

# **ASSIGNMENT 8**

**U21CS089 | Garvit Shah**

## Table Schemas :

```
mysql> desc Product;
```

Field	Type	Null	Key	Default	Extra
product_id	char(2)	NO	PRI	NULL	
product	varchar(20)	YES		NULL	
amount	int	YES		NULL	
quantity_remaining	int	YES		NULL	
category_id	char(2)	YES	MUL	NULL	
seller_id	char(2)	YES	MUL	NULL	
rating	float	YES		0	

```
7 rows in set (0.00 sec)
```

```
mysql> desc Order_Products;
```

Field	Type	Null	Key	Default	Extra
Order_Id	char(2)	NO	PRI	NULL	
Product_Id	char(2)	NO	PRI	NULL	
Quantity	int	NO		NULL	
Seller_Id	char(2)	NO	PRI	NULL	
Original_Amount	float	NO		NULL	
Discount	float	NO		NULL	
Product_Rating	float	YES		0	

```
7 rows in set (0.00 sec)
```

```
mysql> desc Seller;
```

Field	Type	Null	Key	Default	Extra
seller_id	char(2)	NO	PRI	NULL	
seller_name	varchar(20)	YES		NULL	
rating	float	YES		NULL	

```
3 rows in set (0.00 sec)
```

```
mysql> desc Category
-> ;
```

Field	Type	Null	Key	Default	Extra
category_id	char(2)	NO	PRI	NULL	
category	varchar(20)	YES		NULL	

2 rows in set (0.00 sec)

```
mysql> desc Product;
```

Field	Type	Null	Key	Default	Extra
product_id	char(2)	NO	PRI	NULL	
product	varchar(20)	YES		NULL	
amount	int	YES		NULL	
quantity_remaining	int	YES		NULL	
category_id	char(2)	YES	MUL	NULL	
seller_id	char(2)	YES	MUL	NULL	
rating	float	YES		0	

7 rows in set (0.00 sec)

```
mysql> desc Customer;
```

Field	Type	Null	Key	Default	Extra
customer_id	char(2)	NO	PRI	NULL	
name	varchar(20)	YES		NULL	
password	varchar(255)	NO		NULL	

3 rows in set (0.00 sec)

```
mysql> desc Orders;
```

Field	Type	Null	Key	Default	Extra
order_id	char(2)	NO	PRI	NULL	
customer_id	char(2)	YES	MUL	NULL	
amount	int	YES		NULL	
date	date	YES		NULL	
time	time	YES		NULL	

5 rows in set (0.00 sec)

1. Create a Function which returns the seller's name with the highest rating.

```
DELIMITER $$
CREATE FUNCTION get_best_seller() RETURNS VARCHAR(20) READS SQL DATA
BEGIN
    DECLARE best_seller VARCHAR(20);
    SELECT seller_name INTO best_seller FROM Seller
    WHERE rating = (SELECT MAX(rating) FROM Seller);
    RETURN best_seller;
END $$
DELIMITER ;
```

2. Create Stored procedure which takes as an input 'category' and outputs all the products of that category.

```
DELIMITER $$
CREATE PROCEDURE get_products_by_category(IN category_name VARCHAR(20))
BEGIN
    SELECT Product.product_id, Product.product, Product.amount, Product.quantity_remaining,
    Seller.seller_name, Seller.rating, Product.rating
    FROM Product
    JOIN Category ON Product.category_id = Category.category_id
    JOIN Seller ON Product.seller_id = Seller.seller_id
    WHERE Category.category = category_name;
END$$
DELIMITER ;
```

3. Create Stored procedure to take a range of prices as input and output all the products in the provided range.

```
DELIMITER //
CREATE PROCEDURE get_products_by_price_range(IN min_price INT, IN max_price INT)
BEGIN
    SELECT Product.product_id, Product.product, Product.amount, Product.quantity_remaining,
    Seller.seller_name, Seller.rating, Product.rating
    FROM Product
    JOIN Seller ON Product.seller_id = Seller.seller_id
    WHERE Product.amount BETWEEN min_price AND max_price;
END //
DELIMITER ;
```

4. Create function to display all the seller details with rating more than 3.

```
CREATE FUNCTION GetHighlyRatedSellers()
RETURNS TABLE
AS
RETURN
    SELECT seller_id, seller_name, rating
    FROM Seller
    WHERE rating > 3;
```

5. Create a function to display all the products, seller wise.

```
CREATE FUNCTION GetProductsBySeller(sellerId CHAR(2))
RETURNS TABLE
AS
RETURN
    SELECT product_id, product, amount, quantity_remaining, rating
    FROM Product
    WHERE seller_id = sellerId;
```

6. Create a Stored procedure which checks all the entries in Order\_Products table and update seller and product table accordingly.

```
CREATE PROCEDURE update_seller_and_product_tables()
BEGIN
    UPDATE Seller
    SET rating = (SELECT AVG(Product_Rating) FROM Order_Products WHERE
Order_Products.Seller_Id = Seller.seller_id);

    UPDATE Product
    SET rating = (SELECT AVG(Product_Rating) FROM Order_Products WHERE
Order_Products.Product_Id = Product.product_id);

    UPDATE Product
    SET quantity_remaining = (quantity_remaining - Quantity)
    WHERE product_id IN (SELECT Product_Id FROM Order_Products);
END;
```

7. Create Stored procedure which takes as input different filters such as price range, category, product rating, seller rating, out of stock and displays the list of products with all the details after applying filters

```
CREATE PROCEDURE get_filtered_products(IN min_price INT, IN max_price INT, IN
category_name VARCHAR(20), IN product_rating FLOAT, IN seller_rating FLOAT, IN
out_of_stock INT)
BEGIN
```

```

SELECT Product.product_id, Product.product, Product.amount, Product.quantity_remaining,
Seller.seller_name, Seller.rating, Product.rating
FROM Product
JOIN Category ON Product.category_id = Category.category_id
JOIN Seller ON Product.seller_id = Seller.seller_id
WHERE Product.amount BETWEEN min_price AND max_price
AND Category.category = category_name
AND Product.rating >= product_rating
AND Seller.rating >= seller_rating
AND (Product.quantity_remaining = 0 OR Product.quantity_remaining > 0);
END;

```

8. Create a function which takes as input sorting criteria like popularity or lowest price or highest price and display the product list accordingly.

```

CREATE FUNCTION GetSortedProducts(sortingCriteria VARCHAR(20))
RETURNS TABLE
AS
RETURN
SELECT p.product_id, p.product, p.amount, p.quantity_remaining, p.rating, s.seller_name,
s.rating, c.category
FROM Product p
INNER JOIN Seller s ON p.seller_id = s.seller_id
INNER JOIN Category c ON p.category_id = c.category_id
ORDER BY
CASE sortingCriteria
WHEN 'popularity' THEN p.rating DESC
WHEN 'lowest price' THEN p.amount ASC
WHEN 'highest price' THEN p.amount DESC
ELSE p.rating DESC
END;

```

