DATA AND
ARTIFICIAL INTELLIGENCE

Working with Database and Tables

Learning Objectives

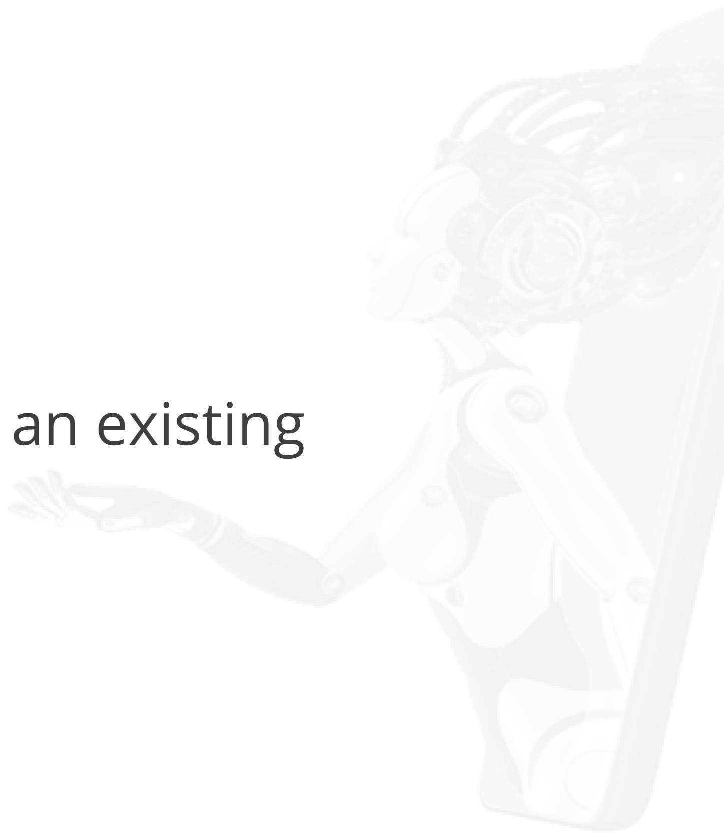
By the end of this lesson, you will be able to:

- List and apply various techniques to manipulate databases and tables in MySQL
- Analyze the importance and types of storage engines in MySQL
- Outline different clauses and operators used in MySQL
- Perform sorting and grouping operations on table data retrieved from MySQL database

Introduction to Database in MySQL

Database Manipulation in MySQL

- A database in MySQL is implemented as a directory that contains all files that correspond to the database's tables.
- Creating databases with MySQL requires admin privileges.
- The terms schema and database are synonyms in MySQL, so they can be used interchangeably.
- The MySQL copy or clone database feature allows us to make a backup copy of an existing database, including table structure, indexes, constraints, and default values.



Database Manipulation in MySQL

Database Manipulation Commands in MySQL

The following are the most important SQL commands for database manipulation:

1. CREATE DATABASE
2. SHOW DATABASES
3. USE
4. DROP DATABASE (or DROP SCHEMA)

Creating Databases in MySQL

- To build a new MySQL database, use the **CREATE DATABASE** statement.
- Syntax:

Syntax

```
CREATE DATABASE [IF NOT EXISTS]database_name
[CHARACTER SET charset_name]
[COLLATE collation_name];
```

Creating Databases in MySQL

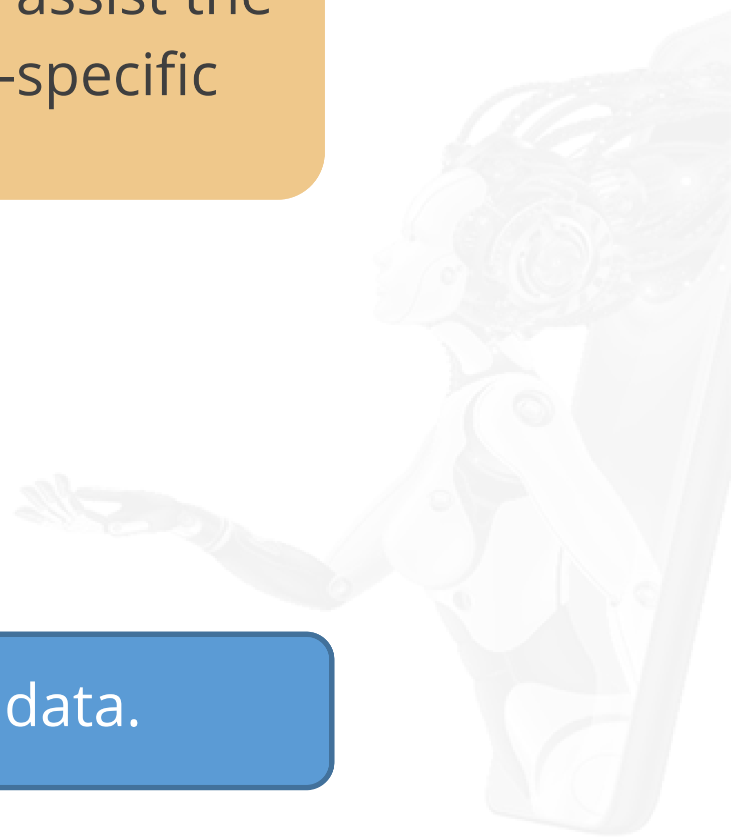
- Parameters:

Keywords	Meaning
database_name	A unique name for a new database
IF NOT EXISTS **(Optional)**	It prevents an error from occurring while creating a database that already exists
charset_name **(Optional)**	It is the name of the character set to store every character in a string
collation_name **(Optional)**	It compares characters in a particular character set

Creating Databases in MySQL: Example

Problem Statement: You work as a junior analyst at your organization. You must assist the HR department to create an employee database for managers to hold manager-specific information about employees.

Objective: Build the appropriate database for storing the managers-specific data.



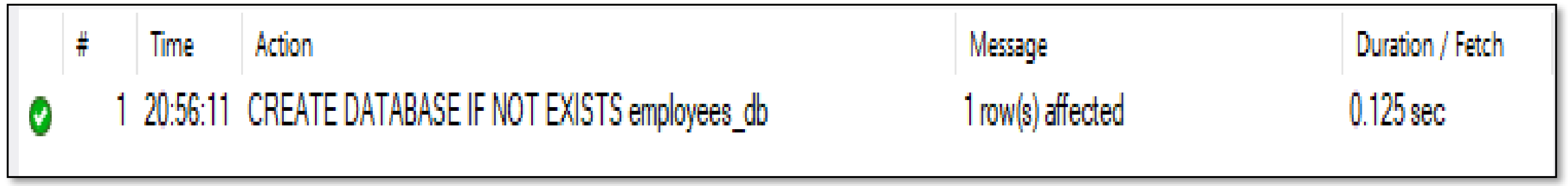
Creating Databases in MySQL: Example

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

SQL Query

```
CREATE DATABASE IF NOT EXISTS employees_db;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	20:56:11	CREATE DATABASE IF NOT EXISTS employees_db	1 row(s) affected	0.125 sec

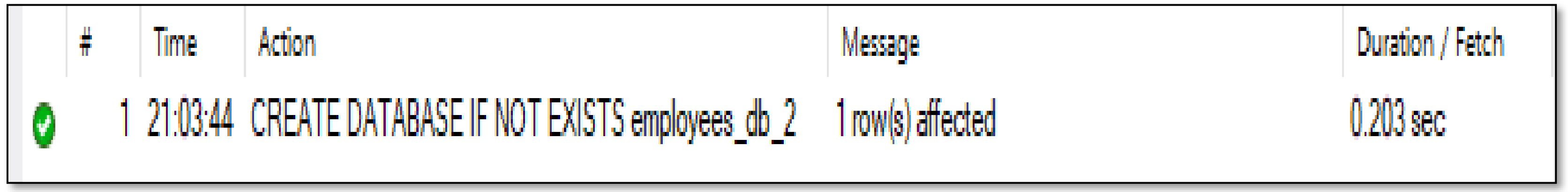
Creating Databases in MySQL: Example

Step 2: Create another database named **employees_db_2** with the **CREATE DATABASE** statement.

SQL Query

```
CREATE DATABASE IF NOT EXISTS employees_db_2;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	21:03:44	CREATE DATABASE IF NOT EXISTS employees_db_2	1 row(s) affected	0.203 sec

Listing Available Databases in MySQL

- To lists all databases in the MySQL database, use the **SHOW DATABASES** statement.
- Syntax:

Syntax

```
SHOW DATABASES;
```

Listing Available Databases in MySQL: Example

Problem Statement: Your manager wants you to provide the list of all the databases available in the organization in their MySQL setup.

Objective: Use the **SHOW DATABASES** statement to get the required list in MySQL.

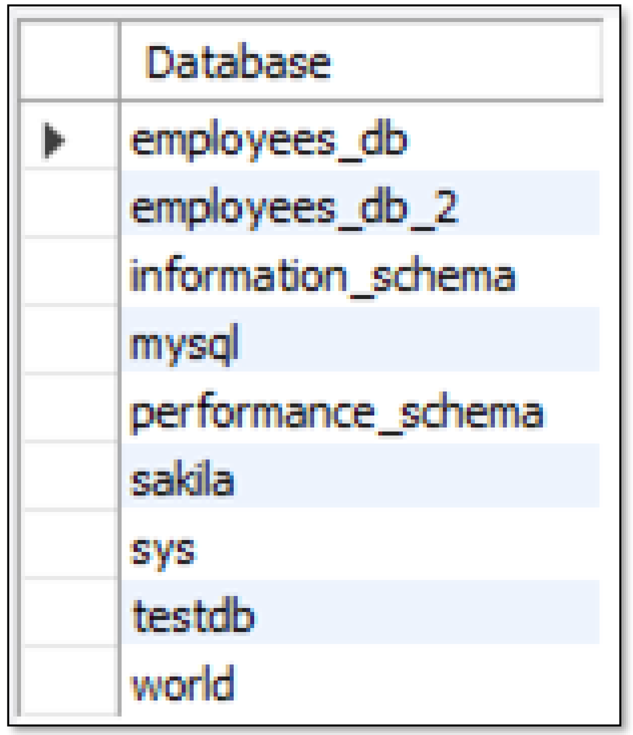
Listing Available Databases in MySQL: Example

Step 1: Use the **SHOW DATABASES** statement to list all the existing databases as given below.

SQL Query

```
SHOW DATABASES;
```

Output:



	Database
▶	employees_db
	employees_db_2
	information_schema
	mysql
	performance_schema
	sakila
	sys
	testdb
	world

Database Selection in MySQL

- It is necessary to notify MySQL about the destination database before interacting with it.
- To select an existing database from the MySQL databases, use the **USE** statement.
- To view the existing selected database from MySQL databases, we use the **SELECT database()** statement.

Database Selection in MySQL

- Syntax for selecting a default database:

Syntax

```
USE database_name;
```

- Syntax for getting the name of the default database:

Syntax

```
SELECT database();
```

Database Selection in MySQL

- Parameters:

Keywords	Meaning
database_name	A unique name for a new database

Database Selection in MySQL: Example

Problem Statement: Your manager wants you to change the default database to one that stores manager-specific data in the MySQL setup.

Objective: Use the **USE** and **SELECT databases()** statements to change the default database and verify the same respectively in MySQL.

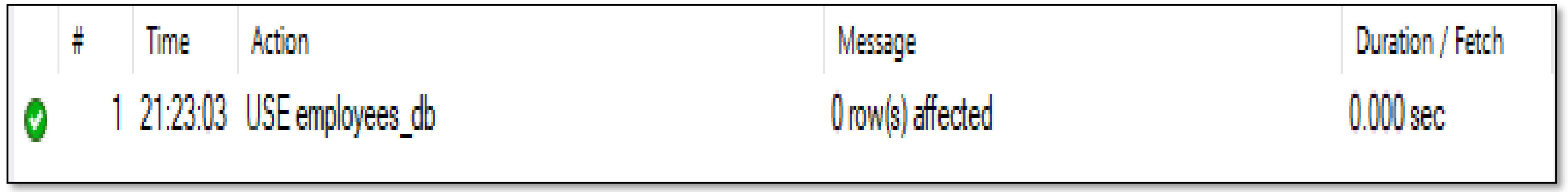
Database Selection in MySQL: Example

Step 1: Use the **USE** statement to change the default database to **employees_db** as given below.

SQL Query

```
USE employees_db;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	21:23:03	USE employees_db	0 row(s) affected	0.000 sec

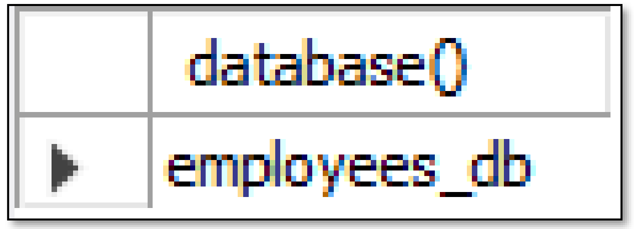
Database Selection in MySQL: Example

Step 2: Use the **SELECT databases()** statement to verify the default database as given below.

SQL Query

```
SELECT database();
```

Output:



	database()
▶	employees_db

Deleting Databases in MySQL

- The **DROP DATABASE** statement is used to delete all tables in an existing database and permanently delete that database.
- Syntax:

Syntax

```
DROP DATABASE [IF EXISTS] database_name;
```

Deleting Databases in MySQL

- Parameters:

Keywords	Meaning
database_name	A unique name of the existing target database to delete
IF EXISTS **(Optional)**	It prevents an error from occurring while removing a database that does not exist

Deleting Databases in MySQL: Example

Problem Statement: Your manager wants you to delete all the unrequired databases from the MySQL setup.

Objective: Use the **DROP DATABASE** and **SHOW DATABASES** statements to drop the **employees_db_2** database and verify the same respectively in MySQL.

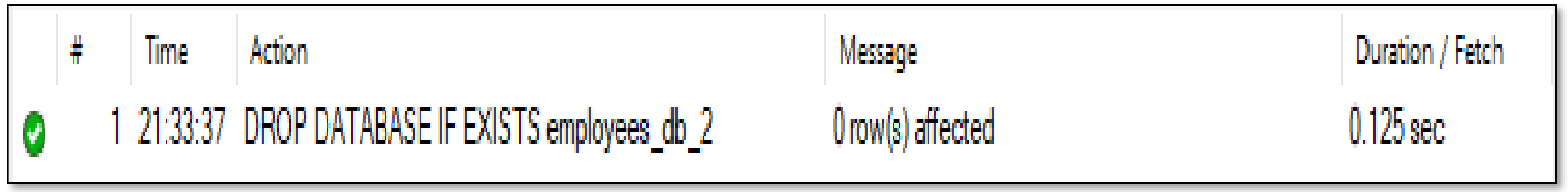
Deleting Databases in MySQL: Example

Step 1: Use the **DROP** statement to drop the **employees_db** database as given below.

SQL Query

```
DROP DATABASE IF EXISTS employees_db_2;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	21:33:37	DROP DATABASE IF EXISTS employees_db_2	0 row(s) affected	0.125 sec

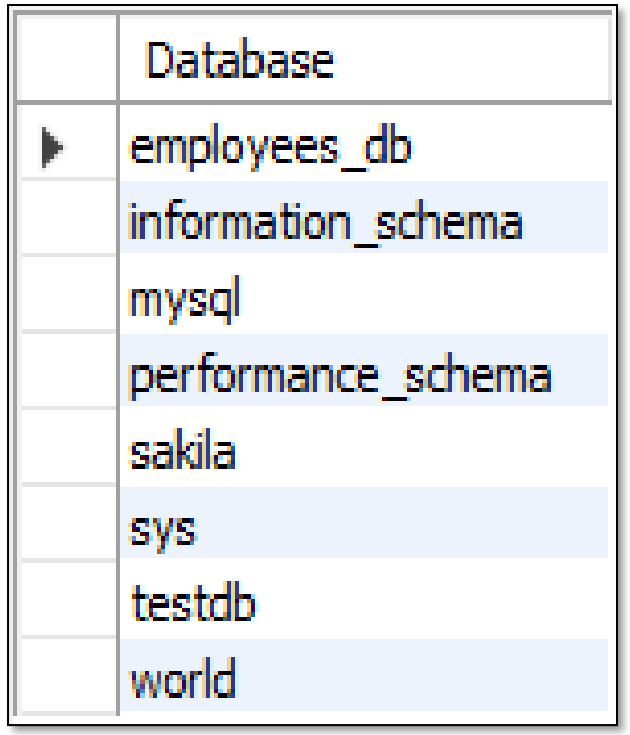
Deleting Databases in MySQL: Example

Step 2: Use the **SHOW DATABASES** statement to verify whether **employees_db** is available among the database in the MySQL setup as given below.

SQL Query

```
SHOW DATABASES;
```

Output:



```
Database
employees_db
information_schema
mysql
performance_schema
sakila
sys
testdb
world
```

Cloning a Database in MySQL

Making a clone of an original database in MySQL includes the following three steps:

- **Step 1:** Create an SQL file backup of the original database
- **Step 2:** Create a new database
- **Step 3:** Copy the data from the SQL file to the new database

Transactions and ACID Properties in MySQL

Transaction

- It is a process that gets executed due to real-world events and changes the state of the information stored in a database.

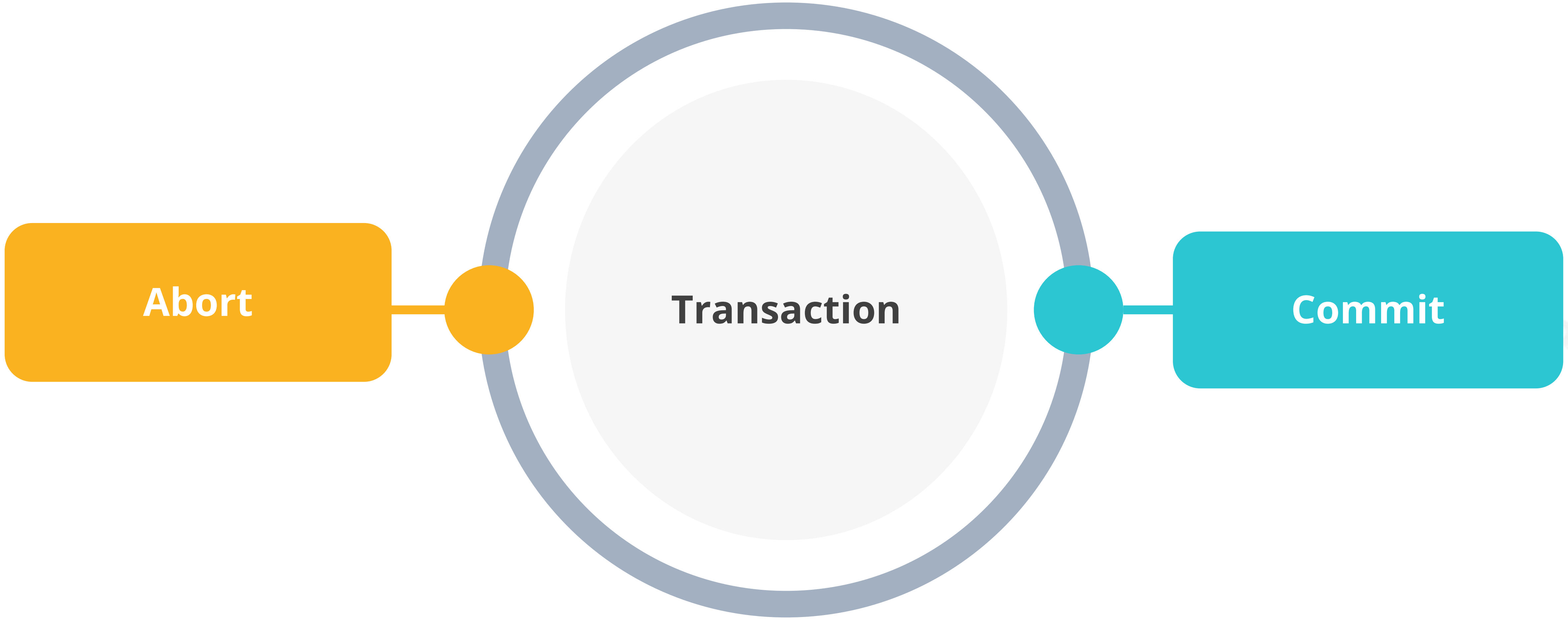
- It is a set of logical operations on data and functions as a single entity.

Transaction: Example

Transferring money between bank accounts is a transaction that includes the debit or credit of some amount from one account to another.

Transaction Components

A transaction is made up of two procedures:



ACID Properties

- These are a set of qualities required in database transactions to ensure that these are handled reliably.

- These enable developers to focus on application logic rather than failures as well as recover the complexity of the environment in which the application is executed.

- These are solely concerned with how a database recovers from any failure (transaction, median, or system failure) that may occur during transaction processing.

ACID Properties

- ACID-compliant databases ensure that only transactions that were completely successful will be processed, and if any failure occurs before a transaction is completed, the data will not be modified.

- In a complete ACID system, the database engine guarantees that the *synchronous write* options are used. Hence, it is preferred to physical media.

ACID Properties

ACID properties include:

A

Atomicity:
Transactions are all or nothing.

C

Consistency:
A transaction is subject to a set of rules.

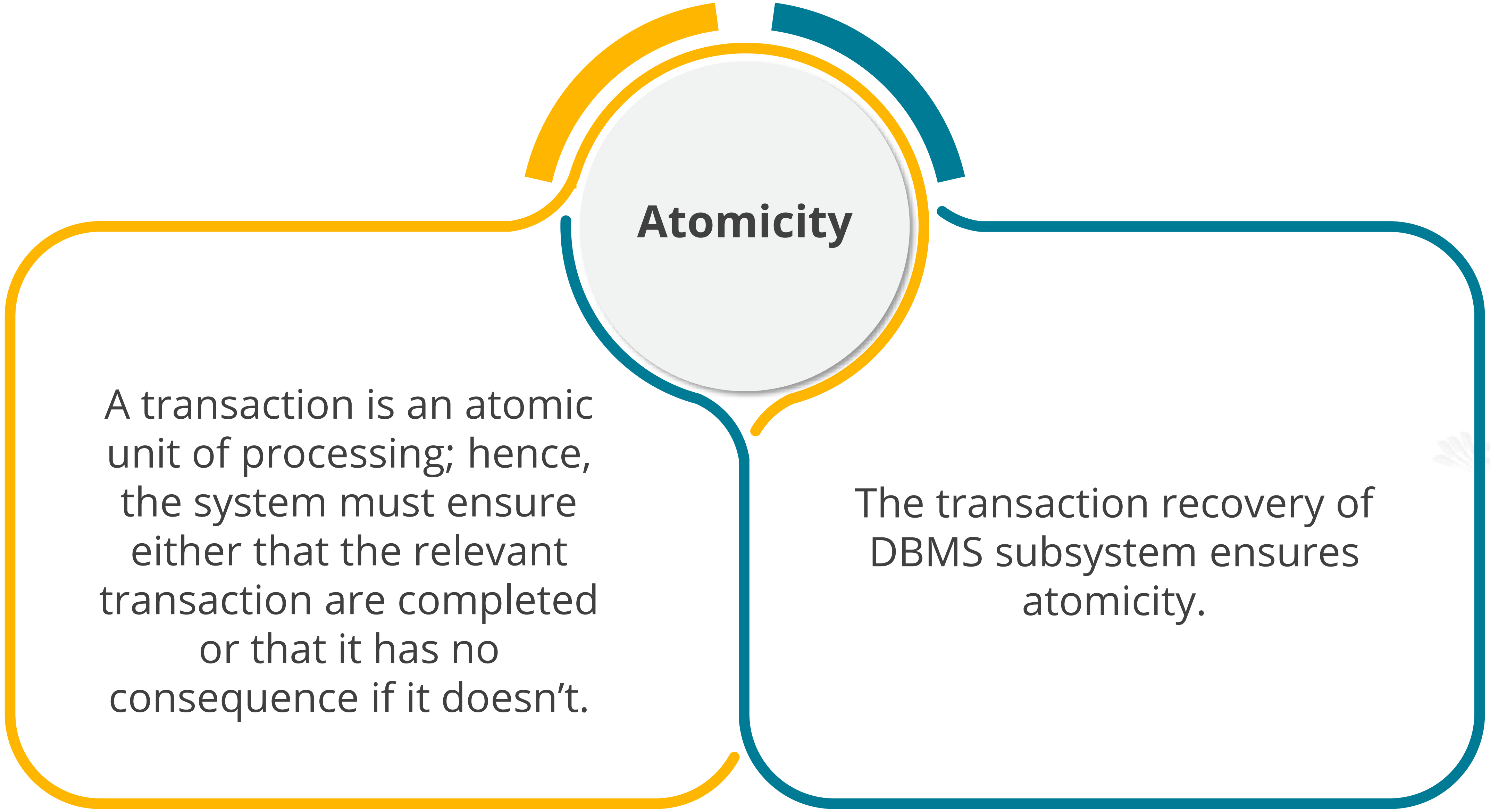
I

Isolation:
Transactions do not affect each other.

D

Durability:
Written data will not get lost.

ACID: Atomicity

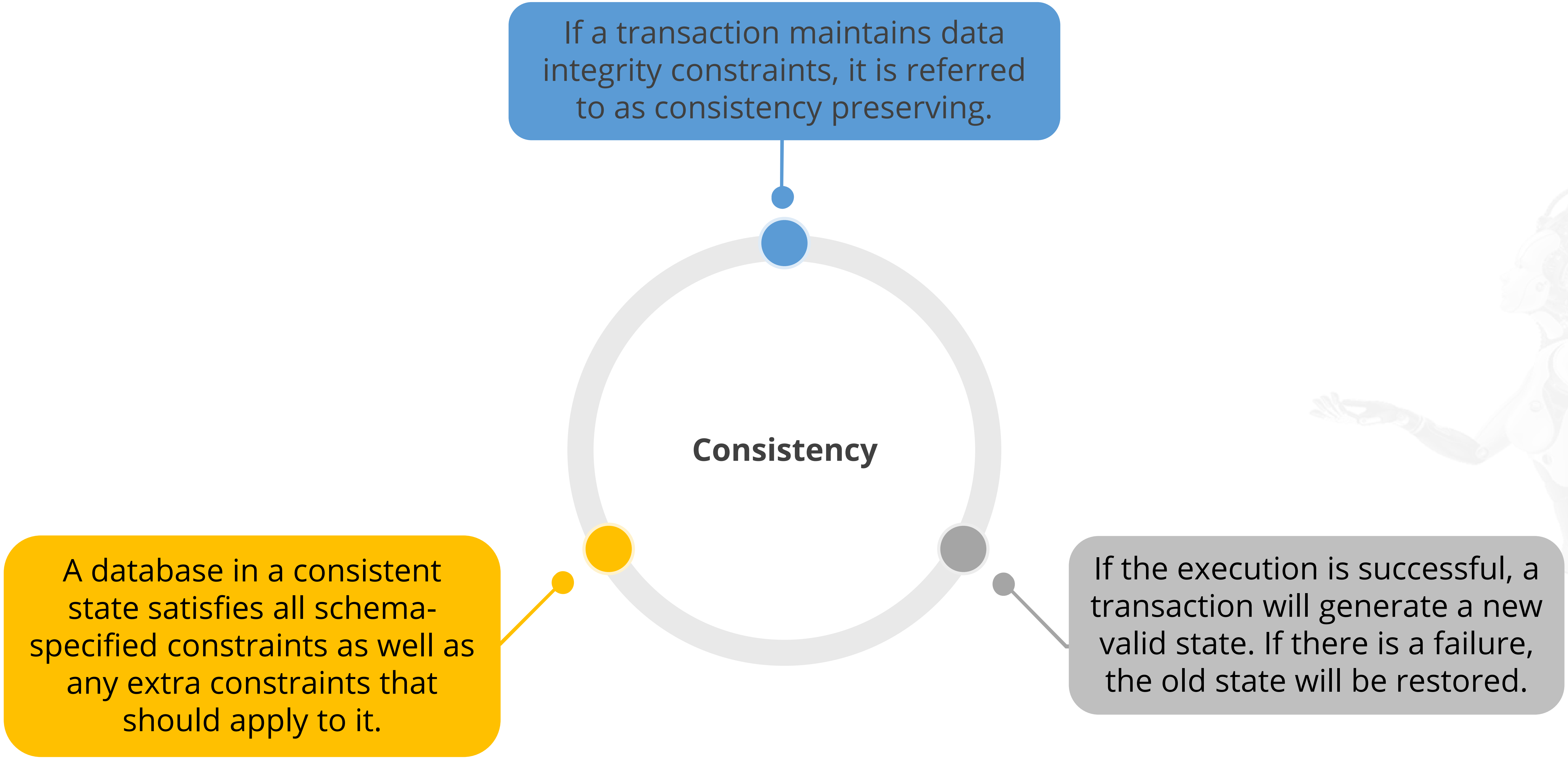


Atomicity

A transaction is an atomic unit of processing; hence, the system must ensure either that the relevant transaction are completed or that it has no consequence if it doesn’t.

The transaction recovery of DBMS subsystem ensures atomicity.

ACID: Consistency



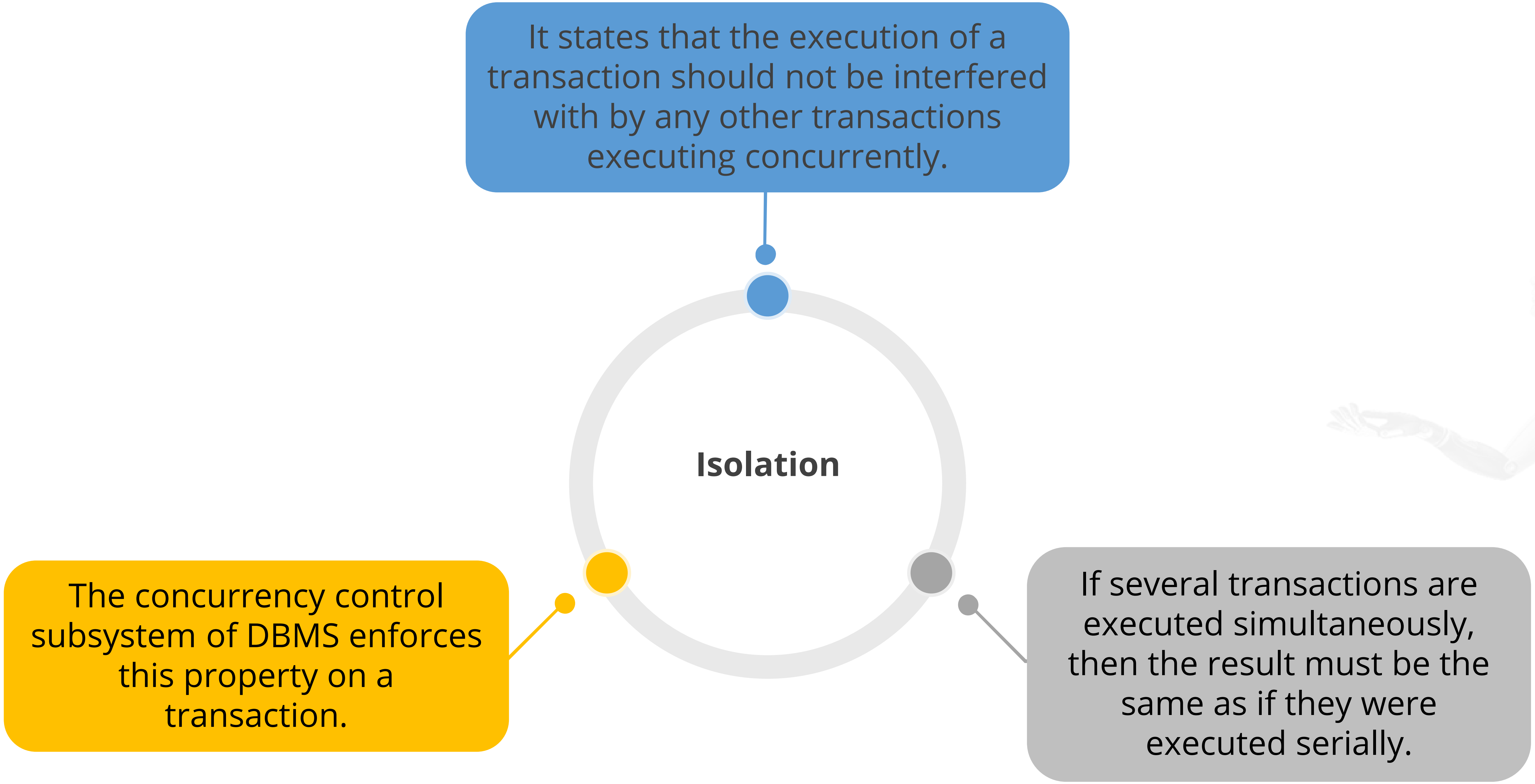
Consistency

If a transaction maintains data integrity constraints, it is referred to as consistency preserving.

A database in a consistent state satisfies all schema-specified constraints as well as any extra constraints that should apply to it.

If the execution is successful, a transaction will generate a new valid state. If there is a failure, the old state will be restored.

ACID: Isolation



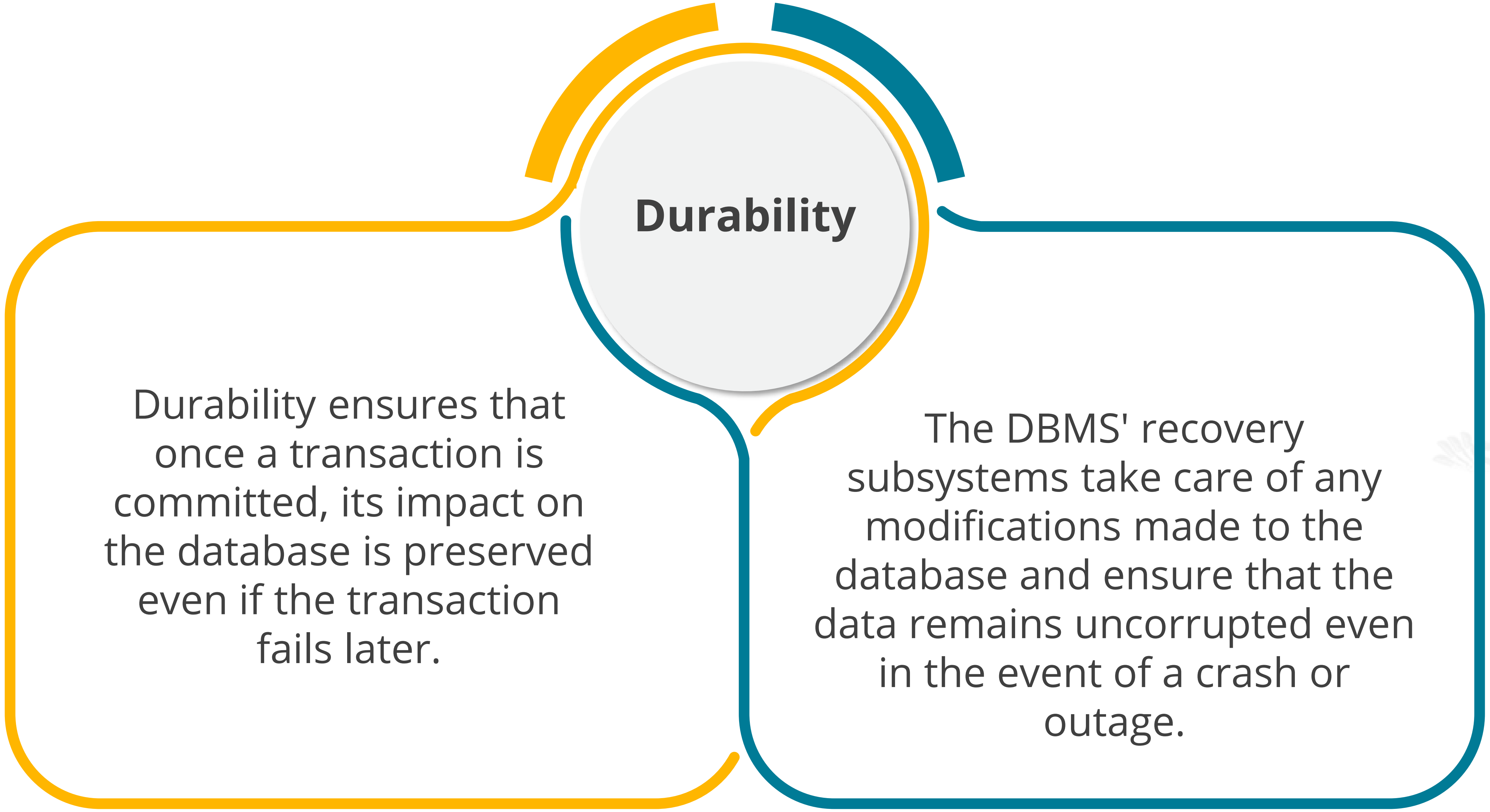
Isolation

It states that the execution of a transaction should not be interfered with by any other transactions executing concurrently.

If several transactions are executed simultaneously, then the result must be the same as if they were executed serially.

The concurrency control subsystem of DBMS enforces this property on a transaction.

ACID: Durability



Durability

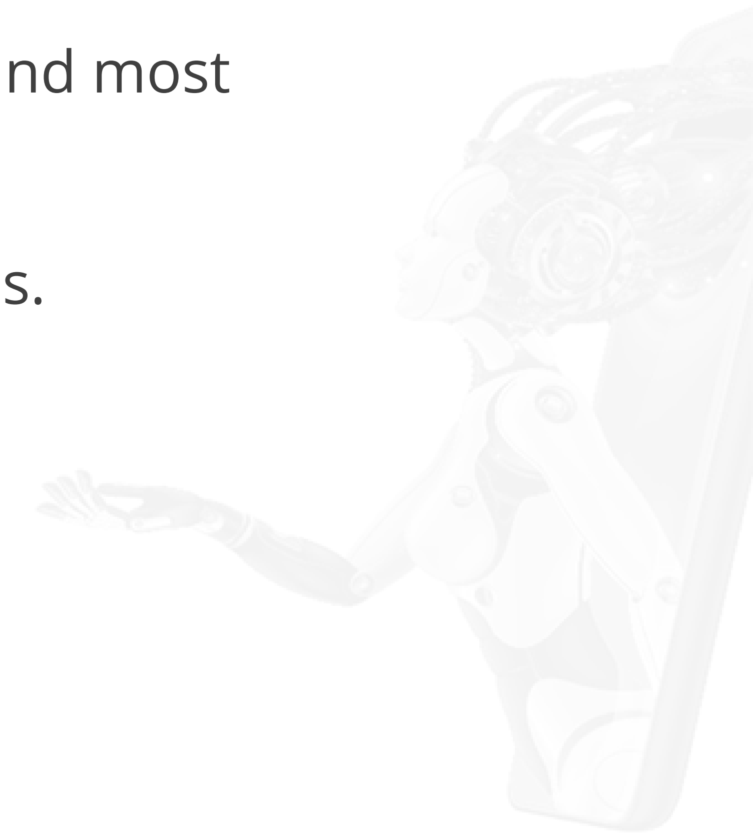
Durability ensures that once a transaction is committed, its impact on the database is preserved even if the transaction fails later.

The DBMS' recovery subsystems take care of any modifications made to the database and ensure that the data remains uncorrupted even in the event of a crash or outage.

Introduction to MySQL Storage Engines

MySQL Storage Engines

- Storage engines are MySQL components which can perform SQL operations on various table types in order to store and manage data in a database.
- Except for specific use situations, Oracle recommends InnoDB as the default and most general-purpose storage engine for tables.
- By default, the CREATE TABLE statement in MySQL 8.0 generates InnoDB tables.



MySQL Storage Engines

- MySQL supports several storage engines for its tables, each of which is utilized for a distinct purpose.

- Each storage engine has advantages and disadvantages of its own.

- To maximize database performance, it's critical to understand each storage engine's features and select one which is most appropriate for the target tables.

Storage Engine Types

Storage Engines Types in MySQL

- Following are some of the MySQL storage engines:

  1. InnoDB
  2. MyISAM
  3. MERGE (MRG_MyISAM)
  4. CSV

InnoDB Storage Engine

- Fully support ACID-compliant and transactions
- Optimal for performance
- Size limited to 64TB

MyISAM Storage Engine

- Optimal for compression and speed
- Portable across various platforms and OS
- Limited performance in read/write workloads
- Size limited to 256TB

MERGE Storage Engine

- Provides virtual table functionality
- Improves performance for join operation
- Provides safe transaction without data loss
- Handles nontransactional tables

CSV Storage Engine

- Supports data storage in CSV format
- Reading data requires a full table scan
- No support for NULL data and table indexing

Selection of Storage Engine

- The various storage engines included with MySQL are tailored to certain use cases.
- You can find a detailed feature summary for different storage engines offered with MySQL under **Course Resources** in the **SELF-LEARNING tab** of this course.

Storage Engine Setup in MySQL

Setting the Storage Engine in a Database

- To set a specific default storage engine for the current session, the **default_storage_engine** variable is set up using the **SET** command.

- Syntax:

Syntax

```
SET default_storage_engine=storage_engine;
```

Setting the Storage Engine in a Database: Example

Problem Statement: Your manager wants you to set the default storage engine for the employee database to **INNODB** in the MySQL setup.

Objective: Set the **default_storage_engine** variable for the **employees_db** database to **INNODB** using the **SET** clause in MySQL.

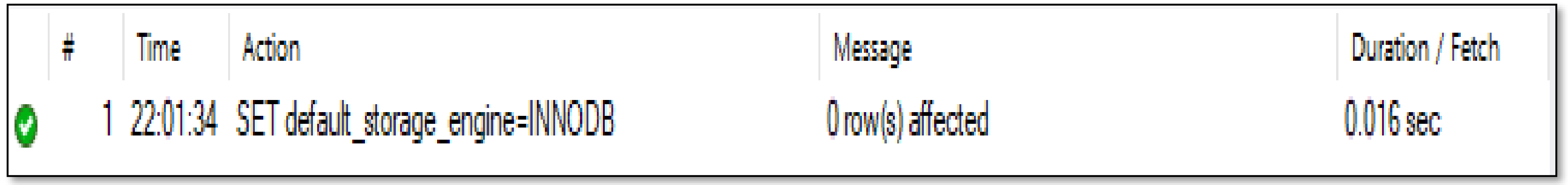
Setting the Storage Engine in a Database: Example

Step 1: Use the **SET** statement to set the value for the **default_storage_engine** variable as **INNODB** to make it the default storage engine for the **employees_db** database as given below.

SQL Query

```
SET default_storage_engine = INNODB;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:01:34	SET default_storage_engine=INNODB	0 row(s) affected	0.016 sec

Setting the Storage Engine for a Table

- The **ENGINE** table option in the **CREATE TABLE** command is used to define a storage engine.
- Syntax:

Syntax

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column_1_definition,
   column_2_definition,
   ...,
   table_constraints
) ENGINE=storage_engine;
```

Setting the Storage Engine for a Table: Example

Problem Statement: You work as a junior analyst at your organization. You must assist the department in creating a basic employee and manager relation table in one of the databases. This will allow managers to keep track of basic employee information and managers allocated to them.

Objective: Build the appropriate table structure in the **employees_db** database for storing the required manager-specific data.

Setting the Storage Engine for a Table: Example

The managers have provided a detailed description of the required employee table given 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)
ROLE	Employee job designation
DEPT	Department of the employee

Setting the Storage Engine for a Table: Example

Consider the employee table given below, which has the columns: EMP_ID, FIRST_NAME, LAST_NAME, GENDER, ROLE, DEPT, and MANAGER_ID.

EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
E260	Roy	Collins	M	SR DATA SCIENTIST	RETAIL	E583
E620	Katrina	Allen	F	SR DATA SCIENTIST	FINANCE	E612
E583	John	Hale	M	MANAGER	RETAIL	E002
E612	Tracy	Norris	F	MANAGER	FINANCE	E002
E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001

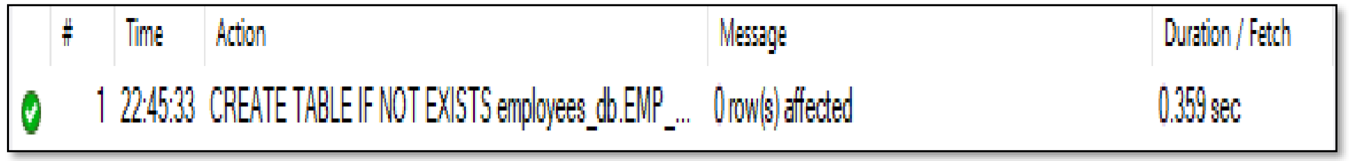
Setting the Storage Engine for a Table: Example

Step 1: Create the required **EMP_TABLE** table in the **employees_db** database with the **CREATE TABLE** statement and set the value for the **ENGINE** variable as **INNODB** as given below.

SQL Query

```
CREATE TABLE IF NOT EXISTS employees_db.EMP_TABLE (
    EMP_ID VARCHAR(4) NOT NULL,
    FIRST_NAME VARCHAR(100) NOT NULL,
    LAST_NAME VARCHAR(100),
    GENDER VARCHAR(1),
    ROLE VARCHAR(100),
    DEPT VARCHAR(100),
    MANAGER_ID VARCHAR(100),
    check(GENDER in ('M', 'F', 'O'))
)  ENGINE=INNODB;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:45:33	CREATE TABLE IF NOT EXISTS employees_db.EMP_...	0 row(s) affected	0.359 sec

Changing the Storage Engine of a Table

- To convert a table from one storage engine to another, the **ALTER TABLE** statement is used.
- Syntax:

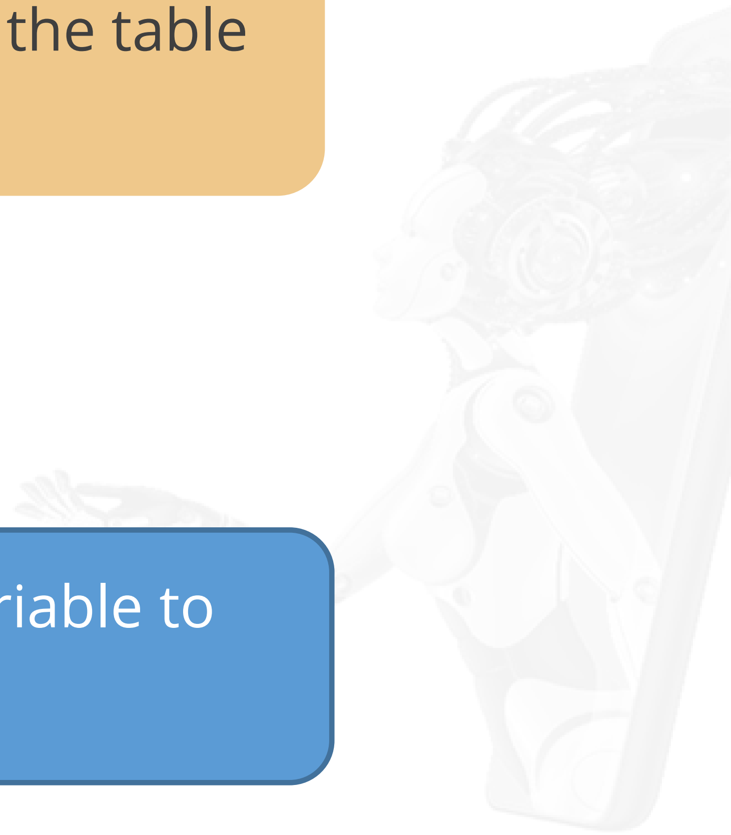
Syntax

```
ALTER TABLE table_name ENGINE = storage_engine;
```

Changing the Storage Engine of a Table: Example

Problem Statement: Your manager wants you to change the storage engine for the table created for employee data to **MERGE** in the MySQL setup.

Objective: Use the **ALTER TABLE** statement to set the value for the **ENGINE** variable to **MERGE** to set it as the default storage engine for the **EMP_TABLE** table.



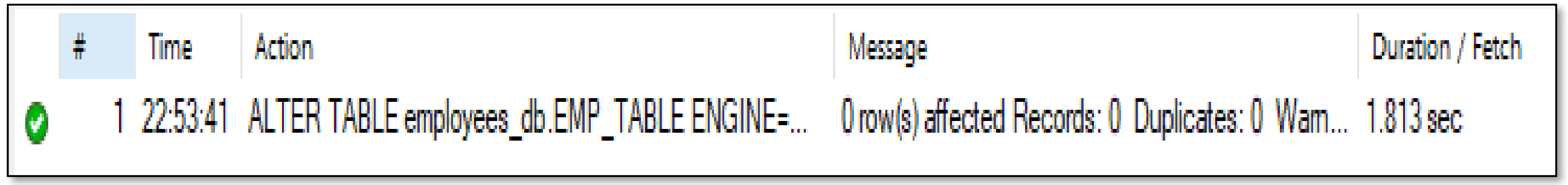
Changing the Storage Engine of a Table: Example

Step 1: Use the **ALTER TABLE** statement to set the value for the **ENGINE** variable to **MERGE** to set it as the default storage engine for the **EMP_TABLE** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_TABLE ENGINE=MERGE;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:53:41	ALTER TABLE employees_db.EMP_TABLE ENGINE=...	0 row(s) affected Records: 0 Duplicates: 0 Warn...	1.813 sec

Creating and Managing Tables in MySQL

Table Manipulation Commands in MySQL

Following are the most important SQL commands for table manipulation:

1. CREATE TABLE
2. DESCRIBE
3. SHOW TABLES
4. ALTER TABLE
5. TRUNCATE TABLE
6. DROP TABLE

Creating Tables in MySQL

- To build a new MySQL table in a specific database, use the **CREATE TABLE** statement.
- Syntax:

Syntax

```
CREATE TABLE [IF NOT EXISTS] [database_name.]table_name(
   column_1_definition,
   column_2_definition,
   ...,
   table_constraints
) ENGINE=storage_engine;
```

Creating Tables in MySQL

- Parameters:

Keywords	Meaning
database_name **(Optional)**	A unique name of the existing database if it is not the default set database
IF NOT EXISTS **(Optional)**	It prevents an error from occurring while creating a table that already exists
table_name	A unique name for a new table

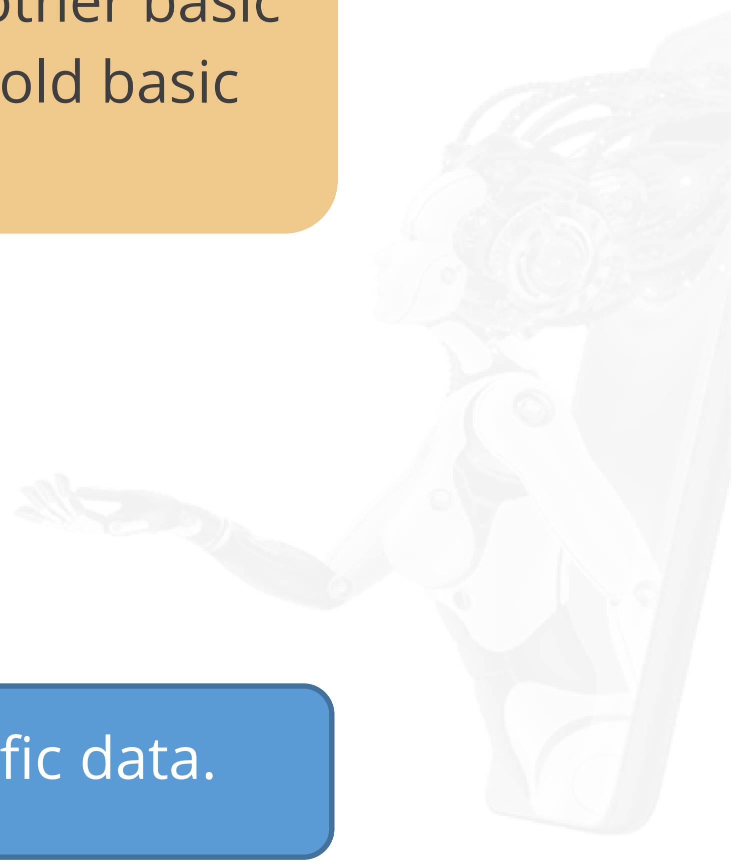
Creating Tables in MySQL

- Parameters:

Keywords	Meaning
column_n_definition	It provides the column name as well as the data types for each column. The comma operator is used to divide the columns in a table definition. Column definition syntax: **column_name1 data_type(size) [NULL | NOT NULL]**
ENGINE	It specifies a specific type of storage engine for the table. The default is **INNODB**.

Creating Tables in MySQL: Example

Problem Statement: The HR department requires your assistance in creating another basic employee and manager relation table in one of the databases for managers to hold basic employee details and managers assigned to them.



Objective: Build the appropriate table structure for storing the managers-specific data.

Creating Tables in MySQL: Example

The managers have provided a detailed description of the required employee table given 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)
ROLE	Employee job designation
DEPT	Department of the employee

Creating Tables in MySQL: Example

Consider the employee table which has the columns: EMP_ID, FIRST_NAME, LAST_NAME, and GENDER.

EMP_ID	FIRST_NAME	LAST_NAME	GENDER
E260	Roy	Collins	M
E620	Katrina	Allen	F
E583	John	Hale	M
E612	Tracy	Norris	F
E002	Cynthia	Brooks	F

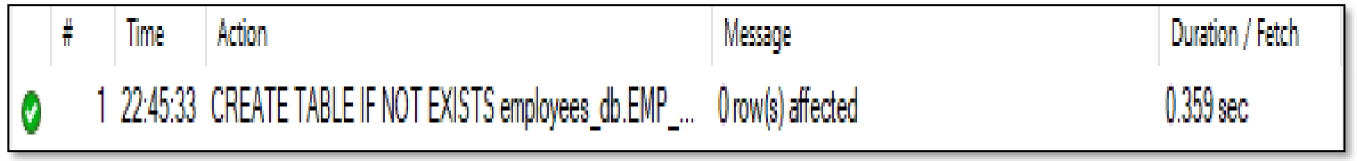
Creating Tables in MySQL: Example

Step 1: Create the required **EMP_RECORDS** table in the **employees_db** database with the **CREATE TABLE** statement as given below.

SQL Query

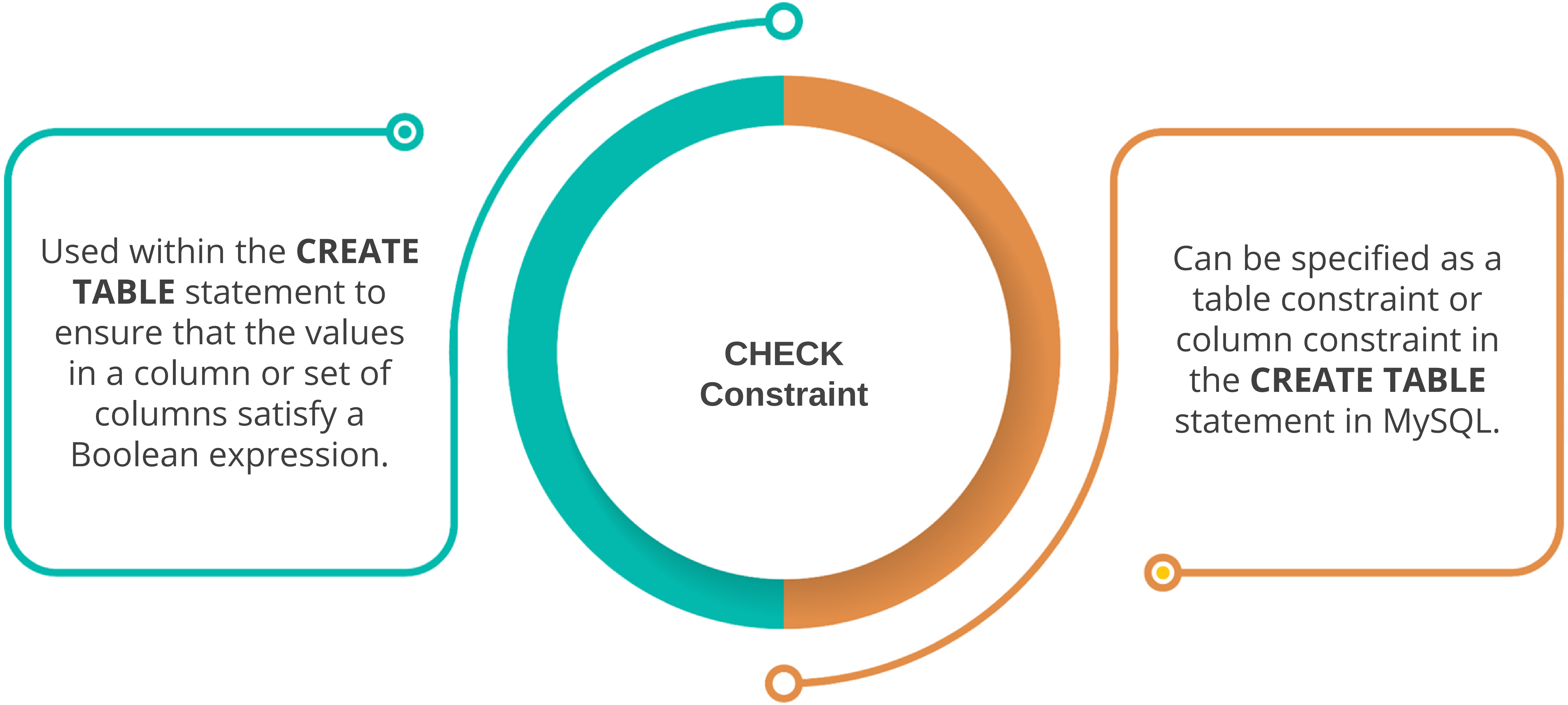
```
CREATE TABLE IF NOT EXISTS employees_db.EMP_RECORDS (
    EMP_ID VARCHAR(4) NOT NULL,
    FIRST_NAME VARCHAR(100) NOT NULL,
    LAST_NAME VARCHAR(100),
    GENDER VARCHAR(1)
)  ENGINE=INNODB;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:45:33	CREATE TABLE IF NOT EXISTS employees_db.EMP_...	0 row(s) affected	0.359 sec

CHECK Constraint in MySQL

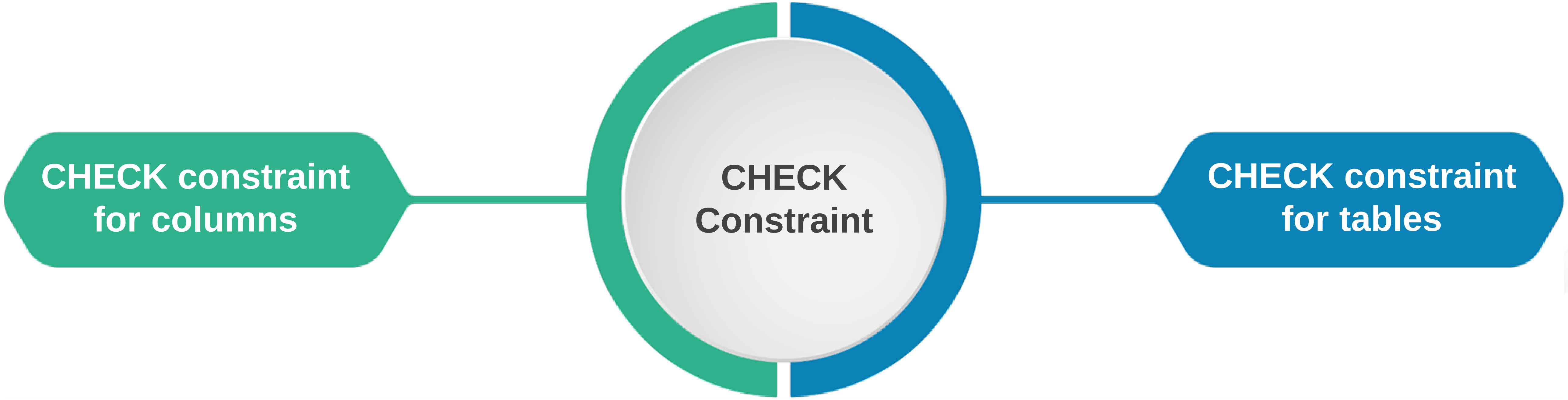


Used within the **CREATE TABLE** statement to ensure that the values in a column or set of columns satisfy a Boolean expression.

CHECK Constraint

Can be specified as a table constraint or column constraint in the **CREATE TABLE** statement in MySQL.

CHECK Constraint in MySQL



CHECK constraint for columns

CHECK Constraint

CHECK constraint for tables

CHECK Constraint in MySQL

- Syntax for adding a column constraint:

Syntax

```
CREATE TABLE [IF NOT EXISTS] [database_name.]table_name(
   column_1_definition,
   column_2_definition CHECK (expression),
   ...,
) ENGINE=storage_engine;
```

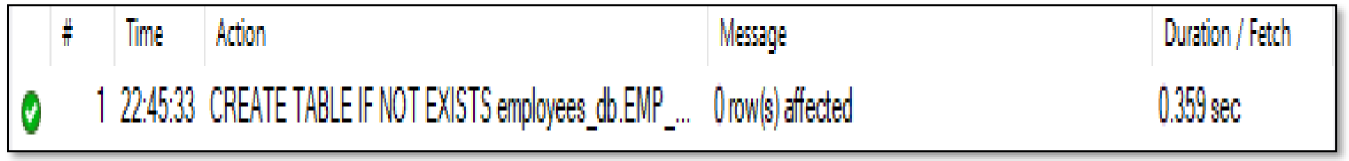
CHECK Constraint in MySQL: Example

Step 2: Use the **CREATE TABLE** statement with the **CHECK** clause and a column constraint for the **EXP** column to create another table similar to **EMP_RECORDS** named **EMP_TABLE_2** as given below.

SQL Query

```
CREATE TABLE IF NOT EXISTS employees_db.EMP_TABLE_2
(
    EMP_ID VARCHAR(4) NOT NULL,
    FIRST_NAME VARCHAR(100) NOT NULL,
    LAST_NAME VARCHAR(100),
    GENDER VARCHAR(1),
    ROLE VARCHAR(100),
    DEPT VARCHAR(100),
    MANAGER_ID VARCHAR(100),
    EXP INTEGER NOT NULL CHECK (EXP >= 0)
)  ENGINE=INNODB;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:45:33	CREATE TABLE IF NOT EXISTS employees_db.EMP_...	0 row(s) affected	0.359 sec

CHECK Constraint in MySQL

- Syntax for adding a table along with a column constraint:

Syntax

```
CREATE TABLE [IF NOT EXISTS] [database_name.]table_name(
   column_1_definition,
   column_2_definition CHECK (expression),
   ...,
   [CONSTRAINT [constraint_name]] CHECK (expression) [[NOT] ENFORCED]
) ENGINE=storage_engine;
```

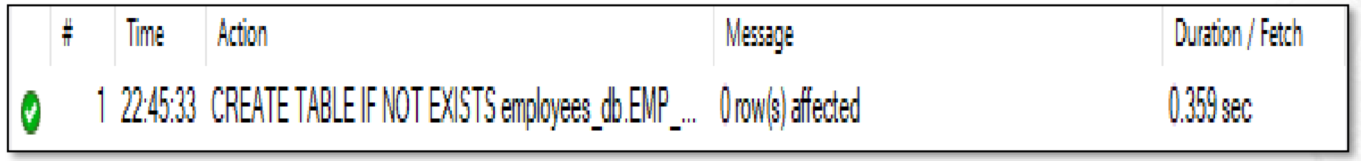
CHECK Constraint in MySQL: Example

Step 3: Use the **CREATE TABLE** statement with the **CHECK** and **CONSTRAINT** clauses along with a table constraint for the **EXP** and **GENDER** columns respectively to create another table similar to **EMP_TABLE_2** named **EMP_TABLE_3** as given below.

SQL Query

```
CREATE TABLE IF NOT EXISTS employees_db.EMP_TABLE_3 (
    EMP_ID VARCHAR(4) NOT NULL,
    FIRST_NAME VARCHAR(100) NOT NULL,
    LAST_NAME VARCHAR(100),
    GENDER VARCHAR(1),
    ROLE VARCHAR(100),
    DEPT VARCHAR(100),
    MANAGER_ID VARCHAR(100),
    EXP INTEGER NOT NULL CHECK (EXP >= 0),
    CONSTRAINT gender_check CHECK(GENDER in ('M', 'F', 'O'))
)  ENGINE=INNODB;
```

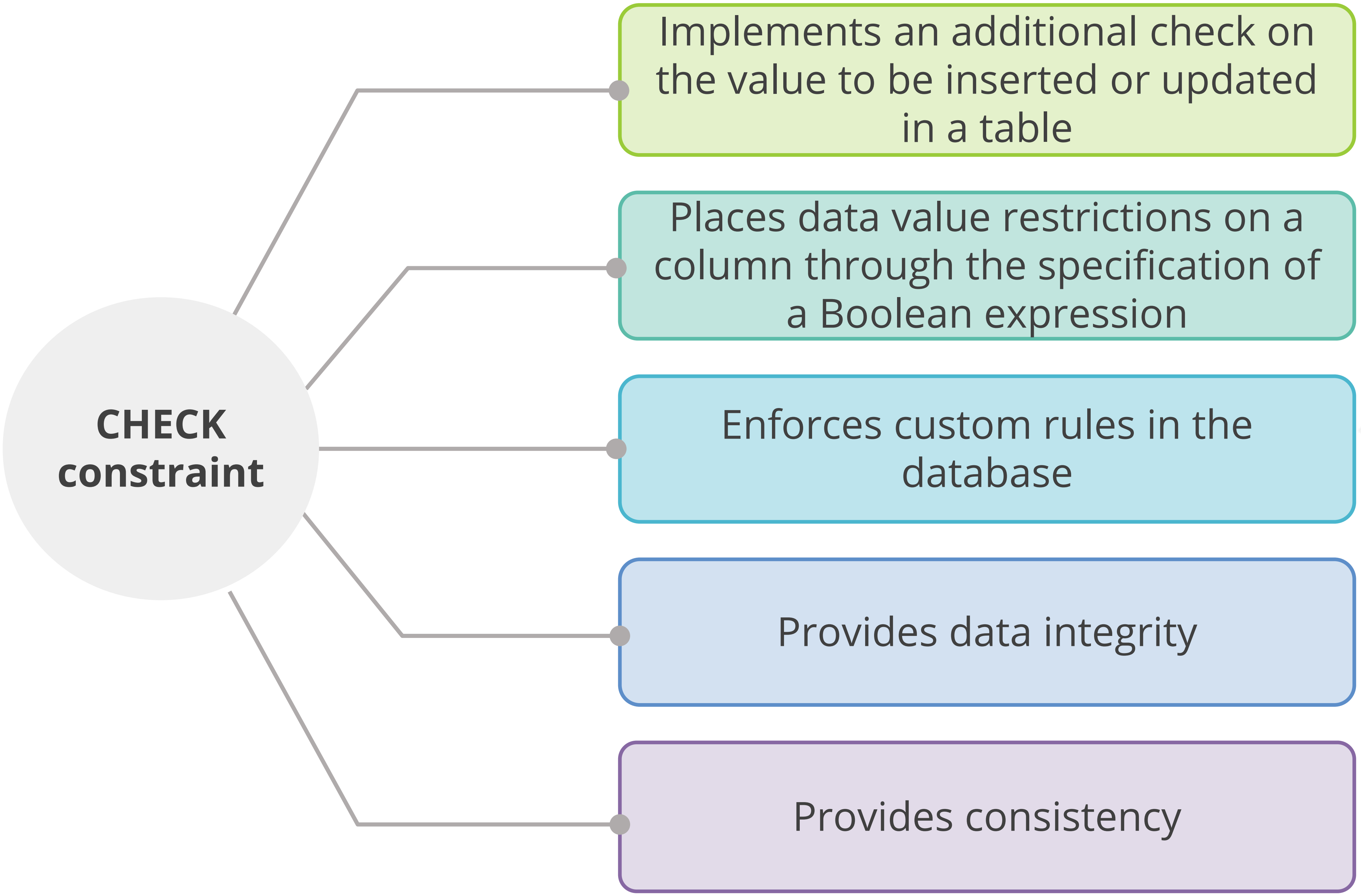
Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:45:33	CREATE TABLE IF NOT EXISTS employees_db.EMP_...	0 row(s) affected	0.359 sec

Benefits of CHECK Constraint

The CHECK Constraint has the following benefits:



CHECK constraint

Implements an additional check on the value to be inserted or updated in a table

Places data value restrictions on a column through the specification of a Boolean expression

Enforces custom rules in the database

Provides data integrity

Provides consistency

Analyzing Table Description in MySQL

- To view the structure of a MySQL table, use the **DESCRIBE** statement.
- Syntax:

Syntax

```
DESCRIBE table_name;
```

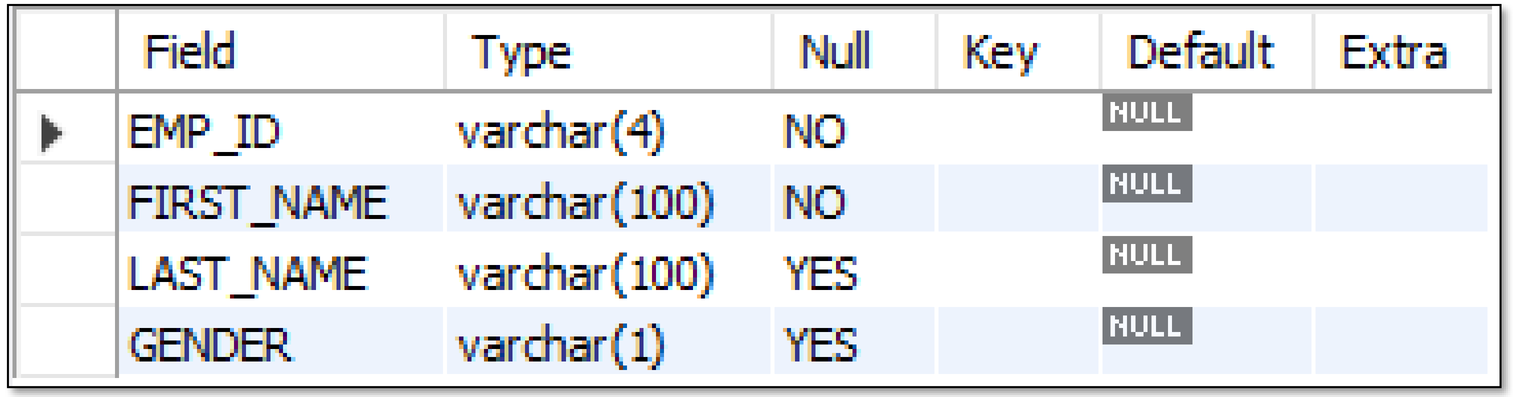
Analyzing Table Description in MySQL: Example

Step 4: Use the **DESCRIBE** statement to analyze the structure of the **EMP_RECORDS** table as given below.

SQL Query

```
DESCRIBE employees_db.EMP_RECORDS;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	YES		NULL	

Listing Existing Tables in MySQL

- To list all databases in the MySQL database, use the **SHOW TABLES** statement.
- Syntax:

Syntax

```
SHOW TABLES;
```

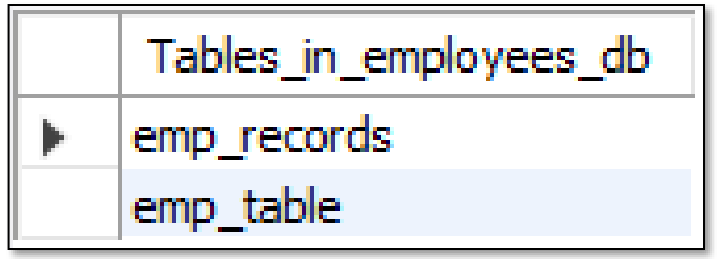
Listing Existing Tables in MySQL: Example

Step 5: Use the **SHOW TABLES** statement to all the available tables in the **employees_db** database as given below.

SQL Query

```
SHOW TABLES;
```

Output:



	Tables_in_employees_db
▶	emp_records
	emp_table

Modifying Table Structure in MySQL

- The **ALTER** statement is used to modify a table by performing actions such as adding a column, altering a column, renaming a column, dropping a column, and renaming a table.

- Following are some of the operations that the **ALTER** statement can perform:

  1. Adding columns to a table
  2. Modifying columns in a table
  3. Renaming a column in a table
  4. Dropping a column
  5. Renaming a table

Adding Columns to a Table

- Two ways to add columns to a table:

  1. Adding a single column to a table

  2. Adding multiple columns to a table

Adding Columns to a Table

- Syntax for adding a single column to a table:

Syntax

```
ALTER TABLE table_name
ADD
    new_column_name column_definition
    [FIRST | AFTER column_name]
```

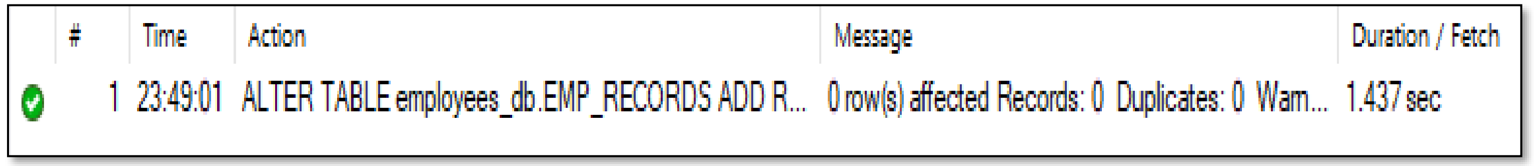
Adding Columns to a Table: Example

Step 6: Use the **ALTER TABLE** statement with the **ADD** clause to add a **ROLE** column in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS
ADD ROLE VARCHAR(100);
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	23:49:01	ALTER TABLE employees_db.EMP_RECORDS ADD R...	0 row(s) affected Records: 0 Duplicates: 0 Warn...	1.437 sec

Adding Columns to a Table

- Syntax for adding multiple columns to a table:

Syntax

```
ALTER TABLE table_name
    ADD new_column_name column_definition
    [FIRST | AFTER column_name],
    ADD new_column_name column_definition
    [FIRST | AFTER column_name],
    ...;
```

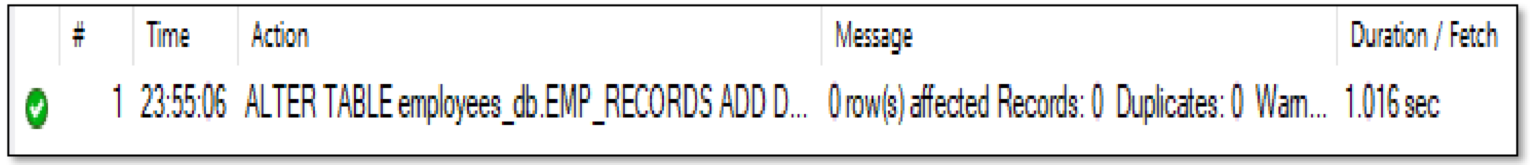
Adding Columns to a Table: Example

Step 7: Use the **ALTER TABLE** statement with the **ADD** clause to add the **DEPT** and **MANAGER_ID** columns in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS
ADD DEPT VARCHAR(100),
ADD MANAGER_ID VARCHAR(100);
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	23:55:06	ALTER TABLE employees_db.EMP_RECORDS ADD D...	0 row(s) affected Records: 0 Duplicates: 0 Warn...	1.016 sec

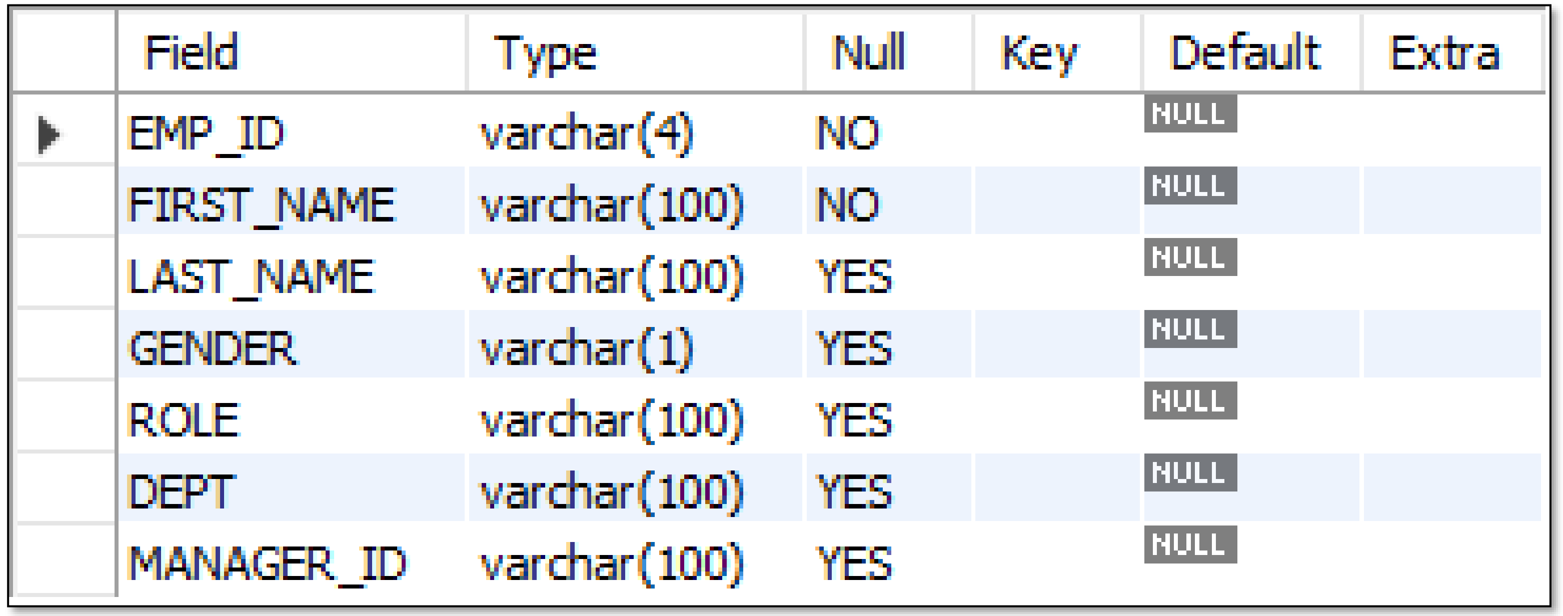
Adding Columns to a Table: Example

Step 8: Use the **DESCRIBE** statement to analyze the structure of the **EMP_RECORDS** table to verify the changes as given below.

SQL Query

```
DESCRIBE employees_db.EMP_RECORDS;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	YES		NULL	
	ROLE	varchar(100)	YES		NULL	
	DEPT	varchar(100)	YES		NULL	
	MANAGER_ID	varchar(100)	YES		NULL	

Modifying Columns in a Table

- Two ways to modify columns in a table:

  1. Modifying a single column in a table

  2. Modifying multiple columns in a table

Modifying Columns in a Table

- Syntax for modifying a single column in a table:

Syntax

```
ALTER TABLE table_name
MODIFY
       column_name column_definition
       [ FIRST | AFTER column_name];
```

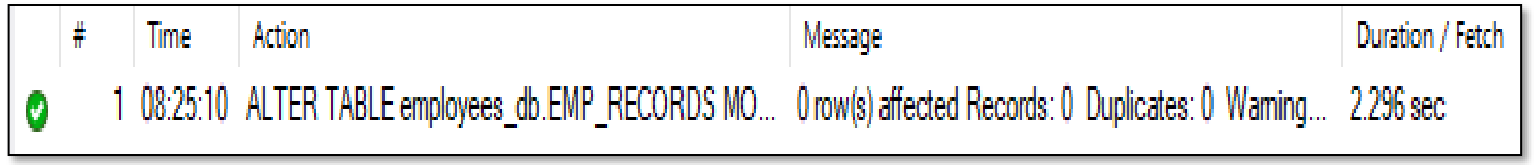
Modifying Columns in a Table: Example

Step 9: Use the **ALTER TABLE** statement with the **MODIFY** clause to add a **NOT NULL** constraint to the **GENDER** column in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS

MODIFY GENDER VARCHAR(1) NOT NULL;
```

Output:



```
    #  Time      Action                                   Message                                                  Duration / Fetch
✓   1  08:25:10  ALTER TABLE employees_db.EMP_RECORDS MO...  0 row(s) affected Records: 0  Duplicates: 0  Warning...  2.296 sec
```

Modifying Columns in a Table

- Syntax for modifying multiple columns in a table:

Syntax

```
ALTER TABLE table_name
    MODIFY column_name column_definition
    [ FIRST | AFTER column_name],
    MODIFY column_name column_definition
    [ FIRST | AFTER column_name],
    ...;
```

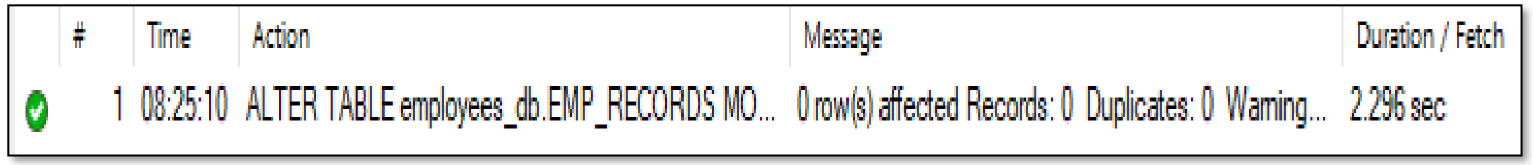
Modifying Columns in a Table: Example

Step 10: Use the **ALTER TABLE** statement with the **MODIFY** clause to add a **NOT NULL** constraint to both **DEPT** and **MANAGER_ID** columns in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS
MODIFY DEPT VARCHAR(100) NOT NULL,
MODIFY MANAGER_ID VARCHAR(100) NOT NULL;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	08:25:10	ALTER TABLE employees_db.EMP_RECORDS MO...	0 row(s) affected Records: 0 Duplicates: 0 Warning...	2.296 sec

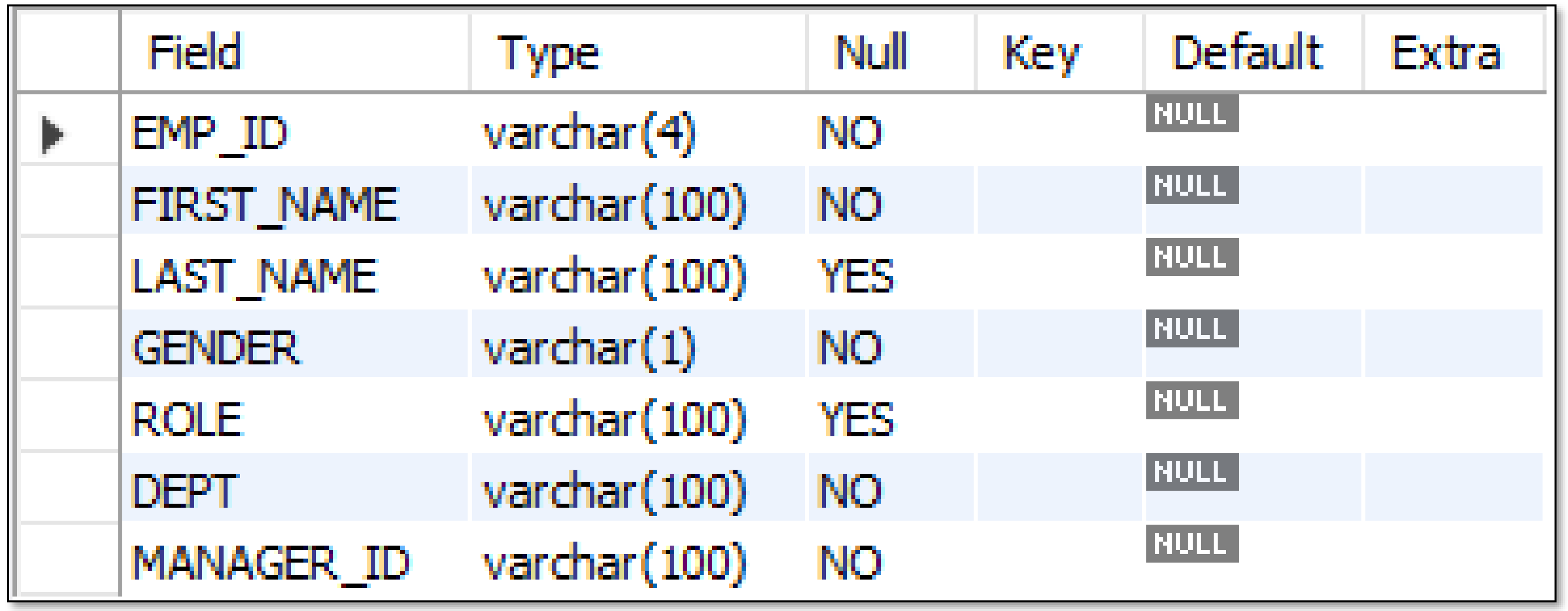
Modifying Columns in a Table: Example

Step 11: Use the **DESCRIBE** statement to analyze the structure of the **EMP_RECORDS** table to verify the changes as given below.

SQL Query

```
DESCRIBE employees_db.EMP_RECORDS;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	NO		NULL	
	ROLE	varchar(100)	YES		NULL	
	DEPT	varchar(100)	NO		NULL	
	MANAGER_ID	varchar(100)	NO		NULL	

Renaming a Column in a Table

- Syntax for renaming a column in a table:

Syntax

```
ALTER TABLE table_name
CHANGE COLUMN
    original_name new_name column_definition
    [FIRST | AFTER column_name];
```

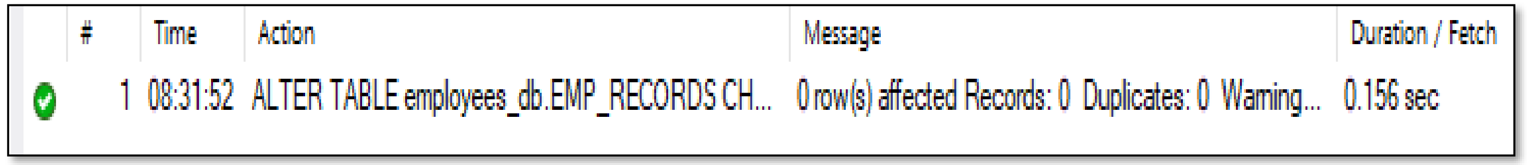
Renaming a Column: Example

Step 12: Use the **ALTER TABLE** statement with the **CHANGE COLUMN** clause to rename the **ROLE** column as **JOB** in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS
CHANGE COLUMN ROLE JOB VARCHAR(100);
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	08:31:52	ALTER TABLE employees_db.EMP_RECORDS CH...	0 row(s) affected Records: 0 Duplicates: 0 Warning...	0.156 sec

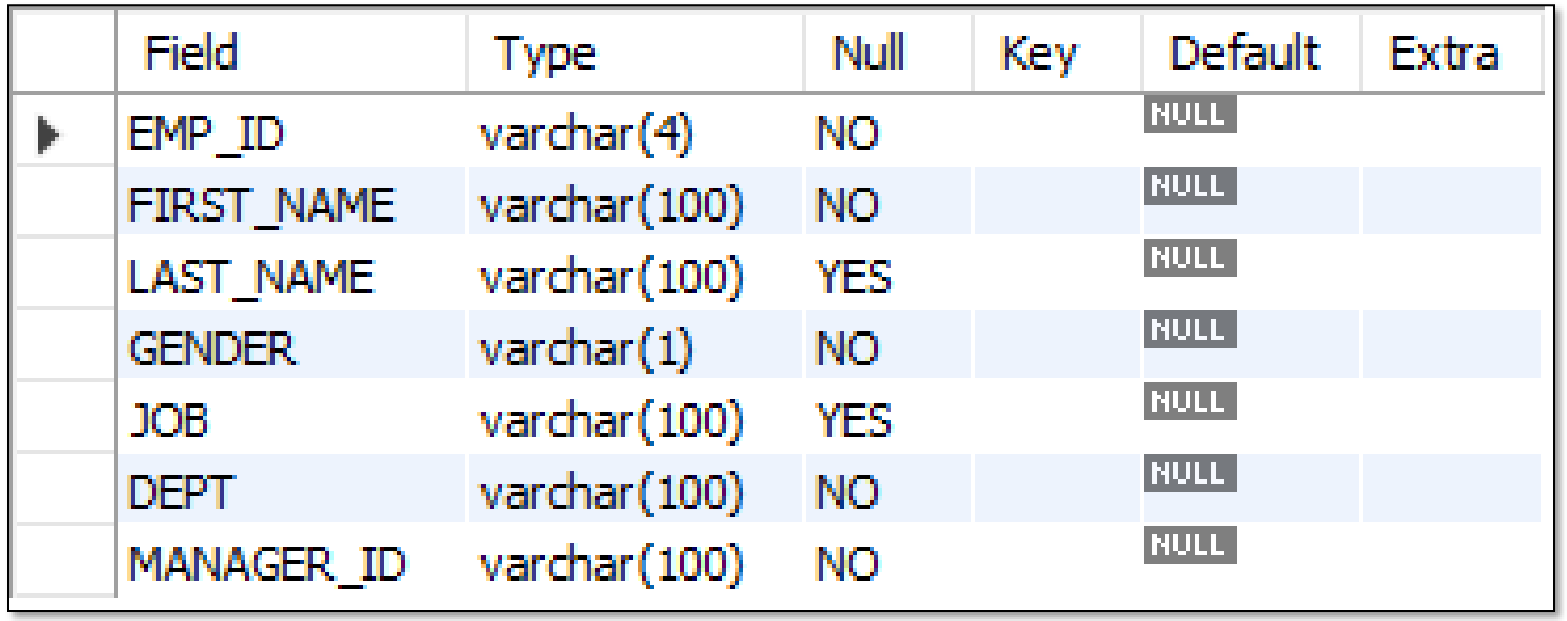
Renaming a Column: Example

Step 13: Use the **DESCRIBE** statement to analyze the structure of the **EMP_RECORDS** table to verify if the changes as given below.

SQL Query

```
DESCRIBE employees_db.EMP_RECORDS;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	NO		NULL	
	JOB	varchar(100)	YES		NULL	
	DEPT	varchar(100)	NO		NULL	
	MANAGER_ID	varchar(100)	NO		NULL	

Dropping a Column in a Table

- Syntax for dropping a column in a table:

Syntax

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

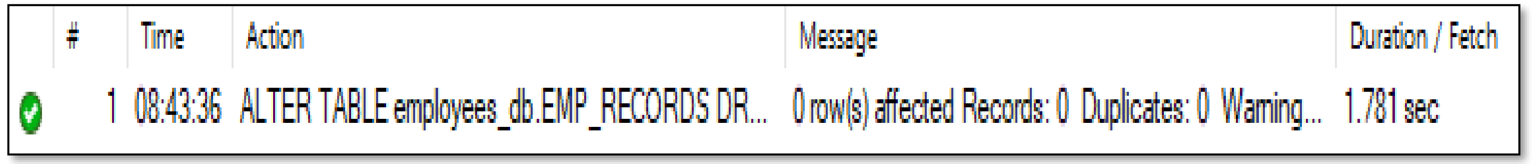
Dropping a Column: Example

Step 14: Use the **ALTER TABLE** statement with the **DROP** clause to drop the **MANAGER_ID** column in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS

DROP COLUMN MANAGER_ID;
```

Output:



```
   #  Time      Action                                    Message                                                  Duration / Fetch
✓  1  08:43:36  ALTER TABLE employees_db.EMP_RECORDS DR...  0 row(s) affected Records: 0  Duplicates: 0  Warning...  1.781 sec
```

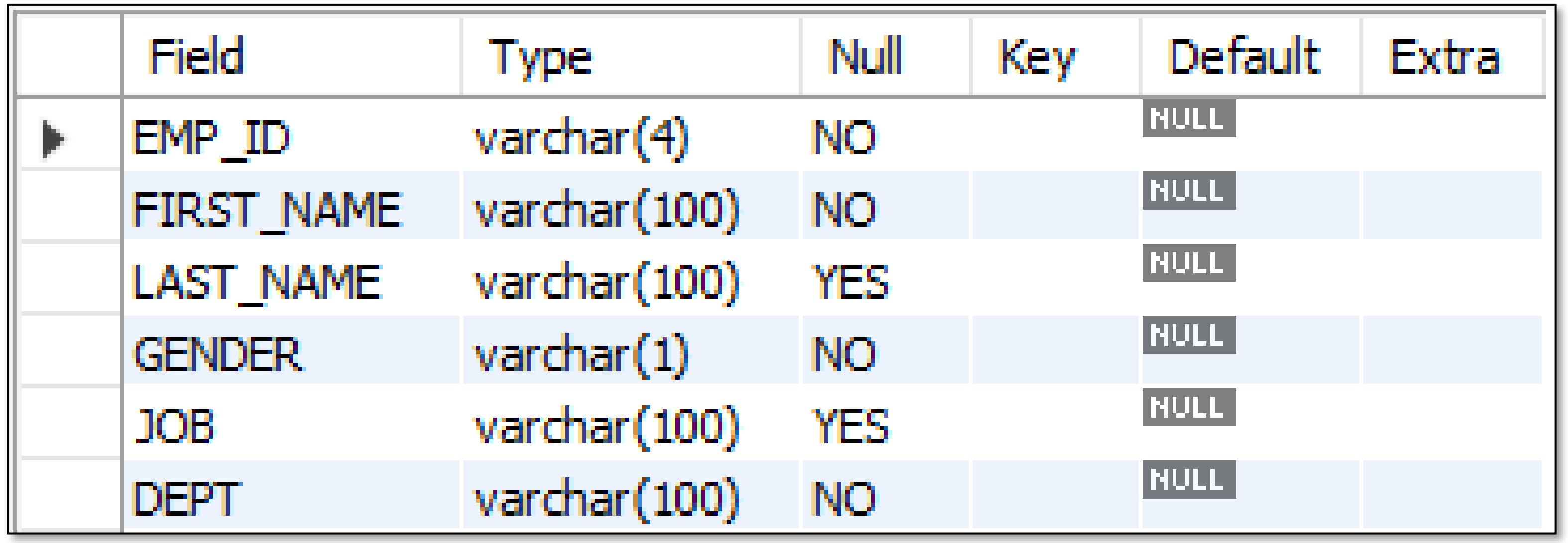
Dropping a Column: Example

Step 15: Use the **DESCRIBE** statement to analyze the structure of the **EMP_RECORDS** table to verify the changes as given below.

SQL Query

```
DESCRIBE employees_db.EMP_RECORDS;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	NO		NULL	
	JOB	varchar(100)	YES		NULL	
	DEPT	varchar(100)	NO		NULL	

Renaming a Table

- Syntax for renaming a table:

Syntax

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

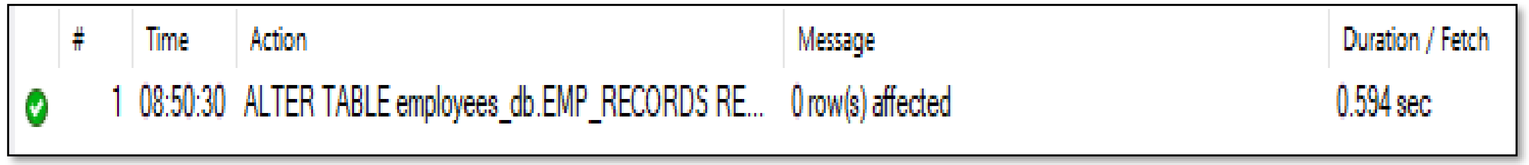
Renaming a Table: Example

Step 16: Use the **ALTER TABLE** statement with the **RENAME TO** clause to rename the **EMP_RECORDS** table to **EMP_DATA** as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_RECORDS
RENAME TO employees_db.EMP_DATA;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	08:50:30	ALTER TABLE employees_db.EMP_RECORDS RE...	0 row(s) affected	0.594 sec

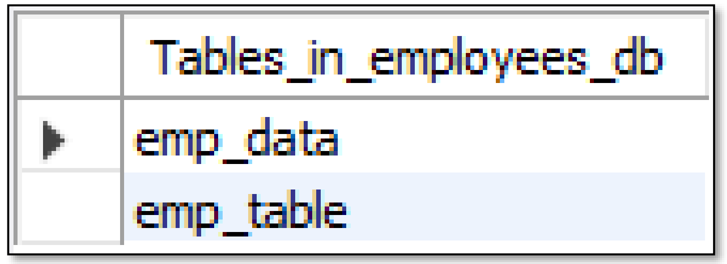
Renaming a Table: Example

Step 17: Use the **SHOW TABLES** statement to list all the available tables in the **employees_db** database to ensure that the changes are applied as given below.

SQL Query

```
SHOW TABLES;
```

Output:



Tables_in_employees_db
emp_data
emp_table

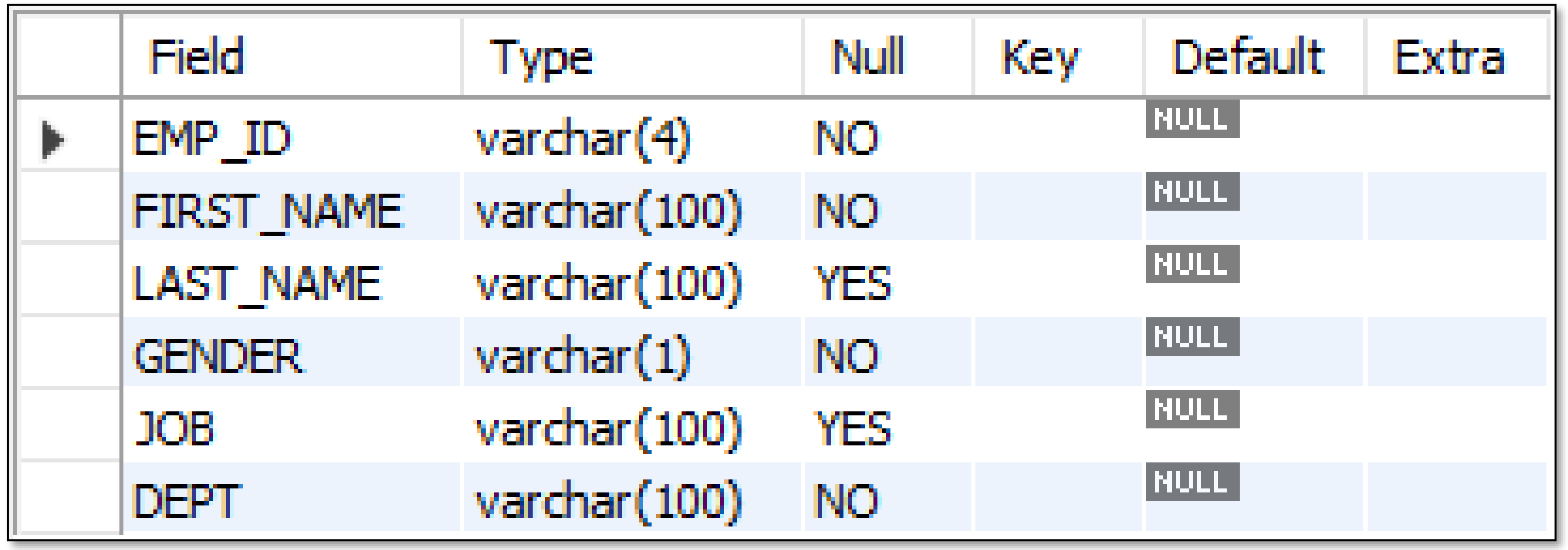
Renaming a Table: Example

Step 18: Use the **DESCRIBE** statement to analyze the structure of the **EMP_DATA** table to verify whether it has the same structure as **EMP_RECORDS** as given below.

SQL Query

```
DESCRIBE employees_db.EMP_DATA;
```

Output:



Field	Type	Null	Key	Default	Extra
EMP_ID	varchar(4)	NO		NULL	
FIRST_NAME	varchar(100)	NO		NULL	
LAST_NAME	varchar(100)	YES		NULL	
GENDER	varchar(1)	NO		NULL	
JOB	varchar(100)	YES		NULL	
DEPT	varchar(100)	NO		NULL	

Generated Columns in MySQL

- Generated columns are those whose data is generated using predefined expressions and computations.
- Syntax:

Syntax

```
col_name data_type [GENERATED ALWAYS] AS (expr)
  [VIRTUAL | STORED] [NOT NULL | NULL]
  [UNIQUE [KEY]] [[PRIMARY] KEY]
  [COMMENT 'string']
```

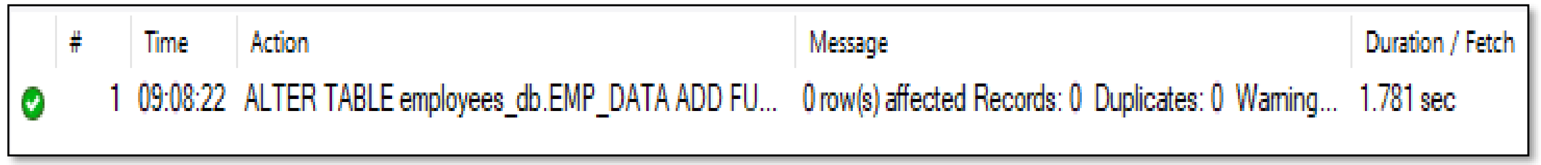
Generated Columns in MySQL: Example

Step 19: Use the **ALTER TABLE** statement with the **ADD** and **GENERATED ALWAYS AS** clauses to add a **FULL_NAME** column generated by combining the **FIRST_NAME** and **LAST_NAME** columns in the **EMP_RECORDS** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_DATA
ADD FULL_NAME VARCHAR(200)
GENERATED ALWAYS AS(CONCAT(FIRST_NAME,' ',LAST_NAME));
```

Output:



```
  #  Time      Action                                 Message                                                    Duration / Fetch
✓ 1  09:08:22  ALTER TABLE employees_db.EMP_DATA ADD FU...  0 row(s) affected Records: 0  Duplicates: 0  Warning...  1.781 sec
```

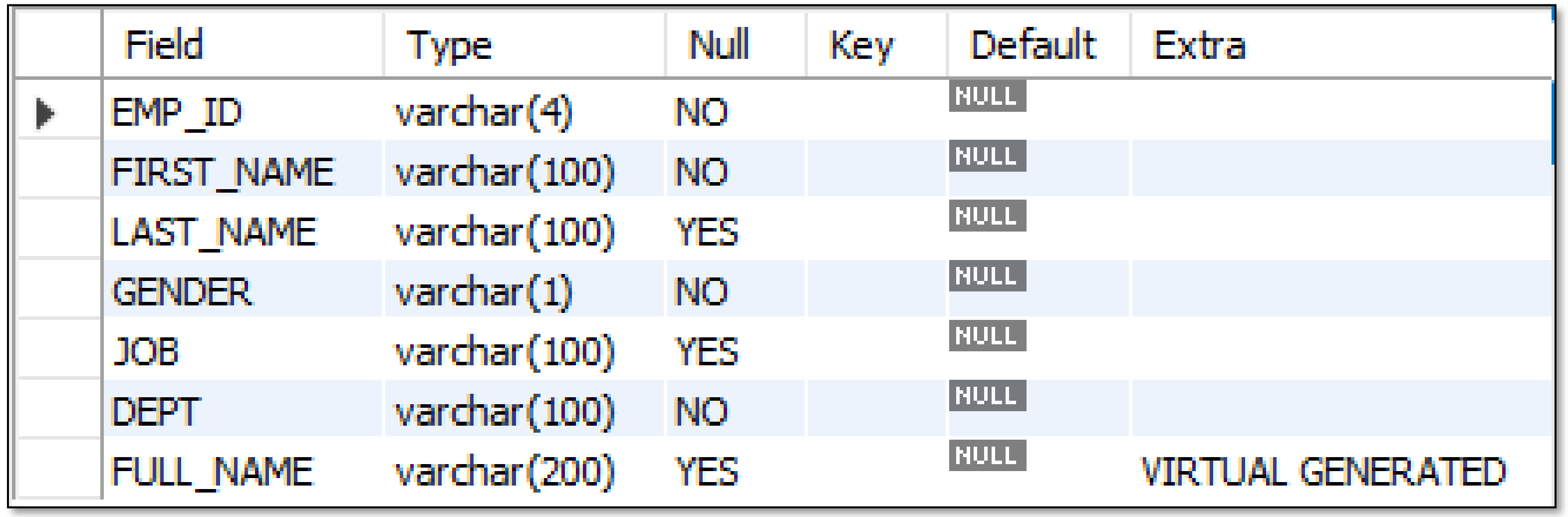
Generated Columns in MySQL: Example

Step 20: Use the **DESCRIBE** statement to analyze the structure of the **EMP_RECORDS** table to verify if the changes as given below.

SQL Query

```
DESCRIBE employees_db.EMP_DATA;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	NO		NULL	
	JOB	varchar(100)	YES		NULL	
	DEPT	varchar(100)	NO		NULL	
	FULL_NAME	varchar(200)	YES		NULL	VIRTUAL GENERATED

Truncating Tables in MySQL

- To delete all data in an existing table in the MySQL database, use the **TRUNCATE TABLE** statement.

- Syntax:

Syntax

```
TRUNCATE [TABLE] table_name;
```

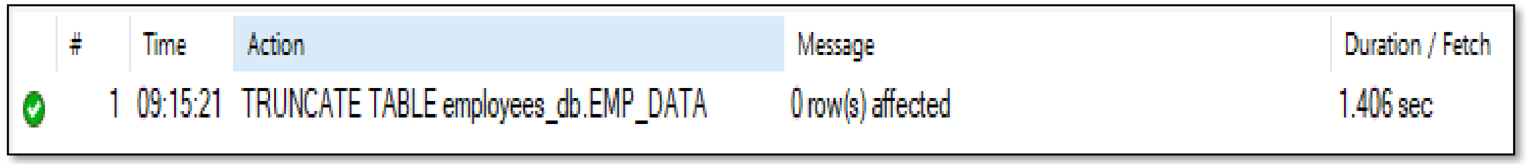
Truncating Tables in MySQL: Example

Step 21: Use the **TRUNCATE TABLE** statement to delete all records from the **EMP_DATA** table as given below.

SQL Query

```
TRUNCATE TABLE employees_db.EMP_DATA;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	09:15:21	TRUNCATE TABLE employees_db.EMP_DATA	0 row(s) affected	1.406 sec

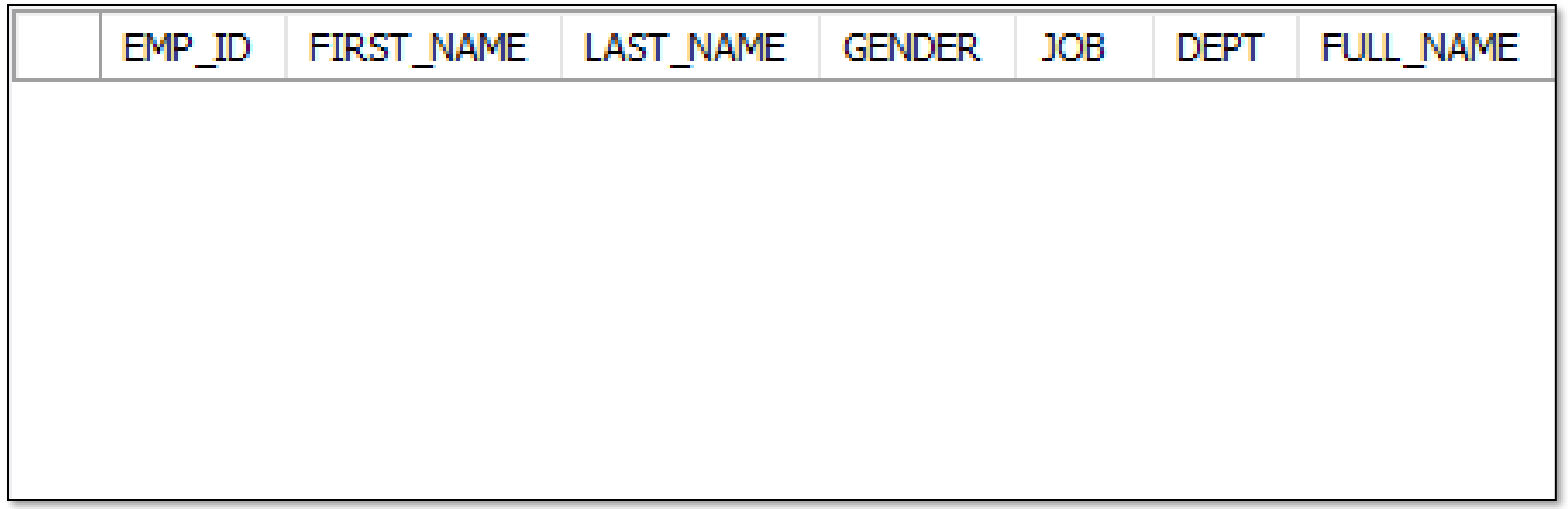
Truncating Tables in MySQL: Example

Step 22: Use the **SELECT** statement with ***** condition to analyze the data available in the **EMP_DATA** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_DATA;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	JOB	DEPT	FULL_NAME

Dropping Tables in MySQL

- To remove existing tables in the MySQL database, use the **DROP TABLE** statement.
- Syntax:

Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS] table_name [, table_name] ...
[RESTRICT | CASCADE]
```

Dropping Tables in MySQL

- The dropping operation can be performed in the following ways:

  1. Dropping a single table

  2. Dropping multiple tables

Dropping Tables in MySQL

- Syntax for dropping a specific table:

Syntax

```
DROP TABLE IF EXISTS
  database_name.table_name;
```

Dropping Tables in MySQL

- Syntax for dropping multiple tables:

Syntax

```
DROP TABLE IF EXISTS
  database_name.table_name_1,
  database_name.table_name_2,
  ...;
```

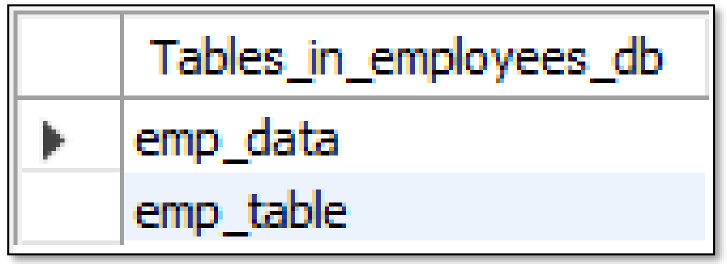
Dropping Tables in MySQL: Example

Step 23: Use the **SHOW TABLES** statement to list all the available tables in the **employees_db** database as given below.

SQL Query

```
SHOW TABLES;
```

Output:



```
Tables_in_employees_db
emp_data
emp_table
```

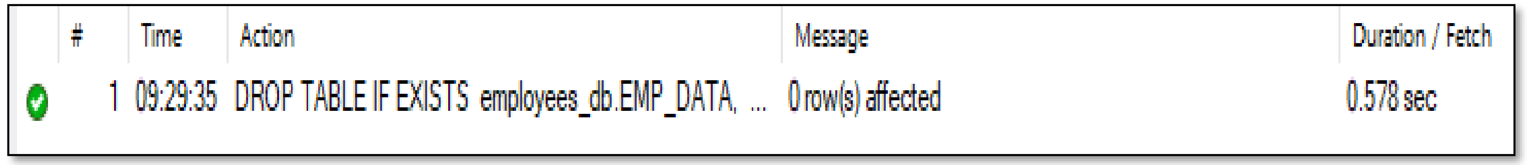
Dropping Tables in MySQL: Example

Step 24: Use the **DROP TABLE** statement to delete both **EMP_DATA** and **EMP_RECORDS** tables from **employees_db** database as given below.

SQL Query

```
DROP TABLE IF EXISTS
 employees_db.EMP_DATA,
 employees_db.EMP_TABLE;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	09:29:35	DROP TABLE IF EXISTS employees_db.EMP_DATA, ...	0 row(s) affected	0.578 sec

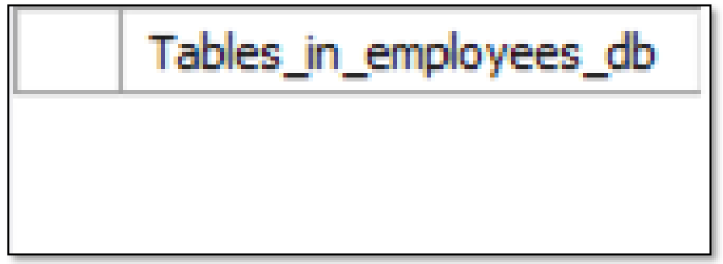
Dropping Tables in MySQL: Example

Step 23: Use the **SHOW TABLES** statement to list all the available tables in the **employees_db** database to verify the changes as given below.

SQL Query

```sql
SHOW TABLES;
```

Output:



```
Tables_in_employees_db
```

DROP vs. TRUNCATE

DROP

- It removes the definition, indexes, data, constraints, and triggers for a table.
- It removes the integrity constraints.
- The table structure does not exist after the drop operation.
- It deletes the table from memory.
- It is slower than the TRUNCATE command.

TRUNCATE

- It deletes all the tuples from the table.
- It doesn't remove the integrity constraints.
- The table structure still exists after the truncate operation.
- It clears the values in the table but does not delete the table from memory.
- It is faster than the DROP command.

Inserting Data in Tables

Inserting Data in Tables

- In MySQL, the **INSERT** statement is used to insert one or more rows to a table.
- The number of columns and values must be the same when using the **INSERT** statement. Furthermore, the columns' positions must correspond to the positions of their values.
- Syntax:

Syntax

```
INSERT INTO table(c1,c2,...,cn)
VALUES (v1,v2,...,vn);
```

Inserting Data in Tables: Example

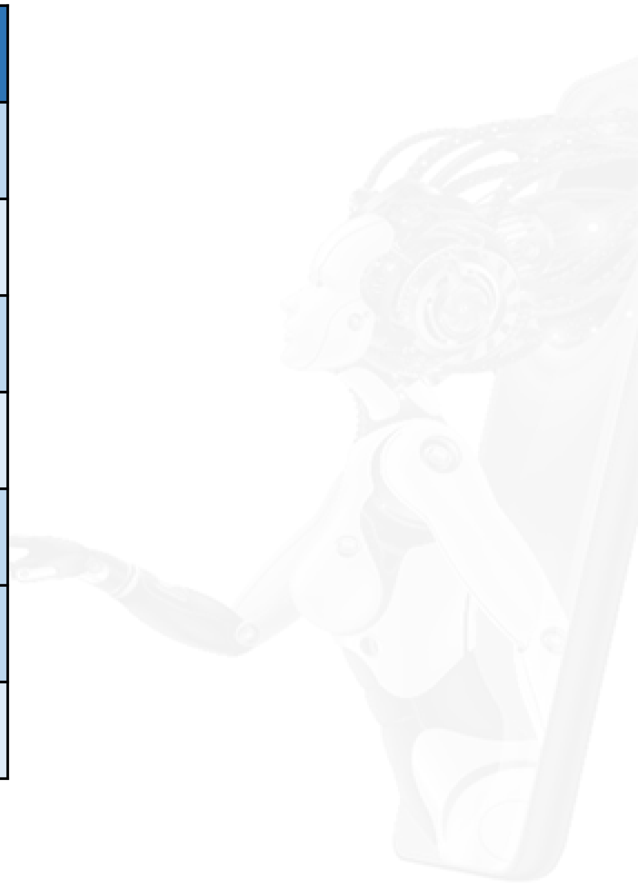
Problem Statement: You work as a junior analyst at your organization. You must assist the department in creating a basic employee and manager relation table in one of the databases for managers to keep track of basic employee information and managers allocated to them.

Objective: Build the appropriate table structure in the **employees_db** database for storing the required manager-specific data.

Inserting Data in Tables: Example

The managers have provided a detailed description of the required employee table given 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)
ROLE	Employee job designation
DEPT	Department of the employee



Inserting Data in Tables: Example

Consider the employee table given below, which has the columns: EMP_ID, FIRST_NAME, LAST_NAME, GENDER, ROLE, DEPT, and MANAGER_ID.

EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
E260	Roy	Collins	M	SR DATA SCIENTIST	RETAIL	E583
E620	Katrina	Allen	F	SR DATA SCIENTIST	FINANCE	E612
E583	John	Hale	M	MANAGER	RETAIL	E002
E612	Tracy	Norris	F	MANAGER	FINANCE	E002
E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001

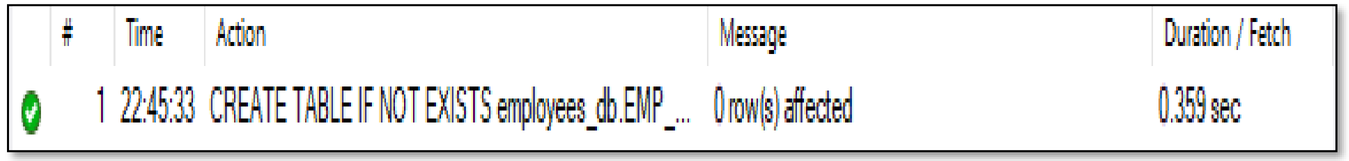
Inserting Data in Tables: Example

Step 1: Create the required **EMP_TABLE** table in the **employees_db** database with the **CREATE TABLE** statement as given below.

SQL Query

```
CREATE TABLE IF NOT EXISTS employees_db.EMP_TABLE (
    EMP_ID VARCHAR(4) NOT NULL,
    FIRST_NAME VARCHAR(100) NOT NULL,
    LAST_NAME VARCHAR(100),
    GENDER VARCHAR(1),
    ROLE VARCHAR(100),
    DEPT VARCHAR(100),
    MANAGER_ID VARCHAR(100),
    check(GENDER in ('M', 'F', 'O'))
)  ENGINE=INNODB;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	22:45:33	CREATE TABLE IF NOT EXISTS employees_db.EMP_...	0 row(s) affected	0.359 sec

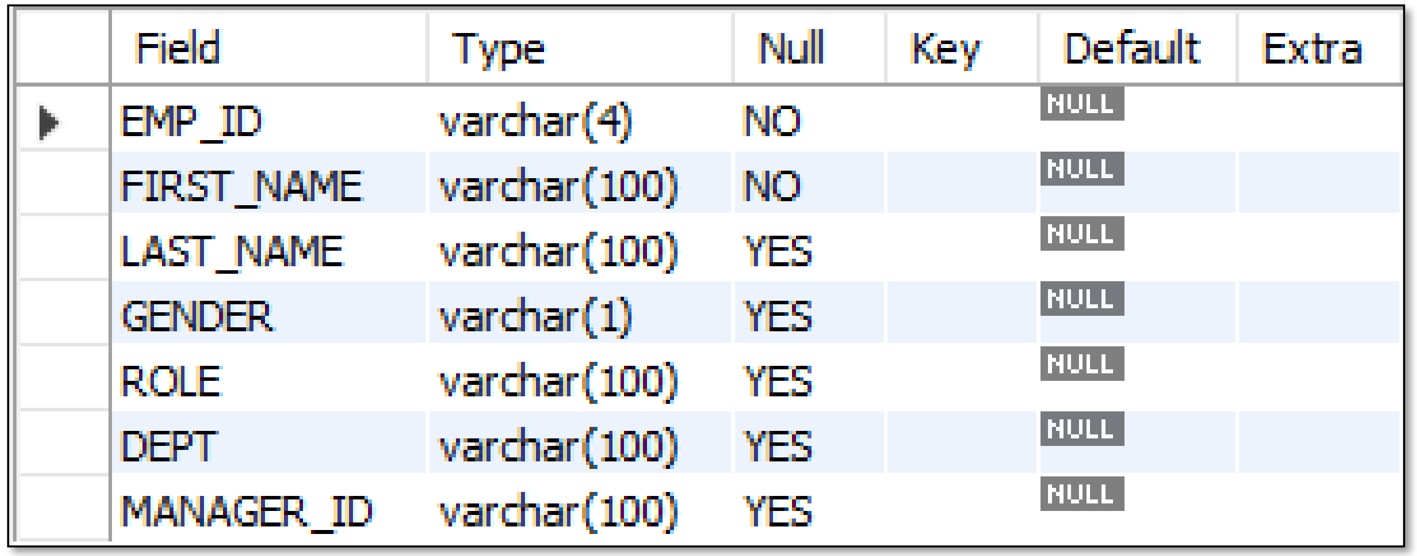
Inserting Data in Tables: Example

Step 2: Use the **DESCRIBE** statement to analyze the structure of the **EMP_TABLE** table as given below.

SQL Query

```
DESCRIBE employees_db.EMP_TABLE;
```

Output:



	Field	Type	Null	Key	Default	Extra
▶	EMP_ID	varchar(4)	NO		NULL	
	FIRST_NAME	varchar(100)	NO		NULL	
	LAST_NAME	varchar(100)	YES		NULL	
	GENDER	varchar(1)	YES		NULL	
	ROLE	varchar(100)	YES		NULL	
	DEPT	varchar(100)	YES		NULL	
	MANAGER_ID	varchar(100)	YES		NULL	

Inserting Data in Tables

- The insert operation can be performed in the following ways:

  1. Inserting the values for a single row in the table
  2. Inserting the values for multiple rows in the table

Inserting Data in Tables

- The syntax for inserting the values for a single row in a table:

Syntax

```
INSERT INTO table(c1,c2,...,cn)
VALUES (v1,v2,...,vn);
```

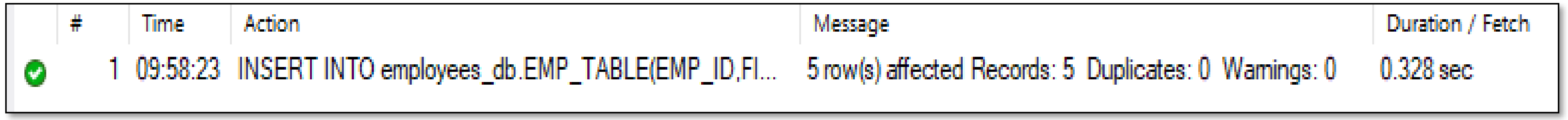
Inserting Data in Tables

Step 3: Use the **INSERT TABLE** statement with **VALUES** clause to insert the required data only for the first row into the **EMP_RECORDS** table as shown below.

SQL Query

```
INSERT INTO
employees_db.EMP_TABLE(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT,MANAGER_ID)

VALUES ("E260", "Roy", "Collins", "M", "SENIOR DATA SCIENTIST", "RETAIL", "E583");
```

Output:



```
     #  Time      Action                                   Message                                                    Duration / Fetch
✓    1  09:58:23  INSERT INTO employees_db.EMP_TABLE(EMP_ID,FI...  5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0  0.328 sec
```

Inserting Data in Tables

- Syntax for inserting the values for multiple rows in a table:

Syntax

```
INSERT INTO table(c1,c2,...,cn)
VALUES
 (v1,v2,...,vn),
 (v1,v2,...,vn),
 ...
 (v1,v2,...,vn);
```

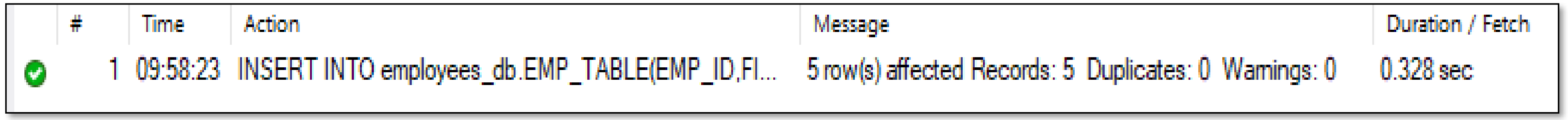
Inserting Data in Tables

Step 4: Use the **INSERT TABLE** statement with **VALUES** clause to insert the required data for multiple rows into the **EMP_RECORDS** table as shown below.

SQL Query

```
INSERT INTO
employees_db.EMP_TABLE(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT,MANAGER_ID)

VALUES

 ("E620", "Katrina", "Allen", "F", "JUNIOR DATA SCIENTIST", "RETAIL", "E612"),

 ("E583", "Janet", "Hale", "F", "MANAGER", "RETAIL", "E002"),

 ("E612", "Tracy", "Norris", "F", "MANAGER", "RETAIL", "E002"),

 ("E002", "Cynthia", "Brooks", "F", "PRESIDENT", "ALL", "E001");
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	09:58:23	INSERT INTO employees_db.EMP_TABLE(EMP_ID,FI...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.328 sec

Querying Table Data

Querying Data From Tables

- To query data from one or more tables, use the **SELECT** statement.
- Syntax:

Syntax

```
SELECT select_list
FROM table_name;
```

Querying Data From Tables

- Parameters:

Keywords	Meaning
select_list	Unique names of one or more target columns separated by a comma, to view their data
table_name	A unique name of the target table having the required columns

Querying Data From Tables: Example

Sample 1: Selecting data from specific columns in MySQL

Problem Statement: Your manager expects you to identify the employee ID, first name, and last name of all employees working in the organization.

Objective: Write an SQL query using the **SELECT** statement to fetch the details of the required columns from the **EMP_TABLE** table.

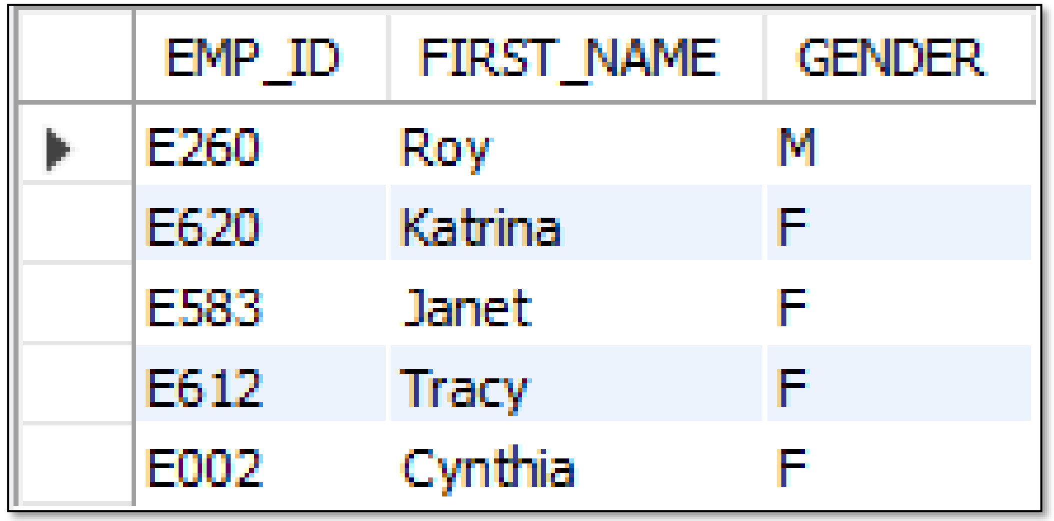
Querying Data From Tables: Example

Step 1: Use the **SELECT** statement to fetch the values for the **EMP_ID**, **FIRST_NAME**, and **GENDER** columns from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT EMP_ID, FIRST_NAME, GENDER

FROM employees_db.EMP_DATA;
```

Output:



	EMP_ID	FIRST_NAME	GENDER
▶	E260	Roy	M
	E620	Katrina	F
	E583	Janet	F
	E612	Tracy	F
	E002	Cynthia	F

Querying Data From Tables: Example

Sample 2: Selecting data from all columns in MySQL

Problem Statement: Your manager expects you to find all the details of each employee in the organization.

Objective: Write an SQL query using the **SELECT** statement with ***** condition to fetch the details of all the columns from the **EMP_TABLE** table.

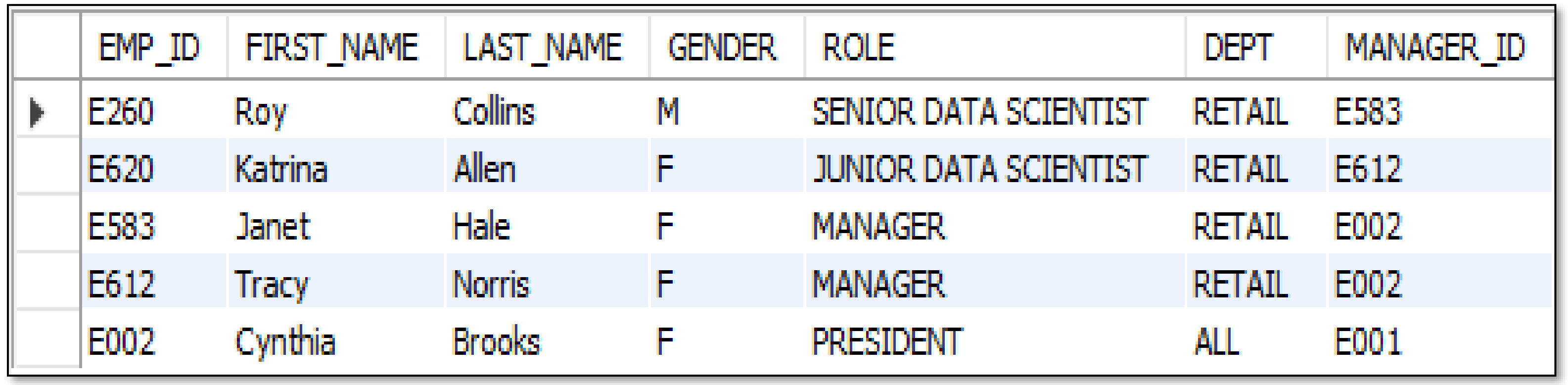
Querying Data From Tables: Example

Step 1: Use the **SELECT** statement with the ***** condition to fetch the values of all the columns from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT *

FROM employees_db.EMP_DATA;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
▶	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583
	E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612
	E583	Janet	Hale	F	MANAGER	RETAIL	E002
	E612	Tracy	Norris	F	MANAGER	RETAIL	E002
	E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001

Filtering Data From Tables

Filtering Data From Tables in MySQL

Following are the most important SQL commands for filtering data from tables:

1. WHERE
2. SELECT DISTINCT
3. AND
4. OR
5. NOT
6. IN
7. NOT IN
8. BETWEEN
9. LIKE
10. LIMIT
11. IS NULL

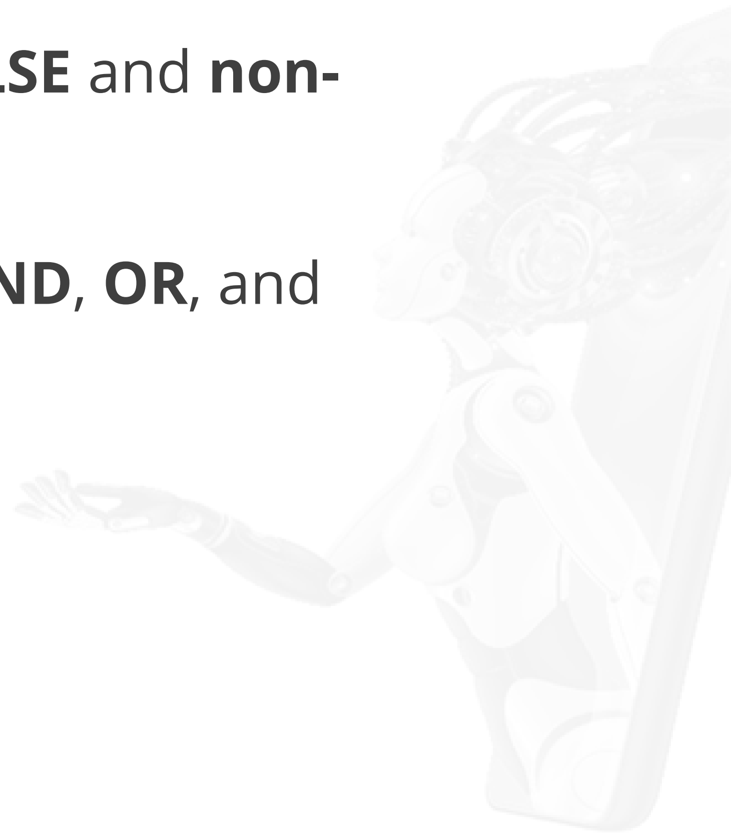
WHERE Clause

WHERE Clause in SELECT Statement

- The **WHERE** clause in the **SELECT** statement allows us to filter rows returned by a query by specifying a search condition.

- There is no built-in Boolean type in MySQL. Instead, it considers **zero** to be **FALSE** and **non-zero** numbers to be **TRUE**.

- The search condition in a **WHERE** clause is a logical expression that uses the **AND**, **OR**, and **NOT** logical operators to combine one or more expressions.



WHERE Clause in SELECT Statement

- Syntax:

Syntax

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

WHERE Clause in SELECT Statement: Example

Problem Statement: Your manager expects you to find all the details of all the managers in the organization.

Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause to fetch the details of all the columns where **ROLE** is set to **MANAGER** from the **EMP_TABLE** table.

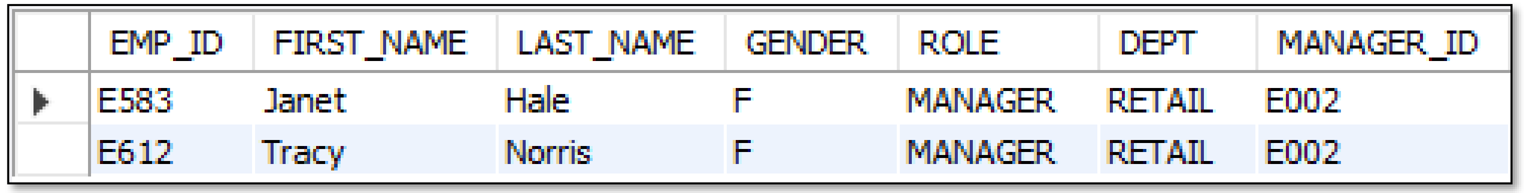
WHERE Clause in SELECT Statement: Example

Step 1: Use the **SELECT** statement with the **WHERE** clause to fetch the values of all the columns where **ROLE** is set to **MANAGER** from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE
WHERE ROLE = "MANAGER";
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
▶	E583	Janet	Hale	F	MANAGER	RETAIL	E002
	E612	Tracy	Norris	F	MANAGER	RETAIL	E002

DISTINCT Clause

DISTINCT Clause in SELECT Statement

- The **DISTINCT** clause in the **SELECT** statement is used to eliminate duplicate rows from the result set that may be produced while querying data from a table.

- When using the **DISTINCT** clause to provide a column with **NULL** values, the **DISTINCT** clause will only preserve one **NULL** value since it considers all **NULL** values to be the same.

DISTINCT Clause in SELECT Statement

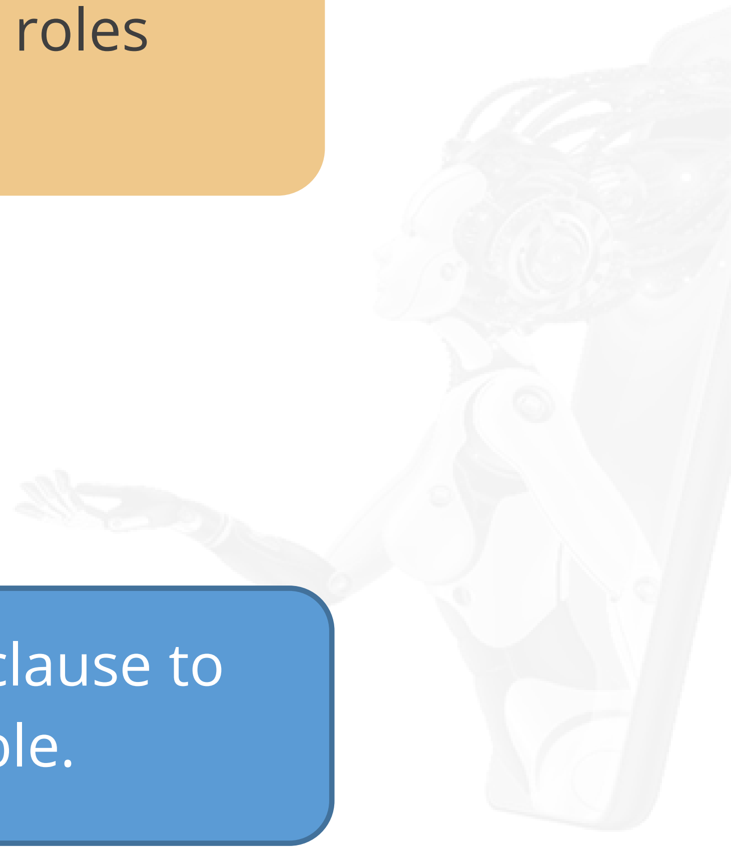
- Syntax:

Syntax

```
SELECT DISTINCT
    select_list
FROM
    table_name
WHERE
    search_condition
ORDER BY
    sort_expression;
```

DISTINCT Clause in SELECT Statement: Example

Problem Statement: Your manager wants you to find all unique types of job roles available in the employee table.



Objective: Write an SQL query using the **SELECT** statement with the **DISTINCT** clause to fetch only the unique values of the **ROLE** column from the **EMP_TABLE** table.

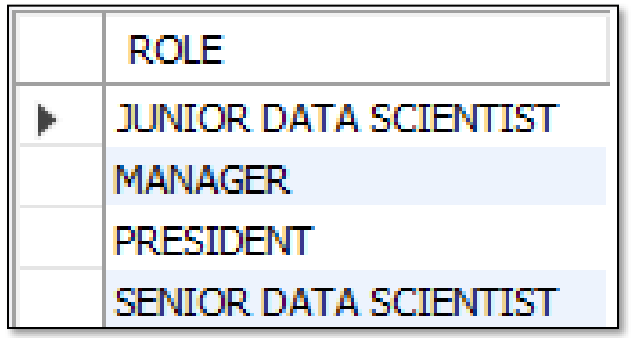
DISTINCT Clause in SELECT Statement: Example

Step 1: Use the **SELECT** statement with the **DISTINCT** clause to get all the unique values of the **ROLE** column from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT DISTINCT ROLE

FROM employees_db.EMP_TABLE

ORDER BY ROLE;
```

Output:



	ROLE
▶	JUNIOR DATA SCIENTIST
	MANAGER
	PRESIDENT
	SENIOR DATA SCIENTIST

AND Operator

AND Operator in MySQL

- The **AND** operator is a logical operator that combines two or more Boolean expressions and returns **1**, **0**, or **NULL**.
- The **AND** operator is used for filtering data with the **WHERE** clause.

AND Operator in MySQL

- Syntax:

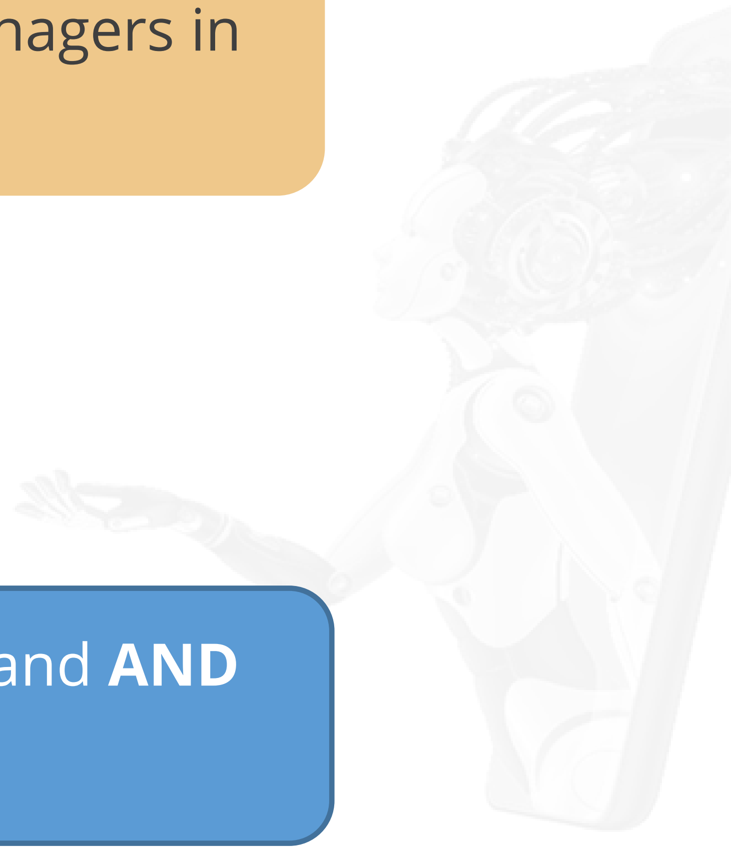
Syntax

```
SELECT select_list
FROM table_name
WHERE
    search_condition_1
AND
   search_condition_2;
```

AND Operator in MySQL: Example

Problem Statement: Your manager expects you to find the details of all the managers in the retail department.

Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **AND** operator to find the required data from the **EMP_TABLE** table.



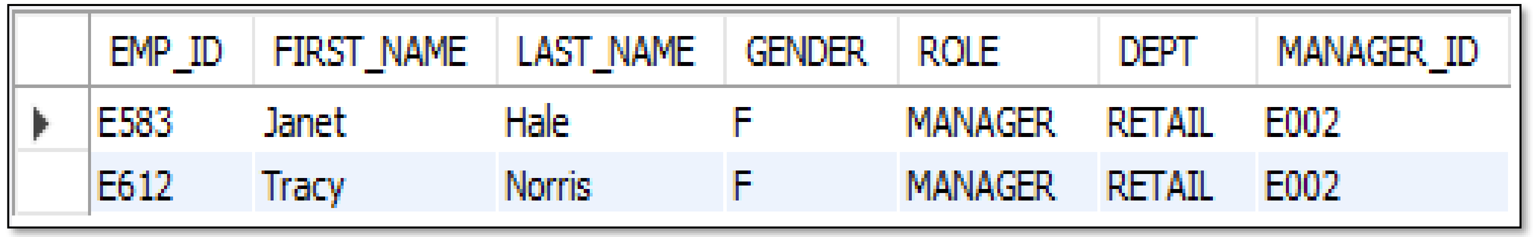
AND Operator in MySQL: Example

Step 1: Use the **SELECT** statement with the **WHERE** clause and **AND** operator to get the values of all the columns where **ROLE** is set to **MANAGER** and **DEPT** is set to **RETAIL** from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE
WHERE ROLE = "MANAGER"
AND DEPT = "RETAIL";
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
▶	E583	Janet	Hale	F	MANAGER	RETAIL	E002
	E612	Tracy	Norris	F	MANAGER	RETAIL	E002

OR Operator

OR Operator in MySQL

- The **OR** operator is a logical operator that combines two Boolean expressions and returns **1 (true)** if either of the expressions is true.
- The **OR** operator is used for filtering data with the **WHERE** clause.

OR Operator in MySQL

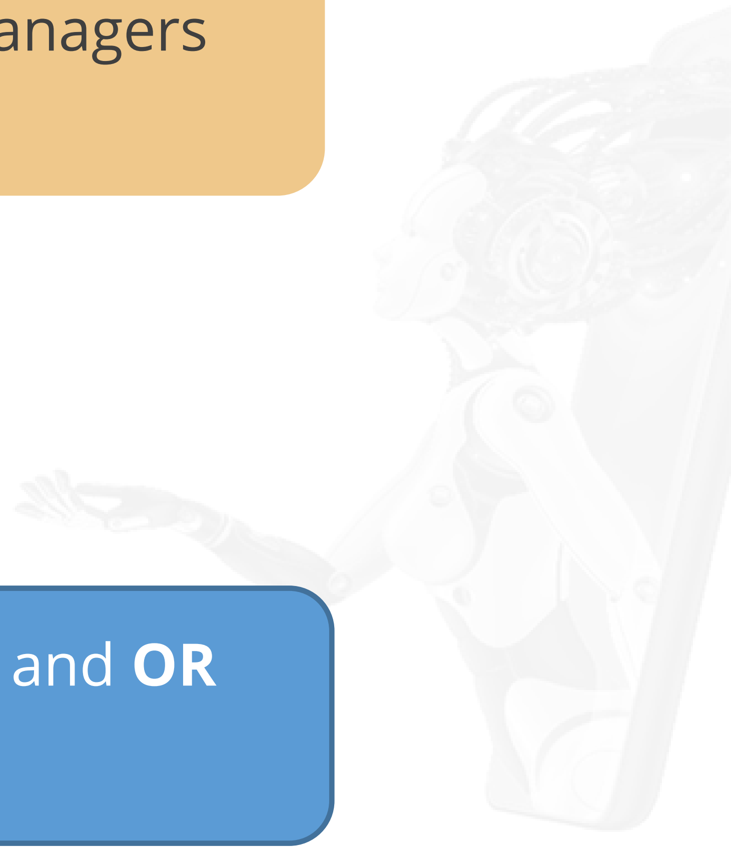
- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE
    search_condition_1
OR
   search_condition_2;
```

OR Operator in MySQL: Example

Problem Statement: Your manager expects you to find the details of all the managers along with the President of the organization.



Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **OR** operator to find the required data from the **EMP_TABLE** table.

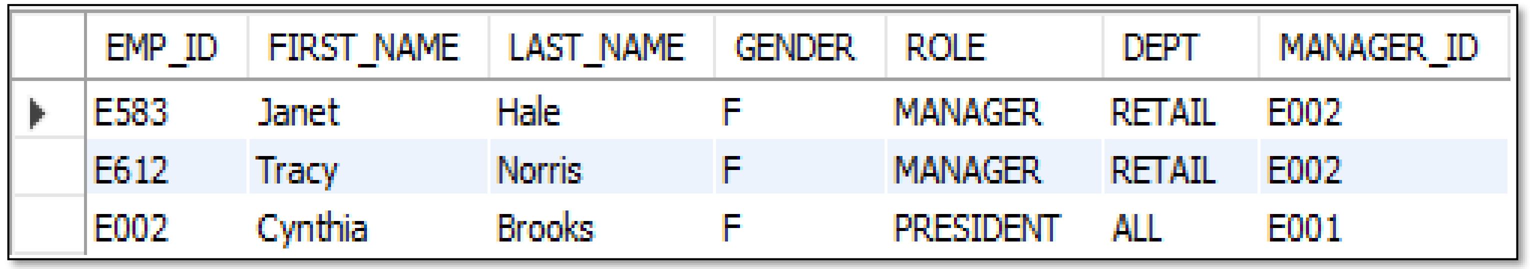
OR Operator in MySQL: Example

Step 1: Use the **SELECT** statement with the **WHERE** clause and **OR** operator to get the values of all the columns where **ROLE** is set to either **MANAGER** or **PRESIDENT** from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

WHERE ROLE = "MANAGER"

OR ROLE = “PRESIDENT";
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
▶	E583	Janet	Hale	F	MANAGER	RETAIL	E002
	E612	Tracy	Norris	F	MANAGER	RETAIL	E002
	E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001

IN Operator

IN Operator in MySQL

- The IN operator is used to determine if a value matches any other value in a list.
- If the value equals any value in the list (value1, value2, value3,...), the IN operator returns **1 (true)**. Otherwise, it returns a value of **0**.

IN Operator in MySQL

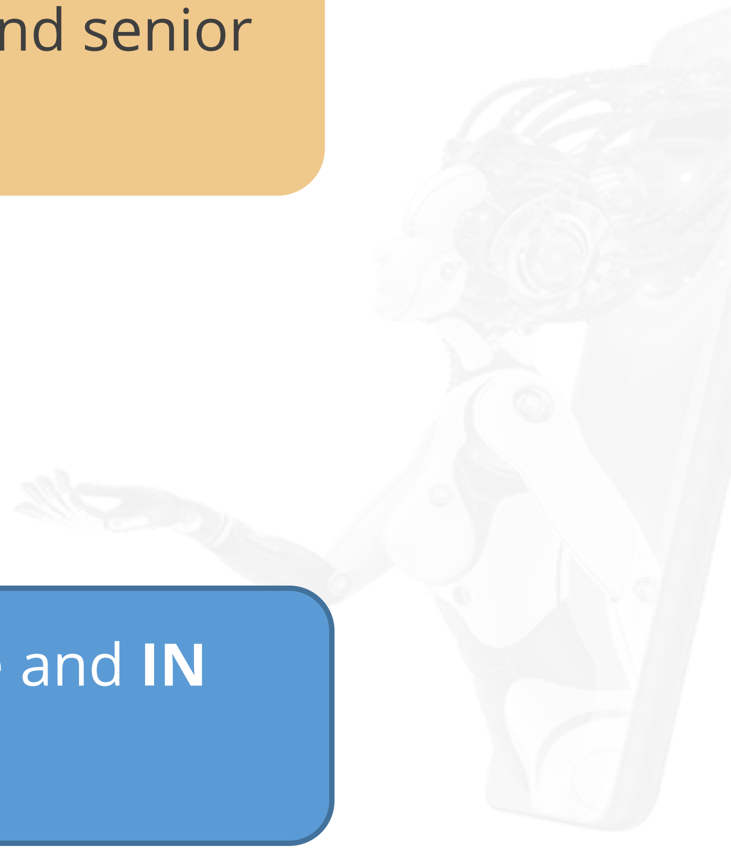
- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE
  value IN (value1, value2, value3,...);
```

IN Operator in MySQL: Example

Problem Statement: Your manager expects you to find the details of all junior and senior data scientist profiles from the employee table.



Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **IN** operator to find the required data from the **EMP_TABLE** table.

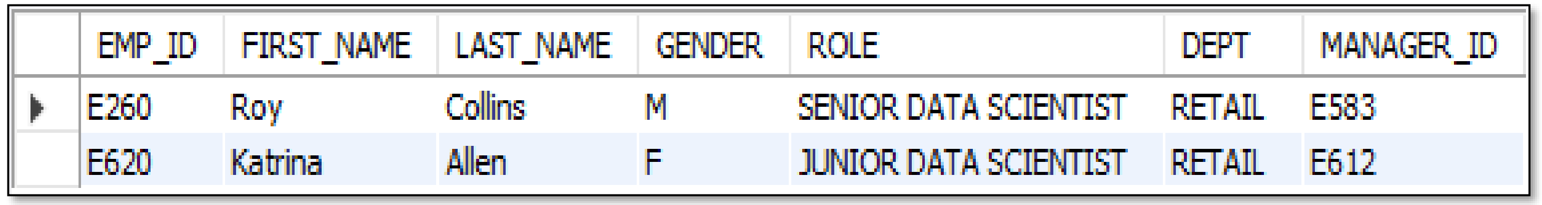
IN Operator in MySQL: Example

Step 1: Use the **SELECT** statement with the **WHERE** clause and **IN** operator to get values of all the columns where **ROLE** is set to the required profiles from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE
WHERE
  ROLE IN ("JUNIOR DATA SCIENTIST", "SENIOR DATA SCIENTIST");
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
▶	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583
	E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612

NOT IN Operator

NOT IN Operator in MySQL

- The **NOT IN** operator is formed by combining the **NOT** and **IN** operators.

- The **NOT IN** operator, in contrast to the IN operator, is used to determine if a value does not match any of the values in a list.

NOT IN Operator in MySQL

- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE
  value NOT IN (value1, value2, value3,...);
```

NOT IN Operator in MySQL: Example

Problem Statement: Your manager expects you to find the details of all the employees except the manager and president from the employee table.

Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **NOT IN** operator to find the required data from the **EMP_TABLE** table.

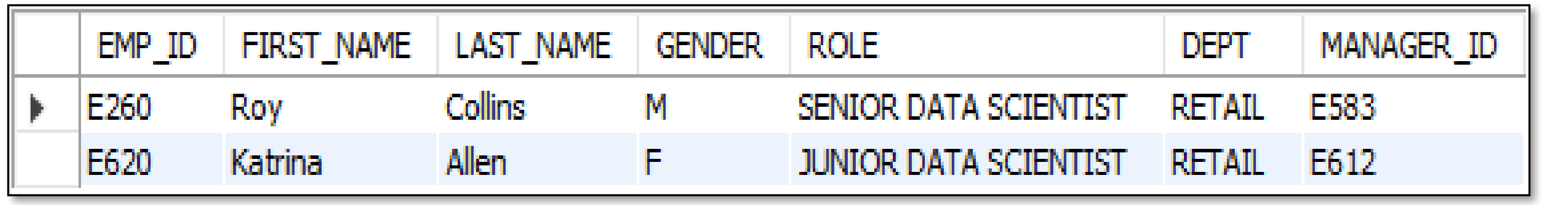
NOT IN Operator in MySQL: Example

Step 1: Use the **SELECT** statement with the **WHERE** clause and **NOT IN** operator to get the values of all the columns where **ROLE** is set to the required profiles from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE
WHERE
  ROLE NOT IN ("MANAGER", "PRESIDENT");
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID
▶	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583
	E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612

Assisted Practice: Filtering

Duration: 20 mins

Problem statement: You have joined Simplilearn as an analyst. Your first task is to answer a few questions based on the quiz organized for learners participating from across the world. Your results would help Simplilearn understand important metrics like the participation rate, winners, and senior citizen participants. The assisted practice questions for this lesson will leverage a quiz dataset. Write a query to print the name and prize choice (shirt_or_hat) of the participants between the ages of 60 and 65.

Assisted Practice: Filtering

Duration: 20 min

Data: This data is about the people who participated in a quiz. It has 1000 rows and 11 columns.

Column Name	Column Description
id_number	Unique identifier
first_name	First name of participant
last_name	Last name of participant
City	City the participant belongs to
state_code	State code of state the participant belongs to
shirt_or_hat	What a participant would want as a prize
quiz_points	Points scored in the quiz
team	Team the participant belongs to
signup	Date of signup
age	Age of participant
company	Company participant works for

Assisted Practice: Filtering

Steps to be performed:

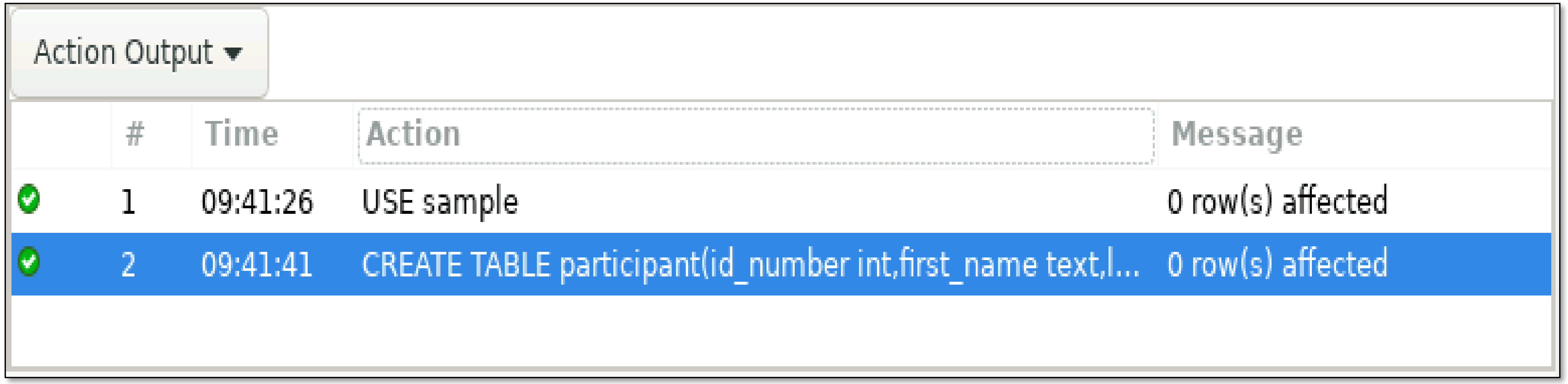
Step 01: Create the table schema using the below query:

CREATE

```
CREATE TABLE participant(
id_number int,first_name text,last_name text,city text,
state_code text,shirt_or_hat text,quiz_points int,team text,
signup date,age int,company text);
```

Assisted Practice: Filtering

Output:



	#	Time	Action	Message
✓	1	09:41:26	USE sample	0 row(s) affected
✓	2	09:41:41	CREATE TABLE participant(id_number int,first_name text,l...	0 row(s) affected

Assisted Practice: Filtering

Step 02: Load the **participant.csv** file following the steps listed in the lab guide

Step 03: Use the SELECT keyword to display the required column names, i.e., first_name, last_name, and shirt_or_hat

Step 04: Use the FROM keyword to direct to the table having the information, which is *participant* here

Step 05: The WHERE clause is used to filter the records and the BETWEEN operator gives a range of values (extreme being inclusive)

Step 06: Filter the records on the *age* column for the range 60-65

Assisted Practice: Filtering

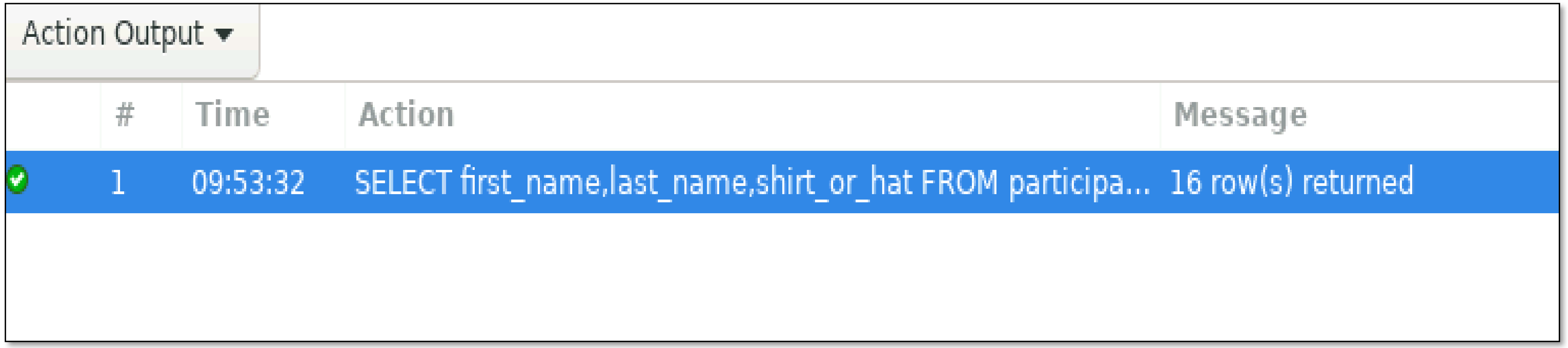
Step 07: Combine the previous steps and create a final query as follows:

```
SELECT first_name, last_name, shirt_or_hat
FROM participant
WHERE age BETWEEN 60 AND 65;
```

Note: The above SQL query should return 16 records with 3 columns.

Assisted Practice: Filtering

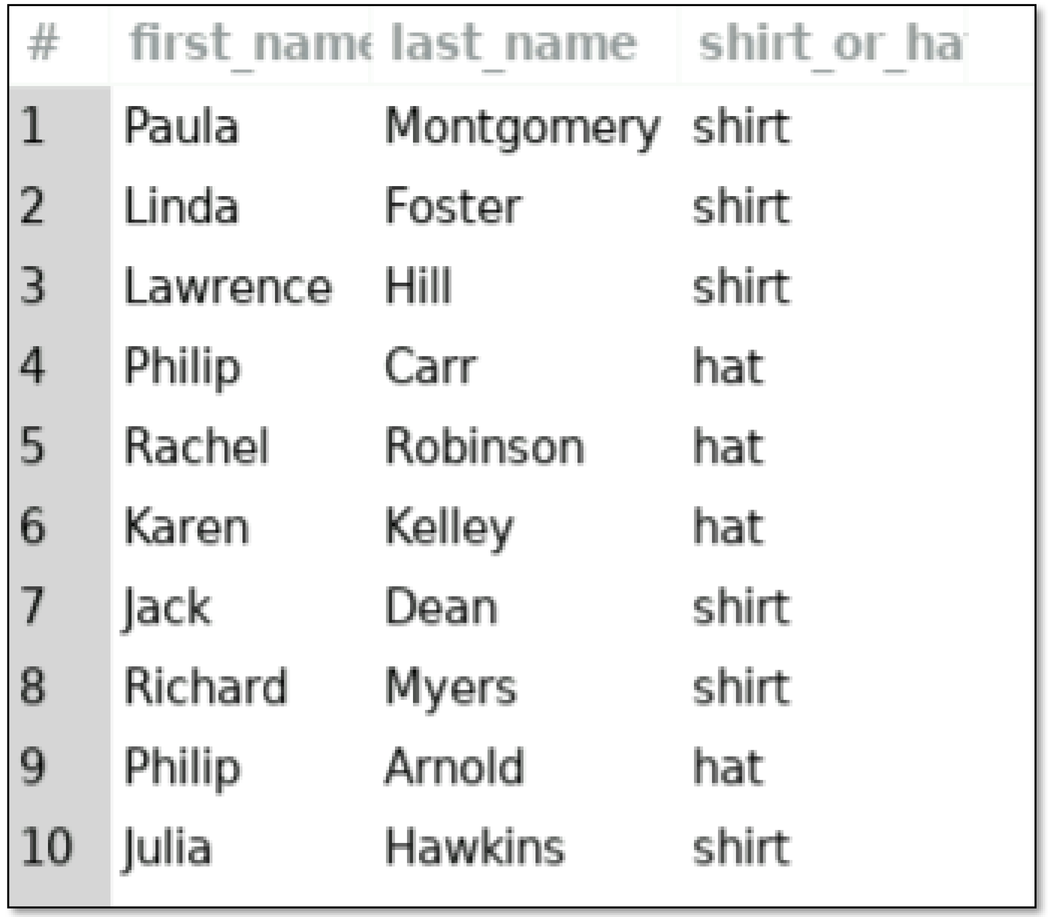
Output:



	#	Time	Action	Message
✓	1	09:53:32	SELECT first_name,last_name,shirt_or_hat FROM participa...	16 row(s) returned

Assisted Practice: Filtering

Output:



#	first_name	last_name	shirt_or_ha
1	Paula	Montgomery	shirt
2	Linda	Foster	shirt
3	Lawrence	Hill	shirt
4	Philip	Carr	hat
5	Rachel	Robinson	hat
6	Karen	Kelley	hat
7	Jack	Dean	shirt
8	Richard	Myers	shirt
9	Philip	Arnold	hat
10	Julia	Hawkins	shirt

Assisted Practice: Aggregation

Duration: 20 mins

Problem statement: You have joined Simplilearn as an analyst. Your first task is to answer a few questions on the quiz organized for learners participating from across the world. Your results would help Simplilearn understand important metrics like the participation rate, winners, and senior citizen participants. The assisted practice questions for this lesson will leverage a quiz dataset. Write a query to display the city name and the average points of the participants enrolled from cities starting with the letter *M*. These participants must have added the name of the company they're working for and should be from the Angry Ants team.

Assisted Practice: Aggregation

Duration: 20 mins

Data: This data is about the people who participated in a quiz. It has 1000 rows and 11 columns.

Column Name	Column Description
id_number	Unique identifier
first_name	First name of participant
last_name	Last name of participant
City	City the participant belongs to
state_code	State code of state the participant belongs to
shirt_or_hat	What a participant would want as a prize
quiz_points	Points scored in the quiz
team	Team the participant belongs to
signup	Date of signup
age	Age of participant
company	Company participant works for

Assisted Practice: Aggregation

Steps to be performed:

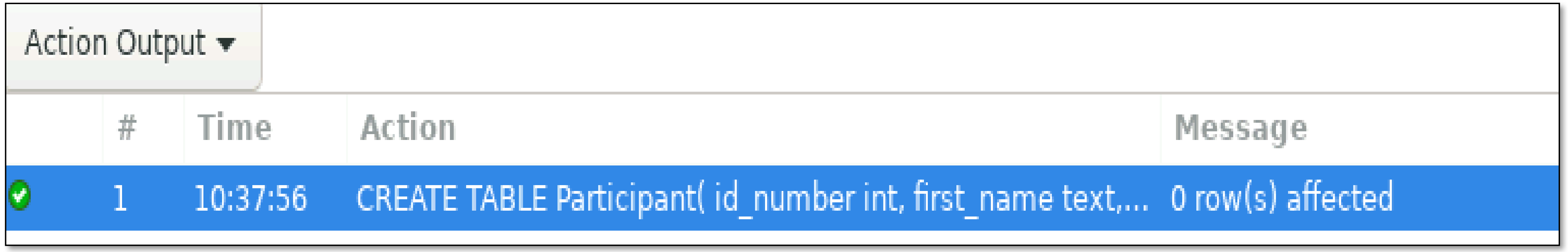
Step 01: Create the table schema using the below query:

CREATE

```
CREATE TABLE participant(
id_number int,first_name text,last_name text,city text,
state_code text,shirt_or_hat text,quiz_points int,team text,
signup date,age int,company text);
```

Assisted Practice: Aggregation

Output:



	#	Time	Action	Message
✓	1	10:37:56	CREATE TABLE Participant(id_number int, first_name text,...	0 row(s) affected

Assisted Practice: Aggregation

Step 02: Load the **participant.csv** file by following the steps listed in the lab guide

Step 03: Use the SELECT keyword to display the names of the required columns, i.e., **city** and **AVG(quiz points)**

Step 04: Use the FROM keyword to navigate to the table containing the data, which is a participant in this case

Step 05: Use the WHERE clause to filter records, the AND operator to apply multiple conditions, the LIKE operator to match a pattern, and the IS NOT NULL condition to return records with the values populated as follows:
WHERE (team="Angry Ants") AND (city LIKE "M%") AND (company IS NOT NULL)

Step 06: Use the GROUP BY keyword to perform aggregation on the nonaggregating columns present with the SELECT statement, i.e., ***city*** in this case

Assisted Practice: Aggregation

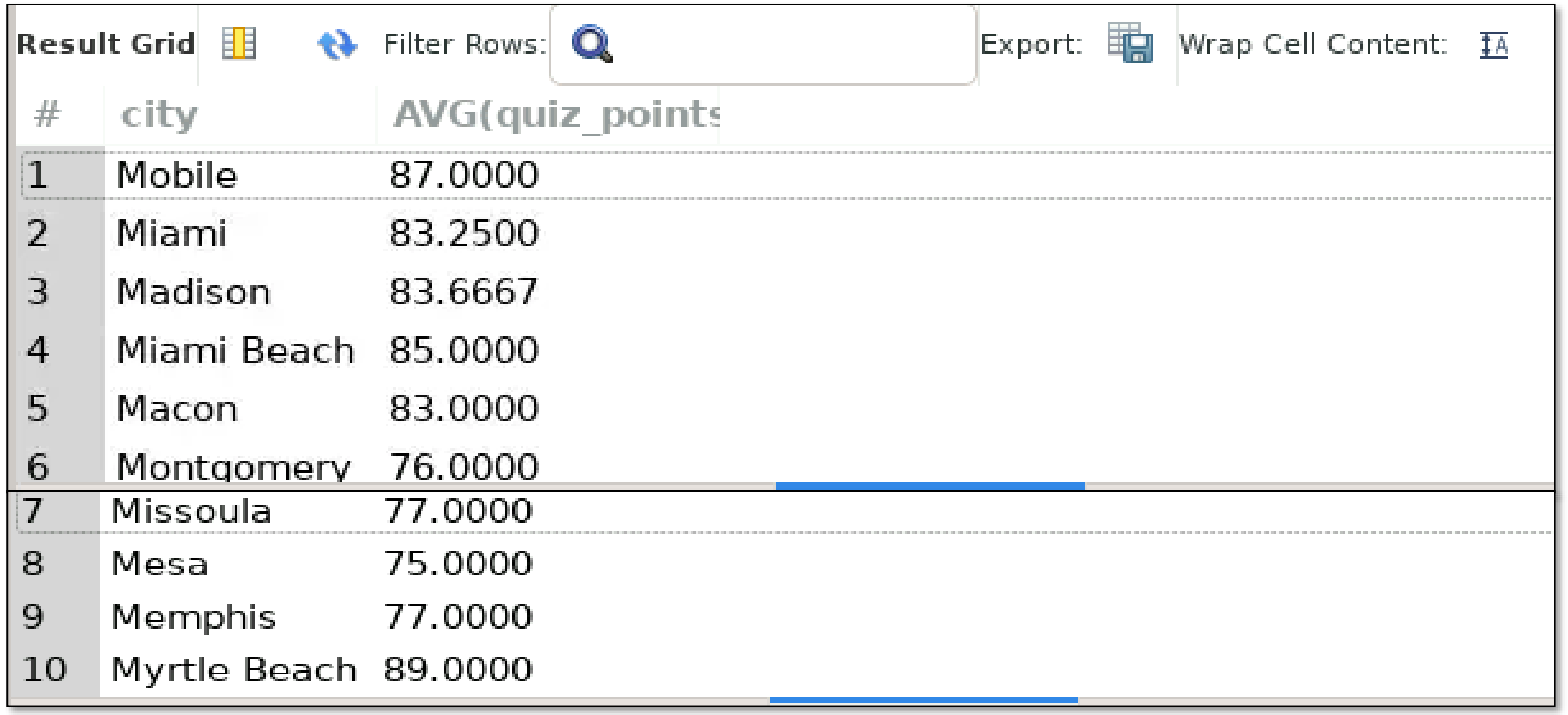
Step 07: Combine the above steps and create a final query as follows:

```
SELECT city, AVG(quiz_points)
FROM participant
WHERE (team="Angry Ants") AND (city LIKE "M%") AND (company IS NOT NULL)
GROUP BY city;
```

Note: The above SQL query should return 10 records with 2 columns.

Assisted Practice: Aggregation

Output:



#	city	AVG(quiz_points
1	Mobile	87.0000
2	Miami	83.2500
3	Madison	83.6667
4	Miami Beach	85.0000
5	Macon	83.0000
6	Montgomery	76.0000
7	Missoula	77.0000
8	Mesa	75.0000
9	Memphis	77.0000
10	Myrtle Beach	89.0000

BETWEEN Operator

BETWEEN Operator in MySQL

- The **BETWEEN** operator is a logical operator that determines whether a value belongs in a range.

- If the following condition is true, the **BETWEEN** operator returns **1**:
  value >= low AND value <= high

- Otherwise, it returns **0**.

- The **BETWEEN** operator returns **NULL** if the **value, low, or high is NULL**.

BETWEEN Operator in MySQL

- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE
  value BETWEEN low AND high;
```

BETWEEN Operator in MySQL: Example

Problem Statement: Your manager expects you to add the experience of each employee in the employee table and then find those having 7 to 14 years of experience.

Objective: Write an SQL query using the **SELECT** statement with **BETWEEN** clause to find the relevant data within a range of values from the **EMP_TABLE** table.

Note: Before performing this task, you must write a sequence of SQL queries to remove all records from the EMP_TABLE and alter its structure. Add the column needed and populate the table with new values.

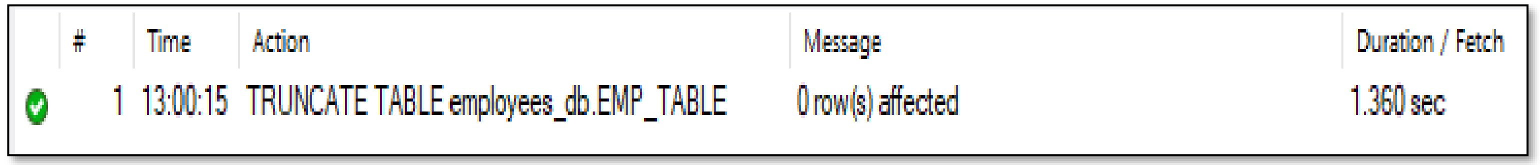
BETWEEN Operator in MySQL: Example

Step 1: Use the **TRUNCATE TABLE** statement to delete all records from the **EMP_TABLE** as given below.

SQL Query

```
TRUNCATE TABLE employees_db.EMP_TABLE;
```

Output:



```
   #  Time      Action                                  Message              Duration / Fetch
✓  1  13:00:15  TRUNCATE TABLE employees_db.EMP_TABLE   0 row(s) affected    1.360 sec
```

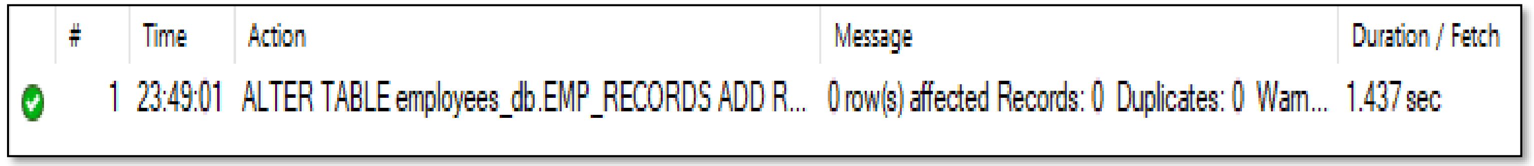
BETWEEN Operator in MySQL: Example

Step 2: Use the **ALTER TABLE** statement with the **ADD** clause to add an **EXP** column to the **EMP_TABLE** table as given below.

SQL Query

```
ALTER TABLE employees_db.EMP_TABLE

ADD EXP INTEGER;
```

Output:



	#	Time	Action	Message	Duration / Fetch
✓	1	23:49:01	ALTER TABLE employees_db.EMP_RECORDS ADD R...	0 row(s) affected Records: 0 Duplicates: 0 Warn...	1.437 sec

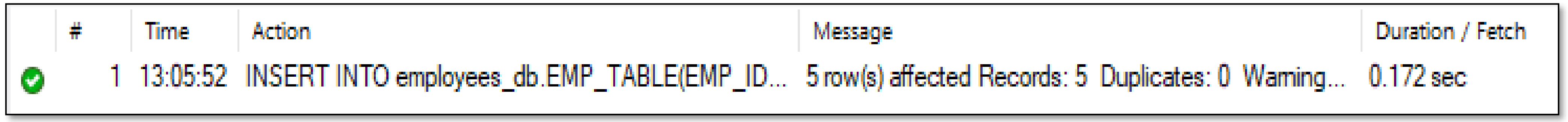
BETWEEN Operator in MySQL: Example

Step 3: Use the **INSERT TABLE** statement with **VALUES** clause to insert the required data for multiple rows into the **EMP_RECORDS** table as shown below.

SQL Query

```
INSERT INTO
employees_db.EMP_TABLE(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT,MANAGER_ID,EXP)

VALUES

 ("E260", "Roy", "Collins", "M", "SENIOR DATA SCIENTIST", "RETAIL", "E583", 7),

 ("E620", "Katrina", "Allen", "F", "JUNIOR DATA SCIENTIST", "RETAIL", "E612", 2),

 ("E583", "Janet", "Hale", "F", "MANAGER", "RETAIL", "E002", 14),

 ("E612", "Tracy", "Norris", "F", "MANAGER", "RETAIL", "E002", 13),

 ("E002", "Cynthia", "Brooks", "F", "PRESIDENT", "ALL", "E001", 17);
```

Output:



```
    #   Time      Action                                     Message                                                Duration / Fetch
✓   1   13:05:52  INSERT INTO employees_db.EMP_TABLE(EMP_ID...  5 row(s) affected Records: 5  Duplicates: 0  Warning...  0.172 sec
```

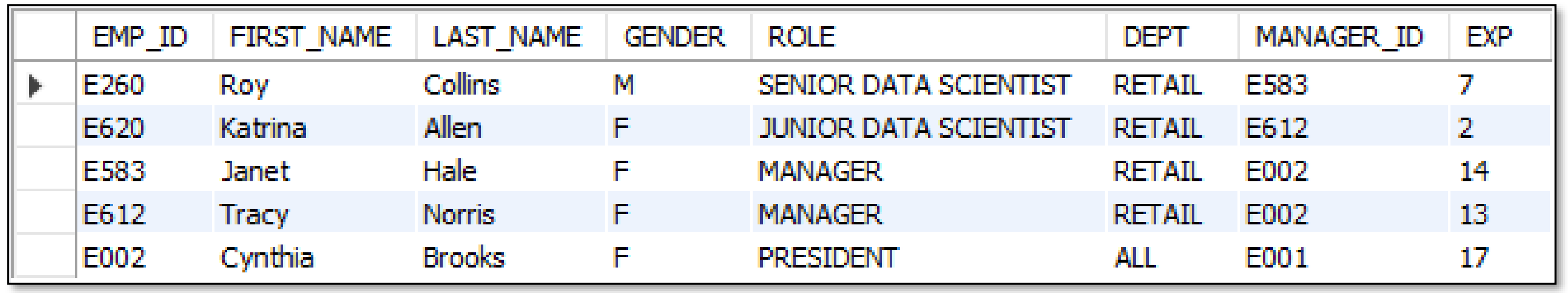
BETWEEN Operator in MySQL: Example

Step 4: Use the **SELECT** statement with the ***** condition to read all the data from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
▶	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583	7
	E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612	2
	E583	Janet	Hale	F	MANAGER	RETAIL	E002	14
	E612	Tracy	Norris	F	MANAGER	RETAIL	E002	13
	E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001	17

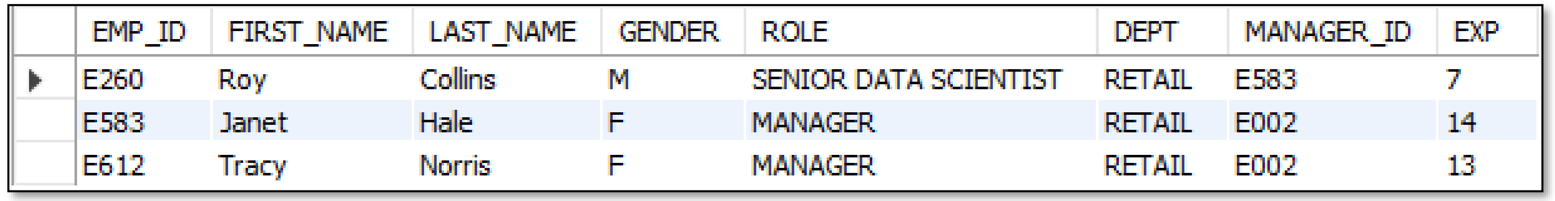
BETWEEN Operator in MySQL: Example

Step 5: Use the **SELECT** statement with the **WHERE** clause and **BETWEEN** operator to get values of all columns where **EXP** is between 7 to 14 from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

WHERE EXP BETWEEN 7 AND 14;
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583	7
E583	Janet	Hale	F	MANAGER	RETAIL	E002	14
E612	Tracy	Norris	F	MANAGER	RETAIL	E002	13

LIKE Operator and Wildcards

LIKE Operator and Wildcards

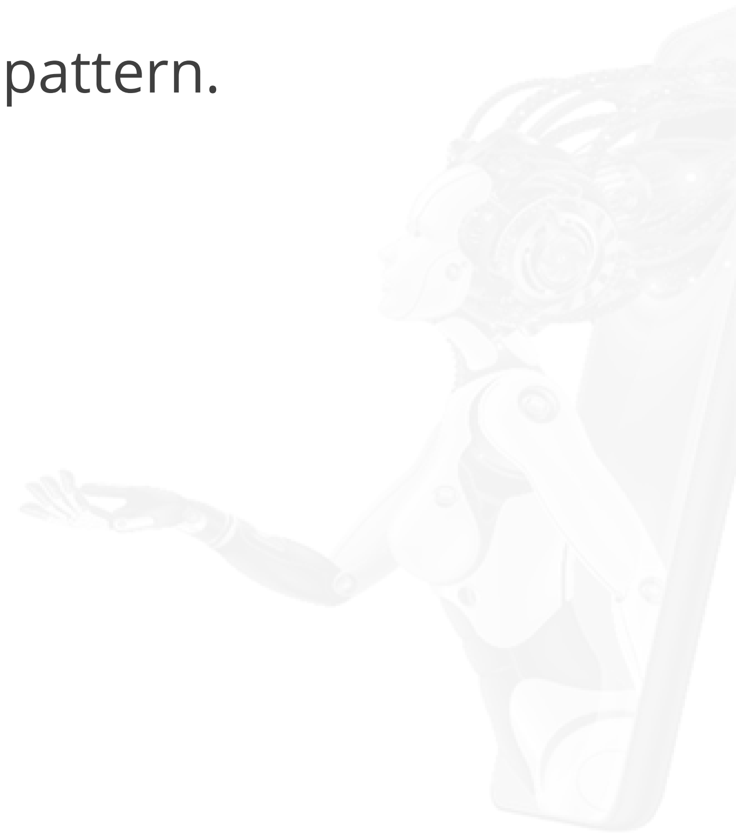
- The **LIKE** operator is a logical operator that determines whether or not a string contains a specified pattern.

- The **LIKE** operator returns **1 (true)** if the expression in the syntax matches the pattern. Otherwise, it returns **0 (false)**.

- For creating patterns, MySQL supports two wildcard characters:

  1. **Percentage (%):** It matches any string of zero or more characters

  1. **Underscore (_):** It matches any single character



LIKE Operator and Wildcards

- When a wildcard character appears in a pattern, the **ESCAPE** clause is used to treat it as a regular character.

- The **LIKE** operator is utilized in the **WHERE** clause of the **SELECT**, **DELETE**, and **UPDATE** statements.

- **Wildcard Example 1:** The **s%** matches any string that starts with the character **s** such as **sun** and **six**.

- **Wildcard Example 2:** The **se_** matches any string that starts with **se** and is followed by any character such as **see** and **sea**.

LIKE Operator and Wildcards

- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE
  expression LIKE pattern ESCAPE escape_character;
```

LIKE Operator and Wildcards: Example

Problem Statement: Your manager expects you to identify all the employees whose last names begin with the letter W.

Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **LIKE** operator to find the relevant data from the **EMP_TABLE** table, satisfying the required condition.

Note: Before performing this task, you must write an SQL query to add more employees in the **EMP_TABLE,** specifically those whose last names begin with **W.**

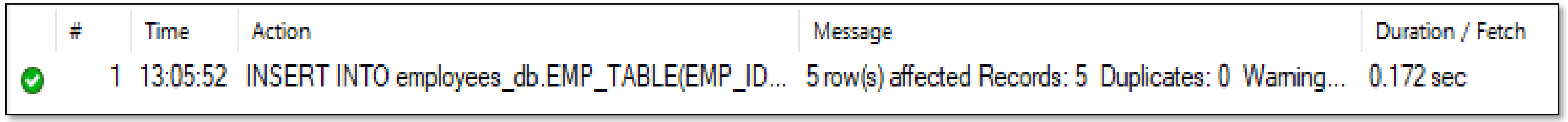
LIKE Operator and Wildcards: Example

Step 1: Use the **INSERT TABLE** statement with **VALUES** clause to insert the required data for multiple rows into the **EMP_TABLE** as shown below.

SQL Query

```
INSERT INTO
employees_db.EMP_TABLE(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT,MANAGER_ID,EXP)
VALUES
("E052", "Dianna", "Wilson", "F", "SENIOR DATA SCIENTIST", "HEALTHCARE", "E083", 6),
("E505", "Chad", "Wilson", "M", "ASSOCIATE DATA SCIENTIST", "HEALTHCARE","E083", 5),
("E083", "Patrick", "Voltz", "M", "MANAGER", "HEALTHCARE", "E002", 15);
```

Output:



```
   #  Time      Action                                      Message                                                    Duration / Fetch
✓  1  13:05:52  INSERT INTO employees_db.EMP_TABLE(EMP_ID...  5 row(s) affected Records: 5  Duplicates: 0  Warning...  0.172 sec
```

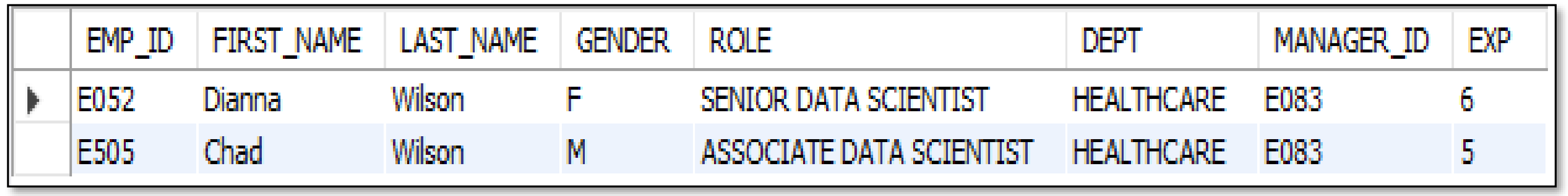
LIKE Operator and Wildcards: Example

Step 2: Use the **SELECT** statement with the **WHERE** clause and **LIKE** operator to get values of all columns where **LAST_NAME** starts with **w** from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE
WHERE LAST_NAME LIKE "w%";
```

Output:



EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	E083	6
E505	Chad	Wilson	M	ASSOCIATE DATA SCIENTIST	HEALTHCARE	E083	5

Assisted Practice: Limiting

Duration: 20 mins

Problem statement: You have joined Simplilearn as an analyst. Your first task is to answer a few questions on the quiz organized for learners participating from across the world. Your results would help Simplilearn understand important metrics like the participation rate, winners, and senior citizen participants. The assisted practice questions for this lesson will leverage the quiz dataset **participant.csv**.

Objective: Write a query to list the name, team, and score of any five participants

Assisted Practice: Limiting

Duration: 20 min

Data: This data is about the people who participated in a quiz. It has 1000 rows and 11 columns.

Column Name	Column Description
id_number	unique identifier
first_name	first name of participant
last_name	last name of participant
City	city, the participant belongs to
state_code	state code, the participant belongs to
shirt_or_hat	participant would want hat or a shirt as a prize
quiz_points	points scored in the quiz
team	team the participant belongs to
signup	date of signup
age	age of participant
company	company of participant

Assisted Practice: Limiting

Steps to be performed:

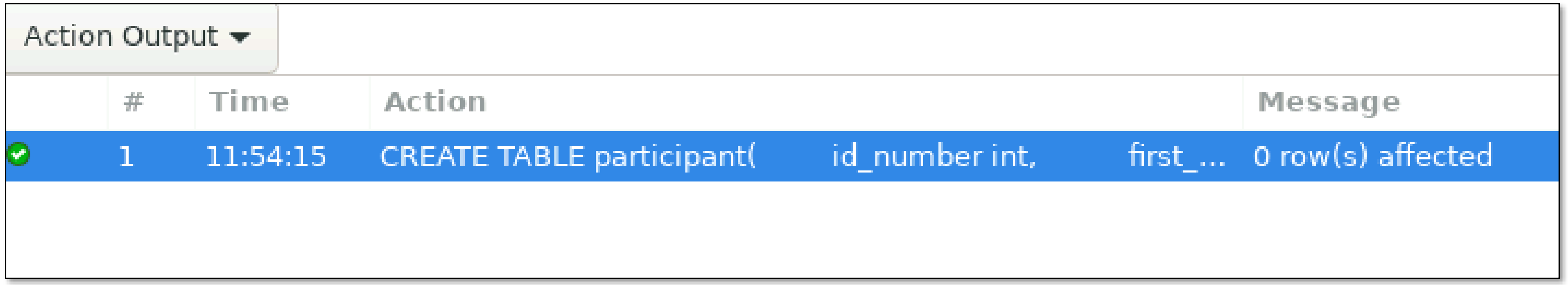
Step 01: Create the table schema using the below query:

Create

```
CREATE TABLE participant(
id_number int,first_name text,last_name text,city text,
state_code text,shirt_or_hat text,quiz_points int,team text,
signup date,age int,company text);
```

Assisted Practice: Limiting

Output:



	#	Time	Action	Message
✓	1	11:54:15	CREATE TABLE participant(id_number int, first_...	0 row(s) affected

Assisted Practice: Limiting

Step 02: Load the **participant.csv** file by following the steps listed in the lab guide

Step 03: Use the SELECT keyword to display the names of the required columns, i.e., first_name, last_name, team, and quiz_points

Step 04: Use the FROM keyword to direct the SELECT request to the table having the information, which is **participant** here

Step 05: Use the LIMIT clause to return the number of records, which is 5 in this case

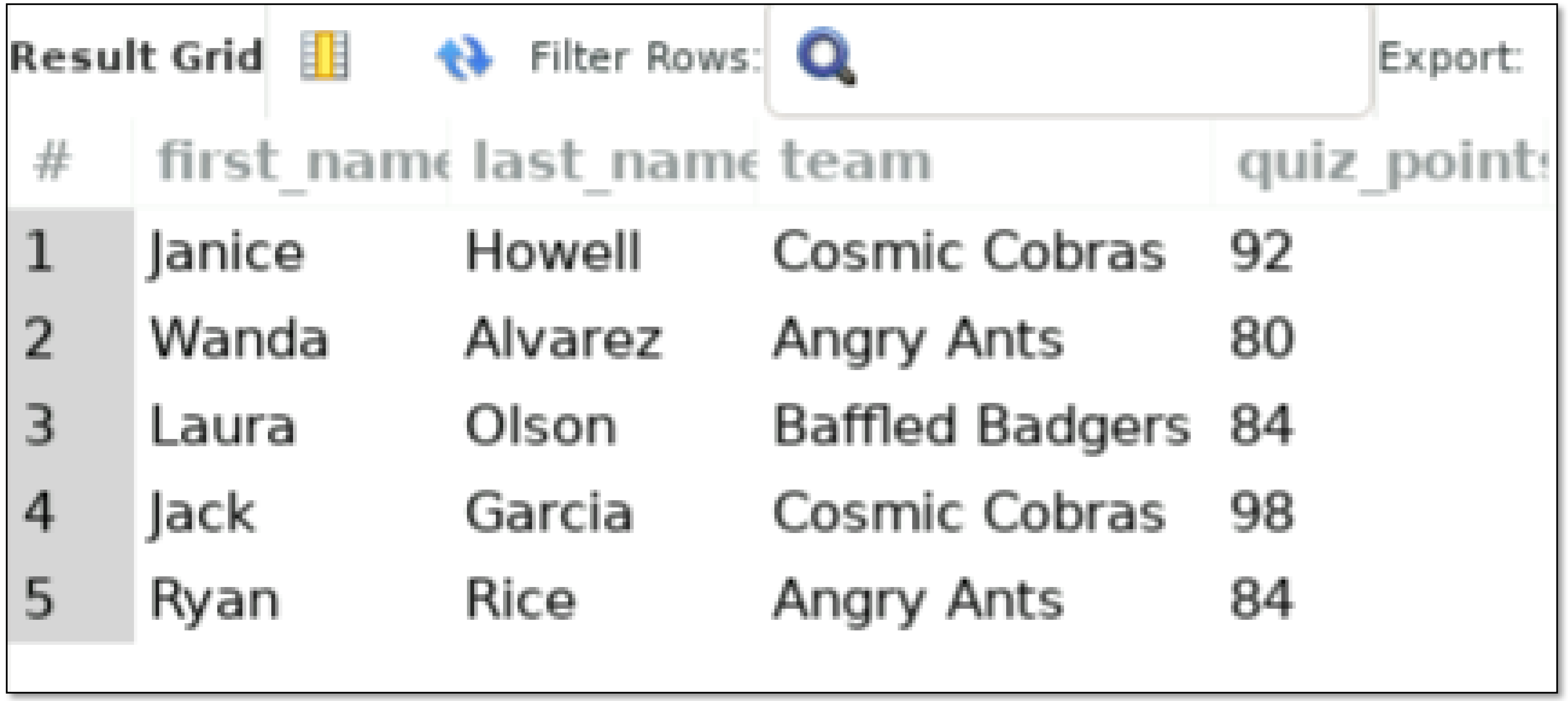
Step 06: Combine the above steps and create a final query as follows:

```
SELECT first_name, last_name, team, quiz_points
FROM participant
LIMIT 5
```

Note: The above query should return five records with four columns

Assisted Practice: Limiting

Output:

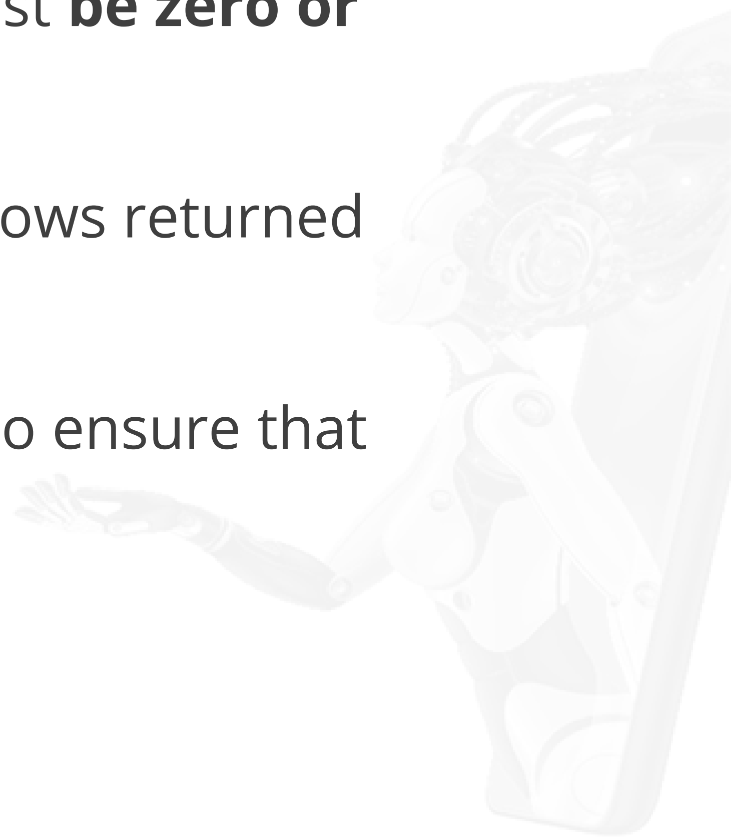


#	first_name	last_name	team	quiz_point
1	Janice	Howell	Cosmic Cobras	92
2	Wanda	Alvarez	Angry Ants	80
3	Laura	Olson	Baffled Badgers	84
4	Jack	Garcia	Cosmic Cobras	98
5	Ryan	Rice	Angry Ants	84

LIMIT Operator

LIMIT Operator in MySQL

- The **LIMIT** clause in the **SELECT** statement is used to limit the number of rows returned.
- **One or two arguments** are accepted by the **LIMIT** clause, and their values must **be zero or positive integers**.
- Since the **SELECT** statement returns rows in an arbitrary order by default, the rows returned are unpredictable when the **LIMIT** clause is added to the **SELECT** statement.
- As a result, the **LIMIT** clause should always be used with an **ORDER BY** clause to ensure that the expected output is produced.



LIMIT Operator in MySQL

- Syntax:

Syntax

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    sort_expression
LIMIT [offset,] row_count;
```

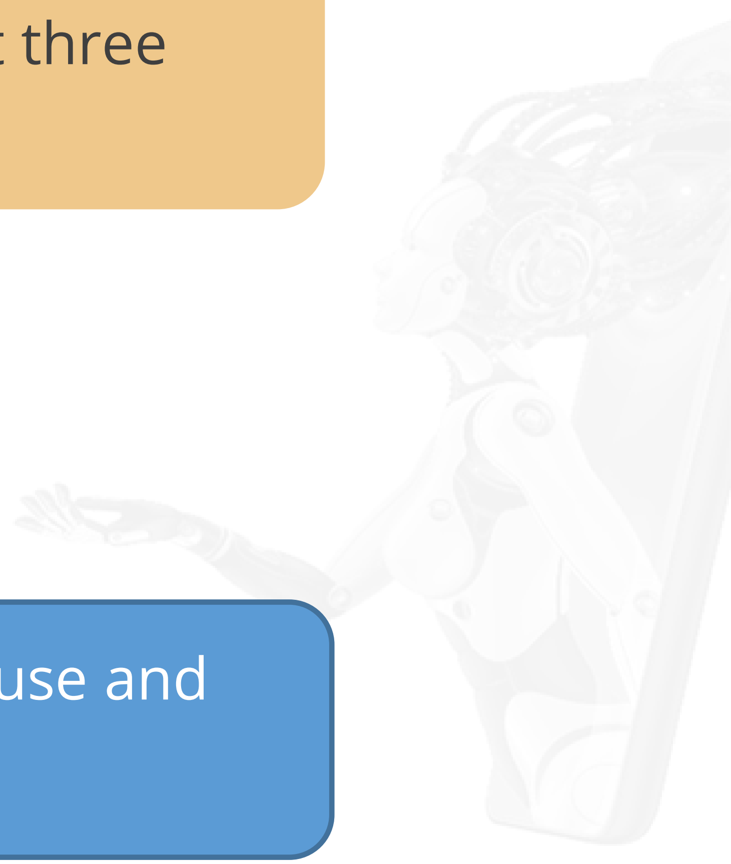
LIMIT Operator in MySQL

- Parameters:

Keywords	**Meaning**
[offset,] **(Optional)**	It specifies the first row's offset to return where the first row's offset is 0 and not 1
row_count	It specifies the maximum number of rows to return

LIMIT Operator in MySQL: Example

Problem Statement: Your manager expects you to find the details of the first three employees with the least experience from the employee table.



Objective: Write an SQL query using the **SELECT** statement with **ORDER BY** clause and **LIMIT** operator to find the required data from the **EMP_TABLE** table.

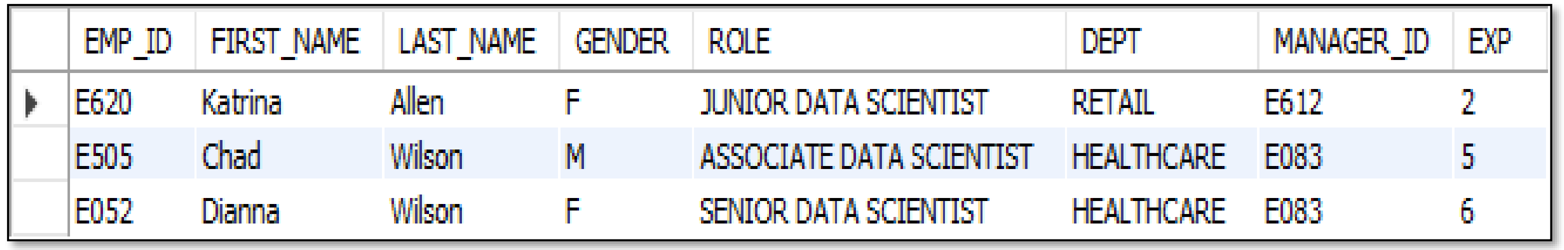
LIMIT Operator in MySQL: Example

Step 1: Use the **SELECT** statement with the **ORDER BY** clause and **LIMIT** operator to get the required values of the rows from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

ORDER BY EXP

LIMIT 3;
```

Output:

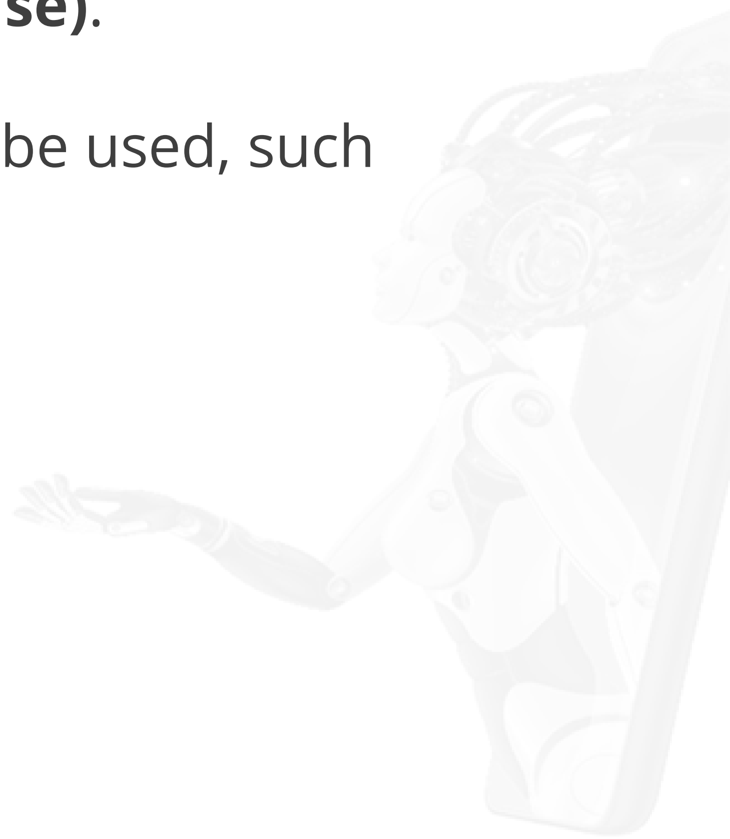


EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612	2
E505	Chad	Wilson	M	ASSOCIATE DATA SCIENTIST	HEALTHCARE	E083	5
E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	E083	6

IS NULL Operator

IS NULL Operator in MySQL

- The IS NULL operator determines whether or not a value is NULL.
- The expression returns **1 (true)** if the value is NULL. Otherwise, it returns **0 (false)**.
- **IS NULL** is a comparison operator that can be used anywhere an operator can be used, such as in a **SELECT** or **WHERE** clause.



IS NULL Operator in MySQL

- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE value IS NULL;
```

IS NULL Operator in MySQL: Example

Problem Statement: Your manager expects you to identify all the employees with no manager assigned in the employee table.

Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **IS NULL** operator to find the relevant data from the **EMP_TABLE** table.

Note: Before performing this task, you must write an SQL query to add one or more employees with no managers.

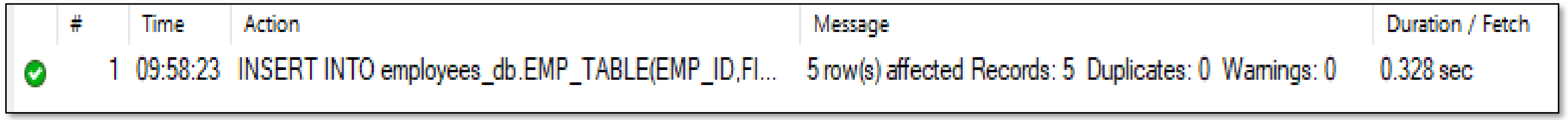
IS NULL Operator in MySQL: Example

Step 1: Use the **INSERT TABLE** statement with **VALUES** clause to insert the required data into the **EMP_RECORDS** table as shown below.

SQL Query

```
INSERT INTO
employees_db.EMP_TABLE(EMP_ID,FIRST_NAME,LAST_NAME,GENDER,ROLE,DEPT)
VALUES
  ("E001", "Arthur", "Black", "M", "CEO", "ALL", ,);
```

Output:



```
      #   Time      Action                                      Message                                                  Duration / Fetch
✓     1   09:58:23  INSERT INTO employees_db.EMP_TABLE(EMP_ID,FI...  5 row(s) affected Records: 5  Duplicates: 0  Warnings: 0   0.328 sec
```

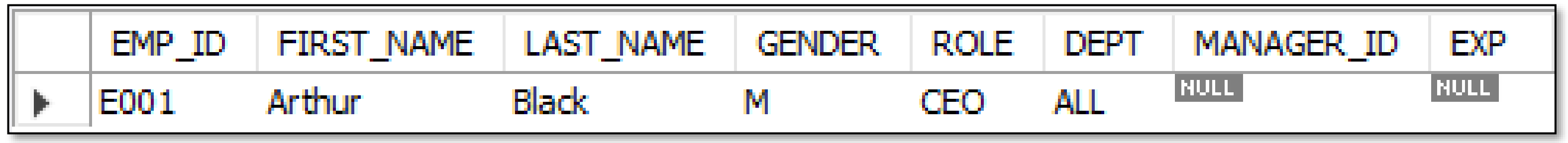
IS NULL Operator in MySQL: Example

Step 2: Use the **SELECT** statement with the **WHERE** clause and **IS NULL** operator to get values of all the columns where **MANAGER_ID** is **NULL** from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

WHERE MANAGER_ID IS NULL;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
▶	E001	Arthur	Black	M	CEO	ALL	NULL	NULL

IS NOT NULL Operator

IS NOT NULL Operator in MySQL

- The **IS NOT NULL** operator is formed by combining the **NOT** and **IS NULL** operators.

- The **IS NOT NULL** operator, in contrast to the **IS NULL** operator, determines if a value is **NOT NULL**.

- The expression, in contrast to the IN operator, returns **1 (true)** if the value is **NOT NULL**. Otherwise, it returns **0 (false)**.

- Like the **IS NULL** operator, the **IS NOT NULL** is also a comparison operator that can be used anywhere an operator can be used, such as in a **SELECT** or **WHERE** clause.

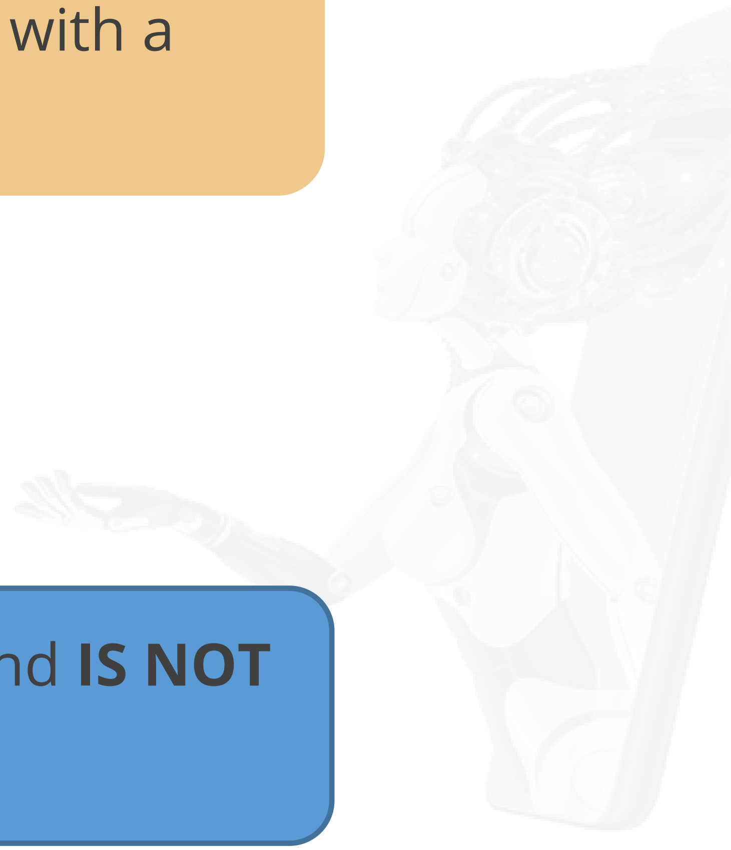
IS NOT NULL Operator in MySQL

- Syntax:

Syntax

```
SELECT select_list
FROM table_name
WHERE value IS NOT NULL;
```

IS NOT NULL Operator in MySQL: Example



Problem Statement: Your manager expects you to identify all the employees with a manager assigned in the employee table.

Objective: Write an SQL query using the **SELECT** statement with **WHERE** clause and **IS NOT NULL** operator to find the relevant data from the **EMP_TABLE** table.

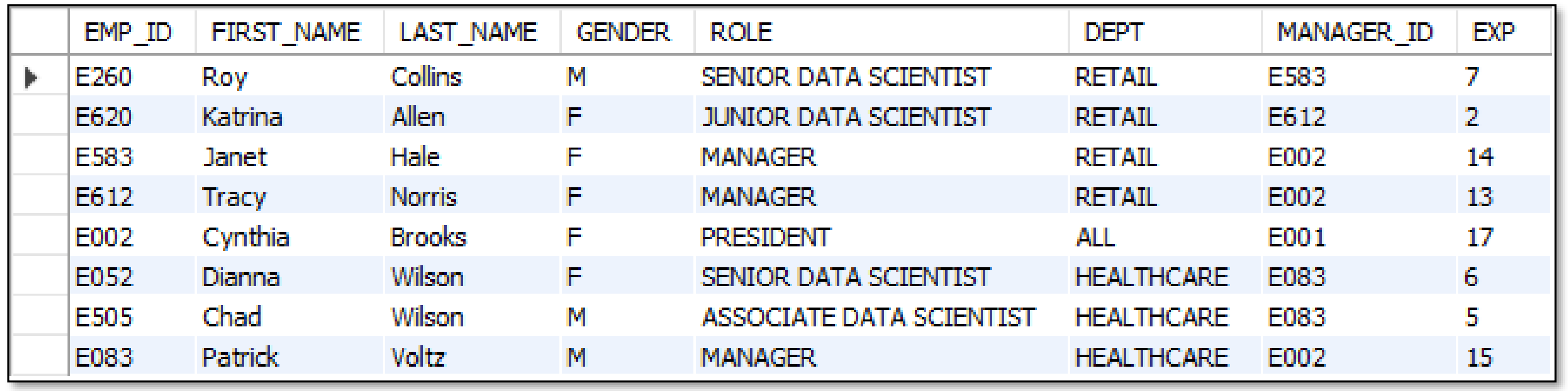
IS NOT NULL Operator in MySQL: Example

Step 1: Use the **SELECT** statement with the **WHERE** clause and **IS NOT NULL** operator to get values of all the columns where **MANAGER_ID** is **NOT NULL** from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

WHERE MANAGER_ID IS NOT NULL;
```

Output:

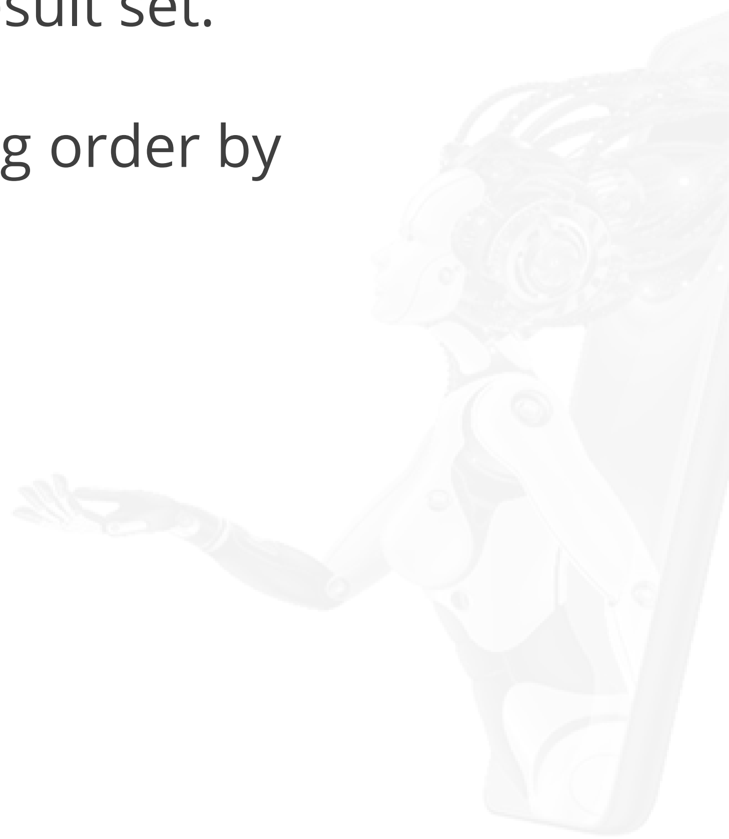


EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583	7
E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612	2
E583	Janet	Hale	F	MANAGER	RETAIL	E002	14
E612	Tracy	Norris	F	MANAGER	RETAIL	E002	13
E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001	17
E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	E083	6
E505	Chad	Wilson	M	ASSOCIATE DATA SCIENTIST	HEALTHCARE	E083	5
E083	Patrick	Voltz	M	MANAGER	HEALTHCARE	E002	15

Sorting Table Data

ORDER BY Clause in MySQL

- The order of rows in the result set obtained from the SELECT statement is unspecified.
- The ORDER BY clause is added to the SELECT statement to sort the rows in the result set.
- The data in the result set produced by the ORDER BY clause is sorted in ascending order by default.



ORDER BY Clause in MySQL

- Syntax:

Syntax

```
SELECT
   select_list
FROM
   table_name
ORDER BY
   column1 [ASC|DESC],
   column2 [ASC|DESC],
   ...;
```

ORDER BY Clause in MySQL

- Parameters:

Keywords	Meaning
ASC	It specifies the to sort the result set in ascending order and is the default value set for sorting.
DESC	It specifies the to sort the result set in descending order.

ORDER BY Clause in MySQL: Example

Problem Statement: Your manager expects you to provide details of all employees with their experience in a descending order.

Objective: Write an SQL query using the **SELECT** statement with **ORDER BY** clause to get the relevant data from the **EMP_TABLE** table.

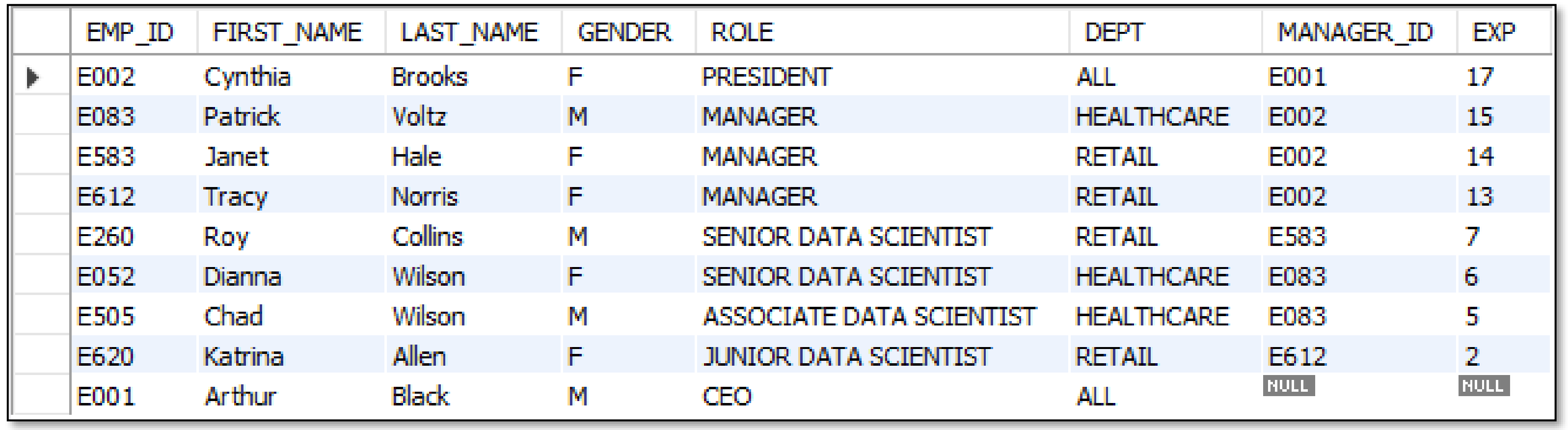
ORDER BY Clause in MySQL: Example

Step 1: Use the **SELECT** statement with the **ORDER BY** clause to get values of all the columns ordered in the required format from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE
ORDER BY EXP DESC;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
▶	E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001	17
	E083	Patrick	Voltz	M	MANAGER	HEALTHCARE	E002	15
	E583	Janet	Hale	F	MANAGER	RETAIL	E002	14
	E612	Tracy	Norris	F	MANAGER	RETAIL	E002	13
	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583	7
	E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	E083	6
	E505	Chad	Wilson	M	ASSOCIATE DATA SCIENTIST	HEALTHCARE	E083	5
	E620	Katrina	Allen	F	JUNIOR DATA SCIENTIST	RETAIL	E612	2
	E001	Arthur	Black	M	CEO	ALL	NULL	NULL

Grouping Table Data

GROUP BY Clause in MySQL

- The **GROUP BY** clause divides a set of rows into subgroups depending on column or expression values.

- For each group, the **GROUP BY** clause produces a single row. In other words, the number of rows in the result set is reduced.

- The GROUP BY clause is an optional clause of the **SELECT** statement.

- The **GROUP BY** clause is used with aggregate functions such as **SUM**, **AVG**, **MAX**, **MIN**, and **COUNT**.

- The information for each group is provided by the aggregate function in the **SELECT** clause.

GROUP BY Clause in MySQL

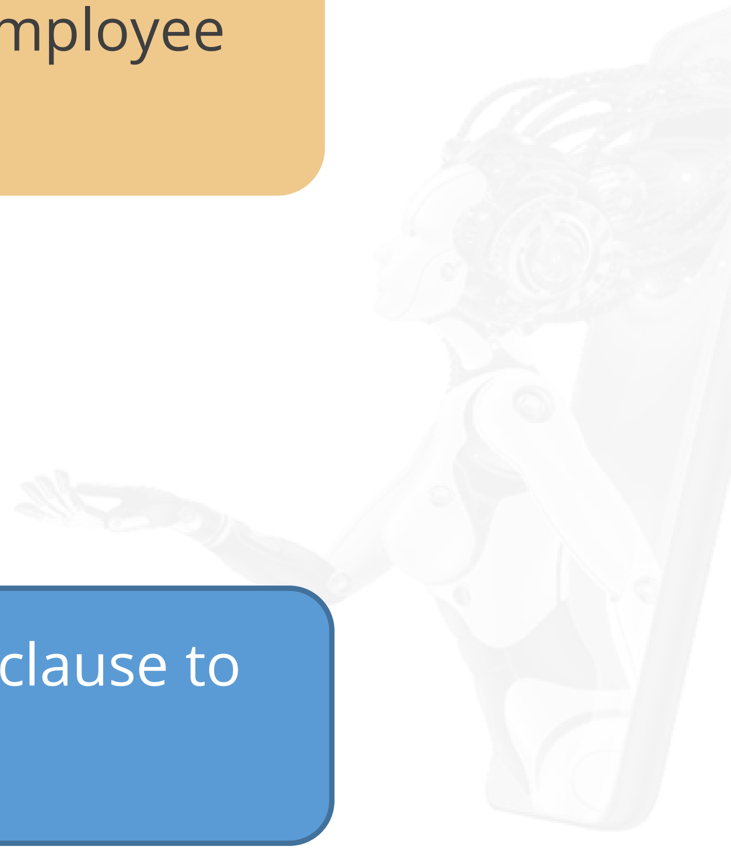
- Syntax:

Syntax

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```

GROUP BY Clause in MySQL: Example

Problem Statement: Your manager expects you to provide the details of one employee from each department from the employee table.



Objective: Write an SQL query using the **SELECT** statement with the **GROUP BY** clause to get the relevant data from the **EMP_TABLE** table.

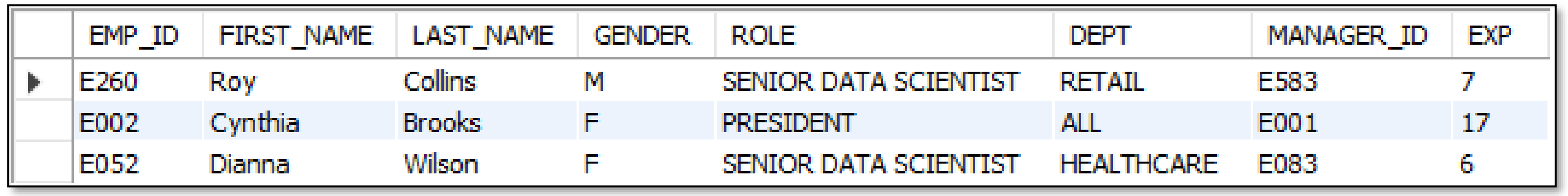
GROUP BY Clause in MySQL: Example

Step 1: Use the **SELECT** statement with the **GROUP BY** clause to get the first row of each group from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

GROUP BY DEPT;
```

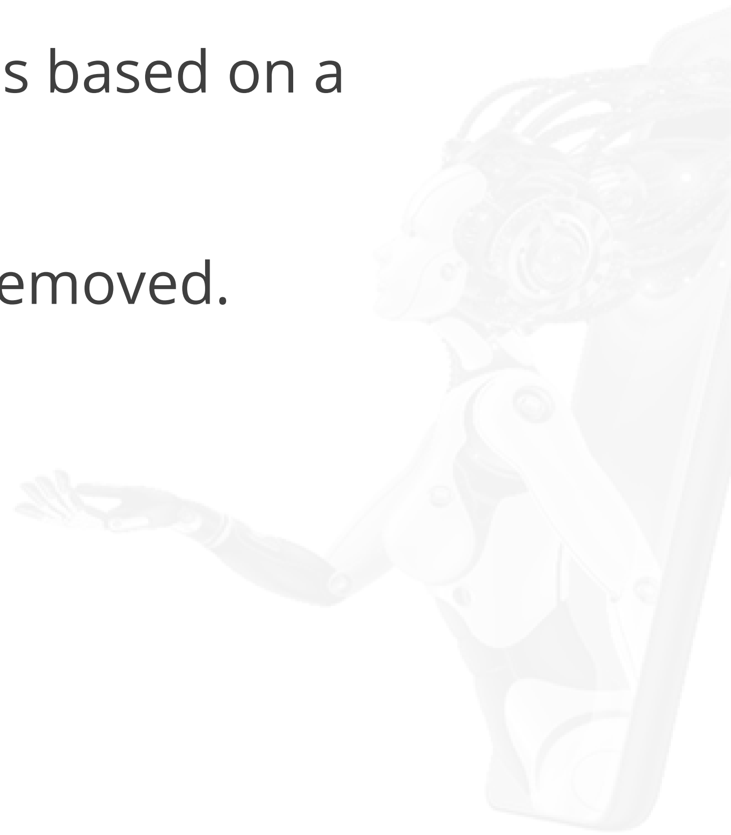
Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
▶	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583	7
	E002	Cynthia	Brooks	F	PRESIDENT	ALL	E001	17
	E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	E083	6

HAVING Clause in MySQL

- The HAVING clause is used to define filter criteria for a group of rows or aggregates in the SELECT statement.
- The HAVING clause is frequently used with the GROUP BY clause to filter groups based on a specified condition.
- The HAVING clause operates like the WHERE clause if the GROUP BY clause is removed.



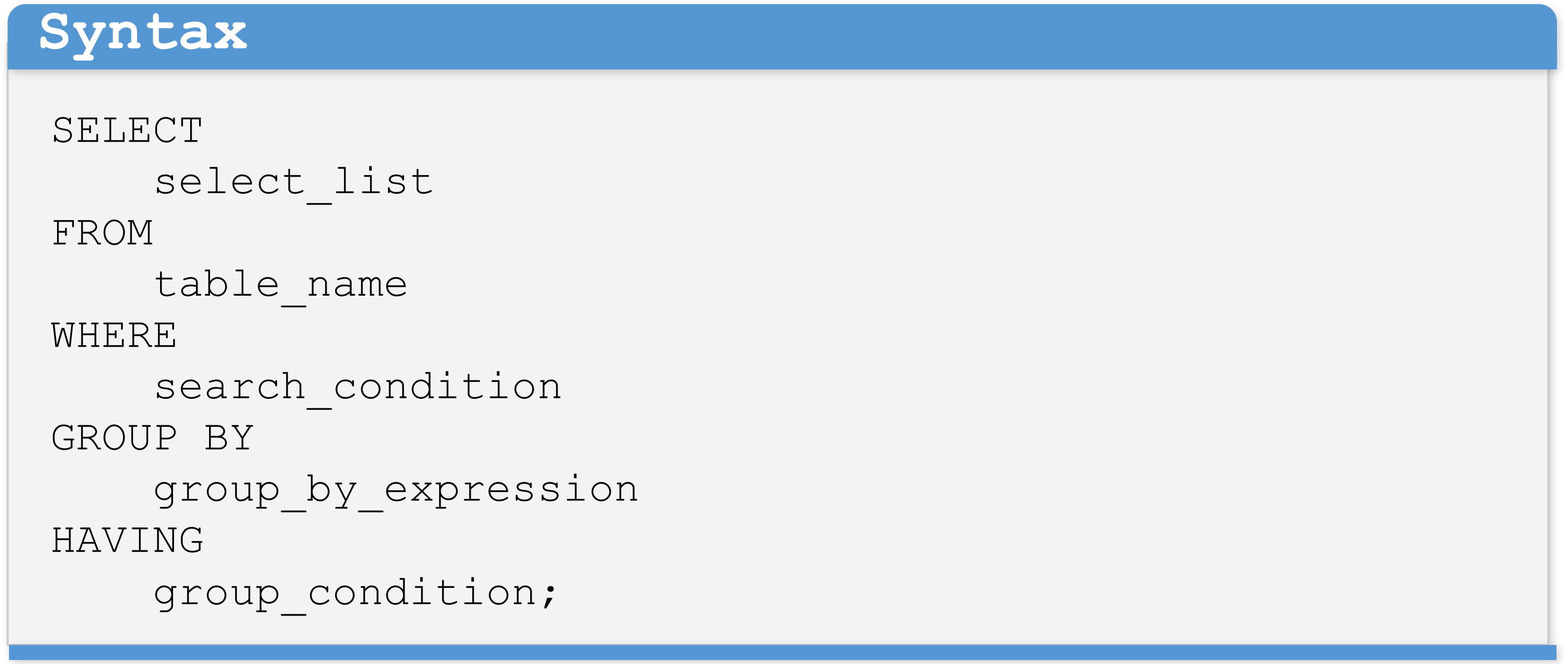
HAVING Clause in MySQL

- Each group produced by the GROUP BY clause is evaluated by the HAVING clause, and if the result is true, the row is included in the result set.

- **Note:** The WHERE clause applies a filter condition to each individual row, whereas the HAVING clause applies it to each group of rows.

HAVING Clause in MySQL

- Syntax:



Syntax

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition
GROUP BY
    group_by_expression
HAVING
    group_condition;
```

HAVING Clause in MySQL: Example

Problem Statement: Your manager expects you to provide the details of one employee from each department having five to ten years of experience.

Objective: Write an SQL query using the **SELECT** statement with **GROUP BY** and **HAVING** clauses to get the relevant data from the **EMP_TABLE** table.

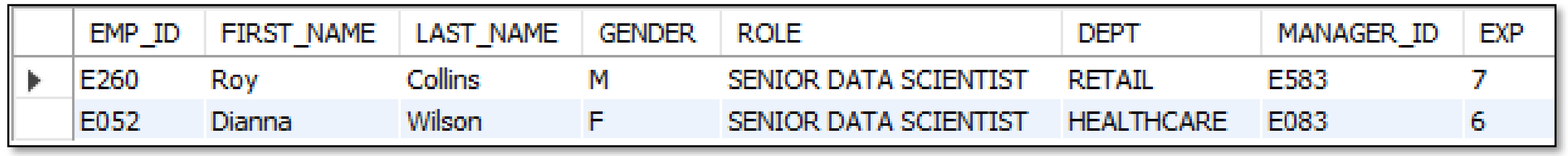
HAVING Clause in MySQL: Example

Step 1: Use the **SELECT** statement with the **GROUP BY** and **HAVING** clauses to get the first row of each group meeting a required condition from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT * FROM employees_db.EMP_TABLE

GROUP BY DEPT

HAVING EXP >= 5 AND EXP <= 10;
```

Output:



	EMP_ID	FIRST_NAME	LAST_NAME	GENDER	ROLE	DEPT	MANAGER_ID	EXP
▶	E260	Roy	Collins	M	SENIOR DATA SCIENTIST	RETAIL	E583	7
	E052	Dianna	Wilson	F	SENIOR DATA SCIENTIST	HEALTHCARE	E083	6

ROLLUP

ROLLUP

ROLLUP does not create all possible grouping sets based on the dimension columns.

ROLLUP provides a shorthand for defining multiple grouping sets.

ROLLUP

An extension of GROUP BY clause

Multiple sets generation

Extra rows represent subtotals

ROLLUP

Syntax

```
SELECT
    column1, column2, column3, ...
FROM
    table_name
GROUP BY
    column1, column2,...WITH ROLLUP;
```

ROLLUP: Example

Problem Statement: You are working as a Junior Database Administrator in an international retail company, and the HR team expects you to find the number of employees in the company in each county along with the total number of employees in the company.

Objective: Write an SQL query using the SELECT statement with GROUP BY and ROLLUP clauses to get the employee count in each country along with the grand total from the **EMP_TABLE** table.

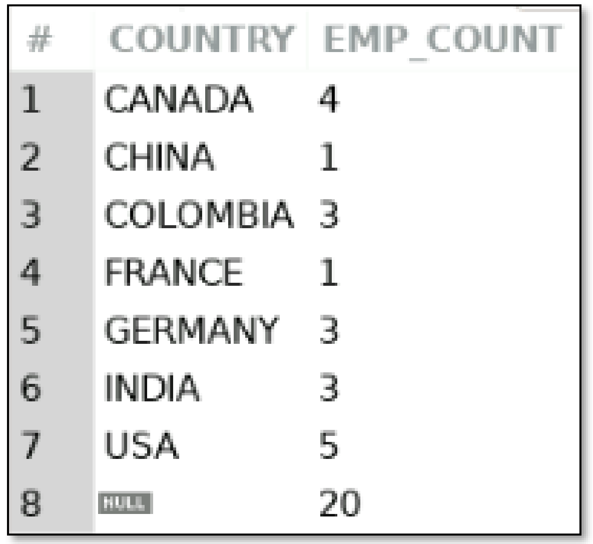
ROLLUP: Example

Step 1: Use the **SELECT** statement with the **GROUP BY** and **ROLLUP** clauses to get the employee count in each country along with the grand total from the **EMP_TABLE** table as given below.

SQL Query

```
SELECT COUNTRY, COUNT(EMP_ID) AS EMP_COUNT

FROM employees_db.EMP_TABLE

GROUP BY COUNTRY WITH ROLLUP
```

Output:



#	COUNTRY	EMP_COUNT
1	CANADA	4
2	CHINA	1
3	COLOMBIA	3
4	FRANCE	1
5	GERMANY	3
6	INDIA	3
7	USA	5
8	NULL	20

Assisted Practice: Grouping and Ordering

Duration: 20 mins

Problem statement: You have joined Simplilearn as an analyst. Your first task is to answer a few questions on the quiz organized for learners participating from across the world. Your results would help Simplilearn understand important metrics like the participation rate, winners, and senior citizen participants. The assisted practice questions for this lesson will leverage a quiz dataset. Write a query to count the number of participants in each team. Also, display the team's name in an alphabetical order, i.e., A to Z.

Assisted Practice: Grouping and Ordering

Duration: 20 min

Data: This data is about the people who participated in a quiz. It has 1000 rows and 11 columns.

Column Name	Column Description
id_number	unique identifier
first_name	first name of participant
last_name	last name of participant
City	city, the participant belongs to
state_code	state code, the participant belongs to
shirt_or_hat	participant would want hat or a shirt as a prize
quiz_points	points scored in the quiz
team	team the participant belongs to
signup	date of signup
age	age of participant
company	company of participant

Assisted Practice: Grouping and Ordering

Steps to be performed:

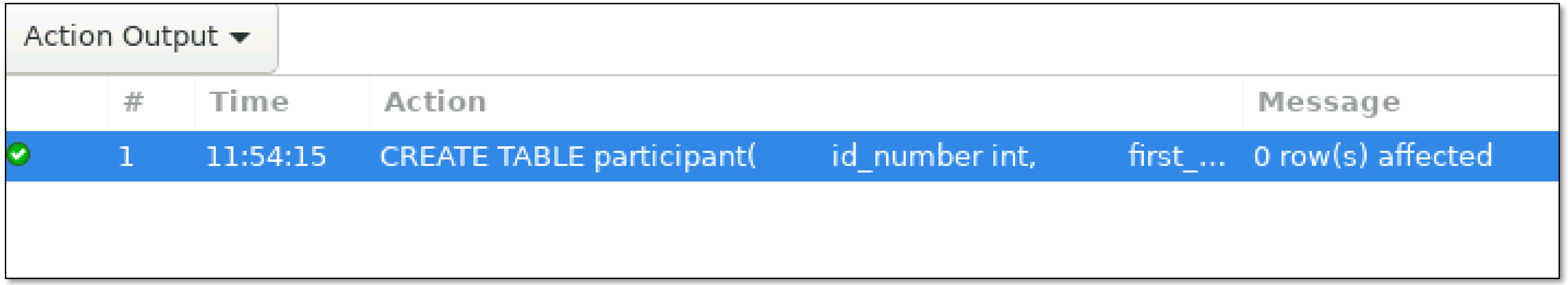
Step 01: Create the table schema using the below query:

CREATE

```
CREATE TABLE participant(
id_number int,first_name text,last_name text,city text,
state_code text,shirt_or_hat text,quiz_points int,team text,
signup date,age int,company text);
```

Assisted Practice: Grouping and Ordering

Output:



```
    #   Time      Action                                                   Message
✓   1   11:54:15  CREATE TABLE participant(    id_number int,     first_...  0 row(s) affected
```

Assisted Practice: Grouping and Ordering

Step 02: Load the **participant.csv** file by following the steps listed in the lab guide.

Step 03: Use the FROM keyword to direct to the table having the information, which is ***participant*** here.

Step 04: Use the GROUP BY keyword to perform aggregation on the nonaggregating columns with the SELECT statement. As we need to count the number of participants for each team, the data will be grouped based on the *team* column.

Step 05: Use the ORDER BY keyword to sort the details in an ascending order (by default) based on the column(s) mentioned. Here, sort the data based on the *team* column.

Assisted Practice: Grouping and Ordering

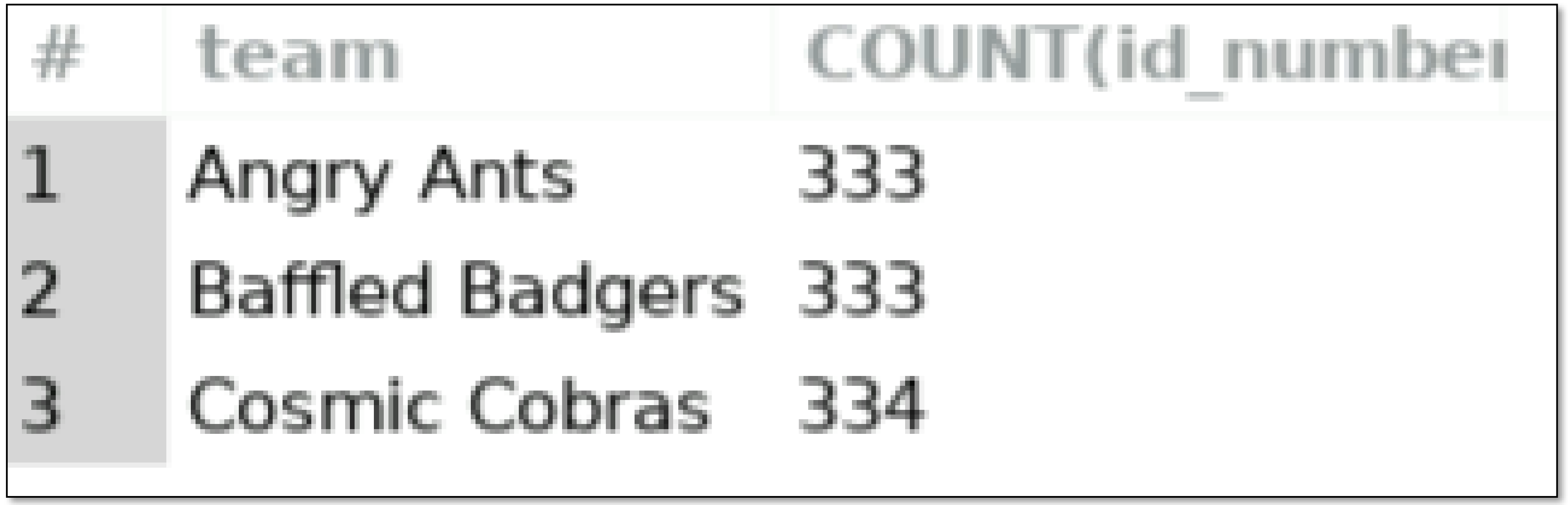
Step 06: Combine the previous steps and create a final query:

```
SELECT team, COUNT(id_number)
FROM participant
GROUP BY team
ORDER BY team
```

Note: The above query should return five records with four columns.

Assisted Practice: Grouping and Ordering

Output:

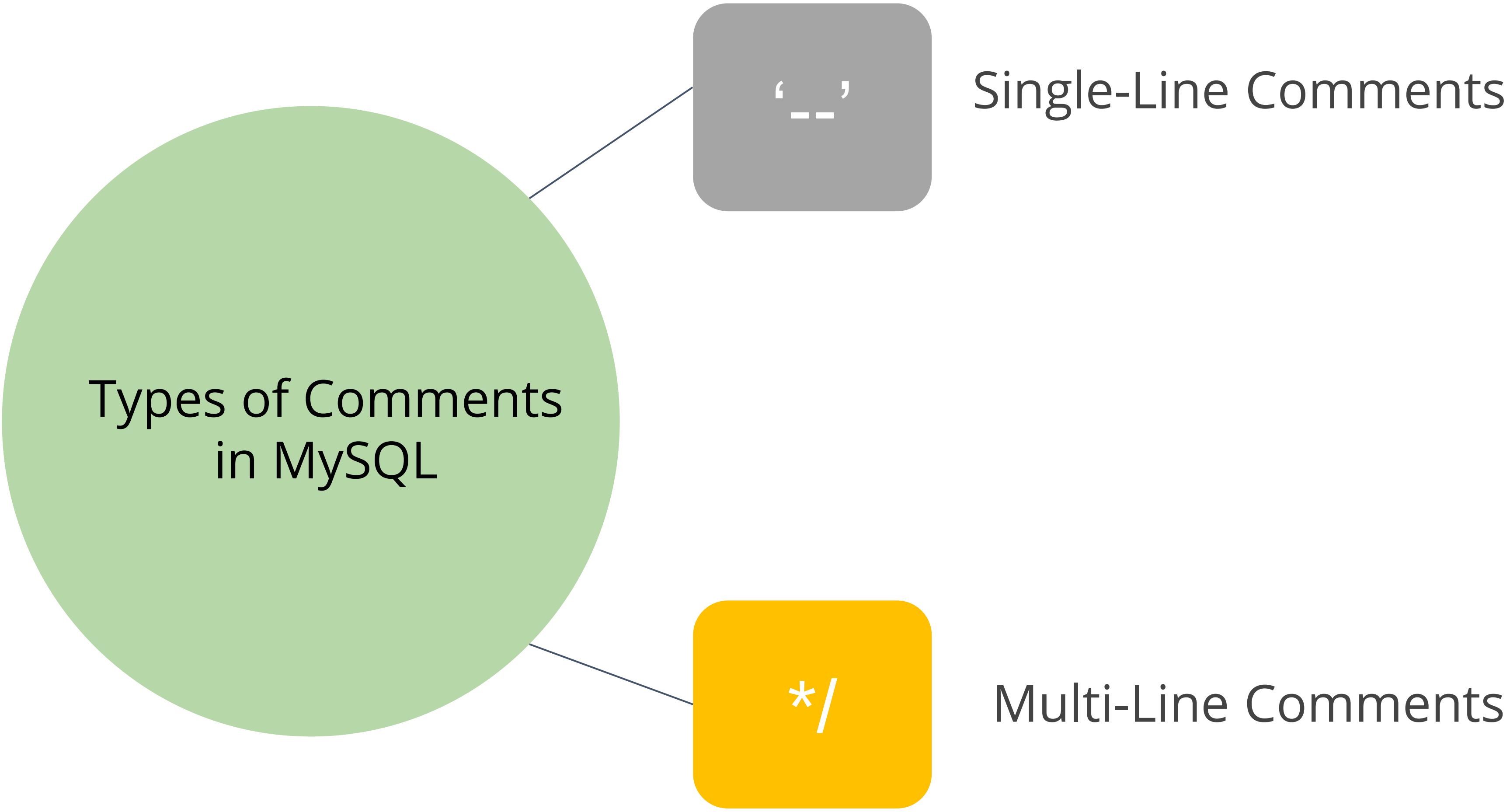


#	team	COUNT(id_number
1	Angry Ants	333
2	Baffled Badgers	333
3	Cosmic Cobras	334

Comments in MySQL

Comments in MySQL

Comments are used to explain sections of SQL statements or to prevent execution of SQL statements.



Single-Line Comments

Single-line comments start with two hyphens (--).

Any text present between the double hyphen and the end of the line will not be executed.

Single-Line Comments

Syntax

```
--Select all:
SELECT * FROM Table_Name;
```

Single-Line Comments: Example

SQL Query

```
A Single-line comment to ignore the end of a line:

SELECT * FROM Customers -- WHERE City='Berlin’;



A Single-line comment to ignore a statement:

--SELECT * FROM Customers;

SELECT * FROM Products;
```

Multi-Line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored.

Multi-Line Comments: Example

SQL Query

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

Multi-Line Comments: Example

SQL Query

```
/*SELECT * FROM table_name;

SELECT * FROM Datacenter;*/

SELECT * FROM Projects;
```

Multi-Line Comments: Example

SQL Query

```
SELECT Customer_Name, /*City,*/ Country FROM Customers;
```

Multi-Line Comments: Example

SQL Query

```
SELECT * FROM Customers WHERE (CustomerName) LIKE 'L%'

OR CustomerName LIKE 'R%’

/* OR CustomerName LIKE 'S%'

AND Country='USA’*/

ORDER BY CustomerName;
```

Knowledge Check

Knowledge Check

1

Which statement should be used to delete all rows from a table without logging the action?

A. DELETE

B. REMOVE

C. DROP

D. TRUNCATE

Knowledge Check

1

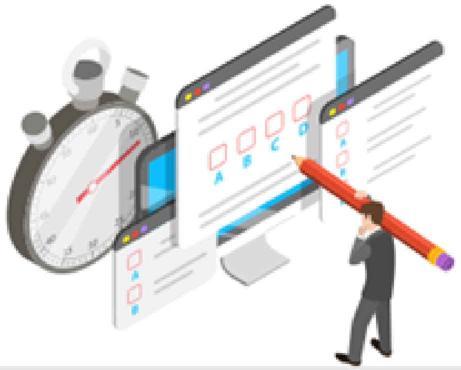
Which statement should be used to delete all rows from a table without logging the action?

A. DELETE

B. REMOVE

C. DROP

D. TRUNCATE



The correct answer is **D**

The TRUNCATE statement deletes all rows from a table without recording the row deletions individually. It is identical to the DELETE statement without the WHERE clause, but it runs quicker since it consumes fewer system and transaction log resources.

Knowledge Check

2

Which operator is used to compare a value to a specified list of values?

A. AND

B. BETWEEN

C. LIKE

D. IN

Knowledge Check

2

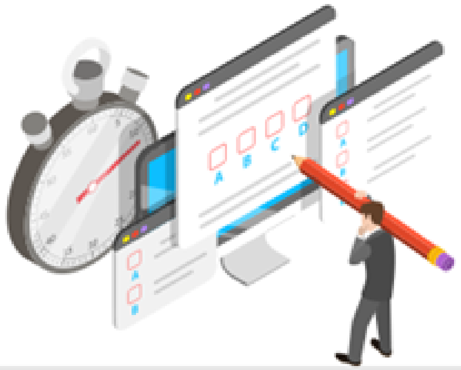
Which operator is used to compare a value to a specified list of values?

A. AND

B. BETWEEN

C. LIKE

D. IN



The correct answer is **D**

The IN operator makes it simple to test an expression and see whether it matches any value in a list of values. This eliminates the need for numerous OR conditions.

Knowledge Check

3

What is the default sorting mechanism specified for ORDER BY clause in MySQL?

A. DESC

B. ASC

C. There is no default value

D. None of the above

Knowledge Check

3

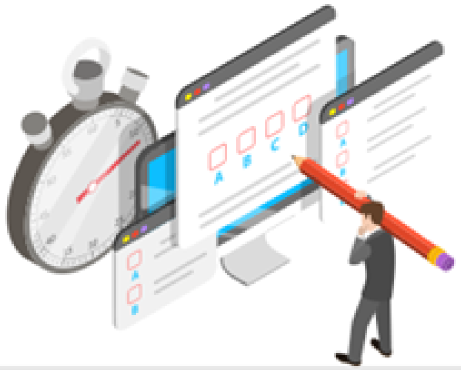
What is the default sorting mechanism specified for ORDER BY clause in MySQL?

A. DESC

B. ASC

C. There is no default value

D. None of the above



The correct answer is **B**

If we do not specify a sorting mechanism while using the ORDER BY clause, MySQL utilizes the ASC as the default sorting order. When sorting, SQL considers Null to be the lowest possible value.

Knowledge Check

4

Which of the following statements regarding the HAVING clause is correct?

A. Similar to the WHERE clause but is used for columns rather than groups.

B. Similar to WHERE clause but is used for rows rather than columns.

C. Similar to WHERE clause but is used for groups rather than rows.

D. Acts exactly like a WHERE clause.

Knowledge Check

4

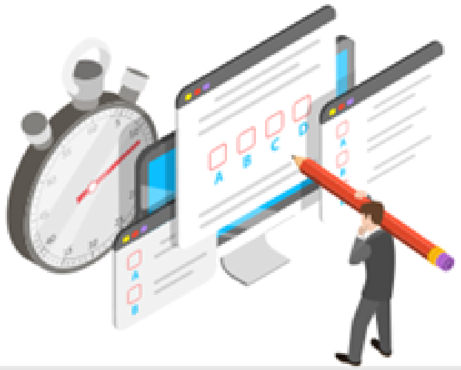
Which of the following statements regarding the HAVING clause is correct?

A. Similar to the WHERE clause but is used for columns rather than groups.

B. Similar to WHERE clause but is used for rows rather than columns.

C. Similar to WHERE clause but is used for groups rather than rows.

D. Acts exactly like a WHERE clause.



The correct answer is **C**

The HAVING clause is always used with the GROUP BY clause and returns the rows where the condition is TRUE.

Lesson-End Project: Retail Mart Management

Problem statement:

A data analyst of a retail shop, Happy Mart, wants to store the product details, customer details, and order details to provide daily insights about customer behavior and product stock details.

Objective:

The objective is to design a database to easily evaluate and identify the performance of the shop to increase the daily sales.

Note: Download the **customer_datasets.csv**, **product_datasets.csv**, and **sales_datasets.csv** files from **Course Resources** to perform the required tasks

LESSON-END PROJECT

Lesson-End Project: Retail Mart Management

Tasks to be performed:

1. Write a query to create a database named **SQL basics**
2. Write a query to select **SQL basics**
3. Write a query to create a **product** table with the fields product code, product name, price, stock, and category, a **customer** table with the fields customer ID, customer name, customer location, and customer phone number, and a **sales** table with the fields date, order number, product code, product name, quantity, and price
4. Write a query to insert values into the **customer**, **product**, and **sales** tables

LESSON-END PROJECT

Lesson-End Project: Retail Mart Management

Tasks to be performed:

5. Write a query to add new columns, such as serial number and categories, to the **sales** table
6. Write a query to change the stock field type to varchar in the **product** table
7. Write a query to change the table name from **customer** to **customer details**
8. Write a query to drop the sl. no. and categories columns from the **sales** table

Lesson-End Project: Retail Mart Management

Tasks to be performed:

9. Write a query to display the order ID, customer ID, order date, price, and quantity columns of the **sales** table
10. Write a query to display the details where the category is stationary from the **product** table
11. Write a query to display the unique category from the **product** table
12. Write a query to display the details of the sales from the **sales** table where quantity is greater than 2 and the price is less than 500
13. Write a query to display every customer whose name ends with an **'a'**

Lesson-End Project: Retail Mart Management

Tasks to be performed:

14. Write a query to display the product details in descending order of price
15. Write a query to display the product code and category from categories that have two or more products
16. Write a query to combine the **sales** and **product** tables based on the order number and customer's name including duplicated rows

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

Lesson-End Project: Churn Analysis

Problem statement:

As an analyst of AKR analytics, you have been asked to analyze the behavior of those who are no longer customers versus those who are still customers of the company.

Objective:

Analyze churn data and share insights about those who are no longer customers of the company so that the existing customers can be retained

Note: Use the existing table in labs named **churn data** to perform the required tasks

Lesson-End Project: Churn Analysis

Tasks to be performed:

Step 01: Upload the **ChurnData.csv** dataset to the lab

Step 02: Import the **ChurnData.csv** into the database as a table

Step 03: Find the number of unique customers present in the data

Step 04: Find the number of customers who have left vs. those who haven’t

Step 05: Find the top two customer IDs with the highest total charges (from the **TotalCharge** column) for those who were taking all the services offered but still decided to leave the company

Lesson-End Project: Churn Analysis

Tasks to be performed:

Step 06: Find the average tenure and average monthly charges by gender for those who stopped being customers

Step 07: Write a query to understand the split of customer IDs by **Churn** and **seniorCitizen** columns and find if the average of the **TotalCharge** column are higher for senior citizens or nonsenior citizens

Step 08: Display the mode of payment preferred by those who are no longer customers (from the **PaymentMethod** column) and those who still are in descending order of **TotalCharge** with only those customers who have paid more than $10,000

LESSON-END PROJECT

Lesson-End Project: Churn Analysis

Tasks to be performed:

Step 09: Find the number of people who are no longer customers that were only taking the phone services, the internet services, or both and exclude those who were using *fiber-optic* internet services

LESSON-END PROJECT

Key Takeaways

- Important SQL commands such as CREATE DATABASE, SHOW DATABASE, USE, and DROP DATABASE are required to manage a database.
- Storage engines store and manage data in a database via SQL operations on several tables.
- In MySQL, the WHERE clause applies a filter condition to each individual row, whereas the HAVING clause applies it to each group of rows.
- In MySQL, the ORDER BY clause performs sorting, whereas the GROUP BY and HAVING clauses perform grouping.