**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)

|  |
| --- |
| delimiter //  create function findStoreByProduct(prod\_name varchar(255))  returns int deterministic  begin  declare storeid int;  select s.store\_id into storeid  from stocks s  join  (  select p.product\_id, p.product\_name, max(st.quantity) as max\_quantity  from products p  join stocks st on p.product\_id=st.product\_id  where p.product\_name=prod\_name  group by p.product\_id, p.product\_name  ) t on s.product\_id= t.product\_id and s.quantity=t.max\_quantity;  return storeid;  end //  delimiter ; |

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)

|  |
| --- |
| DELIMITER //  CREATE PROCEDURE placeOrder(IN customerId INT, IN productId INT, IN qty INT, OUT createdOrderId INT)  BEGIN  /\* Declare your variables. \*/  declare next\_order\_id int;  declare storeId int;  declare orderStatus int;  declare staffId int;  declare listPrice decimal(10,2);  set orderStatus=1;  /\* Calculate the next order id, since this column is not auto-increment. \*/  set next\_order\_id=(select max(order\_id)+1  from orders);  set createdOrderId=next\_order\_id;  /\* Find the store to use for serving this order. \*/  set storeId=(  select s.store\_id  from stocks s  join  (  select p.product\_id, p.product\_name, max(st.quantity) as max\_quantity  from products p  join stocks st on p.product\_id=st.product\_id  where p.product\_id=productId  group by p.product\_id, p.product\_name  ) t on s.product\_id= t.product\_id and s.quantity=t.max\_quantity  );  /\* Pick any staff member that works in the selected store. \*/  set staffId=(  select staff\_id from staffs where store\_id=storeId and active=1 limit 1  );  /\* Create the order row. \*/  insert into orders(order\_id, customer\_id, order\_status, order\_date, required\_date, shipped\_date, store\_id, staff\_id)  values(next\_order\_id, customerId, orderStatus, curdate(), adddate(curdate(), interval 7 day), null, storeId, staffId);  /\* Find the price for the product \*/  set listPrice=(  select list\_price  from products where product\_id=productId  );  /\* Create the item row. \*/  insert into order\_items(order\_id, item\_id, product\_id, quantity, list\_price, discount)  values(next\_order\_id, 1, productId, qty, listPrice, 0);  END //  DELIMITER ; |

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)

|  |
| --- |
| drop procedure placeOrder;  DELIMITER //  CREATE PROCEDURE placeOrder(IN customerId INT, IN productId INT, IN qty INT, IN storeId int, OUT createdOrderId INT)  BEGIN  /\* Declare your variables. \*/  declare next\_order\_id int;  declare orderStatus int;  declare staffId int;  declare listPrice decimal(10,2);  declare \_quantity int;  set orderStatus=1;  /\* Calculate the next order id, since this column is not auto-increment. \*/  set next\_order\_id=(select max(order\_id)+1  from orders);  set createdOrderId=next\_order\_id;    START TRANSACTION;    set staffId=(  select staff\_id from staffs where store\_id=storeId and active=1 limit 1  );  insert into orders(order\_id, customer\_id, order\_status, order\_date, required\_date, shipped\_date, store\_id, staff\_id)  values(next\_order\_id, customerId, orderStatus, curdate(), adddate(curdate(), interval 7 day), null, storeId, staffId);  set listPrice=(  select list\_price  from products where product\_id=productId  );  insert into order\_items(order\_id, item\_id, product\_id, quantity, list\_price, discount)  values(next\_order\_id, 1, productId, qty, listPrice, 0);  /\* verify that the requested store has stock for the requested product and quantity. \*/    set \_quantity=(select quantity  from stocks  where store\_id=storeId and product\_id=productId  );    if \_quantity<qty then  rollback;  set createdOrderId=-1;  else  update stocks set quantity=quantity-qty where store\_id=storeId and product\_id=productId;  commit;  end if;  END //  DELIMITER ; |

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)

|  |
| --- |
| drop procedure placeOrder;  DELIMITER //  CREATE PROCEDURE placeOrder(IN customerId INT, IN productId INT, IN qty INT, IN storeId int, OUT createdOrderId INT)  BEGIN  /\* Declare your variables. \*/  declare next\_order\_id int;  declare orderStatus int;  declare staffId int;  declare listPrice decimal(10,2);  declare \_quantity int;  DECLARE EXIT HANDLER FOR SQLEXCEPTION  BEGIN  ROLLBACK;  set createdOrderId=-1;  END;  set orderStatus=1;  /\* Calculate the next order id, since this column is not auto-increment. \*/  set next\_order\_id=(select max(order\_id)+1  from orders);  set createdOrderId=next\_order\_id;    START TRANSACTION;    set staffId=(  select staff\_id from staffs where store\_id=storeId and active=1 limit 1  );  insert into orders(order\_id, customer\_id, order\_status, order\_date, required\_date, shipped\_date, store\_id, staff\_id)  values(next\_order\_id, customerId, orderStatus, curdate(), adddate(curdate(), interval 7 day), null, storeId, staffId);  set listPrice=(  select list\_price  from products where product\_id=productId  );  insert into order\_items(order\_id, item\_id, product\_id, quantity, list\_price, discount)  values(next\_order\_id, 1, productId, qty, listPrice, 0);  /\* verify that the requested store has stock for the requested product and quantity. \*/    set \_quantity=(select quantity  from stocks  where store\_id=storeId and product\_id=productId  );    if \_quantity<qty then  rollback;  set createdOrderId=-1;  else  update stocks set quantity=quantity-qty where store\_id=storeId and product\_id=productId;  commit;  end if;  END //  DELIMITER ; |

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)

|  |
| --- |
| DELIMITER //  CREATE TRIGGER orders\_update  AFTER insert  ON order\_items FOR EACH ROW  BEGIN  DECLARE storeId INT;  declare staffId int;  select s.store\_id into storeId  from stocks s  join  (  select p.product\_id, p.product\_name, max(st.quantity) as max\_quantity  from products p  join stocks st on p.product\_id=st.product\_id  where p.product\_id=new.product\_id  group by p.product\_id, p.product\_name  ) t on s.product\_id= t.product\_id and s.quantity=t.max\_quantity  ;  select staff\_id into staffId from staffs where store\_id=storeId and active=1 limit 1;  update orders set store\_id=storeId, staff\_id=staffId where order\_id=NEW.order\_id;  END //  DELIMITER ; |

|  |
| --- |
| drop procedure placeOrder;  DELIMITER //  CREATE PROCEDURE placeOrder(IN customerId INT, IN productId INT, IN qty INT, OUT createdOrderId INT)  BEGIN  /\* Declare your variables. \*/  declare next\_order\_id int;  declare orderStatus int;  declare listPrice decimal(10,2);  set orderStatus=1;  /\* Calculate the next order id, since this column is not auto-increment. \*/  set next\_order\_id=(select max(order\_id)+1  from orders);  set createdOrderId=next\_order\_id;  /\* Create the order row. \*/  insert into orders(order\_id, customer\_id, order\_status, order\_date, required\_date, shipped\_date)  values(next\_order\_id, customerId, orderStatus, curdate(), adddate(curdate(), interval 7 day), null);  /\* Find the price for the product \*/  set listPrice=(  select list\_price  from products where product\_id=productId  );  /\* Create the item row. \*/  insert into order\_items(order\_id, item\_id, product\_id, quantity, list\_price, discount)  values(next\_order\_id, 1, productId, qty, listPrice, 0);  END //  DELIMITER ; |

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;

|  |
| --- |
| with no trigger:    with trigger: |