1. Create a New Database and Table for Employees Task: Create a new database named and Create a table named with the following columns.

Column Name	Data Type	Constraint
employee_id	INT	PRIMARY KEY
first_name	VARCHAR(50)	
last_name	VARCHAR(50)	
department	VARCHAR(50)	
salary	INT	
hire_date	DATE	

➔ Snippet to run:

```
CREATE DATABASE company_db;
USE company_db;

CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    department VARCHAR(50),
    salary INT,
    hire_date DATE
);
```

- **CREATE DATABASE**: This command initializes a new storage container named company_db.
- **INT**: Used for employee_id and salary because these fields store whole numbers.
- **VARCHAR(50)**: Used for names and departments to allow variable-length character strings up to 50 characters.
- **PRIMARY KEY**: This constraint ensures that each employee_id is unique and not null, serving as the unique identifier for every row.
- **DATE**: Specifically formatted to store calendar dates (YYYY-MM-DD) for the hire_date column.

2. Insert Data into Employees Table Task: Insert the following sample records into the table.

employee_id	first_name	last_name	department	salary	hire_date
101	Amit	Sharma	HR	50000	2020-01-15
102	Riya	Kapoor	Sales	75000	2019-03-22
103	Raj	Mehta	IT	90000	2018-07-11
104	Neha	Verma	IT	85000	2021-09-01
105	Arjun	singh	Finance	60000	2022-02-10

➔ Snippet to run:

```
INSERT INTO employees (employee_id, first_name, last_name, department, salary, hire_date)
VALUES
(101, 'Amit', 'Sharma', 'HR', 50000, '2020-01-15'),
(102, 'Riya', 'Kapoor', 'Sales', 75000, '2019-03-22'),
(103, 'Raj', 'Mehta', 'IT', 90000, '2018-07-11'),
(104, 'Neha', 'Verma', 'IT', 85000, '2021-09-01'),
(105, 'Arjun', 'Singh', 'Finance', 60000, '2022-02-10');
```

Syntax: The `INSERT INTO` statement specifies the target table and columns, while `VALUES` contains the actual data rows.
Data Formatting: String values (like names and departments) and Dates must be enclosed in **single quotes** (`' '`).
Integrity: Because `employee_id` is the **Primary Key**, MySQL will throw an error if you try to insert a duplicate ID (e.g., trying to add another record with ID 101).

3. Display All Employee Records Sorted by Salary (Lowest to Highest)
   Hint: Use the clause on the salary column.

➔ Snippet to run:

```
SELECT * FROM employees
ORDER BY salary ASC;
```

employee_id	first_name	last_name	department	salary	hire_date
101	Amit	Sharma	HR	50000	2020-01-15
105	Arjun	Singh	Finance	60000	2022-02-10
102	Riya	Kapoor	Sales	75000	2019-03-22
104	Neha	Verma	IT	85000	2021-09-01
103	Raj	Mehta	IT	90000	2018-07-11

- **SELECT ***: This retrieves all columns from the `employees` table.
- **ORDER BY salary**: This tells the database to arrange the rows based on the values in the `salary` column.
- **ASC**: This keyword stands for **Ascending**. While it is the default behavior in MySQL, explicitly including it makes your code more readable for other developers.

Expected Result Set

4. Show Employees Sorted by Department (A–Z) and Salary (High → Low)

➔ Snippet to run:

```
SELECT * FROM employees
ORDER BY department ASC, salary DESC;
```

employee_id	first_name	last_name	department	salary	hire_date
105	Arjun	Singh	Finance	60000	2022-02-10
101	Amit	Sharma	HR	50000	2020-01-15
103	Raj	Mehta	IT	90000	2018-07-11
104	Neha	Verma	IT	85000	2021-09-01
102	Riya	Kapoor	Sales	75000	2019-03-22

- **ORDER BY department ASC**: This sorts the departments alphabetically (A to Z).
- **salary DESC**: For employees in the same department (like the two employees in "IT"), this sorts them by salary from highest to lowest.

5. List All Employees in the IT Department, Ordered by Hire Date (Newest First)

➔ Snippet to run:

```
SELECT * FROM employees
WHERE department = 'IT'
ORDER BY hire_date DESC;
```

employee_id	first_name	last_name	department	salary	hire_date
104	Neha	Verma	IT	85000	2021-09-01
103	Raj	Mehta	IT	90000	2018-07-11
NULL	NULL	NULL	NULL	NULL	NULL

- **WHERE department = 'IT'**: This filters the entire table to only include rows where the department is exactly "IT".
- **ORDER BY hire_date DESC**: This sorts the filtered results by date. In SQL, a "Higher" date is a more recent one (e.g., 2021 is higher than 2018), so `DESC` gives you the newest employees first.

6. Create and Populate a Sales Table
   Task: Create a table to track sales data:

sale_id	customer_name	amount	sale_date
1	Aditi	1500	2024-08-01
2	Rohan	2200	2024-08-03
3	Aditi	3500	2024-09-05
4	Meena	2700	2024-09-15
5	Rohan	4500	2024-09-25

➔ Snippet to run:

```
CREATE TABLE sales (
    sale_id INT PRIMARY KEY,
    amount INT,
    customer_name VARCHAR(50),
    sale_date DATE
);

INSERT INTO sales (sale_id, amount, customer_name, sale_date)
VALUES
(1, 1500, 'Aditi', '2024-08-01'),
(2, 2200, 'Rohan', '2024-08-03'),
(3, 3500, 'Aditi', '2024-09-05'),
(4, 2700, 'Meena', '2024-09-15'),
(5, 4500, 'Rohan', '2024-09-25');
```

sale_id	amount	customer_name	sale_date
1	1500	Aditi	2024-08-01
2	2200	Rohan	2024-08-03
3	3500	Aditi	2024-09-05
4	2700	Meena	2024-09-15
5	4500	Rohan	2024-09-25

- **Table Creation**: We define `sale_id` as the `PRIMARY KEY` to ensure every transaction is unique.
- **Data Types**: `amount` is stored as an `INT` for calculation purposes, and `sale_date` uses the `DATE` type for time-based filtering.

7. Display All Sales Records Sorted by Amount (Highest → Lowest)
   Hint: Use ORDER BY amount DESC.

➔ Snippet to run:

```
SELECT * FROM sales
ORDER BY amount DESC;
```

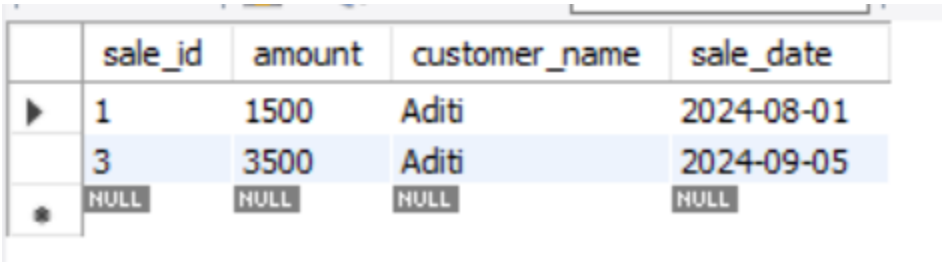
sale_id	amount	customer_name	sale_date
5	4500	Rohan	2024-09-25
3	3500	Aditi	2024-09-05
4	2700	Meena	2024-09-15
2	2200	Rohan	2024-08-03
1	1500	Aditi	2024-08-01

- **ORDER BY amount**: This targets the numerical column containing the transaction values.
- **DESC**: Short for **Descending**. This sorts the data from the largest number to the smallest.

8. Show All Sales Made by Customer “Aditi”
   Hint: Use WHERE customer_name = 'Aditi'.

➔ Snippet to run:

```
SELECT * FROM sales
WHERE customer_name = 'Aditi';
```



sale_id	amount	customer_name	sale_date
1	1500	Aditi	2024-08-01
3	3500	Aditi	2024-09-05
NULL	NULL	NULL	NULL

- **WHERE customer_name = 'Aditi'**: This clause tells MySQL to scan the table and only return rows where the value in the `customer_name` column exactly matches the string "Aditi".
- **Single Quotes**: In SQL, string (text) values must be enclosed in single quotes.

9. What is the Difference Between a Primary Key and a Foreign Key?

➔ **Primary Key (PK)**

- **Definition**: A column (or a set of columns) that uniquely identifies each row in a table.
- **Uniqueness**: Every value in this column must be unique; no two rows can have the same Primary Key.
- **Nullability**: It cannot contain `NULL` values.
- **Quantity**: A table can have only one Primary Key.
- **Example**: In our `employees` table, `employee_id` is the Primary Key.

Foreign Key (FK)

- **Definition**: A column in one table that refers to the Primary Key in another table.
- **Purpose**: It establishes a relationship (link) between two tables, ensuring that the value in the Foreign Key column exists in the referenced table.
- **Uniqueness**: Unlike a Primary Key, a Foreign Key can contain duplicate values.
- **Nullability**: It can contain `NULL` values unless specified otherwise.
- **Example**: If we had a `departments` table, the `department` column in the `employees` table could be a Foreign Key linking to it.

10. What Are Constraints in SQL and Why Are They Used?

➔ Constraints are the **rules** applied to columns in a table to limit the type of data that can be stored. They are essential for maintaining the accuracy and reliability of information within a database.

Why Constraints Are Used

- **Data Integrity**: They prevent invalid or "junk" data from entering the system.
- **Accuracy**: They ensure that the relationships between tables remain consistent.

- **Automation**: The database automatically rejects any `INSERT` or `UPDATE` operation that violates a rule, saving developers from writing manual validation code.