Homework 6: Stored Routines and Transactions

Due: April 22, 2022 @ 11:59PM ET

For this homework we will use the `bike_stores` database you created in homework 5. If you removed it, dropped it, changed its data or lost it in any way, please follow the instructions in Problem 1 for homework 5 to recreate it.

Problem 1: Create a stored function (Points: 20)

Create a stored function called `findStoreByProduct()` that will take a string as input, and return a `store_id.` The `store_id` relates to the store with the largest stock the product identified in the input string.

1. Paste the full body of your stored function. (8 points)

2. Execute the following commands and provide screenshots of the results:

```
SELECT findStoreByProduct("Trek XM700+ - 2018");
(4 points)

SELECT findStoreByProduct("Electra Amsterdam Royal 8i Ladies - 2018");
(4 points)

SELECT findStoreByProduct ("Trek Farley Alloy Frameset - 2017");
(4 points)
```

Problem 2: Create a stored procedure (Points: 30)

Create a stored procedure called `placeOrder()` that can be called to insert a new order in the database. It will receive a `customerId` as an INT, a `productId` as an INT and a `qty` as an INT and return (as an output parameter) the *`order_id`* of the new row created in table `orders`.

This stored procedure will find the store with the largest stock of that particular product and assign that store to the order. The `order_status` should be set to 1 (i.e.

Pending), the current system date (*see function* `CURDATE`*)* will be assigned to `order_date`, column `required_date` will be 7 days from the current system date *(see function* `ADDDATE`*)* and the column `staff_id` will be assigned for anyone that works in the selected store (per previous requirement). Since the `order_id` column is not an auto-incremented column you need to calculate the value for it. You can use `max(order_id)` to find out the highest `order_id` in the table.

For the new record in the order_items table, set the `product_id` and `quantity` to the `productId` and `qty` passed into the stored procedure. The `item_id` shall be set to 1 (since this order will only have one item).The list price should be retrieved from the products table using the passed `productId`. The discount value should be set to 0.

We have provided a rough framework and code comments to help guide you. You do not need to follow this flow if you have other ideas on how to implement it:

```
DELIMITER //
CREATE PROCEDURE placeOrder(IN customerId INT, IN productId INT, IN qty INT, OUT
createdOrderId INT)
 BEGIN

 /* Declare your variables. */


 /* Calculate the next order id, since this column is not auto-increment. */


 /* Find the store to use for serving this order. */


 /* Pick any staff member that works in the selected store. */


 /* Create the order row. */


 /* Find the price for the product */


 /* Create the item row. */


 END //
DELIMITER ;
```

1. Paste the full body of your stored procedure. (20 points)

2. Execute the following commands and provide screenshots of the results: (5 points)

```
CALL placeOrder( 20, 10, 1, @order_id);
SELECT * FROM orders WHERE order_id = @order_id;
SELECT * FROM order_items WHERE order_id = @order_id;
```

3. Execute the following commands and provide screenshots of the results: (5 points)

```
CALL placeOrder( 11, 12, 2, @order_id);
SELECT * FROM orders WHERE order_id = @order_id;
SELECT * FROM order_items WHERE order_id = @order_id;
```

Problem 3: Stored procedure with transaction support (Points: 30)

Modify the `placeOrder()` stored procedure to receive a fourth parameter `store_id` and use transactions. Assign the received `store_id` to the created order. After inserting the `order_items` and `orders` rows, verify that the requested store has stock for the requested product and quantity and if it does not then rollback the transaction and return an `order_id` of -1.

1. Paste the full body of your stored procedure. (20 points)

2. Execute the following commands and provide screenshots of the results: (5 points)

```
CALL placeOrder( 11, 12, 10, 1, @order_id);
SELECT @order_id;
SELECT * FROM orders WHERE order_id = @order_id;
SELECT * FROM order_items WHERE order_id = @order_id;
SELECT max(order_id) FROM orders;
```

3. Execute the following commands and provide screenshots of the results: (5 points)

```
CALL placeOrder( 11, 12, 20, 1, @order_id);
SELECT @order_id;
SELECT * FROM orders WHERE order_id = @order_id;
SELECT * FROM order_items WHERE order_id = @order_id;
SELECT max(order_id) FROM orders;
```

Problem 4: Stored procedure with exception handling (Points: 20)

Modify the `placeOrder()` stored procedure from Problem 3 to handle issues with invalid `customer_id`, `product_id` and `store_id` using an EXIT HANDLER. The exit handler must rollback the transaction and return an `order_id` of -1

1. Paste the full body of the modified stored procedure. (10 points)

2. Execute the following commands and provide screenshots of the results: (5 points)

```
CALL placeOrder( 11, 12, 10, 1, @order_id);
```

```
SELECT @order_id;
SELECT * FROM orders WHERE order_id = @order_id;
SELECT * FROM order_items WHERE order_id = @order_id;
SELECT max(order_id) from orders;
```

3. Execute the following commands and provide screenshots of the results: (5 points)

```
SELECT max(order_id) from orders;
CALL placeOrder( 1500, 12, 10, 1, @order_id);
SELECT @order_id;
CALL placeOrder( 11, 500, 10, 1, @order_id);
SELECT @order_id;
CALL placeOrder( 11, 12, 10, 8, @order_id);
SELECT @order_id;
SELECT max(order_id) from orders;
```

Bonus: Triggers (Points: 10)

Create a trigger object that will assign the store to serve a given order AFTER the order is created. For this to work we will need to modify the `orders` table so that the `store_id` and `staff_id` columns can be left null. Here is the command to do that:

```
ALTER TABLE orders MODIFY store_id INT NULL, MODIFY staff_id INT NULL;
```

Now, take the stored procedure you created in Problem 2 and modify it so it does not assign the `store_id` or the `staff_id`. Invoke it and verify the `store_id` and the `staff_id` are filled by the trigger action on INSERT in the `order_items` table by invoking the modified `placeOrder()` stored procedure.

1. Provide the full body of your trigger. (5 points)

2. Copy and paste the output from invoking your stored procedure and then executing the following commands (5 points):

```
SHOW columns FROM orders;
SELECT * FROM orders WHERE order_id = @order_id;
```