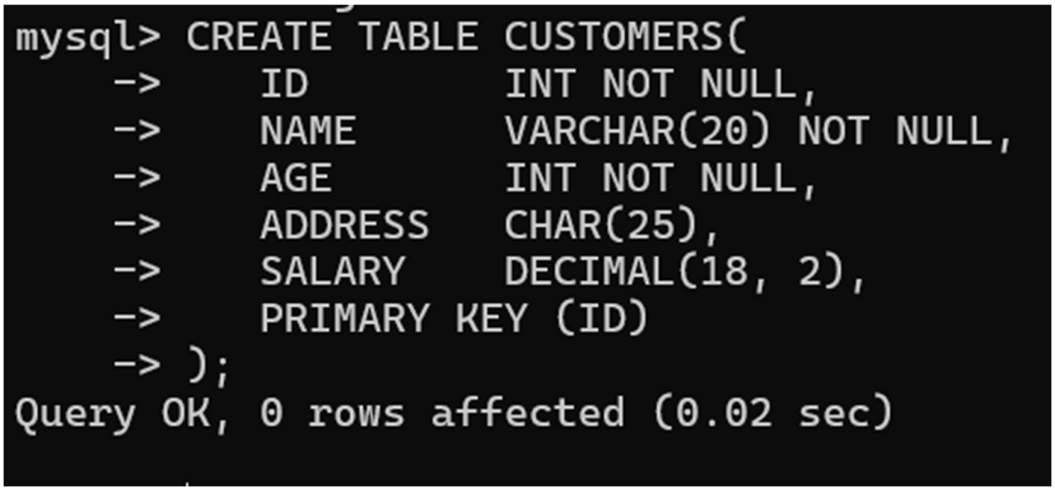
**Q1)Create table customers with the following fileds**

```
SQL> CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```




**2. use the DESC command to check the table customers and view the definition of table**

```
mysql> DESC CUSTOMERS;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| ID      | int          | NO   | PRI | NULL    |       |
| NAME    | varchar(20)  | NO   |     | NULL    |       |
| AGE     | int          | NO   |     | NULL    |       |
| ADDRESS | char(25)     | YES  |     | NULL    |       |
| SALARY  | decimal(18,2)| YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

## 3.insert values to customers table

```
mysql> INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)
    -> VALUES
    -> (1, 'John', 28, 'New York', 45000.00),
    -> (2, 'Mary', 35, 'Los Angeles', 55000.00),
    -> (3, 'David', 22, 'Chicago', 30000.00);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

## 4. alter table customers add column DOB

```
mysql> ALTER TABLE CUSTOMERS RENAME COLUMN NAME TO CUST_NAME;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE CUSTOMERS ADD DOB DATE;
```

## 5. alter table modify columnname "name" as "cust_name"

```
mysql> ALTER TABLE CUSTOMERS RENAME COLUMN NAME TO CUST_NAME;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

## 6. alter table add check constraint as age between 1 and 100

```
mysql> ALTER TABLE CUSTOMERS ADD DOB DATE;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

## 7. alter table drop constrain not null of name column

```
mysql> ALTER TABLE CUSTOMERS MODIFY CUST_NAME VARCHAR(20) NULL;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> select * from customers;
+----+-----------+-----+-------------+----------+------+
| ID | CUST_NAME | AGE | ADDRESS     | SALARY   | DOB  |
+----+-----------+-----+-------------+----------+------+
|  1 | John      |  28 | New York    | 45000.00 | NULL |
|  2 | Mary      |  35 | Los Angeles | 55000.00 | NULL |
|  3 | David     |  22 | Chicago     | 30000.00 | NULL |
+----+-----------+-----+-------------+----------+------+
3 rows in set (0.00 sec)
```

## 8. drop table customers

```
mysql> DROP TABLE CUSTOMERS;
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> select * from customers;
ERROR 1146 (42S02): Table 'mydb.customers' doesn't exist
```

**q2)Create table ORDERS with the following fields:**

**SQL> CREATE TABLE ORDERS (**

**ORDER_ID INT NOT NULL,**

**CUSTOMER_ID INT NOT NULL,**

**ORDER_DATE DATE NOT NULL,**

**AMOUNT DECIMAL(10,2),**

**STATUS VARCHAR(15),**

**PRIMARY KEY (ORDER_ID)**

**);**

```
mysql> CREATE TABLE ORDERS (
    ->     ORDER_ID    INT NOT NULL,
    ->     CUSTOMER_ID INT NOT NULL,
    ->     ORDER_DATE  DATE NOT NULL,
    ->     AMOUNT      DECIMAL(10,2),
    ->     STATUS      VARCHAR(15) NOT NULL,
    ->     PRIMARY KEY (ORDER_ID)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

## 2)Use the DESC command to check the definition of the ORDERS table.

```
mysql> DESC ORDERS;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| ORDER_ID    | int           | NO   | PRI | NULL    |       |
| CUSTOMER_ID | int           | NO   |     | NULL    |       |
| ORDER_DATE  | date          | NO   |     | NULL    |       |
| AMOUNT      | decimal(10,2) | YES  |     | NULL    |       |
| STATUS      | varchar(15)   | NO   |     | NULL    |       |
+-------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

## 3) Insert five sample rows into the ORDERS table.

```
mysql> INSERT INTO ORDERS (ORDER_ID, CUSTOMER_ID, ORDER_DATE, AMOUNT, STATUS)
    -> VALUES
    -> (101, 1, '2025-08-01', 250.00, 'Pending'),
    -> (102, 2, '2025-08-02', 500.00, 'Shipped'),
    -> (103, 3, '2025-08-03', 150.00, 'Delivered'),
    -> (104, 1, '2025-08-05', 300.00, 'Pending'),
    -> (105, 4, '2025-08-06', 450.00, 'Cancelled');
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

## 4) Alter the ORDERS table to add a new column SHIP_DATE of type DATE.

```
mysql> ALTER TABLE ORDERS ADD SHIP_DATE DATE;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

## 5) Rename the column AMOUNT to TOTAL_AMOUNT.

```
mysql> ALTER TABLE ORDERS RENAME COLUMN AMOUNT TO TOTAL_AMOUNT;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

## 6) Add a CHECK constraint to ensure that TOTAL_AMOUNT is greater than 0.

```
mysql> ALTER TABLE ORDERS
    -> ADD CONSTRAINT chk_total_amount CHECK (TOTAL_AMOUNT > 0);
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

## 7) Remove the NOT NULL constraint from the STATUS column.

```
mysql> ALTER TABLE ORDERS MODIFY STATUS VARCHAR(15) NULL;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT * FROM ORDERS;
+----------+-------------+------------+--------------+-----------+-----------+
| ORDER_ID | CUSTOMER_ID | ORDER_DATE | TOTAL_AMOUNT | STATUS    | SHIP_DATE |
+----------+-------------+------------+--------------+-----------+-----------+
|      101 |           1 | 2025-08-01 |       250.00 | Pending   | NULL      |
|      102 |           2 | 2025-08-02 |       500.00 | Shipped   | NULL      |
|      103 |           3 | 2025-08-03 |       150.00 | Delivered | NULL      |
|      104 |           1 | 2025-08-05 |       300.00 | Pending   | NULL      |
|      105 |           4 | 2025-08-06 |       450.00 | Cancelled | NULL      |
+----------+-------------+------------+--------------+-----------+-----------+
5 rows in set (0.00 sec)
```

## 8) Drop the ORDERS table.

```
mysql> DROP TABLE ORDERS;
Query OK, 0 rows affected (0.01 sec)
```

**Create table CUSTOMERS:**

```
CREATE TABLE CUSTOMERS (
  CUST_ID   INT          PRIMARY KEY,
  NAME      VARCHAR(30)   NOT NULL,
  CITY      VARCHAR(20)
);
```

**Create table ORDERS:**

```
CREATE TABLE ORDERS (
  ORDER_ID   INT          PRIMARY KEY,
  CUST_ID    INT          NOT NULL,
  ORDER_DATE DATE,
  AMOUNT     DECIMAL(10,2),
  FOREIGN KEY (CUST_ID) REFERENCES
CUSTOMERS(CUST_ID)
);
```

```
mysql> CREATE TABLE CUSTOMERS (
    ->     CUST_ID   INT PRIMARY KEY,
    ->     NAME      VARCHAR(30) NOT NULL,
    ->     CITY      VARCHAR(20)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE ORDERS (
    ->     ORDER_ID   INT PRIMARY KEY,
    ->     CUST_ID    INT NOT NULL,
    ->     ORDER_DATE DATE,
    ->     AMOUNT     DECIMAL(10,2),
    ->     FOREIGN KEY (CUST_ID) REFERENCES CUSTOMERS(CUST_ID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

**Insert at least 4 customers and 5 orders (make sure some customers have no orders).**

```
mysql> INSERT INTO CUSTOMERS (CUST_ID, NAME, CITY)
    -> VALUES
    -> (1, 'John', 'New York'),
    -> (2, 'Mary', 'Los Angeles'),
    -> (3, 'David', 'Chicago'),
    -> (4, 'Lucy', 'Houston'); -- Lucy has no orders
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO ORDERS (ORDER_ID, CUST_ID, ORDER_DATE, AMOUNT)
    -> VALUES
    -> (101, 1, '2025-08-01', 250.00),
    -> (102, 1, '2025-08-03', 300.00),
    -> (103, 2, '2025-08-04', 150.00),
    -> (104, 3, '2025-08-05', 200.00),
    -> (105, 2, '2025-08-06', 450.00);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

**Write a query using INNER JOIN to display customer names and their orders.**

```
mysql> SELECT C.NAME, O.ORDER_ID, O.ORDER_DATE, O.AMOUNT
    -> FROM CUSTOMERS C
    -> INNER JOIN ORDERS O ON C.CUST_ID = O.CUST_ID;
+-------+----------+------------+--------+
| NAME  | ORDER_ID | ORDER_DATE | AMOUNT |
+-------+----------+------------+--------+
| John  |      101 | 2025-08-01 | 250.00 |
| John  |      102 | 2025-08-03 | 300.00 |
| Mary  |      103 | 2025-08-04 | 150.00 |
| Mary  |      105 | 2025-08-06 | 450.00 |
| David |      104 | 2025-08-05 | 200.00 |
+-------+----------+------------+--------+
5 rows in set (0.00 sec)
```

**Write a query using LEFT JOIN to display all customers and their orders (include customers with no orders).**

```
mysql> SELECT C.NAME, O.ORDER_ID, O.ORDER_DATE, O.AMOUNT
    -> FROM CUSTOMERS C
    -> LEFT JOIN ORDERS O ON C.CUST_ID = O.CUST_ID;
+-------+----------+------------+--------+
| NAME  | ORDER_ID | ORDER_DATE | AMOUNT |
+-------+----------+------------+--------+
| John  |      101 | 2025-08-01 | 250.00 |
| John  |      102 | 2025-08-03 | 300.00 |
| Mary  |      103 | 2025-08-04 | 150.00 |
| Mary  |      105 | 2025-08-06 | 450.00 |
| David |      104 | 2025-08-05 | 200.00 |
| Lucy  |     NULL | NULL       |   NULL |
+-------+----------+------------+--------+
6 rows in set (0.00 sec)
```

**Write a query using RIGHT JOIN to display all orders and their customer names (even if customer data is missing).**

```
mysql> SELECT C.NAME, O.ORDER_ID, O.ORDER_DATE, O.AMOUNT
    -> FROM CUSTOMERS C
    -> RIGHT JOIN ORDERS O ON C.CUST_ID = O.CUST_ID;
+-------+----------+------------+--------+
| NAME  | ORDER_ID | ORDER_DATE | AMOUNT |
+-------+----------+------------+--------+
| John  |      101 | 2025-08-01 | 250.00 |
| John  |      102 | 2025-08-03 | 300.00 |
| Mary  |      103 | 2025-08-04 | 150.00 |
| David |      104 | 2025-08-05 | 200.00 |
| Mary  |      105 | 2025-08-06 | 450.00 |
+-------+----------+------------+--------+
5 rows in set (0.00 sec)
```

**Write a query using FULL OUTER JOIN to list all customers and orders, matching where possible**


```
mysql> SELECT C.NAME, O.ORDER_ID, O.ORDER_DATE, O.AMOUNT
    -> FROM CUSTOMERS C
    -> RIGHT JOIN ORDERS O ON C.CUST_ID = O.CUST_ID;
+-------+----------+------------+--------+
| NAME  | ORDER_ID | ORDER_DATE | AMOUNT |
+-------+----------+------------+--------+
| John  |      101 | 2025-08-01 | 250.00 |
| John  |      102 | 2025-08-03 | 300.00 |
| Mary  |      103 | 2025-08-04 | 150.00 |
| David |      104 | 2025-08-05 | 200.00 |
| Mary  |      105 | 2025-08-06 | 450.00 |
+-------+----------+------------+--------+
5 rows in set (0.00 sec)
```

**Create table DEPARTMENTS:**

```
CREATE TABLE DEPARTMENTS (
  DEPT_ID   INT PRIMARY KEY,
  DEPT_NAME VARCHAR(30)
);

CREATE TABLE EMPLOYEES (
  EMP_ID    INT PRIMARY KEY,
  EMP_NAME  VARCHAR(30),
  DEPT_ID   INT,
  SALARY    DECIMAL(10,2),
  FOREIGN KEY (DEPT_ID) REFERENCES
DEPARTMENTS(DEPT_ID)
);
```

```
mysql> CREATE TABLE DEPARTMENTS (
    ->    DEPT_ID   INT PRIMARY KEY,
    ->    DEPT_NAME VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE EMPLOYEES (
    ->    EMP_ID    INT PRIMARY KEY,
    ->    EMP_NAME  VARCHAR(30),
    ->    DEPT_ID   INT,
    ->    SALARY    DECIMAL(10,2),
    ->    FOREIGN KEY (DEPT_ID) REFERENCES DEPARTMENTS(DEPT_ID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

**Insert 3 departments and 5 employees (make sure one department has no employees).**

```
mysql> -- Insert departments (Dept 3 will have no employees)
mysql> INSERT INTO DEPARTMENTS (DEPT_ID, DEPT_NAME)
    -> VALUES
    -> (1, 'HR'),
    -> (2, 'IT'),
    -> (3, 'Finance');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert employees (one has NULL department)
mysql> INSERT INTO EMPLOYEES (EMP_ID, EMP_NAME, DEPT_ID, SALARY)
    -> VALUES
    -> (101, 'Alice', 1, 50000.00),
    -> (102, 'Bob', 2, 60000.00),
    -> (103, 'Charlie', 2, 55000.00),
    -> (104, 'Diana', 1, 52000.00),
    -> (105, 'Eve', NULL, 45000.00);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

**Write an INNER JOIN query to list employees and their department names.**

```
mysql> SELECT E.EMP_NAME, D.DEPT_NAME
    -> FROM EMPLOYEES E
    -> INNER JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;
+----------+-----------+
| EMP_NAME | DEPT_NAME |
+----------+-----------+
| Alice    | HR        |
| Diana    | HR        |
| Bob      | IT        |
| Charlie  | IT        |
+----------+-----------+
4 rows in set (0.00 sec)
```

**Write a LEFT JOIN to display all employees with their departments (show NULL if no department assigned).**

```
mysql> SELECT E.EMP_NAME, D.DEPT_NAME
    -> FROM EMPLOYEES E
    -> LEFT JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;
+----------+-----------+
| EMP_NAME | DEPT_NAME |
+----------+-----------+
| Alice    | HR        |
| Bob      | IT        |
| Charlie  | IT        |
| Diana    | HR        |
| Eve      | NULL      |
+----------+-----------+
5 rows in set (0.00 sec)
```

**Write a RIGHT JOIN to display all departments and the employees in them (show NULL if no employee).**

```
mysql> SELECT E.EMP_NAME, D.DEPT_NAME
    -> FROM EMPLOYEES E
    -> RIGHT JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;
+----------+-----------+
| EMP_NAME | DEPT_NAME |
+----------+-----------+
| Alice    | HR        |
| Diana    | HR        |
| Bob      | IT        |
| Charlie  | IT        |
| NULL     | Finance   |
+----------+-----------+
5 rows in set (0.00 sec)
```

**Write a FULL OUTER JOIN to display all employees and departments.**

```
mysql> SELECT E.EMP_NAME, D.DEPT_NAME
    -> FROM EMPLOYEES E
    -> LEFT JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID
    ->
    -> UNION ALL
    ->
    -> SELECT E.EMP_NAME, D.DEPT_NAME
    -> FROM EMPLOYEES E
    -> RIGHT JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID
    -> WHERE E.DEPT_ID IS NULL;
+----------+-----------+
| EMP_NAME | DEPT_NAME |
+----------+-----------+
| Alice    | HR        |
| Bob      | IT        |
| Charlie  | IT        |
| Diana    | HR        |
| Eve      | NULL      |
| NULL     | Finance   |
+----------+-----------+
6 rows in set (0.06 sec)
```

## Q5, Online Retail Database (End-to-End)

## Create all tables with proper primary keys, foreign keys, and constraints (check, not null, unique).

## Tables:

## CUSTOMERS(Add EMAIL to CUSTOMERS with a UNIQUE constraint.)

```
mysql> CREATE TABLE CUSTOMERS (
    ->     CUST_ID   INT PRIMARY KEY AUTO_INCREMENT,
    ->     NAME      VARCHAR(50) NOT NULL,
    ->     EMAIL     VARCHAR(100) UNIQUE NOT NULL,
    ->     CITY      VARCHAR(30)
    -> );
Query OK, 0 rows affected (0.08 sec)
```

## PRODUCTS- (Add a CHECK constraint to ensure PRODUCT price > 0.

## )

```
mysql>
mysql> CREATE TABLE PRODUCTS (
    ->     PROD_ID   INT PRIMARY KEY AUTO_INCREMENT,
    ->     PROD_NAME VARCHAR(50) NOT NULL,
    ->     PRICE     DECIMAL(10,2) NOT NULL CHECK (PRICE > 0),
    ->     STOCK_QTY INT NOT NULL
    -> );
```

## ORDERS

```
mysql> CREATE TABLE ORDERS (
    ->    ORDER_ID   INT PRIMARY KEY AUTO_INCREMENT,
    ->    CUST_ID    INT NOT NULL,
    ->    ORDER_DATE DATE NOT NULL,
    ->    FOREIGN KEY (CUST_ID) REFERENCES CUSTOMERS(CUST_ID)
    -> );
```

## ORDER_ITEMS

```
mysql> CREATE TABLE ORDER_ITEMS (
    ->    ITEM_ID   INT PRIMARY KEY AUTO_INCREMENT,
    ->    ORDER_ID  INT NOT NULL,
    ->    PROD_ID   INT NOT NULL,
    ->    QTY       INT NOT NULL CHECK (QTY > 0),
    ->    FOREIGN KEY (ORDER_ID) REFERENCES ORDERS(ORDER_ID),
    ->    FOREIGN KEY (PROD_ID) REFERENCES PRODUCTS(PROD_ID)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

**Insert sample data (at least 5–10 rows per table).**

**Create an INNER JOIN query to get all customer orders with order details and total amount.**

```
mysql> SELECT C.NAME, O.ORDER_ID, P.PROD_NAME, OI.QTY,
    ->        (P.PRICE * OI.QTY) AS TOTAL
    -> FROM CUSTOMERS C
    -> INNER JOIN ORDERS O ON C.CUST_ID = O.CUST_ID
    -> INNER JOIN ORDER_ITEMS OI ON O.ORDER_ID = OI.ORDER_ID
    -> INNER JOIN PRODUCTS P ON OI.PROD_ID = P.PROD_ID;
+-------+----------+------------+-----+---------+
| NAME  | ORDER_ID | PROD_NAME  | QTY | TOTAL   |
+-------+----------+------------+-----+---------+
| John  |        1 | Laptop     |   1 | 1000.00 |
| John  |        1 | Mouse      |   2 |  100.00 |
| John  |        3 | Tablet     |   2 |  800.00 |
| Mary  |        2 | Phone      |   1 |  600.00 |
| David |        4 | Headphones |   1 |  150.00 |
+-------+----------+------------+-----+---------+
5 rows in set (0.00 sec)
```

**Create a LEFT JOIN query to find customers who haven't placed any orders.**

```
mysql> SELECT C.NAME, O.ORDER_ID
    -> FROM CUSTOMERS C
    -> LEFT JOIN ORDERS O ON C.CUST_ID = O.CUST_ID
    -> WHERE O.ORDER_ID IS NULL;
+------+----------+
| NAME | ORDER_ID |
+------+----------+
| Lucy |     NULL |
| Eve  |     NULL |
+------+----------+
2 rows in set (0.00 sec)
```

**Use a FULL OUTER JOIN to list all products and their order quantities (show NULL where not ordered).**

```
mysql> SELECT P.PROD_NAME, SUM(OI.QTY) AS TOTAL_QTY
    -> FROM PRODUCTS P
    -> LEFT JOIN ORDER_ITEMS OI ON P.PROD_ID = OI.PROD_ID
    -> GROUP BY P.PROD_NAME
    ->
    -> UNION
    ->
    -> SELECT P.PROD_NAME, SUM(OI.QTY) AS TOTAL_QTY
    -> FROM PRODUCTS P
    -> RIGHT JOIN ORDER_ITEMS OI ON P.PROD_ID = OI.PROD_ID
    -> GROUP BY P.PROD_NAME;
+------------+-----------+
| PROD_NAME  | TOTAL_QTY |
+------------+-----------+
| Laptop     |         1 |
| Phone      |         1 |
| Tablet     |         2 |
| Headphones |         1 |
| Mouse      |         2 |
+------------+-----------+
5 rows in set (0.00 sec)
```

**Write a subquery to find products whose price is above the average product price.**

```
mysql> SELECT PROD_NAME, PRICE
    -> FROM PRODUCTS
    -> WHERE PRICE > (SELECT AVG(PRICE) FROM PRODUCTS);
+-----------+---------+
| PROD_NAME | PRICE   |
+-----------+---------+
| Laptop    | 1000.00 |
| Phone     |  600.00 |
+-----------+---------+
2 rows in set (0.00 sec)
```

**Write a query with GROUP BY to get total sales per product.**

```
mysql> SELECT P.PROD_NAME, SUM(OI.QTY * P.PRICE) AS TOTAL_SALES
    -> FROM PRODUCTS P
    -> JOIN ORDER_ITEMS OI ON P.PROD_ID = OI.PROD_ID
    -> GROUP BY P.PROD_NAME;
+------------+-------------+
| PROD_NAME  | TOTAL_SALES |
+------------+-------------+
| Laptop     |     1000.00 |
| Phone      |      600.00 |
| Tablet     |      800.00 |
| Headphones |      150.00 |
| Mouse      |      100.00 |
+------------+-------------+
5 rows in set (0.00 sec)
```

**Create a view CUSTOMER_ORDER_SUMMARY showing each customer's total orders and total amount spent.**

```
mysql> CREATE VIEW CUSTOMER_ORDER_SUMMARY AS
    -> SELECT C.CUST_ID, C.NAME,
    ->        COUNT(O.ORDER_ID) AS TOTAL_ORDERS,
    ->        SUM(P.PRICE * OI.QTY) AS TOTAL_SPENT
    -> FROM CUSTOMERS C
    -> LEFT JOIN ORDERS O ON C.CUST_ID = O.CUST_ID
    -> LEFT JOIN ORDER_ITEMS OI ON O.ORDER_ID = OI.ORDER_ID
    -> LEFT JOIN PRODUCTS P ON OI.PROD_ID = P.PROD_ID
    -> GROUP BY C.CUST_ID, C.NAME;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM CUSTOMER_ORDER_SUMMARY;
+---------+-------+--------------+-------------+
| CUST_ID | NAME  | TOTAL_ORDERS | TOTAL_SPENT |
+---------+-------+--------------+-------------+
|       1 | John  |            3 |     1900.00 |
|       2 | Mary  |            1 |      600.00 |
|       3 | David |            1 |      150.00 |
|       4 | Lucy  |            0 |        NULL |
|       5 | Eve   |            0 |        NULL |
+---------+-------+--------------+-------------+
5 rows in set (0.00 sec)
```

**Create an index on ORDER_DATE in the ORDERS table.**

```
mysql> CREATE INDEX idx_order_date ON ORDERS(ORDER_DATE);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

## Create a trigger to automatically reduce PRODUCT.STOCK_QTY when a new order item is inserted.

```
mysql> CREATE TRIGGER reduce_stock_after_insert
    -> AFTER INSERT ON ORDER_ITEMS
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE PRODUCTS
    ->     SET STOCK_QTY = STOCK_QTY - NEW.QTY
    ->     WHERE PROD_ID = NEW.PROD_ID;
    -> END$$
Query OK, 0 rows affected (0.01 sec)
```

## Create a stored procedure to insert a new order with multiple order items in a transaction.

```
mysql>      DELIMITER $$
mysql> CREATE PROCEDURE PlaceOrder(
    ->     IN p_cust_id INT,
    ->     IN p_order_date DATE
    -> )
    -> BEGIN
    ->     DECLARE new_order_id INT;
    ->
    ->
    ->     INSERT INTO ORDERS (CUST_ID, ORDER_DATE)
    ->     VALUES (p_cust_id, p_order_date);
    ->     SET new_order_id = LAST_INSERT_ID();
    ->
    ->
    ->     INSERT INTO ORDER_ITEMS (ORDER_ID, PROD_ID, QTY) VALUES (new_order_id, 1, 1);
    ->     INSERT INTO ORDER_ITEMS (ORDER_ID, PROD_ID, QTY) VALUES (new_order_id, 2, 2);
    -> END$$
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
```

**Q6)**

**University Database**

**Tables:**

**STUDENTS**

**COURSES**

**ENROLLMENTS**

**PROFESSORS**

**Tasks:**

**Create all tables with foreign keys and ON DELETE CASCADE for enrollments.**

```
mysql> CREATE TABLE STUDENTS (
    ->     STUD_ID INT PRIMARY KEY,
    ->     NAME VARCHAR(50) NOT NULL,
    ->     DOB DATE NOT NULL
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE PROFESSORS (
    ->     PROF_ID INT PRIMARY KEY,
    ->     PROF_NAME VARCHAR(50) NOT NULL
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> CREATE TABLE COURSES (
    ->     COURSE_ID INT PRIMARY KEY,
    ->     COURSE_NAME VARCHAR(50) NOT NULL,
    ->     PROF_ID INT,
    ->     FOREIGN KEY (PROF_ID) REFERENCES PROFESSORS(PROF_ID)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> CREATE TABLE ENROLLMENTS (
    ->     ENROLL_ID INT PRIMARY KEY AUTO_INCREMENT,
    ->     STUD_ID INT,
    ->     COURSE_ID INT,
    ->     MARKS DECIMAL(5,2),
    ->     FOREIGN KEY (STUD_ID) REFERENCES STUDENTS(STUD_ID) ON DELETE CASCADE,
    ->     FOREIGN KEY (COURSE_ID) REFERENCES COURSES(COURSE_ID) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.04 sec)
```

## Add a CHECK constraint so that a student's DOB ensures age ≥ 16.

```
mysql> DELIMITER $$
mysql>
mysql> CREATE TRIGGER trg_check_age
    -> BEFORE INSERT ON STUDENTS
    -> FOR EACH ROW
    -> BEGIN
    ->     IF TIMESTAMPDIFF(YEAR, NEW.DOB, CURDATE()) < 16 THEN
    ->         SIGNAL SQLSTATE '45000'
    ->             SET MESSAGE_TEXT = 'Student must be at least 16 years old';
    ->     END IF;
    -> END$$
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> DELIMITER ;
```

**Insert at least 6 students, 4 professors, and 5 courses, and assign professors to courses.**

```
mysql> -- Students
mysql> INSERT INTO STUDENTS VALUES
    -> (1, 'Alice', '2000-05-15'),
    -> (2, 'Bob', '1999-07-20'),
    -> (3, 'Charlie', '2002-03-10'),
    -> (4, 'David', '2001-09-05'),
    -> (5, 'Eva', '1998-12-12'),
    -> (6, 'Frank', '2003-06-25');
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Professors
mysql> INSERT INTO PROFESSORS VALUES
    -> (1, 'Prof. Smith'),
    -> (2, 'Prof. Johnson'),
    -> (3, 'Prof. Brown'),
    -> (4, 'Prof. Taylor');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Courses
mysql> INSERT INTO COURSES VALUES
    -> (101, 'Mathematics', 1),
    -> (102, 'Physics', 2),
    -> (103, 'Chemistry', 3),
    -> (104, 'Biology', 4),
    -> (105, 'Computer Science', 1);
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Enrollments
mysql> INSERT INTO ENROLLMENTS (STUD_ID, COURSE_ID, MARKS) VALUES
```

**Write an INNER JOIN query to list students with the courses they are taking.**

```
mysql> SELECT S.NAME AS Student, C.COURSE_NAME
    -> FROM STUDENTS S
    -> INNER JOIN ENROLLMENTS E ON S.STUD_ID = E.STUD_ID
    -> INNER JOIN COURSES C ON E.COURSE_ID = C.COURSE_ID;
+---------+------------------+
| Student | COURSE_NAME      |
+---------+------------------+
| Alice   | Mathematics      |
| Bob     | Physics          |
| Charlie | Mathematics      |
| David   | Chemistry        |
| Eva     | Computer Science |
+---------+------------------+
5 rows in set (0.00 sec)
```

**Write a LEFT JOIN to find courses with no enrolled students.**

```
mysql> SELECT C.COURSE_NAME, S.NAME AS Student
    -> FROM COURSES C
    -> LEFT JOIN ENROLLMENTS E ON C.COURSE_ID = E.COURSE_ID
    -> LEFT JOIN STUDENTS S ON E.STUD_ID = S.STUD_ID;
+------------------+---------+
| COURSE_NAME      | Student |
+------------------+---------+
| Mathematics      | Alice   |
| Mathematics      | Charlie |
| Physics          | Bob     |
| Chemistry        | David   |
| Biology          | NULL    |
| Computer Science | Eva     |
+------------------+---------+
6 rows in set (0.00 sec)
```

**Write a RIGHT JOIN to list all students and their professors (through the course).**

```
mysql> SELECT S.NAME AS Student, P.PROF_NAME AS Professor
    -> FROM ENROLLMENTS E
    -> RIGHT JOIN STUDENTS S ON E.STUD_ID = S.STUD_ID
    -> LEFT JOIN COURSES C ON E.COURSE_ID = C.COURSE_ID
    -> LEFT JOIN PROFESSORS P ON C.PROF_ID = P.PROF_ID;
+---------+---------------+
| Student | Professor     |
+---------+---------------+
| Alice   | Prof. Smith   |
| Bob     | Prof. Johnson |
| Charlie | Prof. Smith   |
| David   | Prof. Brown   |
| Eva     | Prof. Smith   |
| Frank   | NULL          |
+---------+---------------+
6 rows in set (0.00 sec)
```

**Write a FULL OUTER JOIN between STUDENTS and COURSES to see all students and courses (even if not related).**

```
mysql> SELECT S.NAME AS Student, C.COURSE_NAME
    -> FROM STUDENTS S
    -> LEFT JOIN ENROLLMENTS E ON S.STUD_ID = E.STUD_ID
    -> LEFT JOIN COURSES C ON E.COURSE_ID = C.COURSE_ID
    ->
    -> UNION
    ->
    -> SELECT S.NAME AS Student, C.COURSE_NAME
    -> FROM COURSES C
    -> LEFT JOIN ENROLLMENTS E ON C.COURSE_ID = E.COURSE_ID
    -> LEFT JOIN STUDENTS S ON E.STUD_ID = S.STUD_ID;
+---------+------------------+
| Student | COURSE_NAME      |
+---------+------------------+
| Alice   | Mathematics      |
| Bob     | Physics          |
| Charlie | Mathematics      |
| David   | Chemistry        |
| Eva     | Computer Science |
| Frank   | NULL             |
| NULL    | Biology          |
+---------+------------------+
7 rows in set (0.00 sec)
```

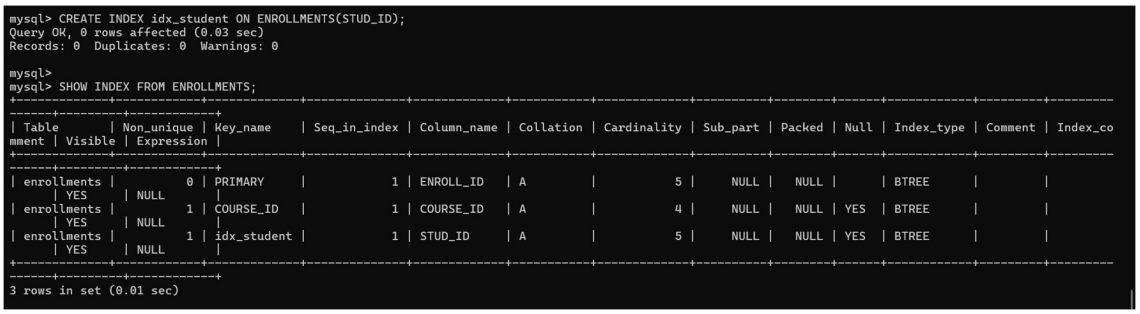
**Write a correlated subquery to find students with marks above their course average.**

```
mysql> SELECT S.NAME, C.COURSE_NAME, E.MARKS
    -> FROM ENROLLMENTS E
    -> JOIN STUDENTS S ON E.STUD_ID = S.STUD_ID
    -> JOIN COURSES C ON E.COURSE_ID = C.COURSE_ID
    -> WHERE E.MARKS > (
    ->     SELECT AVG(MARKS)
    ->     FROM ENROLLMENTS
    ->     WHERE COURSE_ID = E.COURSE_ID
    -> );
+---------+-------------+-------+
| NAME    | COURSE_NAME | MARKS |
+---------+-------------+-------+
| Charlie | Mathematics | 90.00 |
+---------+-------------+-------+
1 row in set (0.00 sec)
```

**Create a view showing each professor and the number of students they are teaching.**

```
mysql> CREATE VIEW Professor_Student_Count AS
    -> SELECT P.PROF_NAME, COUNT(E.STUD_ID) AS Num_Students
    -> FROM PROFESSORS P
    -> LEFT JOIN COURSES C ON P.PROF_ID = C.PROF_ID
    -> LEFT JOIN ENROLLMENTS E ON C.COURSE_ID = E.COURSE_ID
    -> GROUP BY P.PROF_NAME;
Query OK, 0 rows affected (0.00 sec)
```

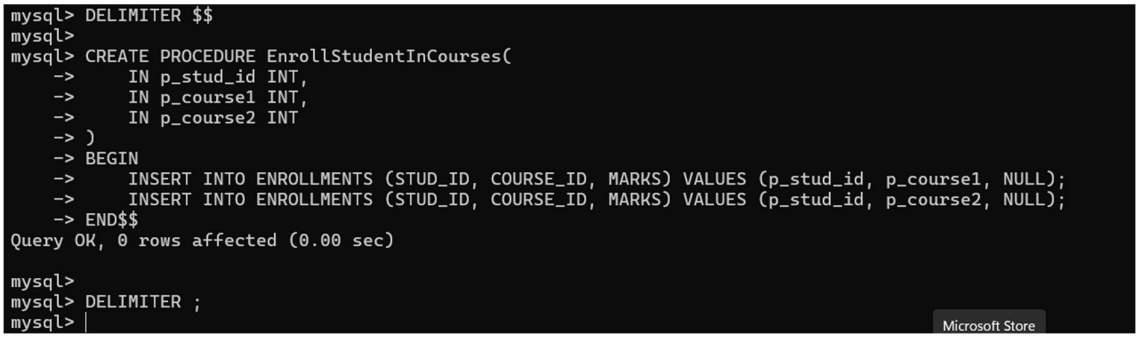
## Create an index on the ENROLLMENTS table for faster lookups by STUD_ID.



```
mysql> CREATE INDEX idx_student ON ENROLLMENTS(STUD_ID);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql> SHOW INDEX FROM ENROLLMENTS;
+-------------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------
------+---------+------------+
| Table       | Non_unique | Key_name    | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_co
mment | Visible | Expression |
+-------------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------
------+---------+------------+
| enrollments |          0 | PRIMARY     |            1 | ENROLL_ID   | A         |           5 |     NULL |   NULL |      | BTREE      |         |
      | YES     | NULL       |
| enrollments |          1 | COURSE_ID   |            1 | COURSE_ID   | A         |           4 |     NULL |   NULL | YES  | BTREE      |         |
      | YES     | NULL       |
| enrollments |          1 | idx_student |            1 | STUD_ID     | A         |           5 |     NULL |   NULL | YES  | BTREE      |         |
      | YES     | NULL       |
+-------------+------------+-------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------
------+---------+------------+
3 rows in set (0.01 sec)
```

## Write a stored procedure to enroll a student in multiple courses at once.



```
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE EnrollStudentInCourses(
    ->     IN p_stud_id INT,
    ->     IN p_course1 INT,
    ->     IN p_course2 INT
    -> )
    -> BEGIN
    ->     INSERT INTO ENROLLMENTS (STUD_ID, COURSE_ID, MARKS) VALUES (p_stud_id, p_course1, NULL);
    ->     INSERT INTO ENROLLMENTS (STUD_ID, COURSE_ID, MARKS) VALUES (p_stud_id, p_course2, NULL);
    -> END$$
Query OK, 0 rows affected (0.00 sec)

mysql>
mysql> DELIMITER ;
mysql>
```



## Create a trigger that prevents inserting an enrollment if the course already has 30 students.