UNION** operator to perform the union of the **EMP_RECORDS** and **PROJ_RECORDS** tables to obtain the required data in MySQL.

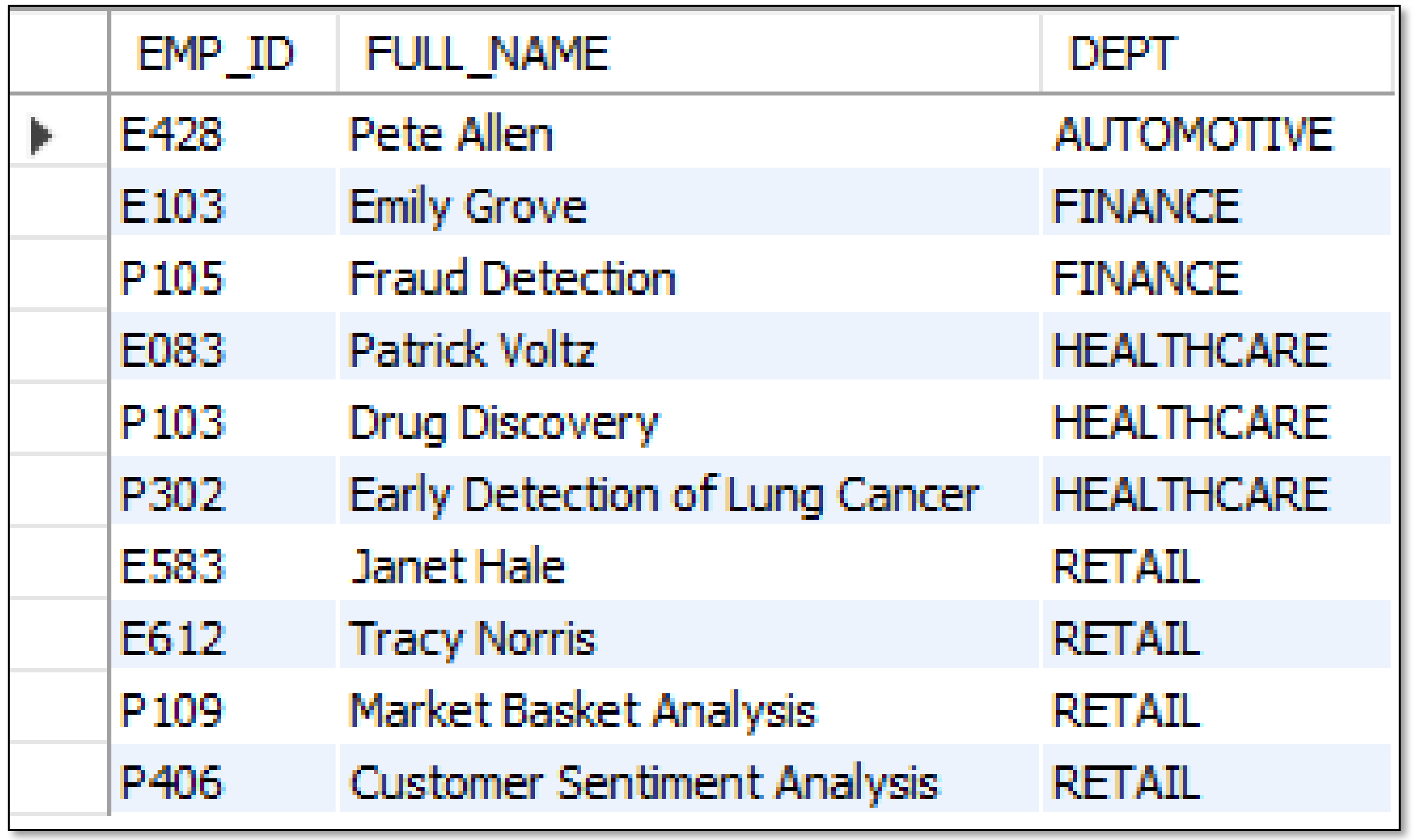
UNION Operator: Example

Step 1: Use the **UNION** operator in the **SELECT** statement to perform the union of the **EMP_RECORDS** and **PROJ_RECORDS** tables as shown below:

SQL Query

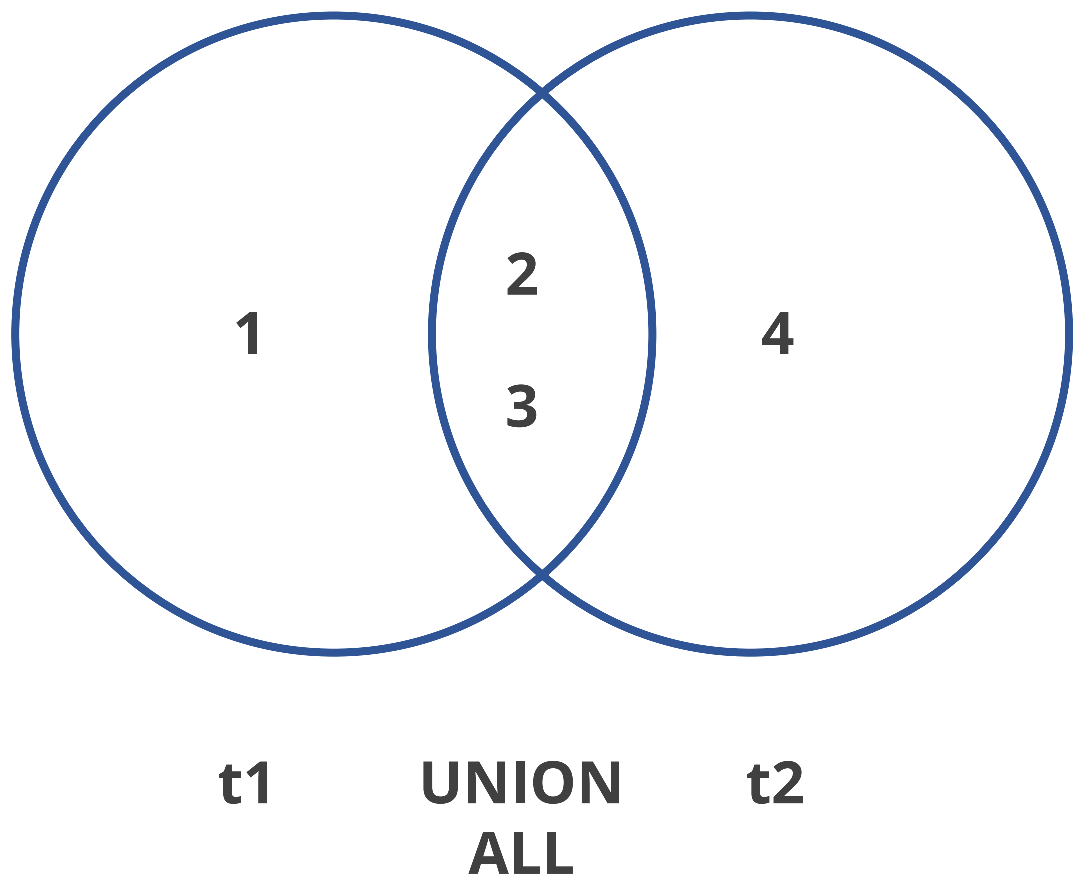
```
SELECT e.EMP_ID,

  CONCAT(e.FIRST_NAME,' ',e.LAST_NAME) AS
`FULL_NAME`,

  e.DEPT

FROM EMP_RECORDS AS e

WHERE e.ROLE IN ("MANAGER")

UNION

SELECT p.PROJ_ID, p.PROJ_NAME, p.DOMAIN

FROM PROJ_RECORDS AS p

ORDER BY DEPT, EMP_ID;
```

Output:



EMP_ID	FULL_NAME	DEPT
E428	Pete Allen	AUTOMOTIVE
E103	Emily Grove	FINANCE
P105	Fraud Detection	FINANCE
E083	Patrick Voltz	HEALTHCARE
P103	Drug Discovery	HEALTHCARE
P302	Early Detection of Lung Cancer	HEALTHCARE
E583	Janet Hale	RETAIL
E612	Tracy Norris	RETAIL
P109	Market Basket Analysis	RETAIL
P406	Customer Sentiment Analysis	RETAIL

UNION ALL Operator



- The UNION operator can be substituted by the UNION ALL operator to maintain the duplicate rows in the result set.
- The UNION ALL operator performs the same functions as the UNION operator, but it is significantly faster since it does not have to deal with duplicate rows.

Note: In this example, t1 contains 1, 2, and 3, while t2 contains 2, 3, and 4. After applying UNION ALL, the output will be 1, 2, 2, 3, 3, and 4.

UNION ALL Operator

Syntax

```
SELECT column_list
UNION ALL [DISTINCT | ALL]
SELECT column_list
UNION ALL [DISTINCT | ALL]
SELECT column_list
...
```

UNION ALL Operator: Example

Problem Statement: Your manager wants you to provide all the available entries with the full names of all employees and projects along with their department or domain as their domain in both the employee and project records tables together.

Objective: Write an SQL query using the **UNION ALL** operator to perform the union of the **EMP_RECORDS** and **PROJ_RECORDS** tables to obtain the required data while considering the duplicate rows in MySQL.

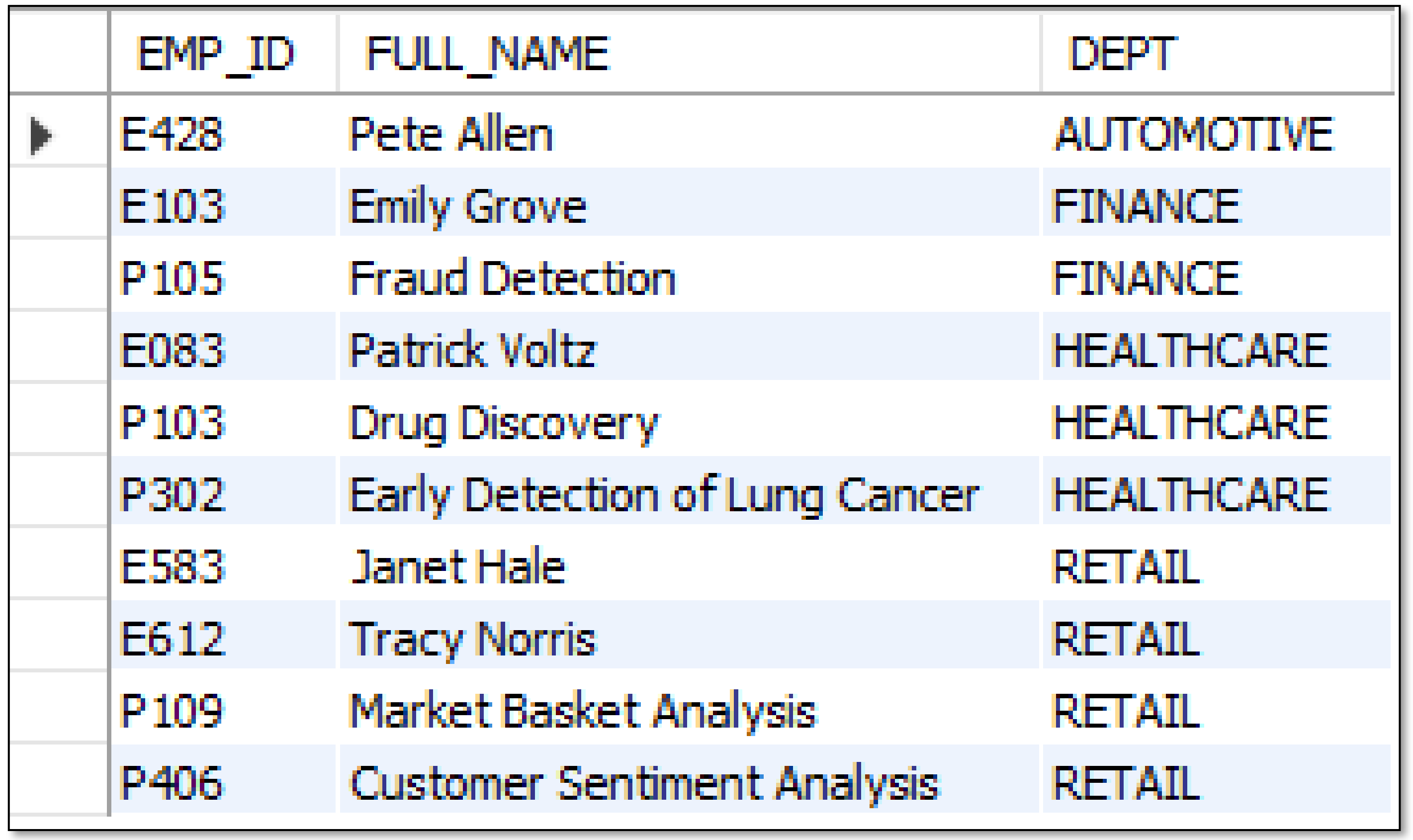
UNION ALL Operator: Example

Step 1: Use the **UNION ALL** operator in the **SELECT** statement to perform the union of the **EMP_RECORDS** and **PROJ_RECORDS** tables while considering the duplicate rows as shown below:

SQL Query

```
SELECT e.EMP_ID,
  CONCAT(e.FIRST_NAME,' ',e.LAST_NAME) AS
`FULL_NAME`,
  e.DEPT
FROM EMP_RECORDS e
WHERE e.ROLE IN ("MANAGER")
UNION ALL
SELECT p.PROJ_ID, p.PROJ_NAME, p.DOMAIN
FROM PROJ_RECORDS p
ORDER BY DEPT, EMP_ID;
```

Output:



EMP_ID	FULL_NAME	DEPT
E428	Pete Allen	AUTOMOTIVE
E103	Emily Grove	FINANCE
P105	Fraud Detection	FINANCE
E083	Patrick Voltz	HEALTHCARE
P103	Drug Discovery	HEALTHCARE
P302	Early Detection of Lung Cancer	HEALTHCARE
E583	Janet Hale	RETAIL
E612	Tracy Norris	RETAIL
P109	Market Basket Analysis	RETAIL
P406	Customer Sentiment Analysis	RETAIL

UNION vs. JOIN

UNION

- Combines the result set of two or more SELECT statements
- Appends result set vertically
- Combines data into new rows
- Number of columns selected from each table must be the same

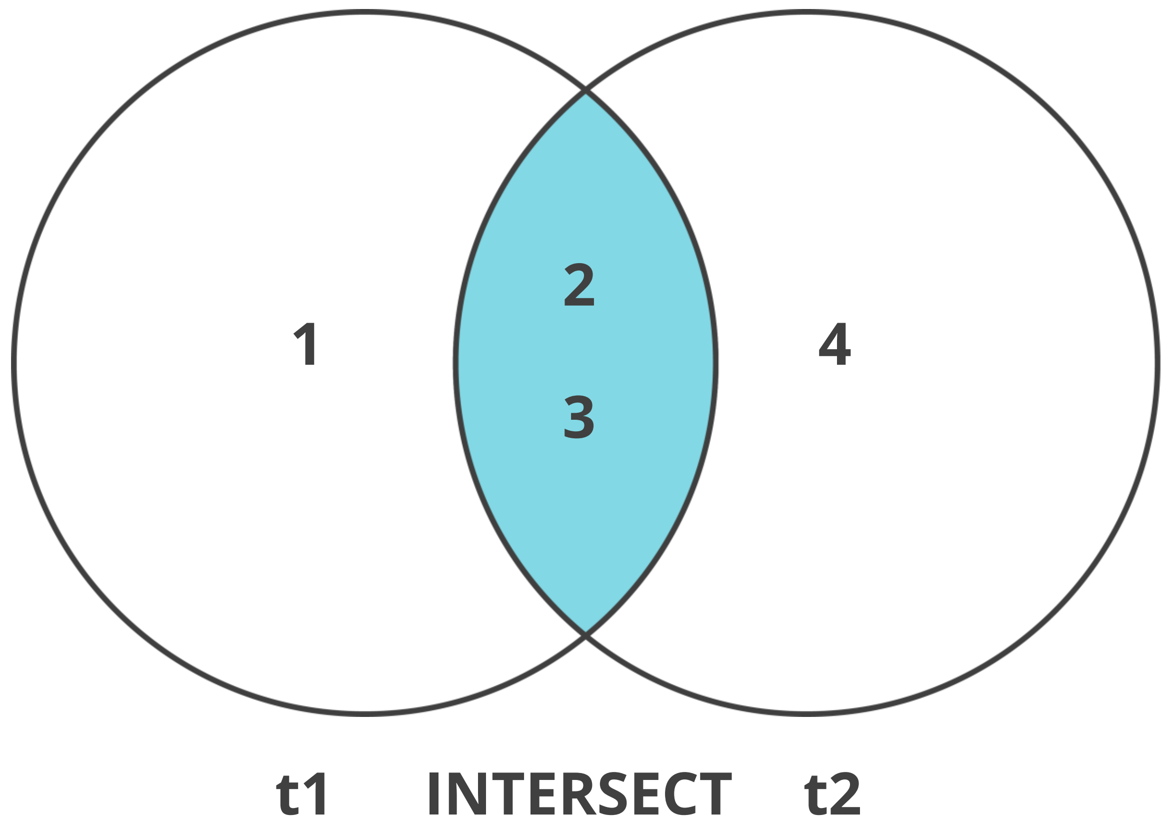
JOIN

- Combines data from various tables based on a matched condition between them
- Combines result sets horizontally
- Combines data into new columns
- Number of columns selected from each table may not be the same

UNION vs. JOIN

UNION	JOIN
Must have the same datatypes in corresponding columns selected from each table	Can have different datatypes in corresponding columns selected from each table
It returns distinct rows	It may not return distinct columns

INTERSECT Operator



- The INTERSECT operator compares the result sets of two or more queries and returns only the distinct rows produced by both queries.
- Unlike the UNION operator, the INTERSECT operator returns the intersection between two circles.

Note: In this example, t1 contains 1, 2, and 3, while t2 contains 2, 3, and 4. After applying INTERSECT, the output will be 2 and 3.

INTERSECT Operator

INTERSECT RULES

01 STEP

The number of columns and their order in the SELECT statement of all the SQL queries must be the same.

02 STEP

The data types of the corresponding columns must be compatible or same.

INTERSECT Operator

Syntax

```
(SELECT column_list
FROM table_1)
INTERSECT
(SELECT column_list
FROM table_2);
```

Emulating INTERSECT in MySQL

The INTERSECT operator is not supported by MySQL; however, it can be emulated.

The INTERSECT operator in MySQL can be emulated in two ways:

1. Using DISTINCT and INNER JOIN clause

2. Using IN and Subquery

INTERSECT Using DISTINCT and INNER JOIN

To perform INTERSECT operation, the INNER JOIN clause can be used to retrieve rows from both the left and right tables, while the DISTINCT operator can be used to eliminate duplicate rows.

Syntax

```
SELECT
   DISTINCT column_name_1
FROM table_1
   INNER JOIN table_2 USING(column_name_1);
```

INTERSECT Using DISTINCT and INNER JOIN: Example

Problem Statement: Your manager wants you to list the employee IDs of all the managers who are involved with at least one project.

Objective: Write an SQL query using either **INNER JOIN** clause with the **DISTINCT** keyword or **IN** operator with **Subquery** to simulate the **INTERSECT** operator on the **EMP_RECORDS** and **PROJ_ASSIGN** tables in MySQL.

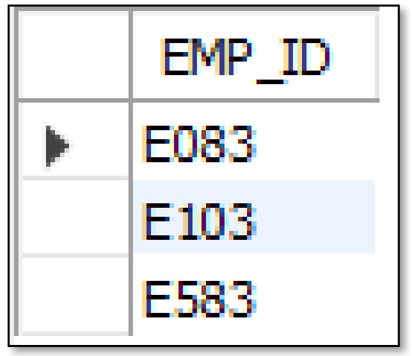
INTERSECT Using DISTINCT and INNER JOIN: Example

Step 1: Use the **INNER JOIN** clause with **DISTINCT** keyword in the **SELECT** statement to simulate the **INTERSECT** operator on the **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query

```
SELECT DISTINCT e.EMP_ID,
FROM EMP_RECORDS e
INNER JOIN PROJ_ASSIGN p ON(EMP_ID)
WHERE e.ROLE IN ("MANAGER")
ORDER BY e.EMP_ID;
```

Output:



```
EMP_ID
E083
E103
E583
```

INTERSECT Using IN and Subquery

To perform INTERSECT operation, the IN operator can be used in the outer query to retrieve rows that exist in first result set, while the DISTINCT operator can be used to ensure that only distinct values are selected.

Syntax

```
SELECT
  DISTINCT column_name_1,
FROM
  table_1
WHERE column_name_1 IN (
  SELECT column_name_1
  FROM table_2
);
```

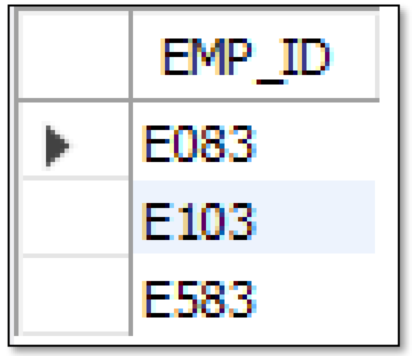
INTERSECT Using IN and Subquery: Example

Step 2: Use the **IN** operator with **Subquery** in the **SELECT** statement to simulate the **INTERSECT** operator on the **EMP_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query

```
SELECT DISTINCT e.EMP_ID
FROM EMP_RECORDS e
WHERE e.EMP_ID IN (
  SELECT a.EMP_ID
  FROM PROJ_ASSIGN a
) AND e.ROLE IN ("MANAGER")
ORDER BY e.EMP_ID;
```

Output:



EMP_ID
E083
E103
E583

MINUS Operator

The MINUS operator compares the results of two queries. It returns distinct rows from the result set of the first query that does not appear in the result set of the second query.

Syntax

```
SELECT select_list1
FROM table_name1
MINUS
SELECT select_list2
FROM table_name2;
```

MINUS Operator

MINUS RULES

01 STEP

The number of columns and their order in the SELECT statement of all the SQL queries must be the same.

02 STEP

The data types of the corresponding columns must be compatible or same.

Emulating MINUS in MySQL

The MINUS operator is not supported by MySQL; however, it can be emulated using the JOIN clause.

Syntax

```
SELECT
    select_list
FROM
    table1
LEFT JOIN table2
    ON join_predicate
WHERE
    table2.column_name IS NULL;
```

MINUS Using LEFT JOIN

To perform INTERSECT operation, the INNER JOIN clause can be used to retrieve rows from both the left and right tables, while the DISTINCT operator can be used to eliminate duplicate rows.

Syntax

```
SELECT
    column_name_1
FROM
    table_1
LEFT JOIN
    table_2 ON (column_name_1)
WHERE
    table_2.column_name_1 IS NULL;
```

MINUS Using LEFT JOIN: Example

Problem Statement: Your manager wants you to list the project IDs of the projects that are not yet assigned to any manager or employee.

Objective: Write an SQL query using **LEFT JOIN** clause to simulate the **MINUS** operator on the **PROJ_RECORDS** and **PROJ_ASSIGN** tables in MySQL.

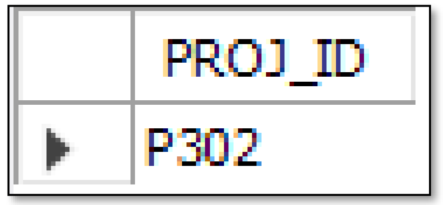
MINUS Using LEFT JOIN: Example

Step 1: Use the **LEFT JOIN** clause in the **SELECT** statement to simulate the **MINUS** operator on the **PROJ_RECORDS** and **PROJ_ASSIGN** tables as shown below:

SQL Query

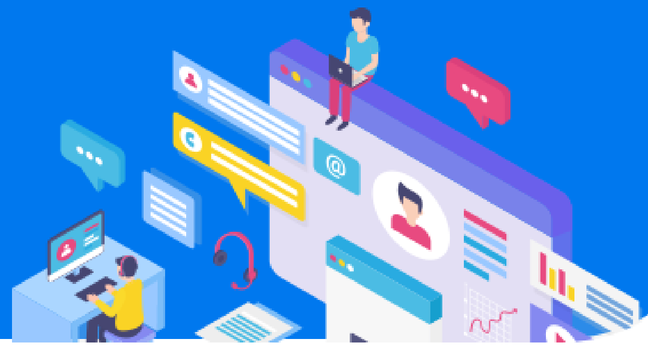
```
SELECT
    p.PROJ_ID
FROM
    PROJ_RECORDS p
LEFT JOIN
    PROJ_ASSIGN a ON (PROJ_ID)
WHERE
    a.PROJ_ID IS NULL;
```

Output:



	PROJ_ID
▶	P302

Assisted Practice: Union



Duration: 15 mins

Problem statement: You have joined as an analyst in the talent acquisition team of Sun Tech. The team collects the data for new hires in a separate table for every month, such as the data hires for January 2025 goes into the **NH_JAN_2025** table and the data for hires for February 2025 goes to **NH_FEB_2025**. You have been asked to bring the data for January and February into a single table.

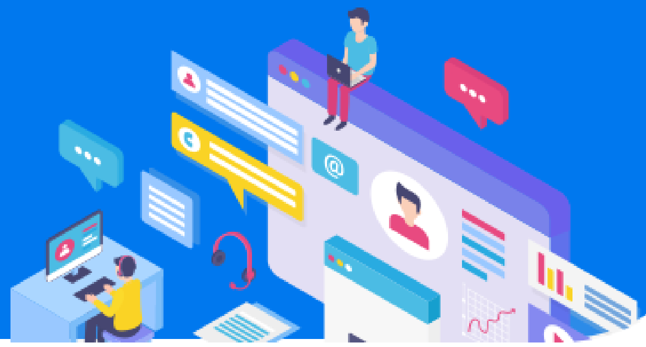
NH_JAN_2025

EMP_ID	EMP_NAME	DEPT
11211	Abhishek	Sales
11244	Rahul	Marketing
11289	Arya	Analytics

NH_FEB_2025

EMP_ID	EMP_NAME	DEPT
11411	Sam	Analytics
11452	John	Marketing
11565	Fatima	Support

Assisted Practice: Union



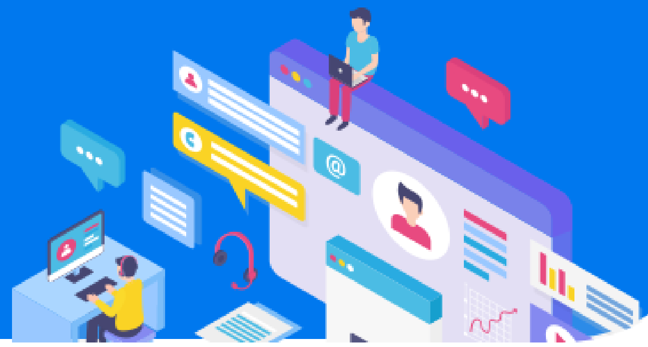
Steps to be performed:

Step 01: Write a query to create two tables (**NH_JAN_2025** and **NH_FEB_2025**) as shown in the problem statement and fill the data shown in those tables in SQL

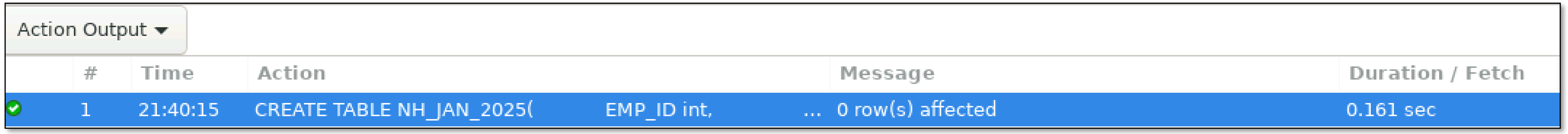
CREATE

```
CREATE TABLE NH_JAN_2025(EMP_ID int, EMP_NAME text, DEPT text);

CREATE TABLE NH_FEB_2025(EMP_ID int, EMP_NAME text, DEPT text);
```

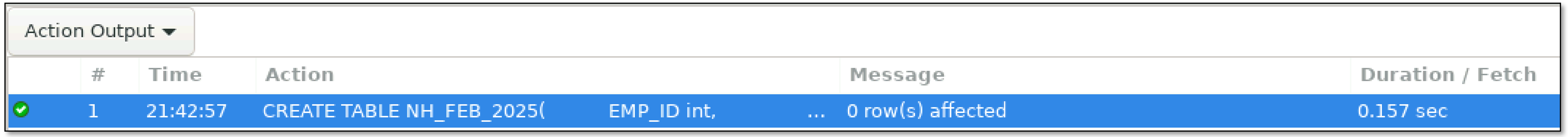
Assisted Practice: Union



Output:



```
Action Output ▾
   #   Time      Action                                            Message              Duration / Fetch
✓  1   21:40:15  CREATE TABLE NH_JAN_2025(      EMP_ID int,     ... 0 row(s) affected   0.161 sec
```



```
Action Output ▾
   #   Time      Action                                            Message              Duration / Fetch
✓  1   21:42:57  CREATE TABLE NH_FEB_2025(      EMP_ID int,     ... 0 row(s) affected   0.157 sec
```

Assisted Practice: Union

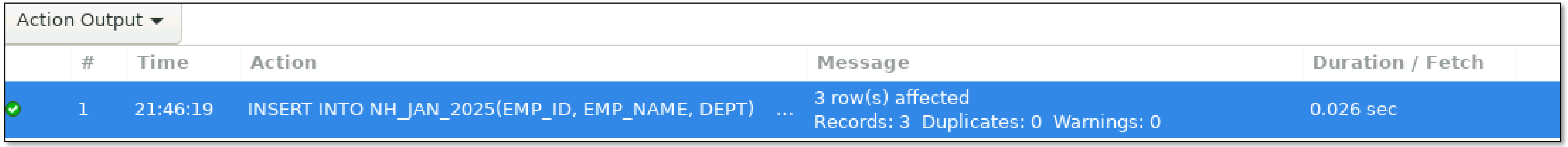
Step 02: Write a query to insert records in both tables

SQL Query

```
INSERT INTO NH_JAN_2025(EMP_ID, EMP_NAME, DEPT)VALUES(11211, "Abhishek", "Sales"),(11244,
"Rahul", "Marketing"),(11289, "Arya" , "Analytics");

INSERT INTO NH_FEB_2025(EMP_ID, EMP_NAME, DEPT)VALUES(11411, "Sam", "Analytics"),(11452,
"John", "Marketing"),  (11565, "Fatima" , "Support");
```

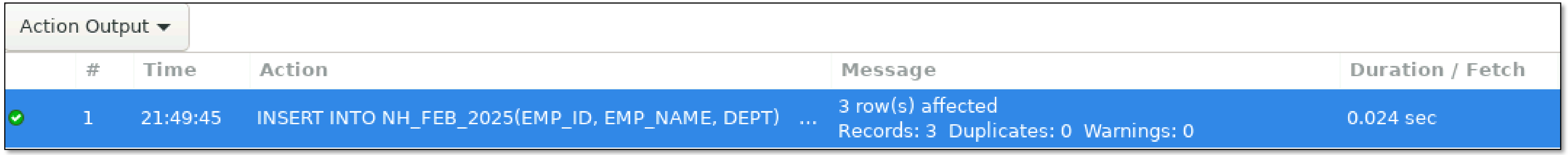
Assisted Practice: Union

Output:



Action Output

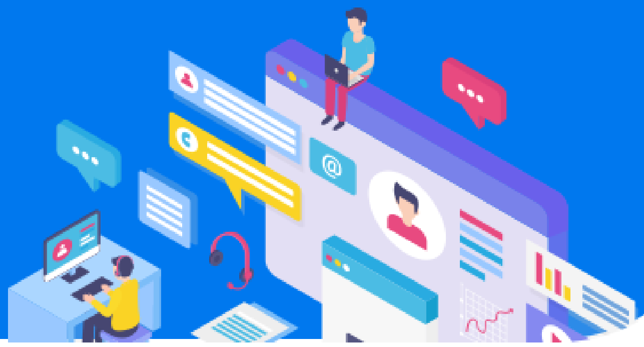
```
   #   Time      Action                                             Message                                       Duration / Fetch
✓  1   21:46:19  INSERT INTO NH_JAN_2025(EMP_ID, EMP_NAME, DEPT) ...  3 row(s) affected                             0.026 sec
                                                                    Records: 3  Duplicates: 0  Warnings: 0
```



Action Output

```
   #   Time      Action                                             Message                                       Duration / Fetch
✓  1   21:49:45  INSERT INTO NH_FEB_2025(EMP_ID, EMP_NAME, DEPT) ...  3 row(s) affected                             0.024 sec
                                                                    Records: 3  Duplicates: 0  Warnings: 0
```

Assisted Practice: Union

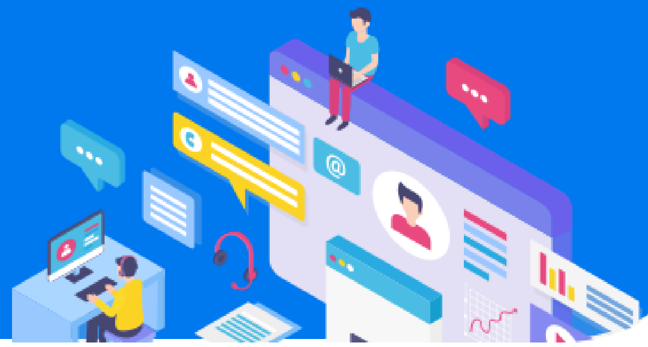


Step 03: Write a query to show the two tables (**NH_JAN_2025** and **NH_FEB_2025**)

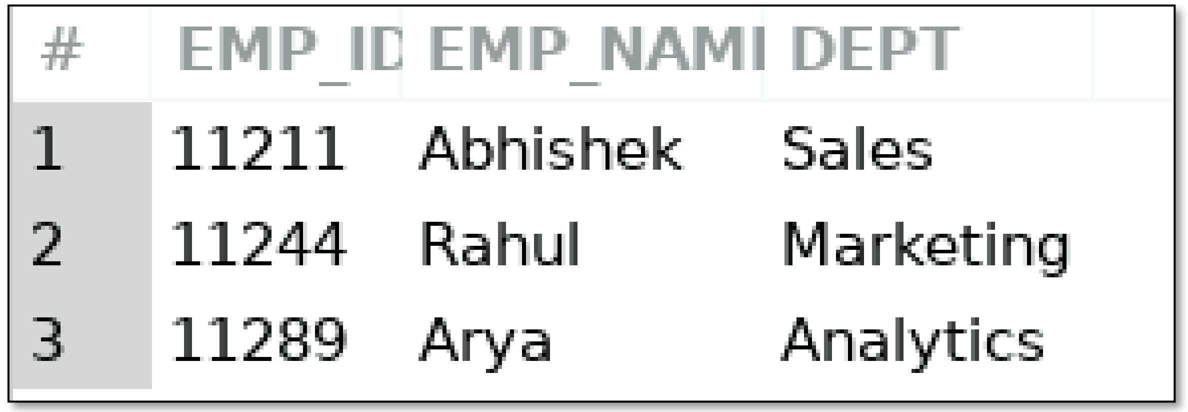
SQL Query

```
SELECT * FROM NH_JAN_2025;
SELECT * FROM NH_FEB_2025;
```

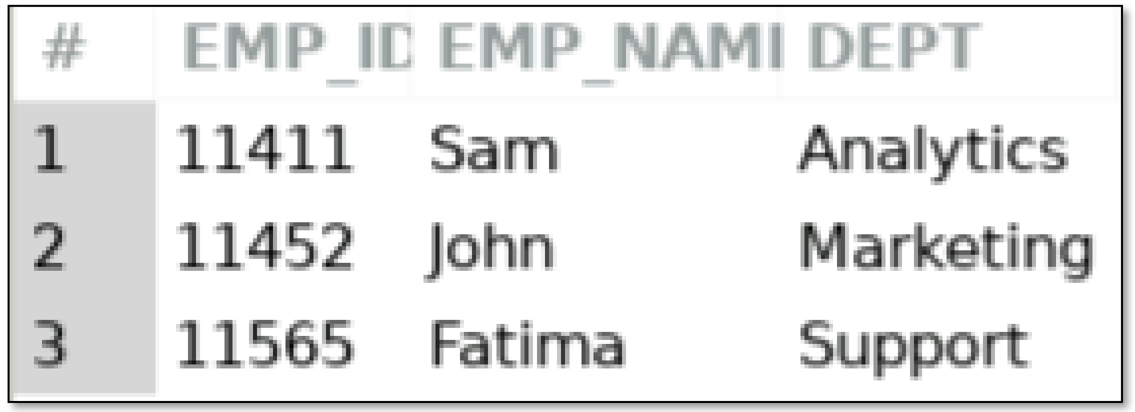
Assisted Practice: Union



Output:

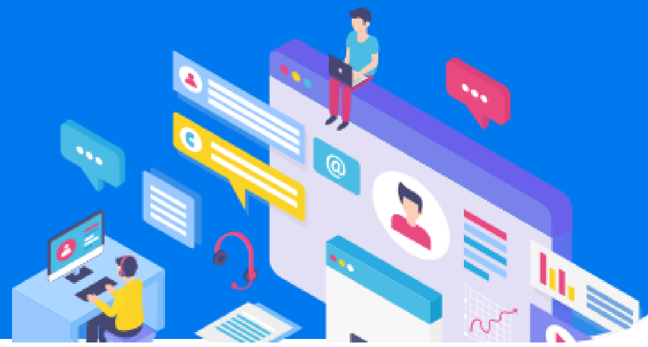


#	EMP_ID	EMP_NAMI	DEPT
1	11211	Abhishek	Sales
2	11244	Rahul	Marketing
3	11289	Arya	Analytics



#	EMP_ID	EMP_NAMI	DEPT
1	11411	Sam	Analytics
2	11452	John	Marketing
3	11565	Fatima	Support

Assisted Practice: Union



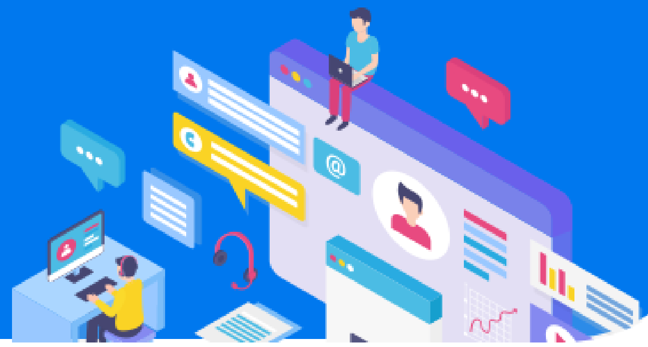
Step 04: Write a query to club the data of both tables into a single table along with the **MONTH** column

Hint: The **MONTH** column can be created by passing a string as a column name. This column should have the month and year values in the data.

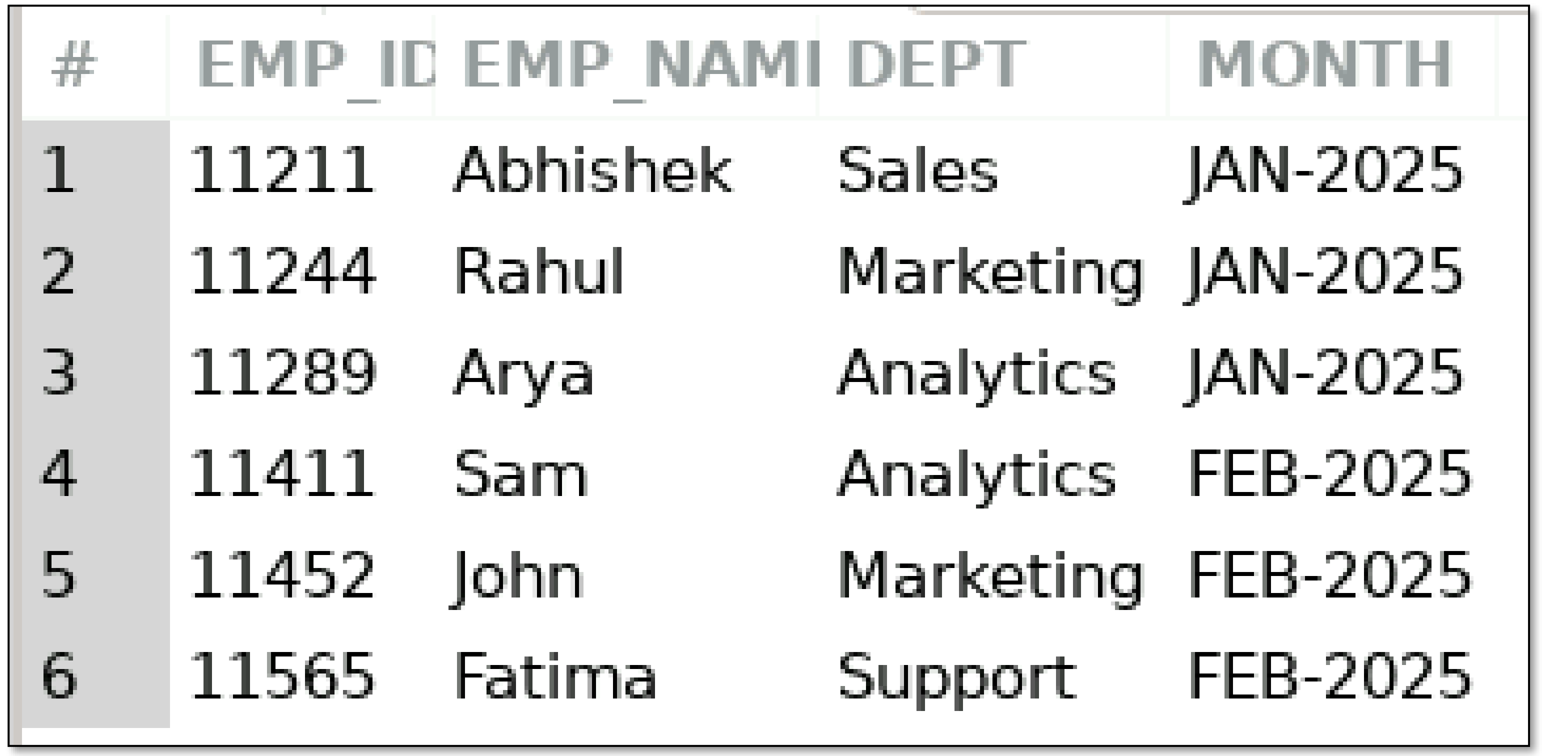
SQL Query

```
SELECT *, "JAN-2025" as MONTH FROM NH_JAN_2025
UNION
SELECT *, "FEB-2025" as MONTH FROM NH_FEB_2025
```

Assisted Practice: Union



Output:



#	EMP_ID	EMP_NAMI	DEPT	MONTH
1	11211	Abhishek	Sales	JAN-2025
2	11244	Rahul	Marketing	JAN-2025
3	11289	Arya	Analytics	JAN-2025
4	11411	Sam	Analytics	FEB-2025
5	11452	John	Marketing	FEB-2025
6	11565	Fatima	Support	FEB-2025

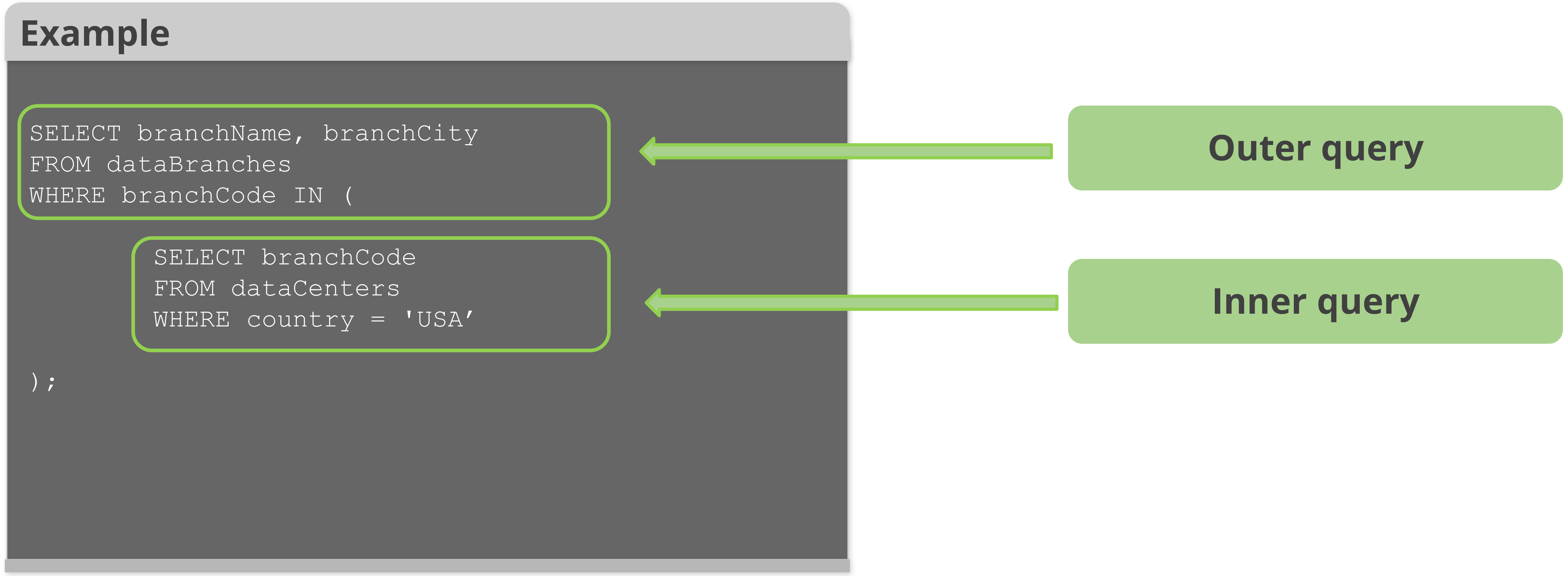
Subquery in SQL

Subquery in SQL

SQL Subquery

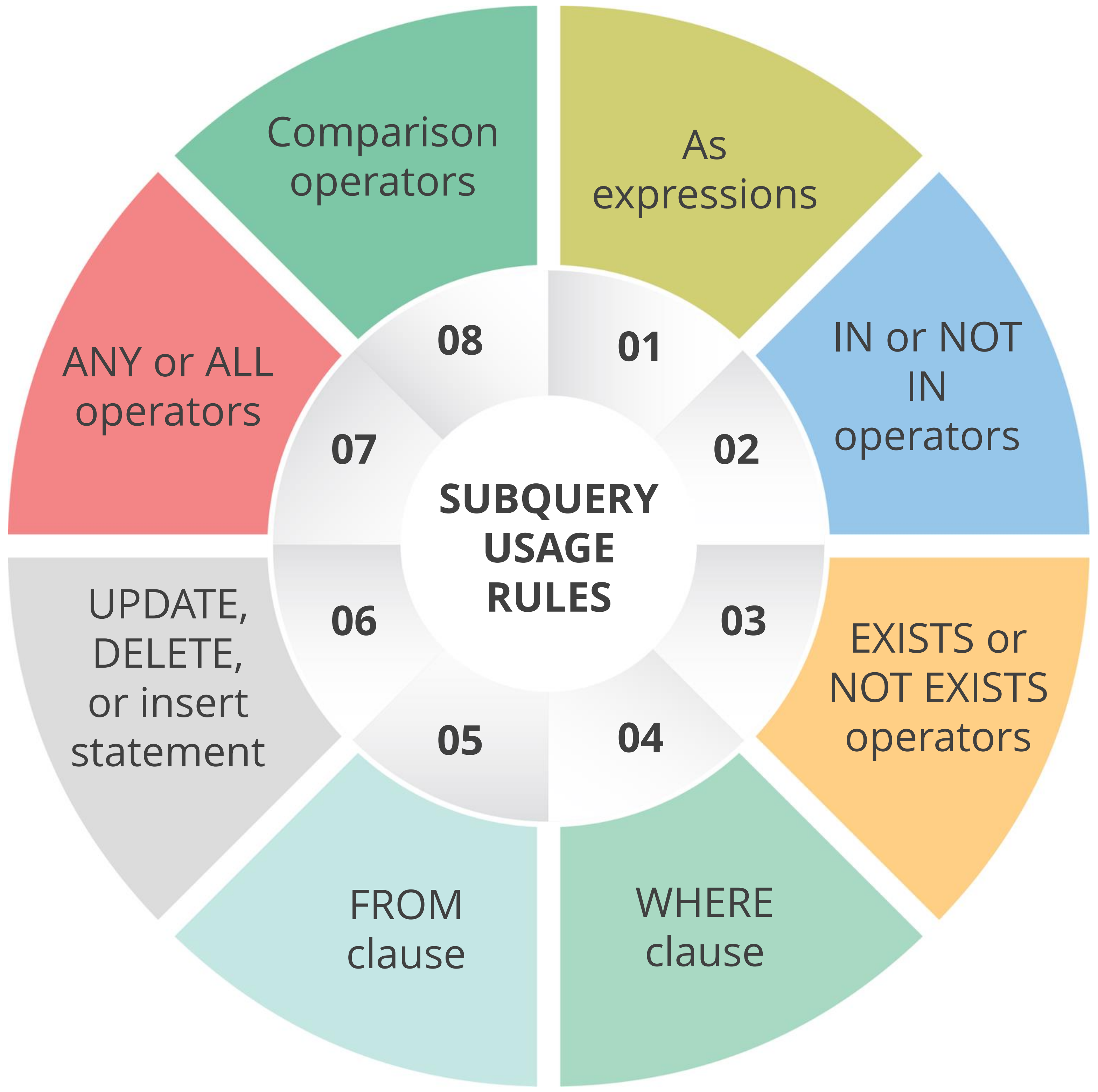
- A subquery is a query nested within another query such as SELECT, INSERT, UPDATE, or DELETE.
- It is also called an **Inner Query** or **Inner Select** while the statement that contains the subquery is called an outer query or outer select.
- It can be used anywhere an expression is used and must be closed in parentheses.
- A subquery can also be nested within another subquery.

Subquery in SQL



```
SELECT branchName, branchCity
FROM dataBranches
WHERE branchCode IN (

        SELECT branchCode
        FROM dataCenters
        WHERE country = 'USA’

);
```

Subquery Rules



Subqueries with Statements and Operators

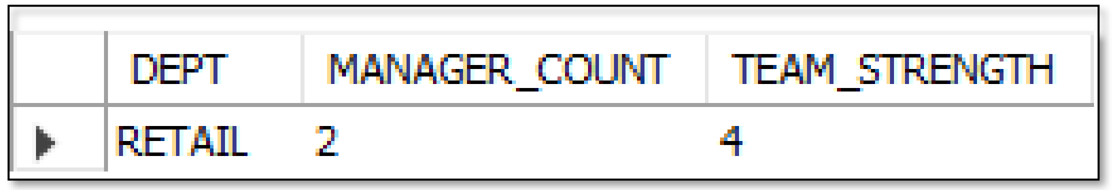
Subquery as Expressions

A subquery that returns a single value can be used as an expression.

SQL Query

```
SELECT m.DEPT, COUNT(DISTINCT m.EMP_ID) AS
`MANAGER_COUNT`,
  ( SELECT COUNT(DISTINCT e.EMP_ID) FROM EMP_RECORDS e
    WHERE e.ROLE NOT IN ("MANAGER", "PRESIDENT", "CEO")
    AND e.DEPT IN ("RETAIL") ) AS `TEAM_STRENGTH`
FROM EMP_RECORDS m
WHERE m.ROLE IN ("MANAGER") AND m.DEPT IN ("RETAIL");
```

Output:



DEPT	MANAGER_COUNT	TEAM_STRENGTH
RETAIL	2	4

Let's say you need to determine the count of managers and the total team strength excluding them in the retail domain in MySQL.

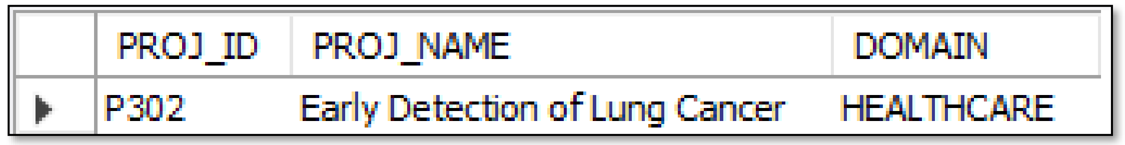
Subquery with WHERE Clause

A subquery can be used with a WHERE clause.

SQL Query

```
SELECT p.PROJ_ID, p.PROJ_NAME, p.DOMAIN
FROM PROJ_RECORDS p
WHERE p.PROJ_ID NOT IN (
  SELECT DISTINCT a.PROJ_ID
  FROM PROJ_ASSIGN a
) AND p.STATUS IN ("YTS");
```

Output:



PROJ_ID	PROJ_NAME	DOMAIN
P302	Early Detection of Lung Cancer	HEALTHCARE

Let's say you need to determine the list of upcoming projects with no manager and team member assigned to them in MySQL.

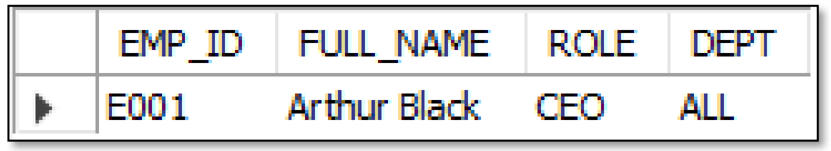
Subquery with Comparison Operators

Comparison operators can be used to compare a single value returned by a subquery with the expression in the WHERE clause.

SQL Query

```
SELECT
  e.EMP_ID,
  CONCAT(e.FIRST_NAME,' ',e.LAST_NAME) AS `FULL_NAME`,
  e.ROLE, e.DEPT
FROM EMP_RECORDS e
WHERE e.EXP = (SELECT MAX(EXP) FROM EMP_RECORDS);
```

Output:



EMP_ID	FULL_NAME	ROLE	DEPT
E001	Arthur Black	CEO	ALL

Let's say you need to determine the employee with the highest experience in the organization in MySQL.

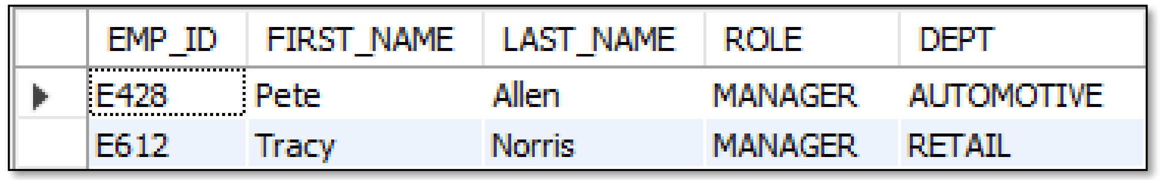
Subquery with IN and NOT IN Operators

A subquery that returns more than one value can be used with IN or NOT IN operators in the WHERE clause.

SQL Query

```
SELECT
  e.EMP_ID, e.FIRST_NAME, e.LAST_NAME, e.ROLE, e.DEPT
FROM EMP_RECORDS e
WHERE e.EMP_ID NOT IN (
    SELECT DISTINCT a.EMP_ID FROM PROJ_ASSIGN a
  ) AND e.ROLE IN ("MANAGER");
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	ROLE	DEPT
E428	Pete	Allen	MANAGER	AUTOMOTIVE
E612	Tracy	Norris	MANAGER	RETAIL

Let’s say you need to determine the list of all managers who have not been assigned to any projects in the organization in MySQL.

Subquery with ANY Operator

A subquery that returns a list of values that can be used with ANY operator in the WHERE clause.

The ANY operator compares each value provided by the subquery with the comparison expression and returns TRUE if any comparison pair evaluates to TRUE; otherwise, it returns FALSE.

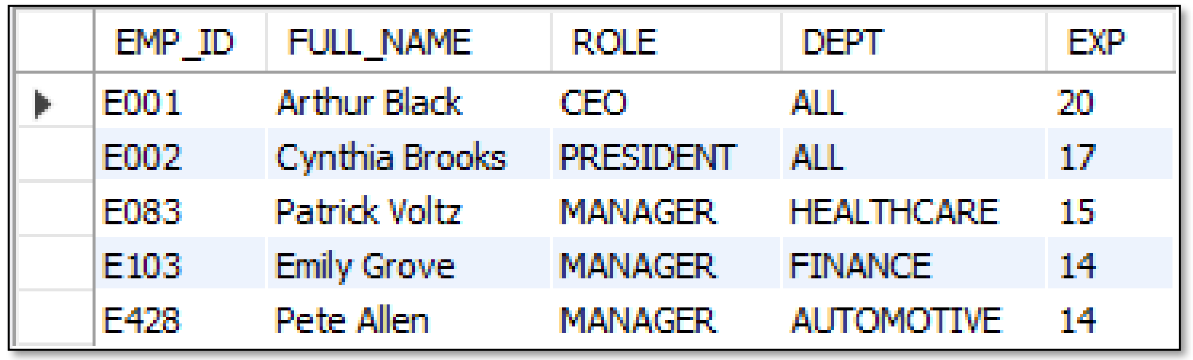
Subquery with ANY Operator

Let's say you need to determine any five employees with more than or equal to the average experience of all employees in the organization in MySQL.

SQL Query

```
SELECT

  e.EMP_ID,

  CONCAT(e.FIRST_NAME,' ',e.LAST_NAME) AS
`FULL_NAME`,

  e.ROLE, e.DEPT, e.EXP

FROM EMP_RECORDS e

WHERE e.EXP >= ANY (SELECT AVG(EXP) FROM EMP_RECORDS)

LIMIT 5;
```

Output:



EMP_ID	FULL_NAME	ROLE	DEPT	EXP
E001	Arthur Black	CEO	ALL	20
E002	Cynthia Brooks	PRESIDENT	ALL	17
E083	Patrick Voltz	MANAGER	HEALTHCARE	15
E103	Emily Grove	MANAGER	FINANCE	14
E428	Pete Allen	MANAGER	AUTOMOTIVE	14

Note: In this example, Employee Experience is compared with the average experience, which is a single value. Alternatively, you can use ANY to compare different values with multiple values. For example, when you want to compare the experience of any employee with that of the manager(s).

Subquery with ALL Operator

A subquery that returns a list of values can also be used with ALL operators in the WHERE clause.

The ALL operator compares each value provided by the subquery with the comparison expression and returns TRUE if all the comparison pairs evaluate to TRUE; otherwise, it returns FALSE.

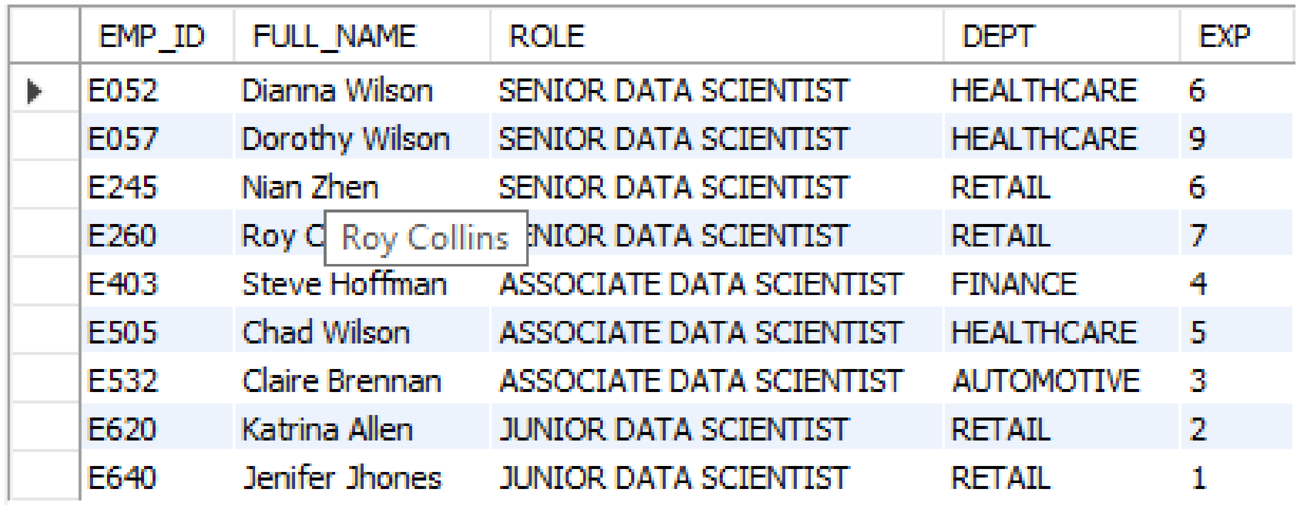
Subquery with ALL Operator

Let's say you need to determine all the employees with less than the average experience of all employees in MySQL.

SQL Query

```
SELECT

  e.EMP_ID,

  CONCAT(e.FIRST_NAME,' ',e.LAST_NAME) AS
`FULL_NAME`,

  e.ROLE, e.DEPT, e.EXP

FROM EMP_RECORDS e

WHERE e.EXP < ALL (SELECT AVG(EXP) FROM EMP_RECORDS);
```

Output:



EMP_ID	FULL_NAME	ROLE	DEPT	EXP
E052	Dianna Wilson	SENIOR DATA SCIENTIST	HEALTHCARE	6
E057	Dorothy Wilson	SENIOR DATA SCIENTIST	HEALTHCARE	9
E245	Nian Zhen	SENIOR DATA SCIENTIST	RETAIL	6
E260	Roy C Roy Collins	NIOR DATA SCIENTIST	RETAIL	7
E403	Steve Hoffman	ASSOCIATE DATA SCIENTIST	FINANCE	4
E505	Chad Wilson	ASSOCIATE DATA SCIENTIST	HEALTHCARE	5
E532	Claire Brennan	ASSOCIATE DATA SCIENTIST	AUTOMOTIVE	3
E620	Katrina Allen	JUNIOR DATA SCIENTIST	RETAIL	2
E640	Jenifer Jhones	JUNIOR DATA SCIENTIST	RETAIL	1

Note: In this example, Employee Experience is compared with the average experience, which is a single value. Alternatively, you can use ANY to compare different values with multiple values. For example, when you want to compare the experience of any employee with that of the manager(s).

Subquery with EXISTS or NOT EXISTS Operators

A subquery can also be used with the EXISTS and NOT EXISTS operators.

The EXISTS operator returns TRUE if the subquery returns the results; otherwise, it returns FALSE. The NOT EXISTS operator is opposite to the EXISTS operator.

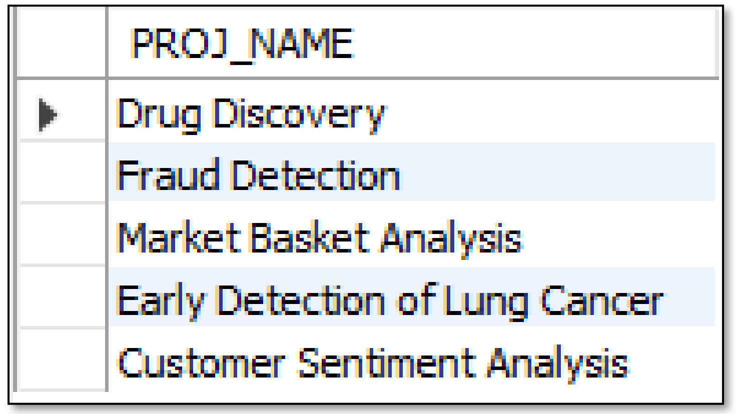
Subquery with EXISTS or NOT EXISTS Operators

Let's say you need to print the names of all the projects only if even one project is assigned to any employee in MySQL.

SQL Query

```
SELECT PROJ_NAME
FROM PROJ_RECORDS
WHERE EXISTS (
  SELECT PROJ_ID
  FROM PROJ_ASSIGN
) ORDER BY PROJ_ID;
```

Output:



PROJ_NAME
Drug Discovery
Fraud Detection
Market Basket Analysis
Early Detection of Lung Cancer
Customer Sentiment Analysis

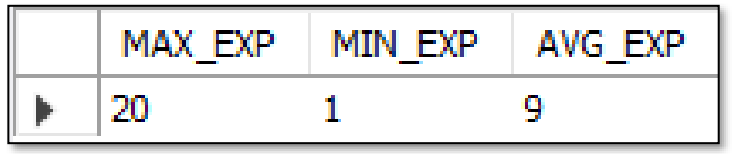
Subquery in the FROM Clause

The FROM clause creates a temporary table from the result set returned by a subquery, often known as a derived table or materialized subquery.

SQL Query

```
SELECT
  MAX(EXP) AS `MAX_EXP`, MIN(EXP) AS `MIN_EXP`,
  FLOOR(AVG(EXP)) AS `AVG_EXP`
FROM (
     SELECT EMP_ID, EXP FROM EMP_RECORDS
     GROUP BY EXP ORDER BY EXP
) AS TOTAL_EXP;
```

Output:



	MAX_EXP	MIN_EXP	AVG_EXP
▶	20	1	9

Let's say you need to determine the maximum, minimum, and average employee experience in the organization in MySQL.

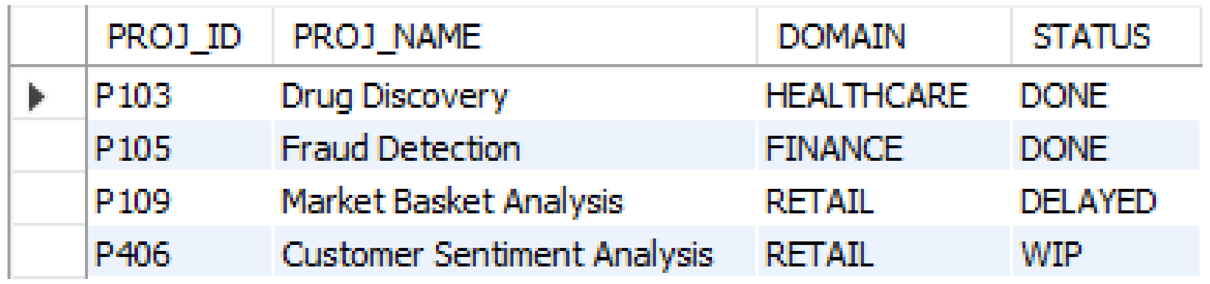
Subquery with SELECT Statement

Subqueries are frequently used with the SELECT statement.

SQL Query

```
SELECT
  PROJ_ID, PROJ_NAME,
  DOMAIN, STATUS
FROM PROJ_RECORDS
WHERE PROJ_ID IN (
  SELECT DISTINCT PROJ_ID FROM PROJ_ASSIGN
) ORDER BY PROJ_ID, DOMAIN;
```

Output:



PROJ_ID	PROJ_NAME	DOMAIN	STATUS
P103	Drug Discovery	HEALTHCARE	DONE
P105	Fraud Detection	FINANCE	DONE
P109	Market Basket Analysis	RETAIL	DELAYED
P406	Customer Sentiment Analysis	RETAIL	WIP

Let’s say you need to determine all those projects that are assigned to at least one of the employees in MySQL.

Subquery with INSERT Statement

Let us make a table comparable to **PROJ_RECORDS** with the name **PROJ_RECORDS_BKUP** in the **PROJ_DB** database in MySQL.

SQL Query

```
CREATE TABLE IF NOT EXISTS PROJ_DB.PROJ_RECORDS_BKUP (
    PROJ_ID VARCHAR(4) NOT NULL CHECK (SUBSTR(PROJ_ID,1,1) = 'P'),
    PROJ_NAME VARCHAR(200) NOT NULL,
    DOMAIN VARCHAR(100) NOT NULL,
    START_DATE DATE NOT NULL CHECK (START_DATE >= '2021-04-01'),
    CLOSURE_DATE DATE NOT NULL CHECK (CLOSURE_DATE <= '2022-03-30'),
    DEV_QTR VARCHAR(2) NOT NULL,
    STATUS VARCHAR(7),
    CONSTRAINT chk_qtr_2 CHECK (DEV_QTR IN ('Q1', 'Q2', 'Q3', 'Q4')),
    CONSTRAINT chk_status_2 CHECK (STATUS IN ('YTS', 'WIP', 'DONE', 'DELAYED'))
)  ENGINE=INNODB;
```

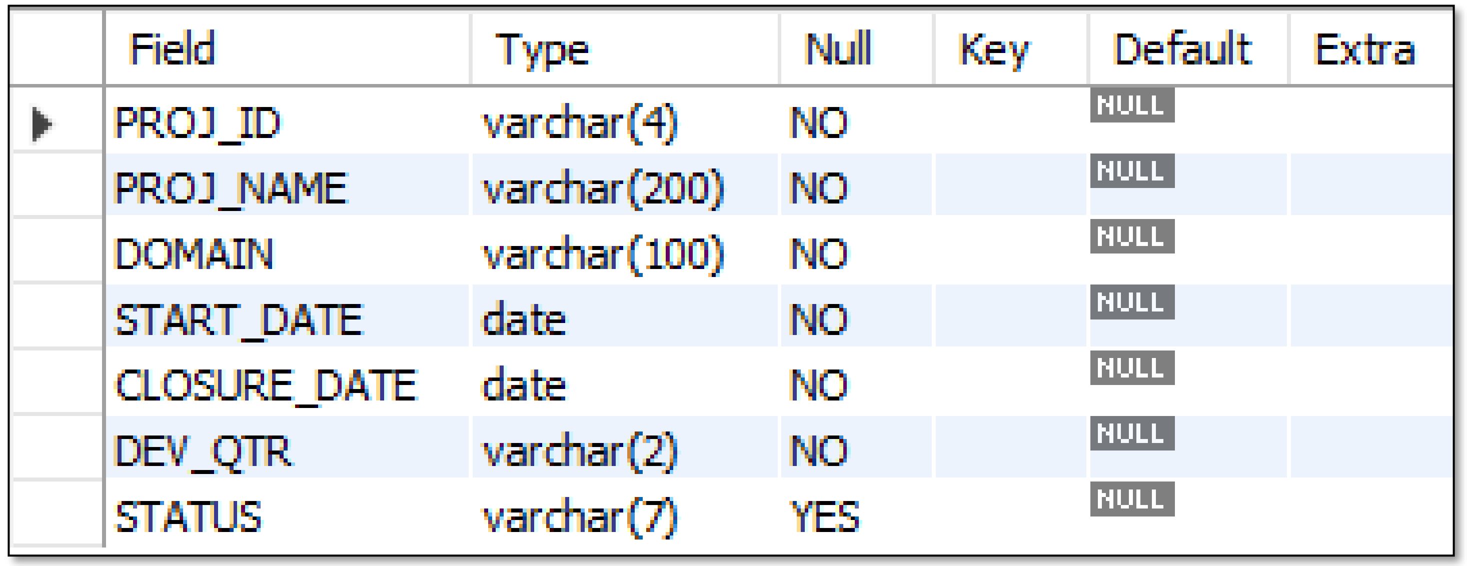
Subquery with INSERT Statement

Let us now analyze the structure of the **PROJ_RECORDS_BKUP** table created in MySQL.

SQL Query

```
DESCRIBE PROJ_DB.PROJ_RECORDS_BKUP;
```

Output:



Field	Type	Null	Key	Default	Extra
PROJ_ID	varchar(4)	NO		NULL	
PROJ_NAME	varchar(200)	NO		NULL	
DOMAIN	varchar(100)	NO		NULL	
START_DATE	date	NO		NULL	
CLOSURE_DATE	date	NO		NULL	
DEV_QTR	varchar(2)	NO		NULL	
STATUS	varchar(7)	YES		NULL	

Subquery with INSERT Statement

Let’s say you need to back up the data of all those projects which have no employee assigned into the new **PROJ_RECORDS_BKUP** table in MySQL.

SQL Query

```
INSERT INTO PROJ_RECORDS_BKUP
SELECT * FROM PROJ_RECORDS p
WHERE p.PROJ_ID NOT IN (
  SELECT DISTINCT PROJ_ID FROM PROJ_ASSIGN
);
```

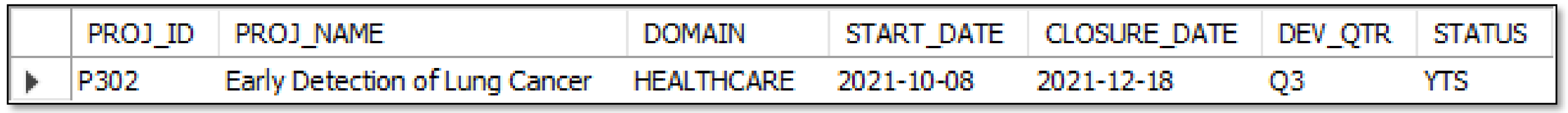
Subquery with INSERT Statement

Now, you need to fetch the data for all the columns from the **PROJ_RECORDS_BKUP** table in MySQL to verify the table data.

SQL Query

```
SELECT * FROM PROJ_DB.EMP_RECORDS_BKUP;
```

Output:



PROJ_ID	PROJ_NAME	DOMAIN	START_DATE	CLOSURE_DATE	DEV_QTR	STATUS
P302	Early Detection of Lung Cancer	HEALTHCARE	2021-10-08	2021-12-18	Q3	YTS

Subquery with UPDATE Statement

The subquery can be used in conjunction with the UPDATE statement to update either single or multiple columns in a table.

SQL Query

```
UPDATE PROJ_RECORDS_BKUP
SET DEV_QTR = 'Q2'
WHERE (START_DATE, CLOSURE_DATE) IN (
  SELECT START_DATE, CLOSURE_DATE
  FROM PROJ_RECORDS
  WHERE DEV_QTR = 'Q3'
);
```

Let’s say you need to change the development quarter along with its start and closure dates for one of the projects in the **PROJ_RECORDS_BKUP** table in MySQL.

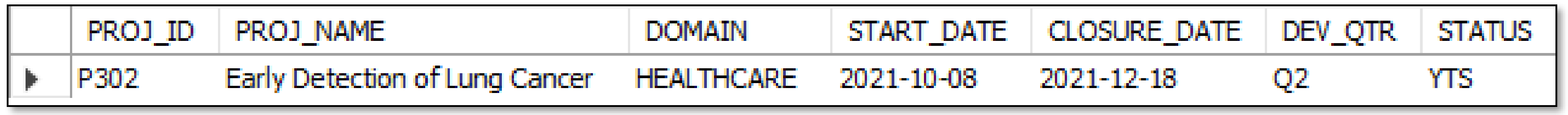
Subquery with UPDATE Statement

Now, you need to check the updated data in all the columns of the **PROJ_RECORDS_BKUP** table in MySQL to verify the update.

SQL Query

```
SELECT * FROM PROJ_DB.EMP_RECORDS_BKUP;
```

Output:



	PROJ_ID	PROJ_NAME	DOMAIN	START_DATE	CLOSURE_DATE	DEV_QTR	STATUS
▶	P302	Early Detection of Lung Cancer	HEALTHCARE	2021-10-08	2021-12-18	Q2	YTS

Subquery with DELETE Statement

The subquery can also be used with the DELETE statement.

SQL Query

```
DELETE FROM PROJ_RECORDS_BKUP
WHERE PROJ_ID IN (
  SELECT p.PROJ_ID
  FROM PROJ_RECORDS p
  WHERE p.STATUS = 'YTS'
);
```

Let's say you need to remove a project from the **PROJ_RECORDS_BKUP** table that has the status YTS in the **PROJ_RECORDS** table in MySQL.

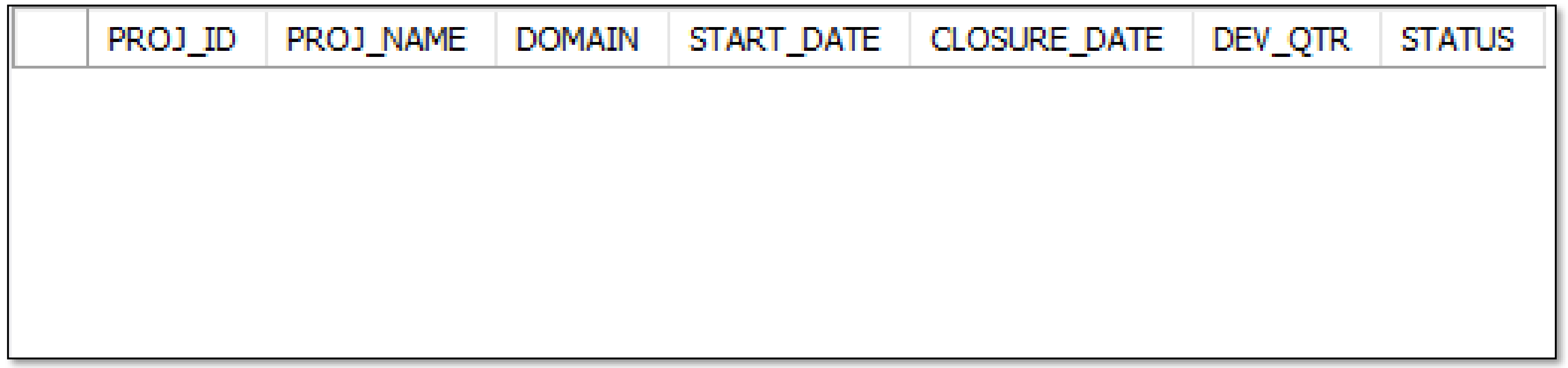
Subquery with DELETE Statement

Now, you need to check the updated data for all the columns in the **PROJ_RECORDS_BKUP** table in MySQL to verify the update.

SQL Query

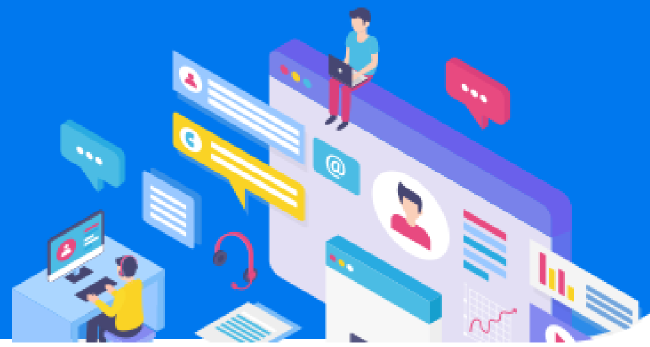
```
SELECT * FROM PROJ_DB.EMP_RECORDS_BKUP;
```

Output:



	PROJ_ID	PROJ_NAME	DOMAIN	START_DATE	CLOSURE_DATE	DEV_QTR	STATUS

Assisted Practice: Subquery

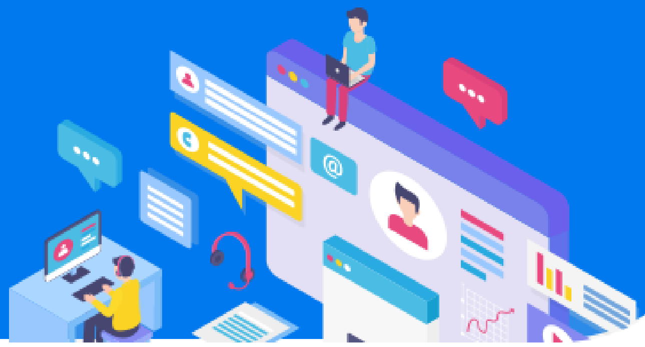


Duration: 15 mins

Problem statement: You're an airport operations specialist, and you have access to the data tables with the flight details of the different airport hubs of your company. You've been asked to analyze only those flights where the origin or destination point is in **SFO airport**.

Objective: Write a query to retrieve all passenger information for the flight numbers operating to or from **SFO airport**

Assisted Practice: Subquery



Duration: 15 mins

Datasets: You've been asked to create the following tables to perform the required tasks:

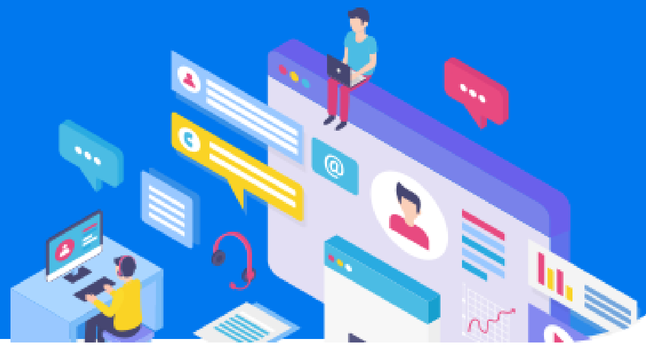
PNR	PAX_NAME	FLT_NO	FLT_DATE	INTL_or_DOM
A123BC	Virat	LF121	2022-01-05	DOM
P565DF	Shane	LF333	2022-02-12	DOM
GF22RF	Brett	LF081	2022-02-24	INTL
P561WF	Rohit	LF081	2022-02-24	INTL
GN22RF	Smith	LF643	2022-03-04	DOM

Passenger table

FLT_NO	FLT_ORG	FLT_DEST
LF121	SFO	ORD
LF333	ORD	EWR
LF081	DEL	SFO
LF643	IAD	ORD
LF999	BOS	EWR

Flight table

Assisted Practice: Subquery



Duration: 15 mins

Steps to be performed:

Step 01: Create the **Passenger** table per the given structure

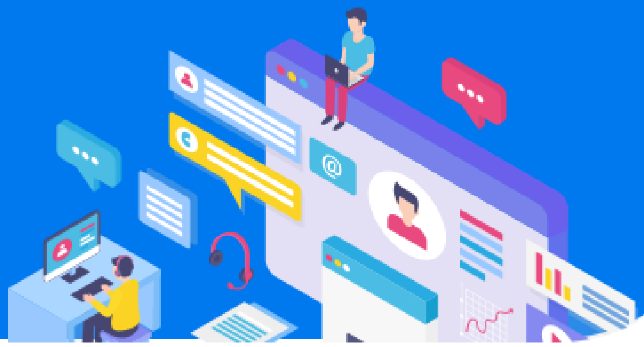
Step 02: Create the **Flight** table per the given structure

Step 03: Insert records in the **Passenger** table

Step 04: Insert records in the **Flight** table

Step 05: Retrieve the data of the passengers who have traveled on planes departing from or arriving to **SFO airport**

Assisted Practice: Subquery

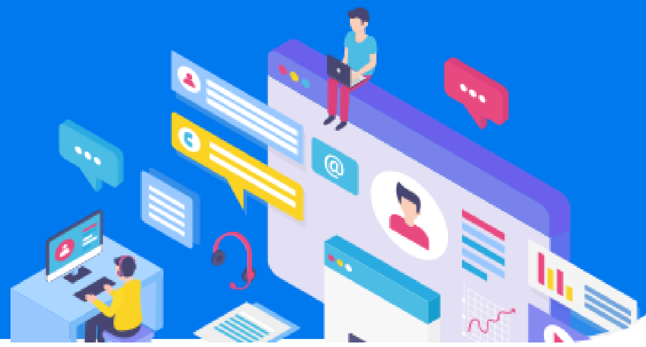


Step 01: Create the **Passenger** table per the given structure

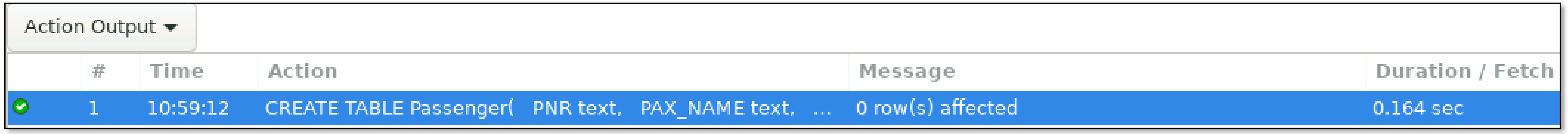
Query

```
CREATE TABLE Passenger(
  PNR text,
  PAX_NAME text,
  FLT_NO text,
  FLT_DATE date,
  INTL_or_DOM text
);
```

Assisted Practice: Subquery

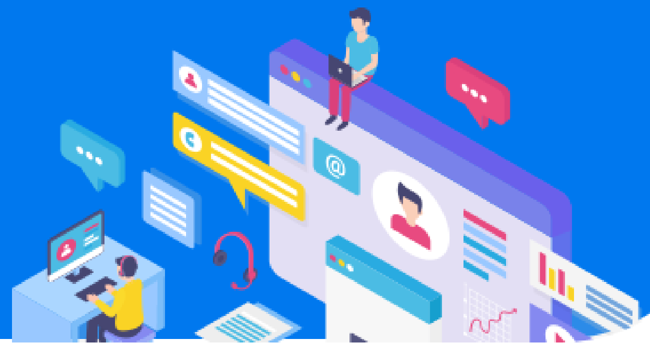


Output:



```
Action Output ▾
    #   Time       Action                                              Message              Duration / Fetch
✓   1   10:59:12   CREATE TABLE Passenger(   PNR text,   PAX_NAME text,   ...   0 row(s) affected   0.164 sec
```

Assisted Practice: Subquery



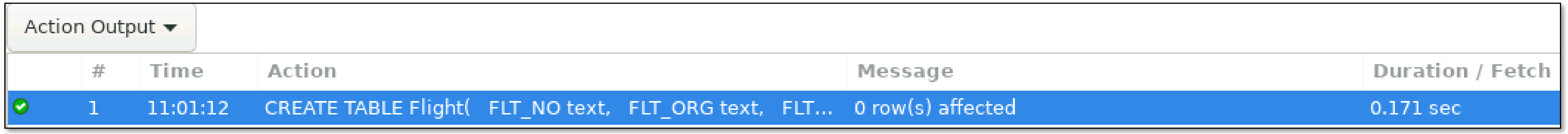
Step 02: Create the **Flight** table per the given structure

Query

```
CREATE TABLE Flight(
  FLT_NO text,
  FLT_ORG text,
  FLT_DEST text
);
```

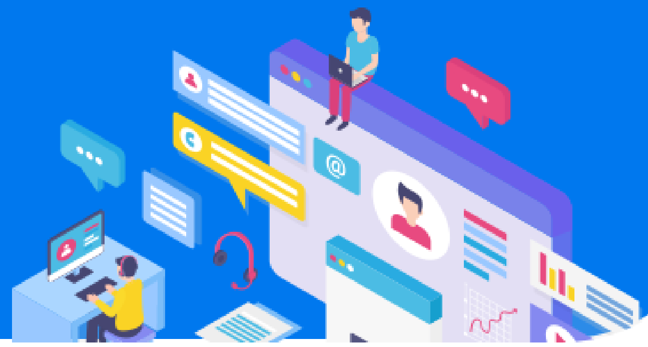
Assisted Practice: Subquery

Output:



```
    #   Time       Action                                              Message               Duration / Fetch
✓   1   11:01:12   CREATE TABLE Flight(   FLT_NO text,   FLT_ORG text,   FLT...   0 row(s) affected     0.171 sec
```

Assisted Practice: Subquery



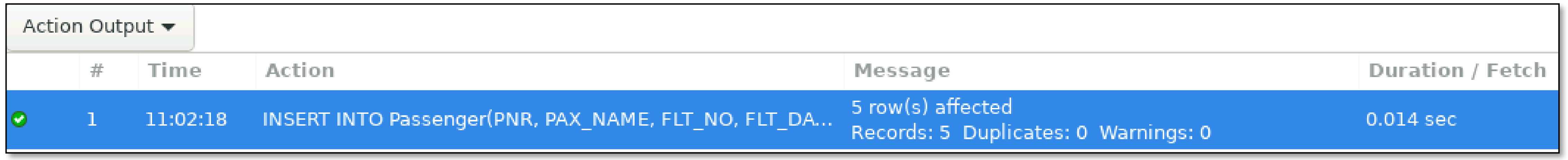
Step 03: Insert records in the **Passenger** table

Query

```
INSERT INTO Passenger(PNR, PAX_NAME, FLT_NO, FLT_DATE,
INTL_or_DOM)
VALUES
("A123BC", "Virat", "LF121", "2022-01-05", "DOM"),
("P565DF" ,"Shane ", "LF333 ", "2022-02-12" ,"DOM"),
("GF22RF",  "Brett", "LF081", "2022-02-24", "INTL"),
("P561WF","Rohit", "LF081", "2022-02-24", "INTL"),
("GN22RF", "Smith", "LF643", "2022-03-04","DOM");
```

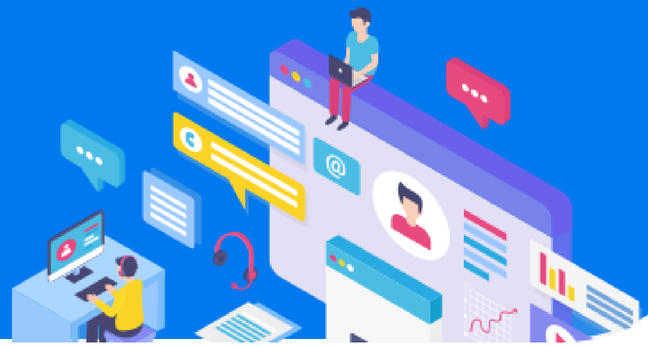
Assisted Practice: Subquery

Output:



```
#  Time      Action                                              Message                                        Duration / Fetch
1  11:02:18  INSERT INTO Passenger(PNR, PAX_NAME, FLT_NO, FLT_DA...  5 row(s) affected                          0.014 sec
                                                                 Records: 5  Duplicates: 0  Warnings: 0
```

Assisted Practice: Subquery

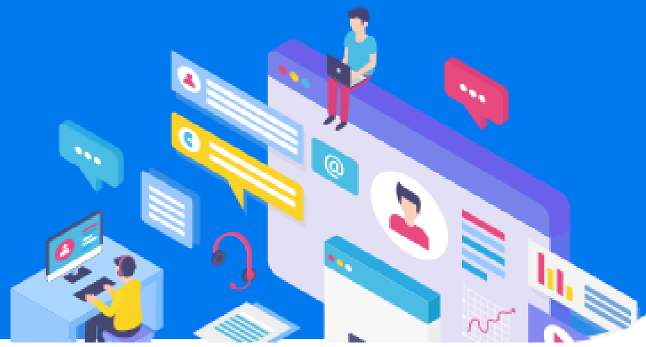


Step 04: Insert records in the **Flight** table

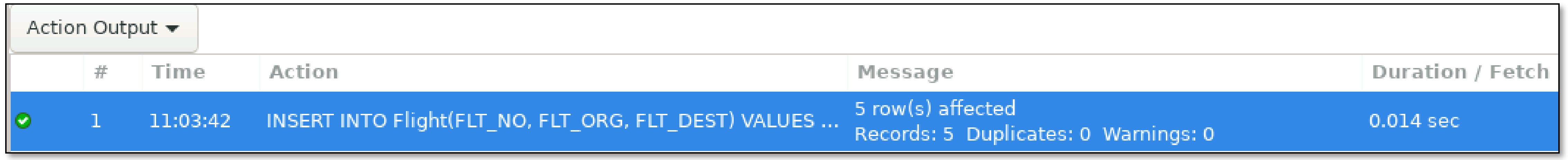
Query

```
INSERT INTO Flight(FLT_NO, FLT_ORG, FLT_DEST)
VALUES
("LF121","SFO","ORD"),
("LF333","ORD","EWR"),
("LF081","DEL","SFO"),
("LG643","IAD","ORD"),
("LF999","BOS","EWR");
```

Assisted Practice: Subquery

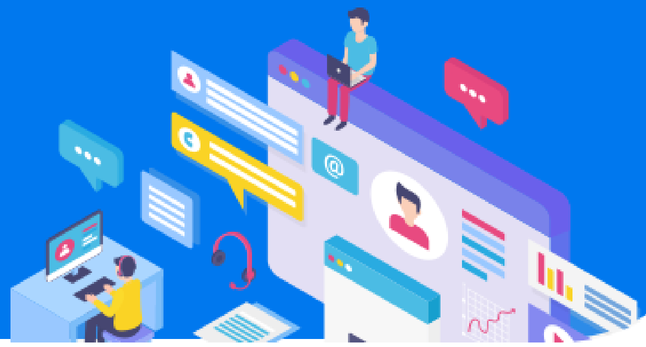


Output:



```
Action Output ▾
    #   Time       Action                                                     Message                                          Duration / Fetch
✓   1   11:03:42   INSERT INTO Flight(FLT_NO, FLT_ORG, FLT_DEST) VALUES ...   5 row(s) affected                                0.014 sec
                                                                              Records: 5  Duplicates: 0  Warnings: 0
```

Assisted Practice: Subquery



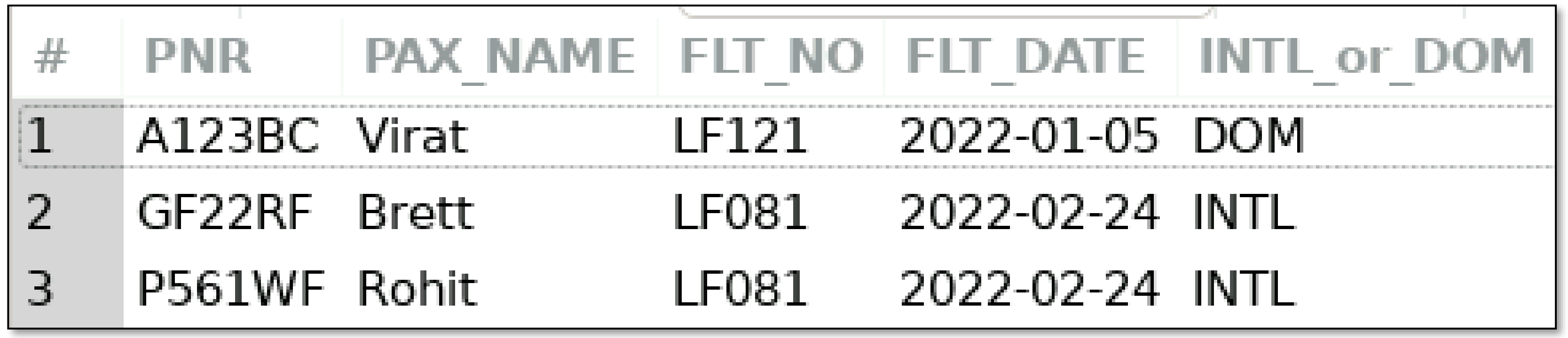
Step 05: Retrieve the data of the passengers who have traveled on planes departing from or arriving to **SFO airport**

Query

```
SELECT *
FROM Passenger
WHERE FLT_NO IN (
    SELECT FLT_NO
    FROM Flight
    WHERE FLT_ORG="SFO" OR FLT_DEST="SFO"
);
```

Assisted Practice: Subquery

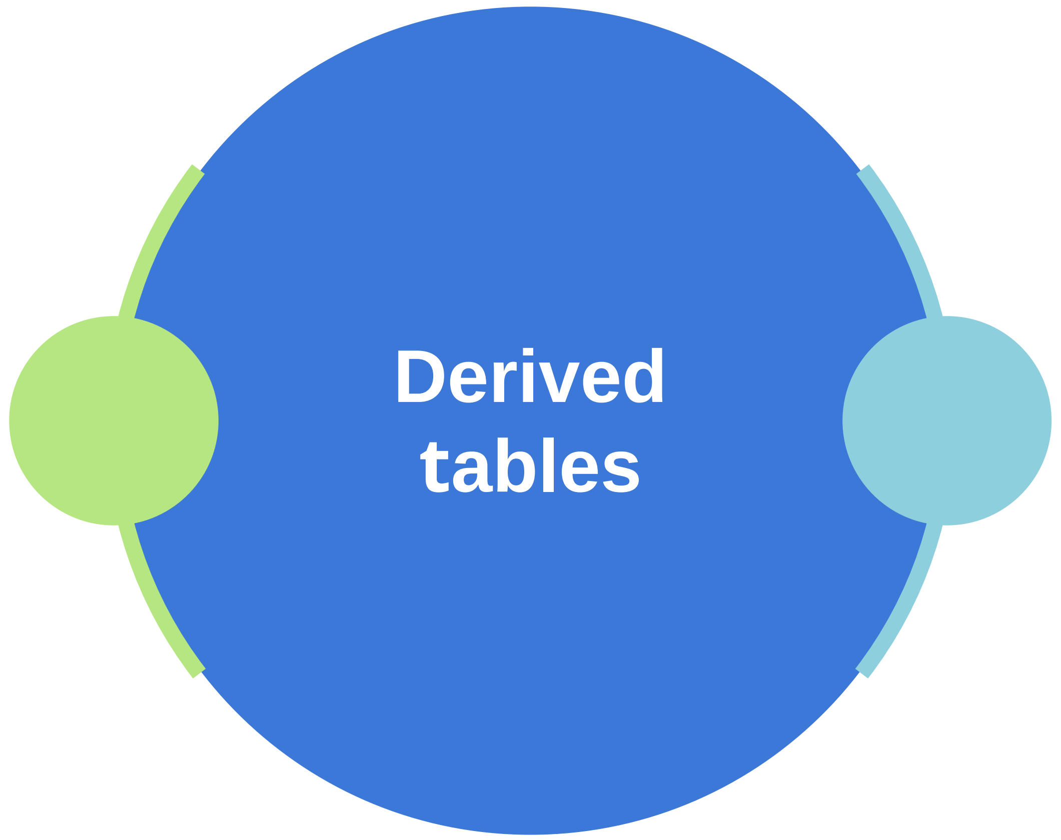
Output:



#	PNR	PAX_NAME	FLT_NO	FLT_DATE	INTL_or_DOM
1	A123BC	Virat	LF121	2022-01-05	DOM
2	GF22RF	Brett	LF081	2022-02-24	INTL
3	P561WF	Rohit	LF081	2022-02-24	INTL

Derived Tables in SQL

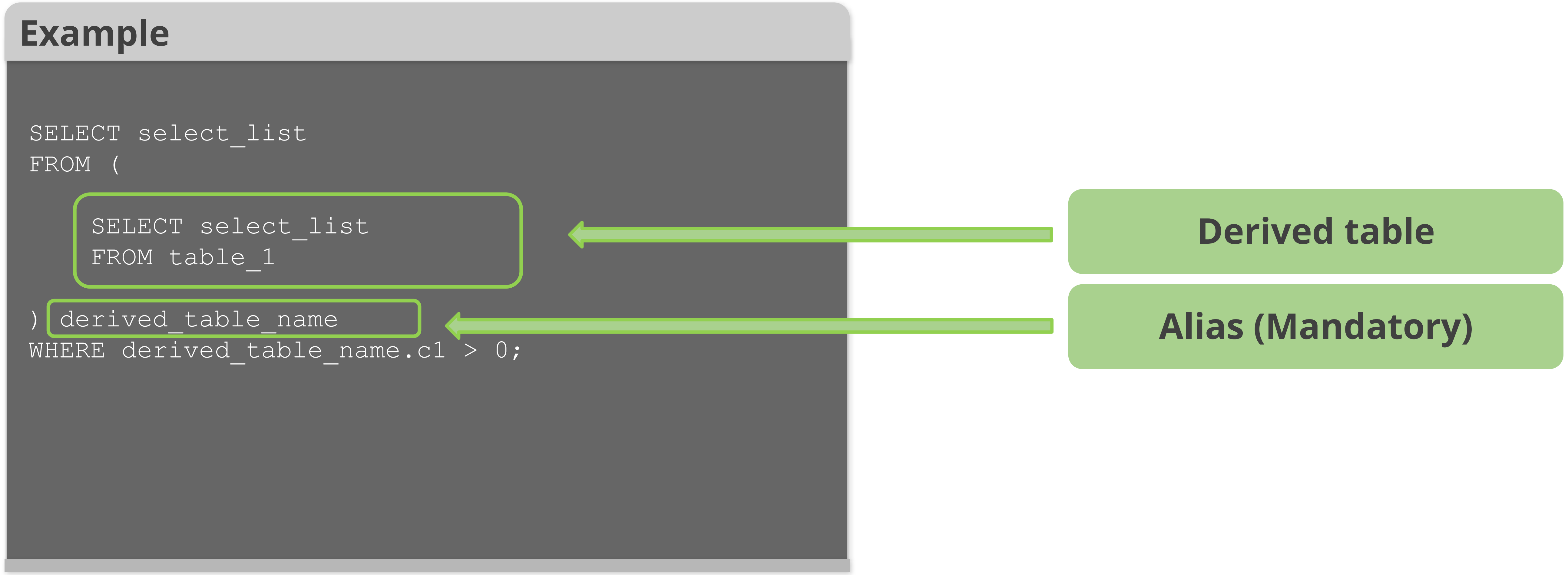
Derived Tables



A derived table is a virtual table returned by a SELECT statement.

A derived table is created by using a stand-alone subquery in the FROM clause of a SELECT statement.

Derived Tables



Example

```
SELECT select_list
FROM (

    SELECT select_list
    FROM table_1

) derived_table_name
WHERE derived_table_name.c1 > 0;
```

Derived Tables

MySQL raises an error in the absence of an alias for a derived table.

Error

```
Every derived table must have its own alias
```

It is mandatory for a derived table to have an alias so that it can be referenced in the query.

Derived Tables: Example

Problem Statement: Your manager wants you to find the total number of managers in the organization.

Objective: Write an SQL query using the **COUNT** function with the **DISTINCT** keyword on the output returned by a **subquery** which creates a **DERIVED TABLE** to return the **EMP_ID** of all managers from the **EMP_RECORDS** tables in MySQL.

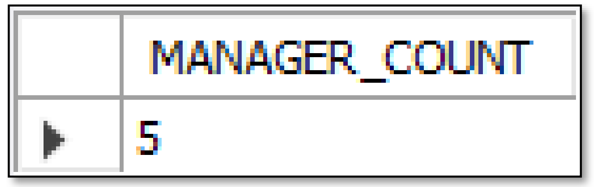
Derived Tables: Example

Step 1: Use the **COUNT** function with the **DISTINCT** keyword on the output returned by a subquery which creates a **DERIVED TABLE** for returning the **EMP_ID** of all managers as shown below:

SQL Query – Par 1

```
SELECT COUNT(DISTINCT EMP_ID) AS `MANAGER_COUNT`
FROM (
  SELECT DISTINCT EMP_ID FROM EMP_RECORDS
  WHERE ROLE IN ("MANAGER")
) employees
WHERE EMP_ID = employees.EMP_ID;
```

Output:



	MANAGER_COUNT
▶	5

EXISTS Operator

EXISTS Operator

The EXISTS operator is a Boolean operator that returns true or false and is frequently used to check if rows returned by a subquery exist.

Syntax

```
SELECT
    select_list
FROM
    a_table
WHERE
    [NOT] EXISTS(subquery);
```

EXISTS Operator: Example

Problem Statement: Your manager wants you to provide the basic information of all the managers in the organization.

Objective: Write an SQL query using the **EXISTS** operator to verify the existence of managers and return their details if available from the **EMP_RECORDS** tables in MySQL.

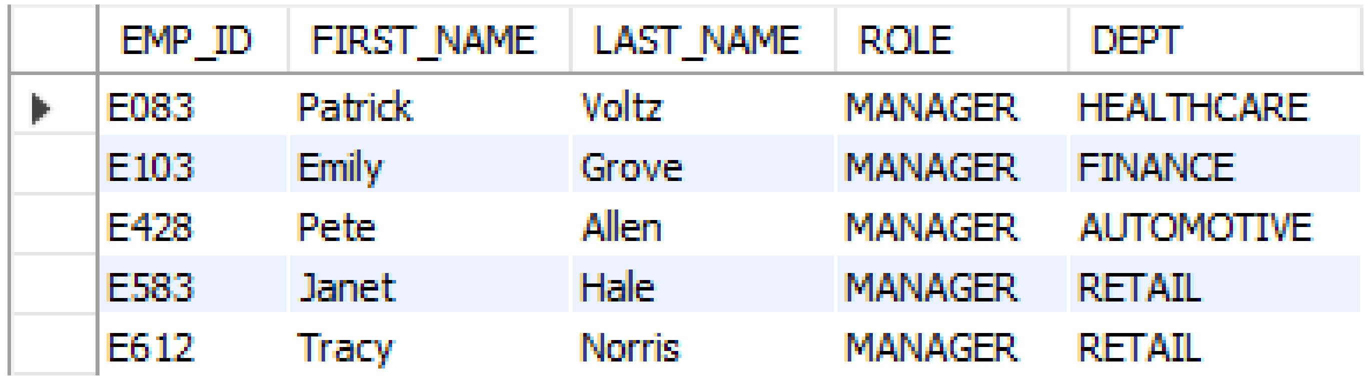
EXISTS Operator: Example

Step 1: Use the **EXISTS** operator in the **SELECT** statement to verify the existence of managers and return their details if available from the **EMP_RECORDS** tables as shown below:

SQL Query

```
SELECT m.EMP_ID, m.FIRST_NAME, m.LAST_NAME,
       m.ROLE, m.DEPT
FROM EMP_RECORDS m
WHERE EXISTS(
  SELECT 1 FROM EMP_RECORDS WHERE ROLE IN
("MANAGER")
) AND ROLE IN ("MANAGER");
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	ROLE	DEPT
E083	Patrick	Voltz	MANAGER	HEALTHCARE
E103	Emily	Grove	MANAGER	FINANCE
E428	Pete	Allen	MANAGER	AUTOMOTIVE
E583	Janet	Hale	MANAGER	RETAIL
E612	Tracy	Norris	MANAGER	RETAIL

Note: SELECT 1 is frequently used in situations where you wish to verify the existence of record.

Or simply retrieve a constant value without specifying a particular column or table.

NOT EXISTS Operator

Alternatively, the NOT EXISTS operator is the opposite of EXISTS operator.

Syntax

```
SELECT
    select_list
FROM
    a_table
WHERE
    NOT EXISTS(subquery);
```

NOT EXISTS Operator: Example

Problem Statement: Your manager wants you to provide the basic information of all the employees with one year or less than one year of experience in the organization.

Objective: Write an SQL query that verifies if there is no entry for a negative experience in the **EMP_RECORDS** tables using the **NOT EXISTS** operator, and then returns the basic information for all employees in that table with an experience of less than or equal to one in MySQL.

NOT EXISTS Operator: Example

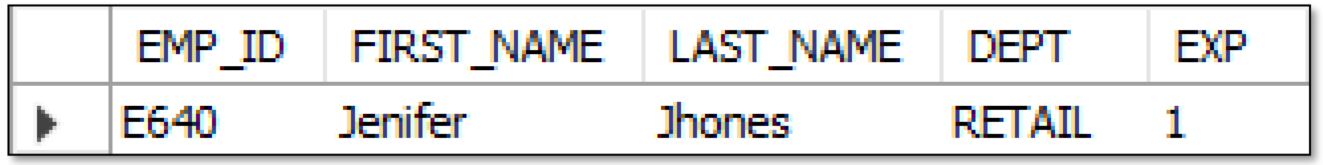
Step 1: Use the **NOT EXISTS** operator in the **SELECT** statement to verify if there is no negative entry experience.

Step 2: Return the basic information of all the employees with an experience of less than or equals to one in the **EMP_RECORDS** table as shown below:

SQL Query

```
SELECT m.EMP_ID, m.FIRST_NAME, m.LAST_NAME,
       m.DEPT, m.EXP
FROM EMP_RECORDS m
WHERE NOT EXISTS(
  SELECT 1 FROM EMP_RECORDS WHERE EXP < 0
) AND m.EXP <= 1;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	DEPT	EXP
▶	E640	Jenifer	Jhones	RETAIL	1

EXISTS vs. IN Operators

EXISTS vs. IN

EXISTS operator

- Faster than IN operator
- Does not depend on subquery
- Works on the at least found principle
- Stops scanning the table as soon as a matching row is found

IN operator

- Slower than EXISTS operator
- Depends on subquery
- Used in conjunction with a subquery
- Waits for MySQL to complete the execution of the subquery to utilize its result

Knowledge Check

Knowledge Check

1

The JOIN where all possible row combinations are produced is called ________.

A. INNER JOIN

B. OUTER

C. NATURAL

D. CROSS

Knowledge Check

1

The JOIN where all possible row combinations are produced is called ________.

A. INNER JOIN

B. OUTER

C. NATURAL

D. CROSS

The correct answer is **D**

In the 'cross product', each row of each table is combined with each row of every other table to generate every possible combination. Since the number is the product of rows, this results in an extremely high number of rows.

Knowledge Check

2

What is joining a table to itself called?

A. SELF JOIN

B. COMPLETE

C. OBSOLETE

D. CROSS

Knowledge Check

2

What is joining a table to itself called?

A. SELF JOIN

B. COMPLETE

C. OBSOLETE

D. CROSS

The correct answer is **A**

The term "self join" refers to the joining of a tables to itself in a database. A table name qualifier is not required when doing a self-join because the table is utilized several times within the query.

Knowledge Check

3

In which JOIN do all the rows from the left table appear in the output irrespective of the content of the other table?

A. RIGHT JOIN

B. LEFT JOIN

C. INNER JOIN

D. OUTER JOIN

Knowledge Check

3

In which JOIN do all the rows from the left table appear in the output irrespective of the content of the other table?

A. RIGHT JOIN

B. LEFT JOIN

C. INNER JOIN

D. OUTER JOIN

The correct answer is **B**

A 'LEFT JOIN' produces output for every row in the left table, even if that row does not exist in the right table. This is why it's referred to as a 'LEFT JOIN.' The 'LEFT JOIN' is a type of OUTER JOIN.

Knowledge Check

4

Which clause is used to sort a UNION result as a whole?

A. LIMIT

B. GROUP BY

C. ORDER BY

D. SORT

Knowledge Check

4

Which clause is used to sort a UNION result as a whole?

A. LIMIT

B. GROUP BY

C. ORDER BY

D. SORT

The correct answer is **C**

To sort a 'UNION' result as a whole, the 'ORDER BY' clause is used with the 'UNION' statement. It is placed after the final SELECT statement which is enclosed in the parentheses.

Lesson-End Project: Employee Data Analysis

Problem statement:

You are a part of the HR department in a company, and you have been asked to extract, update, and delete the employees’ details to maintain the records for further analysis.

Objective:

The objective is to design a database to analyze the performance of the employees on a quarterly basis.

Note: Download the **employee_datasets.csv** file from **Course Resources** to perform the required tasks

Lesson-End Project: Employee Data Analysis

Tasks to be performed:

1. Write a query to create an **employee** table with employee ID, first name, last name, job ID, salary, manager ID, and department ID fields
2. Write a query to insert values into the **employee** table
3. Write a query to find the first and last names of every employee whose salary is higher than the employee with the last name Kumar
4. Write a query to display the employee ID and last name of every employee whose salary is greater than the average

Lesson-End Project: Employee Data Analysis

Tasks to be performed:

5. Write a query to display the employee ID and first name of every employee whose salary is higher than the salary of the shipping clerks (JOB_ID = HP122) and sort the results in the ascending order of the salary
6. Write a query to display the first name, employee ID, and salary of the three employees with the highest salaries

Note: Download the solution document from the **Course Resources** section and follow the steps given in the document

Key Takeaways

- MySQL supports four different types of joins: INNER, LEFT, RIGHT, and CROSS.
- Because the INTERSECT and MINUS set operators are not supported by MySQL, they must be mimicked by combining other MySQL components.
- A derived table is formed by inserting a stand-alone subquery into the FROM clause of a SELECT statement.
- The EXISTS operator is commonly used to determine whether rows returned by a subquery exist by returning a Boolean value.
- The EXISTS operator operates on the least found principle, whereas the IN operator is employed with a subquery.