Assignment On

Course Title: Database Management System

Course Code: CSEL-2204

[Assignment 2 - MySQL Functions]

Submitted To,

Professor **Dr. Md. Manowarul Islam**

Department of CSE

Jagannath University, Dhaka

Submitted By,

Atik Jawad

ID: B220305043

Department of Computer Science & Engineering

Jagannath University, Dhaka

Date of Submission: 5/11/2025

Table of Contents

Introduction

The purpose of this assignment is to explore and apply various **SQL functions and procedural constructs** available in a Database Management System (DBMS). It focuses on performing operations such as **string manipulation, numeric calculations, date and time handling, aggregate and conditional functions**, and implementing **stored procedures, loops, triggers, and error handling mechanisms** using SQL.

Table Creation & Data Insertion

Customers Table:

```sql
CREATE TABLE customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(50),
  email VARCHAR(50),
  home_phone VARCHAR(15),
  work_phone VARCHAR(15),
  alternate_phone VARCHAR(15)
);

INSERT INTO customers VALUES
(1, 'John Doe', 'john@example.com', '1234567890', '9876543210', NULL),
(2, 'Alice Smith', 'alice@example.com', NULL, '8765432109', '7890123456'),
(3, 'Bob Brown', 'bob@example.com', '5432167890', NULL, NULL);
```

Employees Table:

```sql
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  first_name VARCHAR(30),
  last_name VARCHAR(30),
  salary DECIMAL(10,2)
);

INSERT INTO employees VALUES
(1, 'Michael', 'Johnson', 55000),
(2, 'Sarah', 'Williams', 62000),
(3, 'Robert', 'Brown', 70000);
```

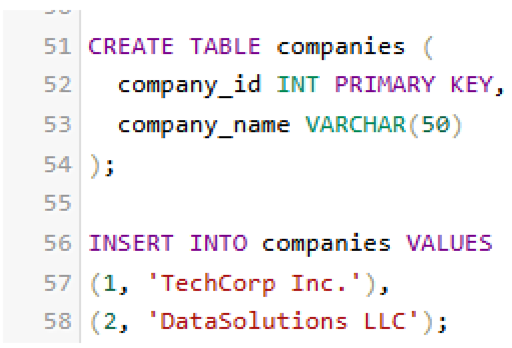
Products Table:

```sql
CREATE TABLE products (
  product_id INT PRIMARY KEY,
  product_name VARCHAR(50),
  price DECIMAL(10,2)
);

INSERT INTO products VALUES
(1, 'Laptop', 1200),
(2, 'Smartphone', 800),
(3, 'Keyboard', 50);
```

Orders Table:

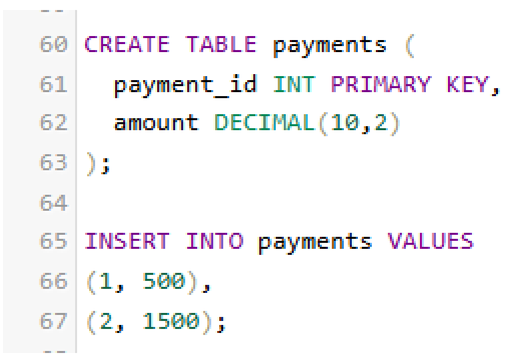
```sql
CREATE TABLE orders (
  order_id INT PRIMARY KEY,
  customer_id INT,
  order_date DATE,
  order_value DECIMAL(10,2),
  discount DECIMAL(10,2),
  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);

INSERT INTO orders VALUES
(1, 1, '2024-03-01', 1500, 50),
(2, 2, '2024-03-05', 3000, 100),
(3, 3, '2024-03-10', 500, NULL);
```

Companies Table:



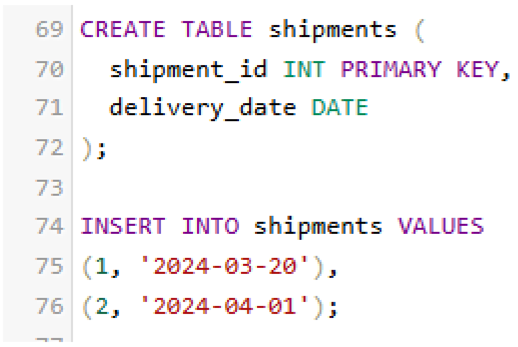
```sql
CREATE TABLE companies (
  company_id INT PRIMARY KEY,
  company_name VARCHAR(50)
);

INSERT INTO companies VALUES
(1, 'TechCorp Inc.'),
(2, 'DataSolutions LLC');
```

Payments Table:



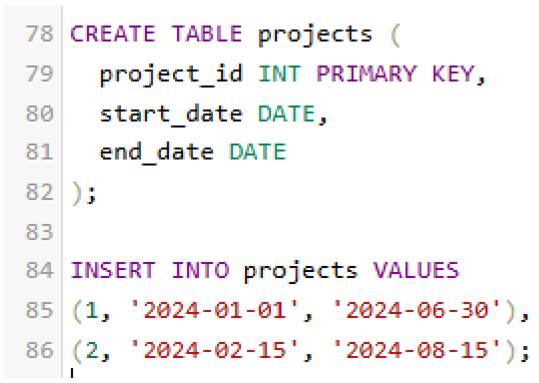
```sql
CREATE TABLE payments (
  payment_id INT PRIMARY KEY,
  amount DECIMAL(10,2)
);

INSERT INTO payments VALUES
(1, 500),
(2, 1500);
```

Shipments Table:



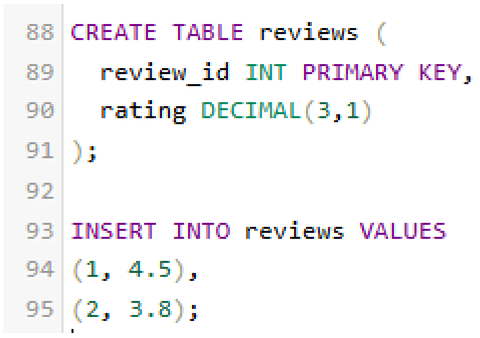
```sql
CREATE TABLE shipments (
  shipment_id INT PRIMARY KEY,
  delivery_date DATE
);

INSERT INTO shipments VALUES
(1, '2024-03-20'),
(2, '2024-04-01');
```

Projects Table:



```sql
CREATE TABLE projects (
  project_id INT PRIMARY KEY,
  start_date DATE,
  end_date DATE
);

INSERT INTO projects VALUES
(1, '2024-01-01', '2024-06-30'),
(2, '2024-02-15', '2024-08-15');
```

Reviews Table:

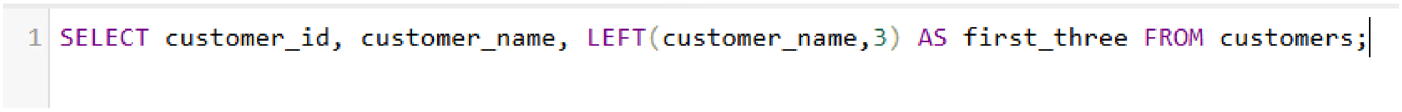


```sql
CREATE TABLE reviews (
  review_id INT PRIMARY KEY,
  rating DECIMAL(3,1)
);

INSERT INTO reviews VALUES
(1, 4.5),
(2, 3.8);
```

Part A – String Functions

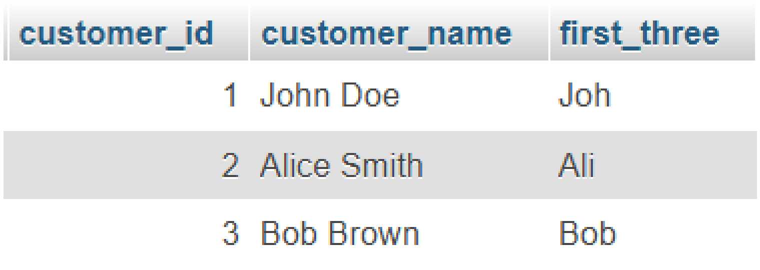
1 – Extract first three characters

SQL Query:



```
SELECT customer_id, customer_name, LEFT(customer_name,3) AS first_three FROM customers;
```

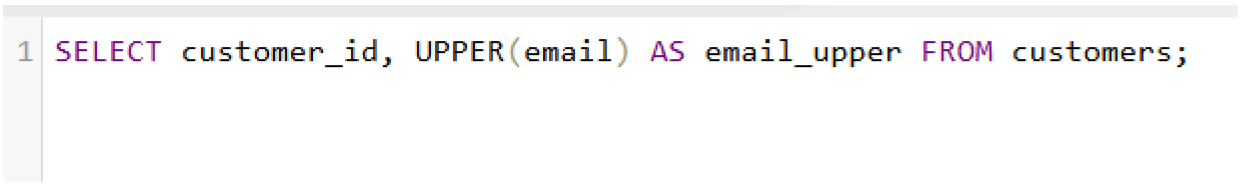
Explanation: Extracts the first 3 characters of each customer's name.



customer_id	customer_name	first_three
1	John Doe	Joh
2	Alice Smith	Ali
3	Bob Brown	Bob

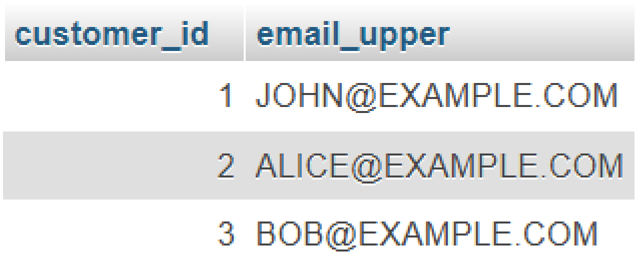
2 – Convert email to uppercase

SQL Query:



```
SELECT customer_id, UPPER(email) AS email_upper FROM customers;
```

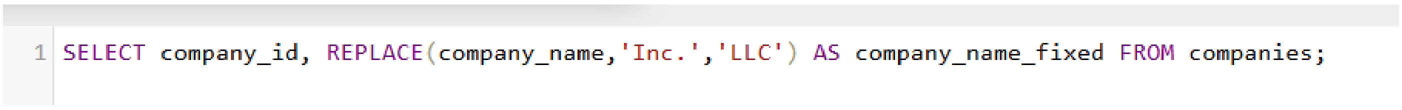
Explanation: Converts all email addresses to uppercase.



customer_id	email_upper
1	JOHN@EXAMPLE.COM
2	ALICE@EXAMPLE.COM
3	BOB@EXAMPLE.COM

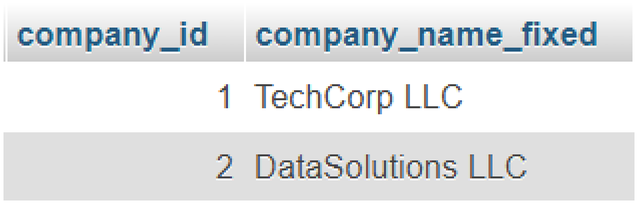
3 – Replace 'Inc.' with 'LLC'

SQL Query:



```
SELECT company_id, REPLACE(company_name,'Inc.','LLC') AS company_name_fixed FROM companies;
```

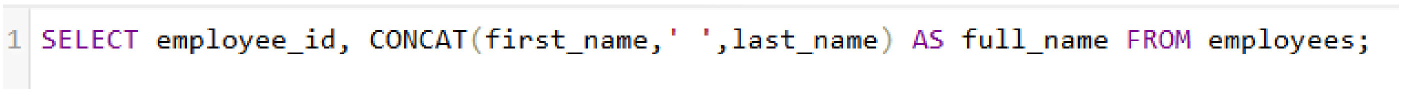
Explanation: Replaces 'Inc.' with 'LLC' in company names.



company_id	company_name_fixed
1	TechCorp LLC
2	DataSolutions LLC

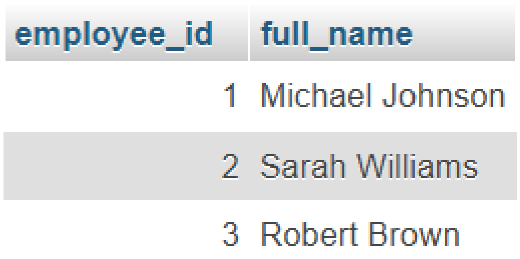
4 – Concatenate first and last name

SQL Query:



```sql
SELECT employee_id, CONCAT(first_name,' ',last_name) AS full_name FROM employees;
```

Explanation: Combines first name and last name into one full name.

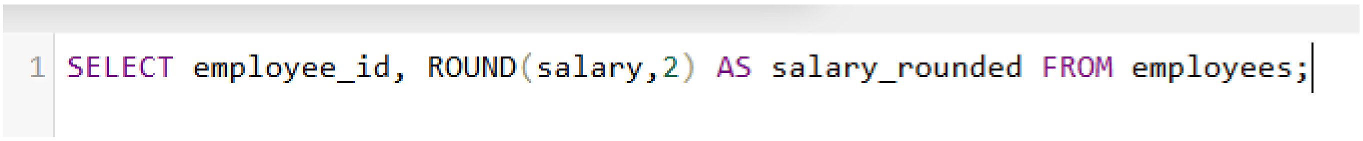


employee_id	full_name
1	Michael Johnson
2	Sarah Williams
3	Robert Brown

Part B – Numeric Functions

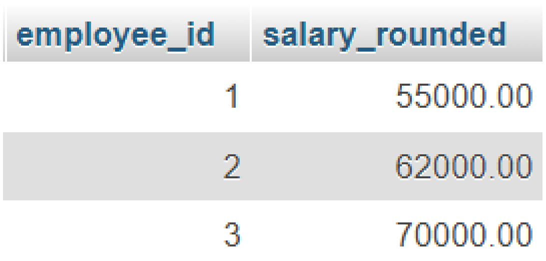
1 – Round salary

SQL Query:



```sql
SELECT employee_id, ROUND(salary,2) AS salary_rounded FROM employees;
```

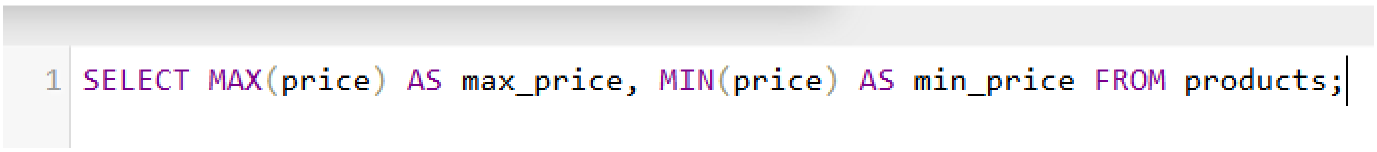
Explanation: Rounds salary to two decimal places.



employee_id	salary_rounded
1	55000.00
2	62000.00
3	70000.00

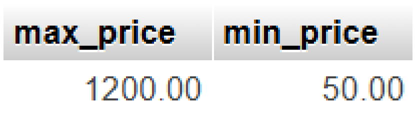
2 – Max and Min price

SQL Query:



```sql
SELECT MAX(price) AS max_price, MIN(price) AS min_price FROM products;
```

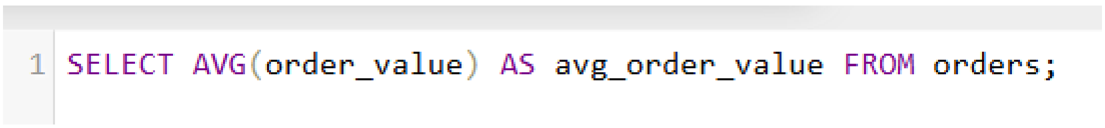
Explanation: Displays highest and lowest product prices.



max_price	min_price
1200.00	50.00

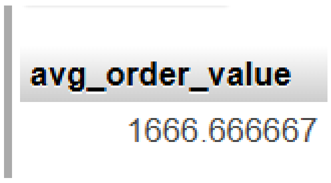
3 – Average order value

SQL Query:



```sql
SELECT AVG(order_value) AS avg_order_value FROM orders;
```

Explanation: Calculates average order value from orders table.



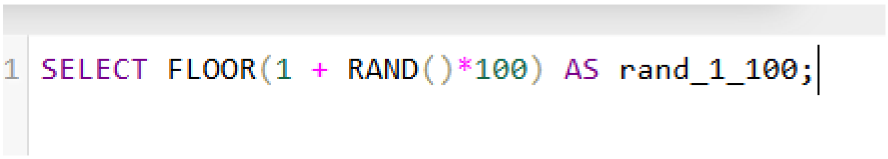
avg_order_value
1666.666667

Random = FLOOR(X + (RAND() * (Y - X + 1)))

General formula for randoms

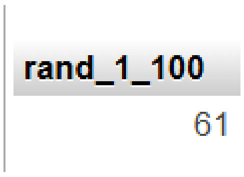
4 – Random number between 1 and 100

SQL Query:



```sql
SELECT FLOOR(1 + RAND()*100) AS rand_1_100;
```

Explanation: Generates a random integer between 1 and 100.

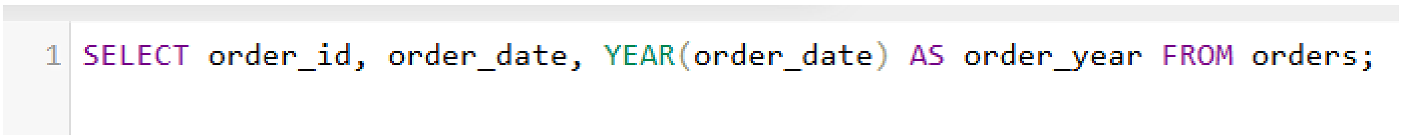


rand_1_100
61

Part C – Date and Time Functions

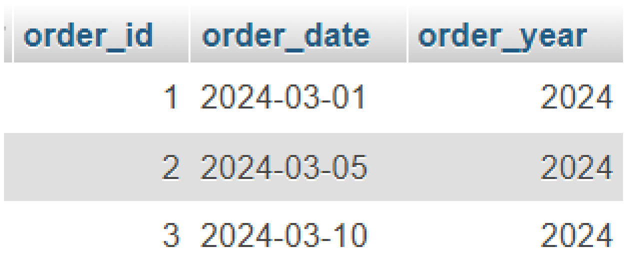
1 – Extract year from order date

SQL Query:



```sql
SELECT order_id, order_date, YEAR(order_date) AS order_year FROM orders;
```

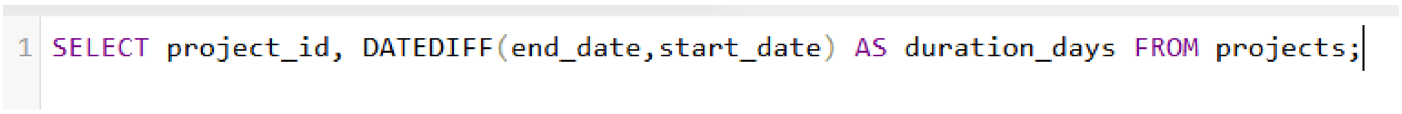
Explanation: Extracts year portion from the date of order.



order_id	order_date	order_year
1	2024-03-01	2024
2	2024-03-05	2024
3	2024-03-10	2024

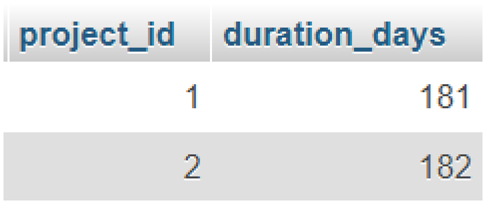
2 – Calculate project duration in days

SQL Query:



```sql
SELECT project_id, DATEDIFF(end_date,start_date) AS duration_days FROM projects;
```

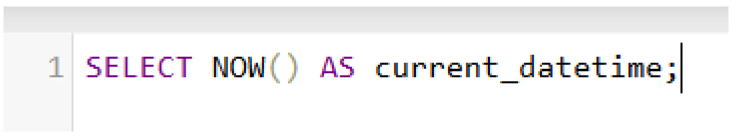
Explanation: Finds number of days between start and end date.



project_id	duration_days
1	181
2	182

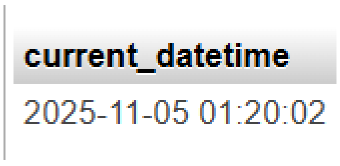
3 – Current system date and time

SQL Query:



```sql
SELECT NOW() AS current_datetime;
```

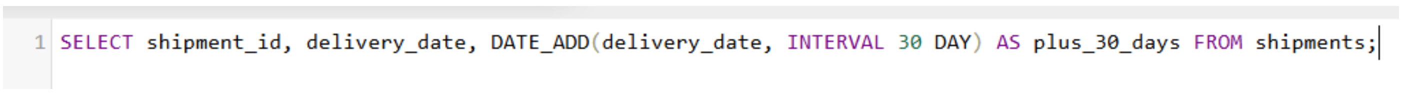
Explanation: Returns the current system date and time.



current_datetime
2025-11-05 01:20:02

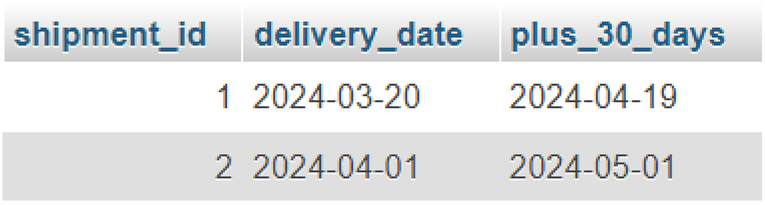
4 – Add 30 days to shipment date

SQL Query:



```sql
SELECT shipment_id, delivery_date, DATE_ADD(delivery_date, INTERVAL 30 DAY) AS plus_30_days FROM shipments;
```

Explanation: Adds 30 days to each shipment delivery date.

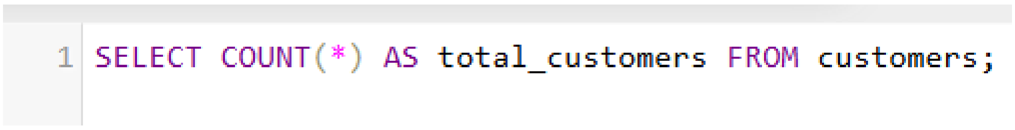


shipment_id	delivery_date	plus_30_days
1	2024-03-20	2024-04-19
2	2024-04-01	2024-05-01

Part D – Aggregate Functions

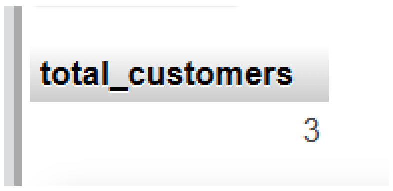
1 – Count total customers

SQL Query:



```sql
SELECT COUNT(*) AS total_customers FROM customers;
```

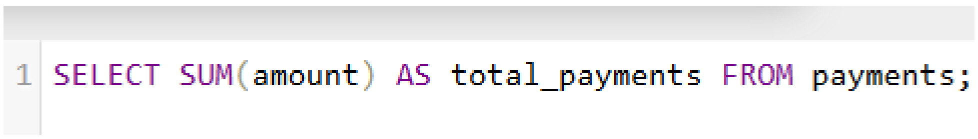
Explanation: Counts the total number of customers in the table.



total_customers
3

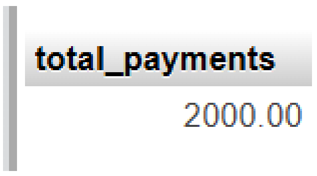
2 – Sum of payment amounts

SQL Query:



```sql
SELECT SUM(amount) AS total_payments FROM payments;
```

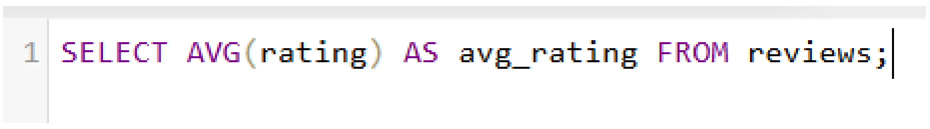
Explanation: Calculates total of all payment amounts.



total_payments
2000.00

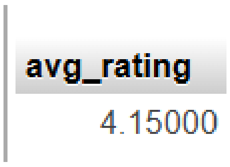
3 – Average rating

SQL Query:



```sql
SELECT AVG(rating) AS avg_rating FROM reviews;
```

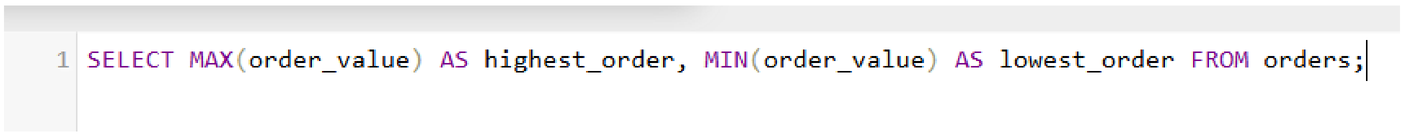
Explanation: Calculates the average of all ratings given by users.



avg_rating
4.15000

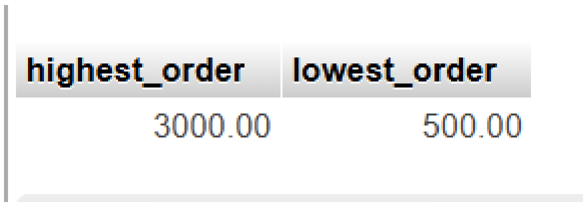
4 – Highest and lowest order value

SQL Query:



```sql
SELECT MAX(order_value) AS highest_order, MIN(order_value) AS lowest_order FROM orders;
```

Explanation: Finds highest and lowest order values.

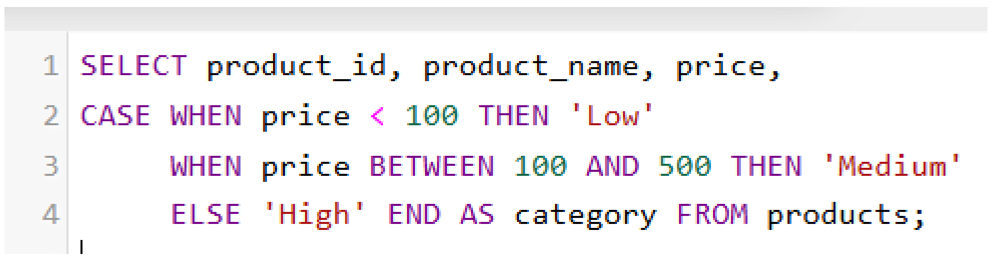


highest_order	lowest_order
3000.00	500.00

Part E – Conditional Functions

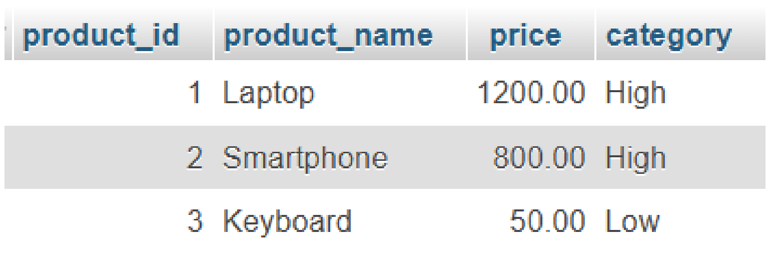
1 – Product price category (CASE)

SQL Query:



```sql
SELECT product_id, product_name, price,
CASE WHEN price < 100 THEN 'Low'
     WHEN price BETWEEN 100 AND 500 THEN 'Medium'
     ELSE 'High' END AS category FROM products;
```

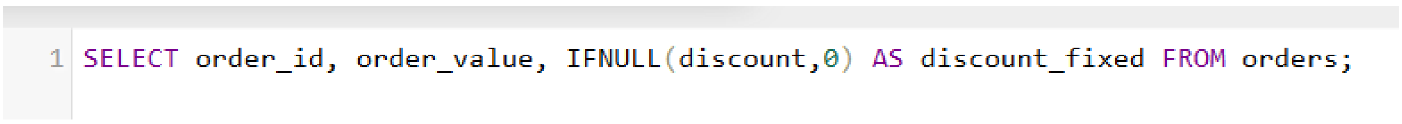
Explanation: Categorizes products by their price range.



product_id	product_name	price	category
1	Laptop	1200.00	High
2	Smartphone	800.00	High
3	Keyboard	50.00	Low

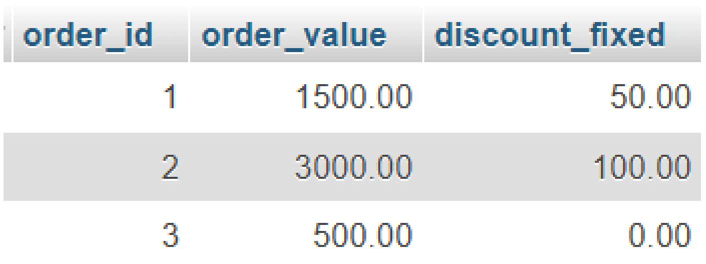
2 – Replace NULL discount with 0 (IF*NULL*)

SQL Query:



```
SELECT order_id, order_value, IFNULL(discount,0) AS discount_fixed FROM orders;
```

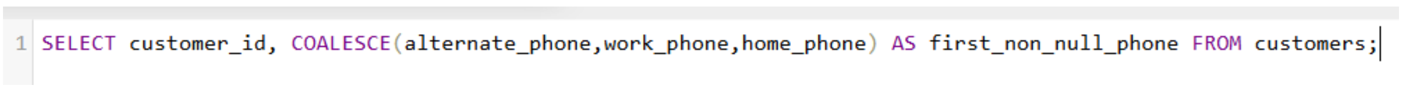
Explanation: Replaces NULL discount values with 0.



order_id	order_value	discount_fixed
1	1500.00	50.00
2	3000.00	100.00
3	500.00	0.00

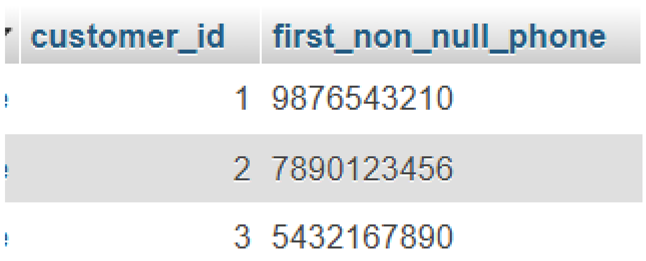
3 – Show first available phone (*COALESCE*)

SQL Query:



```
SELECT customer_id, COALESCE(alternate_phone,work_phone,home_phone) AS first_non_null_phone FROM customers;
```

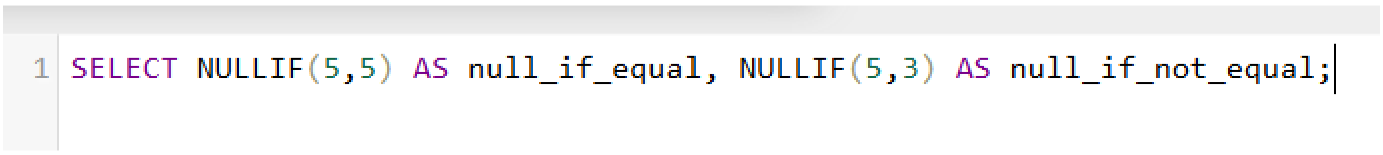
Explanation: Displays the first non-NULL phone number for each customer.



customer_id	first_non_null_phone
1	9876543210
2	7890123456
3	5432167890

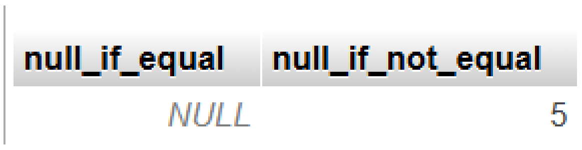
4 – NULLIF example

SQL Query:



```
SELECT NULLIF(5,5) AS null_if_equal, NULLIF(5,3) AS null_if_not_equal;
```

Explanation: Returns NULL if two values are equal, else returns first value.

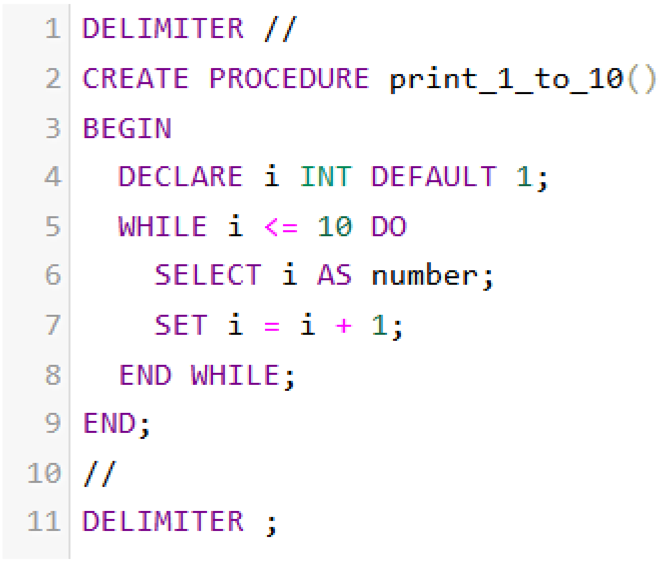


null_if_equal	null_if_not_equal
NULL	5

Part F – WHILE Loops / Stored Procedures

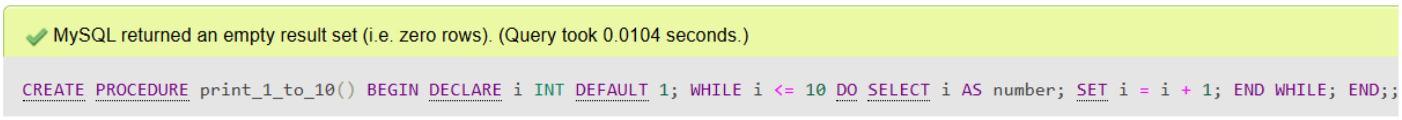
1 – Print numbers 1 to 10

SQL Query:



```sql
DELIMITER //
CREATE PROCEDURE print_1_to_10()
BEGIN
  DECLARE i INT DEFAULT 1;
  WHILE i <= 10 DO
    SELECT i AS number;
    SET i = i + 1;
  END WHILE;
END;
//
DELIMITER ;
```

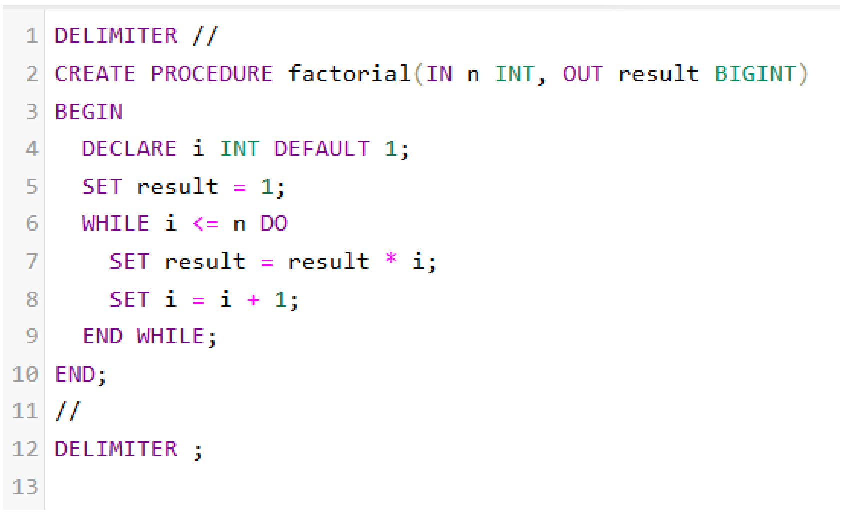
Explanation: Uses WHILE loop to print numbers from 1 to 10.



```
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0104 seconds.)

CREATE PROCEDURE print_1_to_10() BEGIN DECLARE i INT DEFAULT 1; WHILE i <= 10 DO SELECT i AS number; SET i = i + 1; END WHILE; END;;
```

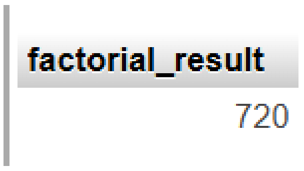
2 – Factorial of a number

SQL Query:



```sql
DELIMITER //
CREATE PROCEDURE factorial(IN n INT, OUT result BIGINT)
BEGIN
  DECLARE i INT DEFAULT 1;
  SET result = 1;
  WHILE i <= n DO
    SET result = result * i;
    SET i = i + 1;
  END WHILE;
END;
//
DELIMITER ;

```

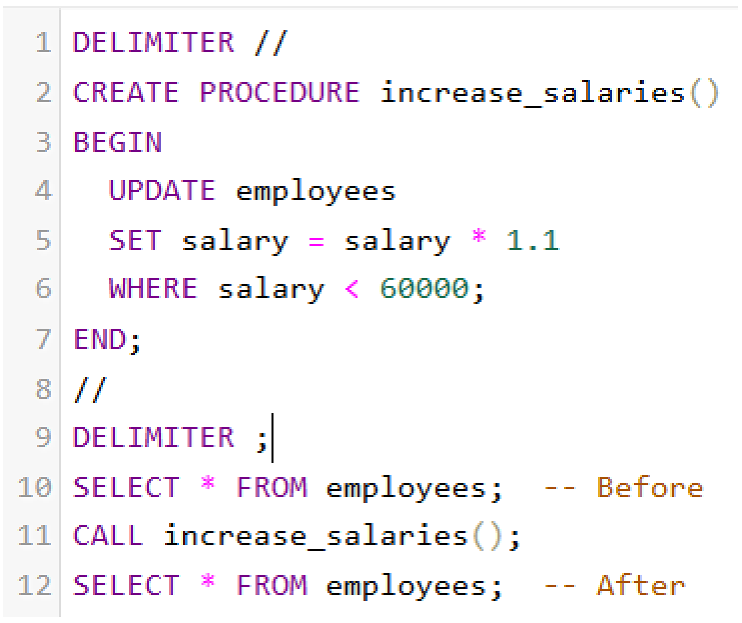
Explanation: Calculates factorial of given number using WHILE loop.



factorial_result
720

3 – Salary increase (10% if salary<60000)

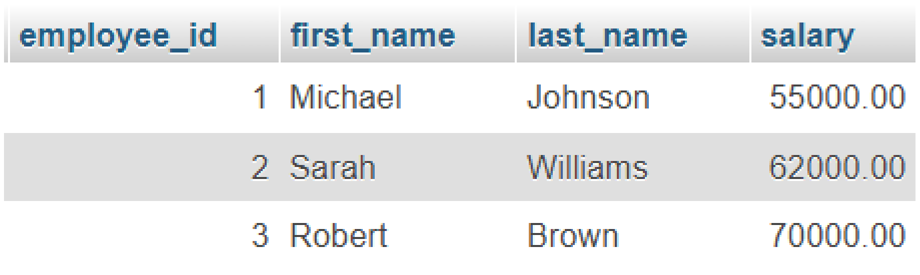
SQL Query:



```sql
DELIMITER //
CREATE PROCEDURE increase_salaries()
BEGIN
  UPDATE employees
  SET salary = salary * 1.1
  WHERE salary < 60000;
END;
//
DELIMITER ;
SELECT * FROM employees;  -- Before
CALL increase_salaries();
SELECT * FROM employees;  -- After
```

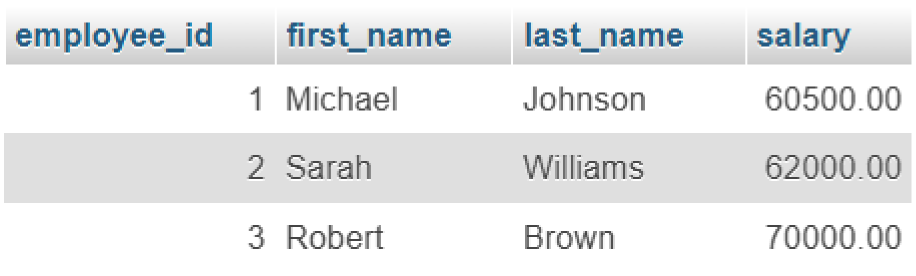
Explanation: Increases salary by 10% for employees earning below 60000.

Before:



employee_id	first_name	last_name	salary
1	Michael	Johnson	55000.00
2	Sarah	Williams	62000.00
3	Robert	Brown	70000.00

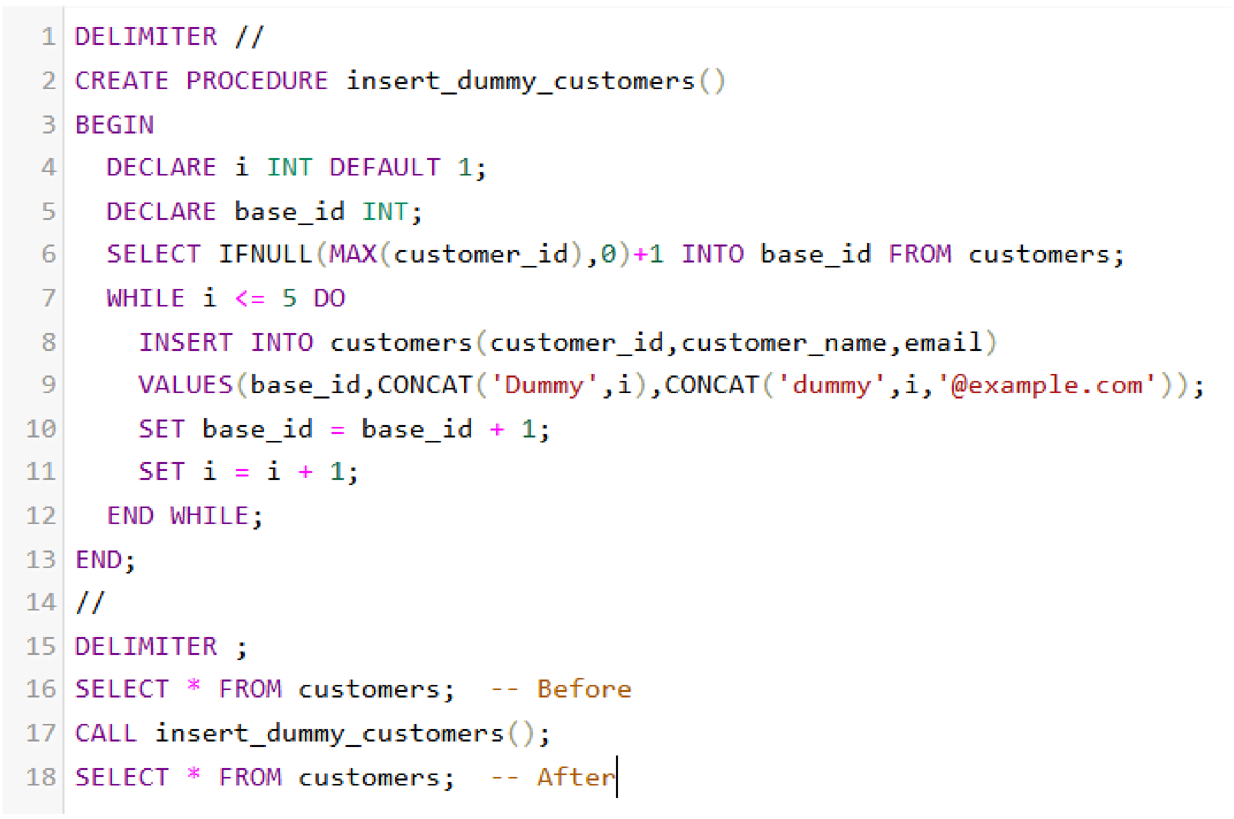
After:



employee_id	first_name	last_name	salary
1	Michael	Johnson	60500.00
2	Sarah	Williams	62000.00
3	Robert	Brown	70000.00

4 – Insert 5 dummy customers

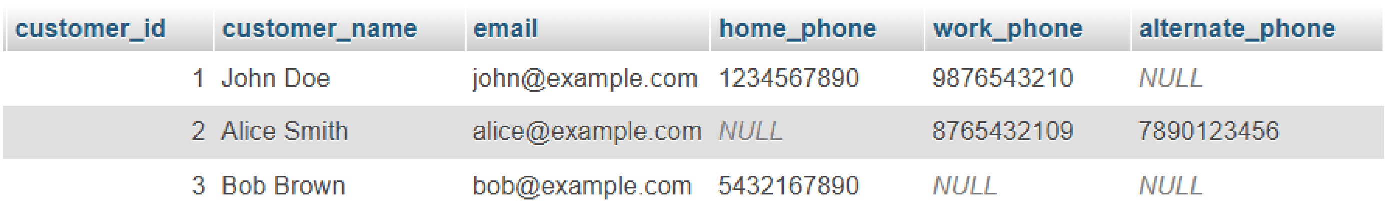
SQL Query:



```sql
DELIMITER //
CREATE PROCEDURE insert_dummy_customers()
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE base_id INT;
  SELECT IFNULL(MAX(customer_id),0)+1 INTO base_id FROM customers;
  WHILE i <= 5 DO
    INSERT INTO customers(customer_id,customer_name,email)
    VALUES(base_id,CONCAT('Dummy',i),CONCAT('dummy',i,'@example.com'));
    SET base_id = base_id + 1;
    SET i = i + 1;
  END WHILE;
END;
//
DELIMITER ;
SELECT * FROM customers;  -- Before
CALL insert_dummy_customers();
SELECT * FROM customers;  -- After
```

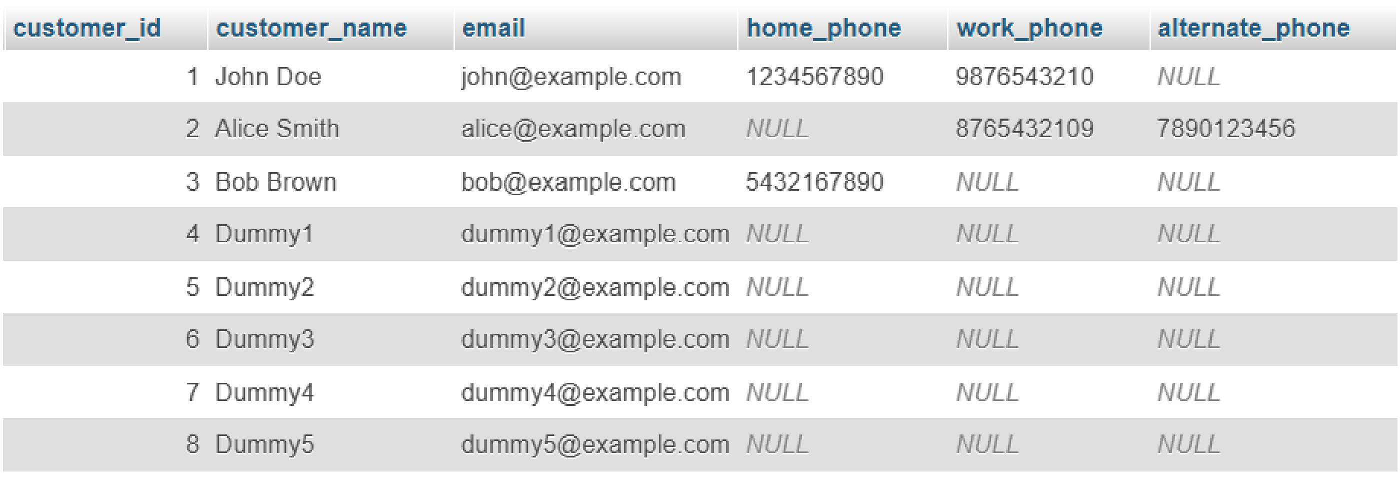
Explanation: Inserts 5 new dummy customers using a WHILE loop.

Before:



customer_id	customer_name	email	home_phone	work_phone	alternate_phone
1	John Doe	john@example.com	1234567890	9876543210	*NULL*
2	Alice Smith	alice@example.com	*NULL*	8765432109	7890123456
3	Bob Brown	bob@example.com	5432167890	*NULL*	*NULL*

After:

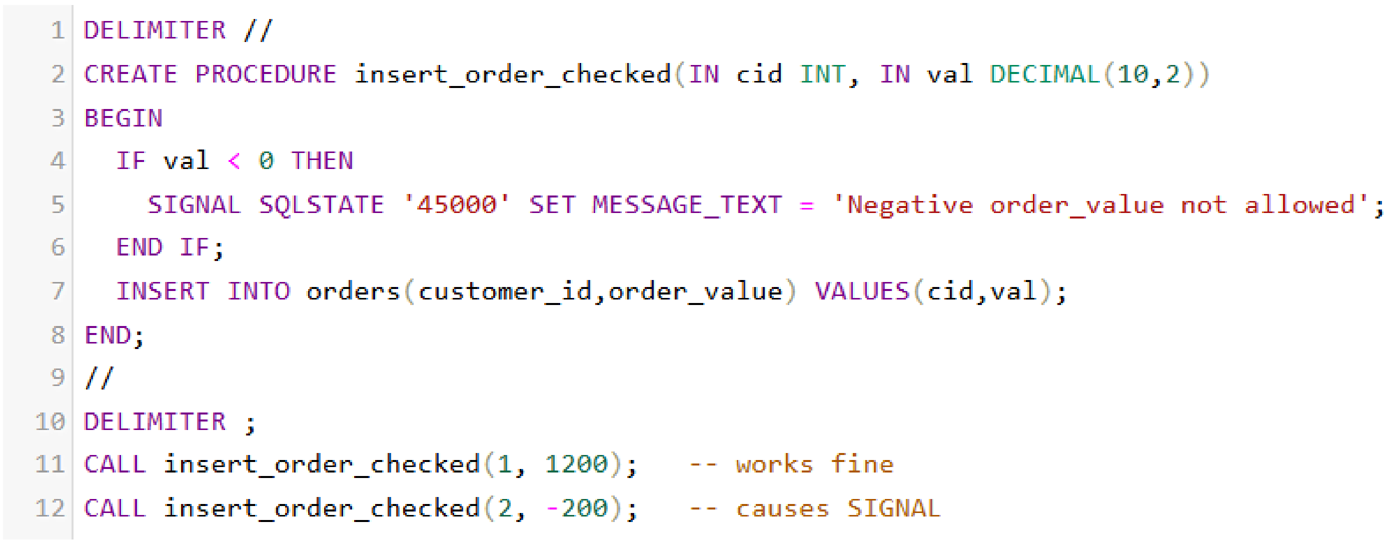


customer_id	customer_name	email	home_phone	work_phone	alternate_phone
1	John Doe	john@example.com	1234567890	9876543210	*NULL*
2	Alice Smith	alice@example.com	*NULL*	8765432109	7890123456
3	Bob Brown	bob@example.com	5432167890	*NULL*	*NULL*
4	Dummy1	dummy1@example.com	*NULL*	*NULL*	*NULL*
5	Dummy2	dummy2@example.com	*NULL*	*NULL*	*NULL*
6	Dummy3	dummy3@example.com	*NULL*	*NULL*	*NULL*
7	Dummy4	dummy4@example.com	*NULL*	*NULL*	*NULL*
8	Dummy5	dummy5@example.com	*NULL*	*NULL*	*NULL*

Part G – SIGNAL / Error Handling Examples

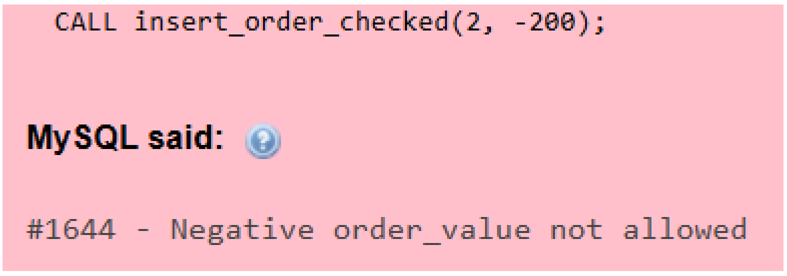
1 – Prevent negative order value

SQL Query:



```
DELIMITER //
CREATE PROCEDURE insert_order_checked(IN cid INT, IN val DECIMAL(10,2))
BEGIN
  IF val < 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Negative order_value not allowed';
  END IF;
  INSERT INTO orders(customer_id,order_value) VALUES(cid,val);
END;
//
DELIMITER ;
CALL insert_order_checked(1, 1200);    -- works fine
CALL insert_order_checked(2, -200);    -- causes SIGNAL
```

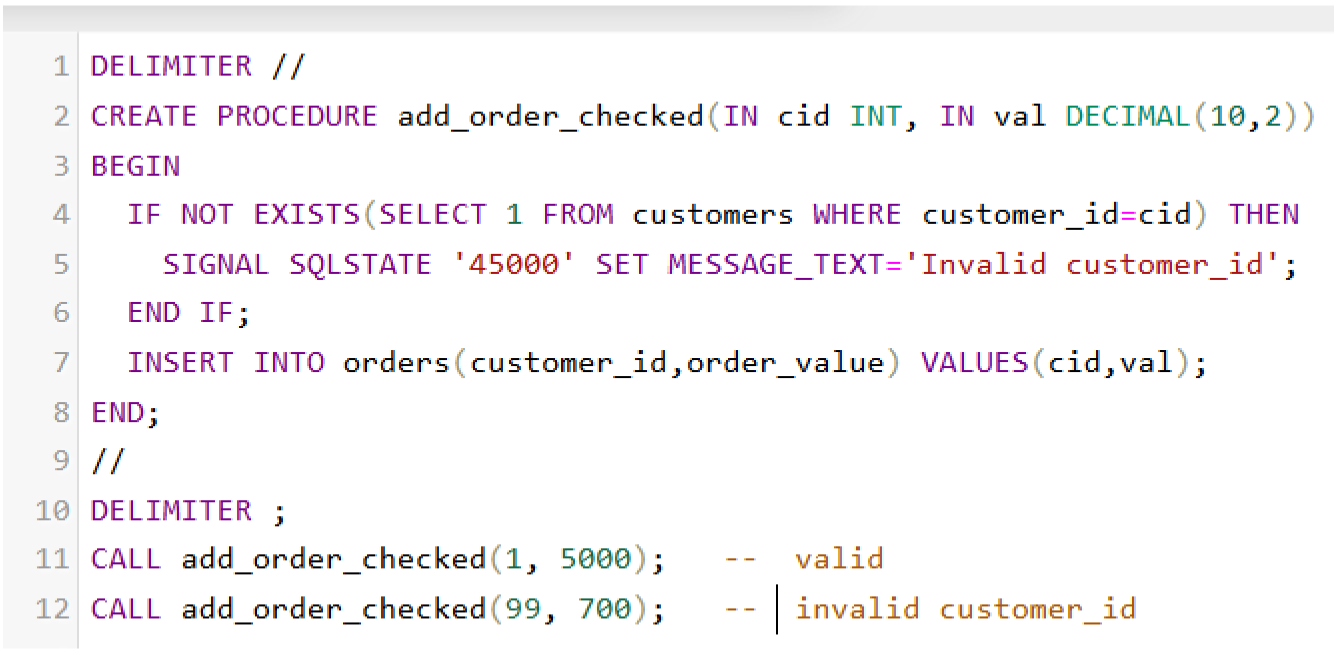
Explanation: Raises an error if a negative order value is provided.



```
CALL insert_order_checked(2, -200);

MySQL said:

#1644 - Negative order_value not allowed
```

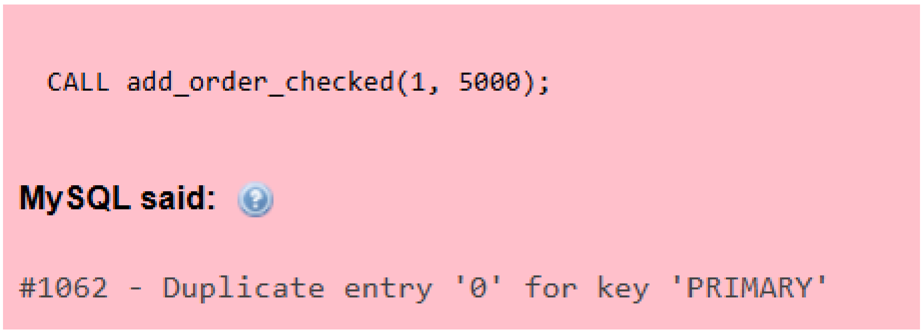
2 – Invalid customer_id SIGNAL

SQL Query:



```
DELIMITER //
CREATE PROCEDURE add_order_checked(IN cid INT, IN val DECIMAL(10,2))
BEGIN
  IF NOT EXISTS(SELECT 1 FROM customers WHERE customer_id=cid) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT='Invalid customer_id';
  END IF;
  INSERT INTO orders(customer_id,order_value) VALUES(cid,val);
END;
//
DELIMITER ;
CALL add_order_checked(1, 5000);    --  valid
CALL add_order_checked(99, 700);    -- invalid customer_id
```

Explanation: Raises custom error if customer_id does not exist.



```
CALL add_order_checked(1, 5000);
```

MySQL said:

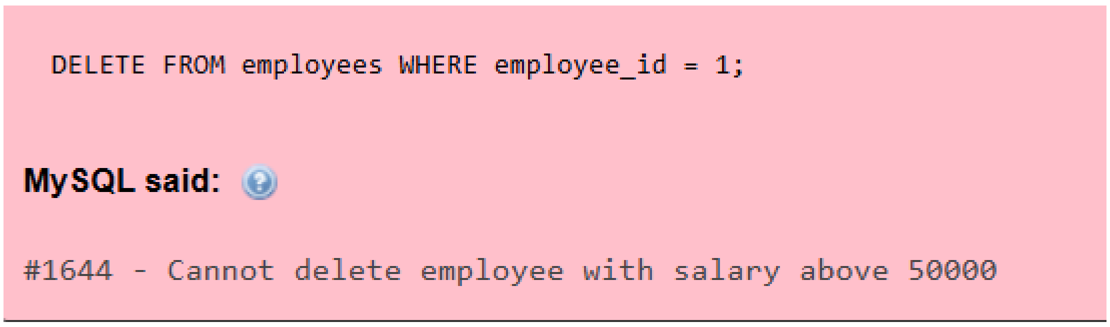
```
#1062 - Duplicate entry '0' for key 'PRIMARY'
```

3 – Prevent deletion of high-salary employee

SQL Query:

```
DELIMITER //
CREATE TRIGGER before_employee_delete
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
  IF OLD.salary > 50000 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT='Cannot delete employee with salary above 50000';
  END IF;
END;
//
DELIMITER ;
DELETE FROM employees WHERE employee_id = 1;  -- salary = 55000
DELETE FROM employees WHERE employee_id = 3;  -- salary = 70000
```

Explanation: Trigger prevents deleting employees earning more than 50000.



```
DELETE FROM employees WHERE employee_id = 1;
```

MySQL said:

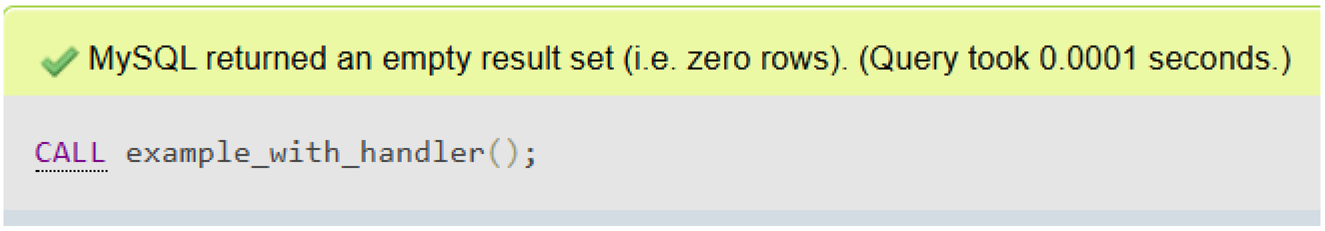
```
#1644 - Cannot delete employee with salary above 50000
```

4 – SQLEXCEPTION handler block

SQL Query:

```sql
DELIMITER //
CREATE PROCEDURE example_with_handler()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    ROLLBACK;
    SELECT 'An error occurred, operation rolled back' AS message;
  END;
  START TRANSACTION;
  -- operations here
  COMMIT;
END;
//
DELIMITER ;
CALL example_with_handler();
```

Explanation: The procedure handles the SQL exception, rolls back, and prints the message instead of crashing.



```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0001 seconds.)

CALL example_with_handler();
```

Conclusion

This assignment helped me gain a comprehensive understanding of the practical aspects of SQL programming. I learned how different SQL functions simplify complex data processing tasks and how procedural constructs such as **WHILE loops, CASE statements, and SIGNAL handlers** enhance control and error management within the database.

Atik Jawad

ID: B220305043

(Thank You)