Test plan for InOrder Project

I. Brief Introduction

We have created basic CRUD operations and stored procedures for the InOrder Project. We can divide the operations into 3 parts. operate tests in part(3) accordingly.

To ensure that we have the same original data before we test every procedure in part(3), we create a stored procedure called **testData**. In the procedure testData, we delete all data in the tables and insert data. Before we test procedures, we call testData, so that we ensure that before each test we have the same data in the database.

We should run part(1) and part(2) first [from 1-885]. When the SQL Server sends the warning: “Unsafe query: 'Delete' statement without 'where' clears all data in the table” , we should select “Execute All”. This is because in the testData, we try to delete all data within each table, so the warning is expected and not a design mistake.

After runnning the two parts, we have created tables and stored procedures. **We should test the functionality of each procedure and trigger one by one in part(3).**

Procedure And Trigger List

index	**procedure name**	**index**	**procedure name**
0	testdata	15	createOrder
1	createCustomer	16	changePreOrderUnits
2	createProuduct	17	createOrderRecord
3	listProduct	18	calculateOrderPrice
4	unlistProduct	19	FinalizeOrderPrice
5	readListedProduct	20	createOneOrderWithManyItems
6	searchProductName	21	updateShipmentDate
7	readProductInventory	22	cancelOrder
8	readAllProductInventory	23	calculateTotalRevenue
9	readInventorySpecifyingUnit	24	readCustomerOrders
10	changeInventoryUnits	25	readCustomerOrdersSpecifyingTime
11	changeDiscount	26	readOrderDetail
12	changePrice	27	readSpecificProductInventory
13	readPriceHistory	28	applyPromotion Trigger
14	readPriceHistorySpecifyingTime	29	unlistProductIfNotAvailable Trigger

II. Tests for Stored Procedures and Triggers

We use “testData” procedure to help to test other 27 procedures and 2 triggers. The test and results are shown below.

(1) createCustomer

[Base Situation] First we run the following code, we will have the information of the Customer table printed, and there are 3 customers.

```
CALL testData();
SELECT * from customer;
```

[Valid Cases] Then we test a valid case. After we run the following code, we find that there are 5 customers in Customer table now.

```
  CALL createCustomer(489325, '4Sally', '231 Maple St.', 'Stockton', 'California','United States','95201');
  CALL createCustomer(365903, 'Stowed012', '221B Baker Street', 'Vancouver', 'British Columbia','Canada','98607');
  SELECT * from customer; # we have 5 customers in the customer table.
```

[Invalid Cases] Finally, we test invalid cases.

```
#duplicate primary key
CALL createCustomer(123456, 'STEVE1', '11 Park Rd.', 'New Haven', 'Connecticut','United States','46593');
# Invalid country name
CALL createCustomer(657848, 'Jenny22', '102 rainbow Dr.', 'Chihuahua', 'Sichuan', 'SABCD','365342');
# Invalid state name
CALL createCustomer(489324, 'Echo_1', '2209 Poughkeepsie Rd.', 'New Haven', 'Yukon Territory','United States','C1A 0A9');
SELECT * from customer;# customers not added, we have 5 customers in the customer table.
```

(2)createProuduct

[Base Situation] First we run the following code, and find that there are 8 products.

```
CALL testData();
SELECT * from Product; # We have 8 products in Product table
```

[Valid Cases] Then we have 2 valid cases. After we run the following code, we find that there are 10 products in Product table now.

```
CALL createProduct('Levis Hat 075','One size fit comfy hat', 'LE-000002-HA');
CALL createProduct('Levis Shirt 1178', 'Comfortable 100% cotton shirt', 'LE-238843-SH');
SELECT * from Product; # we have 10 products in Product table
```

[Invalid Cases] Finally, we test invalid cases.

```
#duplicate primary key
CALL createProduct('Levis Shirt 034', 'Comfortable 100% cotton shirt', 'LE-594092-SH');
#name cannot be empty string
CALL createProduct('', 'Comfortable 100% cotton shirt', 'LE-865421-SH');
#inputSKU does not match regex
CALL createProduct('Levis Hat 075','One size fit comfy hat', 'LE-000002');
CALL createProduct('Levis Hat 075','One size fit comfy hat', '34-564567-12');
SELECT * from Product; # Product not added, we have 10 products in Product table
```

(3)listProduct

[Base Situation] First we run the following code, and find that there are 8 products, and 6 are listed(the column listed = 1), and 2 are unlisted.

```
CALL testData();
SELECT * from Product;
```

[Valid Cases] Then we test 2 valid cases, we list the unlisted products. After running the code, we find that 8 products are all listed.

```
CALL listProduct('LE-420493-SW');
CALL listProduct('LE-497893-PA');
SELECT * from Product; # all 8 products are listed
```

[Invalid Cases] Finally, we test invalid cases.

```
# list an already listed product
CALL listProduct('LE-420422-JE');
CALL listProduct('LE-420423-JE');
#create 2 unlisted products
CALL createProduct('Levis shirt 1105', 'Comfortable 100% cotton shirt', 'LE-456885-SH');
CALL createProduct('Levis shirt 4365', 'Comfortable 100% cotton shirt', 'LE-897643-SH');
#can not be listed since price has not been set yet
CALL listProduct('LE-456885-SH');
CALL listProduct('LE-897643-SH');
SELECT * from Product; # we have 2 unlisted newly created productS in Product table
```

(4)unlistProduct

[Base Situation] First we run the following code, and find that there are 8 products, and 6 are listed(the column listed = 1), and 2 are unlisted.

```
CALL testData();
SELECT * from Product; # we have 2 unlisted products in Product table
```

[Valid Cases] Then we run 2 valid cases, and find that there are 4 products unlisted now.

```
CALL unListProduct('LE-497892-PA');
CALL unListProduct('LE-594092-SH');
SELECT * from Product; #we have 4 unlisted products in Product table
```

(5)readListedProduct

[Base Situation] First we run the following code, and find that there are 8 products shown. The unlisted products are now shown.

```
CALL testData();
SELECT * from Product;
```

[Valid Cases] We can check all listed products, and after unlisting all products, we check again and find no products, which indicates that the procedure functions well.

```
CALL readListedProducts();
SELECT * from Product; # show 8 products in Product table
#unlist all products
CALL unListProduct('LE-420422-JE');
CALL unListProduct('LE-420423-JE');
CALL unListProduct('LE-420492-SW');
CALL unListProduct('LE-497892-PA');
CALL unListProduct('LE-594092-SH');
CALL unListProduct('LE-594382-HA');
CALL readListedProducts();# no listed products found
```

(6)searchProductName

[Base Situation] First we run the following code, and find all products.

```
CALL testData();
SELECT * from Product;
```

[Valid Cases] Then we have two test cases.

For the first case we search for “Jeans”, and we can find 2 products with the name of “Jeans”.

```
CALL searchProductName('jeans');
```

For the Second case we search for “shirt”, and we find 2 products with the name of “shirt”.

```
CALL searchProductName('shirt');
```

. **[Invalid Cases]** Finally, we have one invalid case. We can not find any product with the information of “FSAGAWRGFIJAG” or “fgirtugrtg”.

```
CALL searchProductName(FSAGAWRGFIJAG);
CALL searchProductName(fgirtugrtg);
```

(7)readProductInventory

[Base Situation] First we run the following code, and find 8 product inventories in the table of InventoryRecord.

```
CALL testData();
SELECT * from InventoryRecord;
```

[Valid Cases] Then we have 2 valid cases.

For the first case, we check the inventory of products that are listed. We find 6 products.

```
CALL readProductInventory(TRUE);
```

For the first case, we check the inventory of products that are unlisted. We find 2 products.

```
CALL readProductInventory(FALSE);
```

(8)readAllProductInventory

[Base Situation] First we run the following code, and find 8 product inventories in the table of InventoryRecord.

```
CALL testData();
SELECT * from InventoryRecord;
```

[Valid Cases] Then we run the code and find all 8 product inventories.

```
CALL readAllProductInventory();
```

(9)readInventorySpecifyingUnit

[Base Situation] First we run the following code, and find 8 product inventories in the table of InventoryRecord.

```
CALL testData();
SELECT * from InventoryRecord;
```

[Valid Cases] Then we have valid cases.

For the first case, we check the products that have inventory between 50 and 90. And we find 2 products.

```
CALL readInventorySpecifyingUnit(50, 90);
```

For the second case, we check the products that have inventory between 100and 150. And we find 3 products.

```
CALL readInventorySpecifyingUnit(100, 150);
```

For the second case, no product of the range found.

```
CALL readInventorySpecifyingUnit(1000, 2000); #no product between this range found
```

[Invalid Cases] Finally, we test invalid cases.

```
#Invalid unit range, beginUnit must smaller than endUnit
CALL readInventorySpecifyingUnit(100, 80);
CALL readInventorySpecifyingUnit(200, 70);
```

(10)changeInventoryUnits and Trigger unlistProductIfNotAvailable

[Base Situation] First we run the following code, and find 8 product inventories in the table of InventoryRecord.

```
CALL testData();
SELECT * FROM InventoryRecord;
```

[Valid Cases] Then we have 2 valid cases and test cases for trigger unlistProductIfNotAvailable.

For the first case, we increase the inventory of the product 'LE-420422-JE' by 100. And the inventory increases from 100 to 200.

```
CALL changeInventoryUnits('LE-420422-JE', 100);
SELECT * FROM InventoryRecord;
```

For the second case, we decrease the inventory of the product 'LE-420423-JE' by 50. And the inventory decreases from 60 to 10.

```
CALL changeInventoryUnits('LE-420423-JE', -50);
SELECT * FROM InventoryRecord;

## PLUS* TEST Trigger unlistProductIfNotAvailable
SELECT * FROM Product; # 2 unlisted products in Product table
CALL changeInventoryUnits('LE-420423-JE', -10); # product 'LE-420423-JE' units becomes 0
SELECT * FROM Product; # we can see 3 unlisted products now
```

[Invalid Cases] Finally, we test invalid cases.

```
#cannot find the inout SKU since the product does not exist
CALL changeInventoryUnits('LE-420499-SW', 10);
# SQLException: invalid input. decrease number larger than current units
CALL changeInventoryUnits('LE-420492-SW', -1000);
CALL changeInventoryUnits('LE-497892-PA', -500);
SELECT * FROM InventoryRecord; # units number of 2 products above unchanged
```

(11)changeDiscount

[Base Situation] First we run the following code, and find 8 product inventories in the table of InventoryRecord.

```
CALL testData();
SELECT * FROM InventoryRecord;
```

[Valid Cases] Then we have two valid cases.

```
CALL changeDiscount('LE-420422-JE', 0.6);
CALL changeDiscount('LE-420423-JE', 0.9);
SELECT * FROM InventoryRecord;
```

[Invalid Cases] Finally, we have invalid cases.

```
# Cannot find the input SKU, no such product exists
CALL changeDiscount('LE-387453-JE', 0.8);
# SQLException: Discount cannot be negative or greater than 1
CALL changeDiscount('LE-420422-JE', -0.1);
CALL changeDiscount('LE-420422-JE', 1.2);
SELECT * FROM InventoryRecord; # discount of 2 products above unchanged in InventoryRecord table
```

(12)changePrice

[Base Situation] First we run the following code, and find 8 product inventories in the table of InventoryRecord.

```
CALL testData();
SELECT * FROM InventoryRecord;
```

[Valid Cases] Then we change the price of 2 products.

```
CALL changePrice('LE-594092-SH', 120);
CALL changePrice('LE-594092-SH', 100);
CALL changePrice('LE-420492-SW', 150);
SELECT * FROM InventoryRecord;
```

[Invalid Cases] Finally, we have invalid cases. The price cannot be negative.

```
# Cannot find the input SKU because such product does not exist
CALL changePrice('LE-765392-JE', 90);
# SQLException: Price cannot be negative
CALL changePrice('LE-594092-SH', -10);
CALL changePrice('LE-594092-SH', -100);
SELECT * FROM InventoryRecord; # price of product 'LE-594092-SH' unchanged in InventoryRecord table
```

(13)readPriceHistory

[Base Situation] First we check the PriceHistory table. There are 12 rows.

```
CALL testData();
SELECT * FROM PriceHistory;
```

[Valid Cases] Then we try to read price history of two products.

```
CALL readPriceHistory('LE-420422-JE'); # 8 price historys of product 'LE-420422-JE' found
CALL readPriceHistory('LE-420492-SW'); # 1 price history of product 'LE-420492-SW' found
```

[Invalid Cases] Finally, we try invalid cases.

test for invalid SKU

```
CALL readPriceHistory('LE-438753-JE'); # No such product exists
```

(14)readPriceHistorySpecifyingTime

[Base Situation] First we check the data of the PriceHistory table.

```
CALL testData();
SELECT * FROM PriceHistory;
```

[Valid Cases] Then we try two valid cases.

```
CALL readPriceHistorySpecifyingTime('LE-420422-JE', '2019-01-01', '2019-06-29'); # 2 history found
CALL readPriceHistorySpecifyingTime('LE-420422-JE', '2019-06-30', '2019-11-30'); # 4 history found
```

[Invalid Cases] Finally, we have two invalid cases.

```
#Invalid time range, start datetime cannot be later than end datetime
CALL readPriceHistorySpecifyingTime('LE-420422-JE', '2019-10-11', '2019-06-29');
# no such product exists
CALL readPriceHistorySpecifyingTime('LE-424865-LA','2019-01-01', '2019-09-29');
```

(15)createOrder

[Base Situation] First we can check 12 rows in the Orders table.

```
CALL testData();
SELECT * FROM Orders;
```

[Valid Cases] We have 2 valid cases to create 2 orders.

```
CALL createOrder('123456', '#show 14 records in Orders table',NOW());
CALL createOrder('123456', '475686901',NOW());
SELECT * FROM Orders; #show 14 records in Orders table
```

[Invalid Cases] We have invalid cases.

```
#SQLException: Duplicate Primary Key
CALL createOrder('123457','197354610', NOW());
#SQLException: a foreign key constraint fails because customer '234223' does not exists
```

```
CALL createOrder('234223', '435421456', NOW());
# SQLException: OrderID cannot be empty string.
CALL createOrder('123457','', NOW());
SELECT * FROM Orders; #show 14 records in Orders table
```

(16)updateInventoryUnits

[Base Situation] First we check the status of the InventoryRecord table.

```
CALL testData();
SELECT * FROM InventoryRecord;
```

[Valid Cases] Then we test 3 valid cases. The preorder units of the two products has been updated.

```
CALL changePreOrderUnits('LE-420422-JE',   100);
CALL changePreOrderUnits('LE-420492-SW', 200);
CALL changePreOrderUnits('LE-420492-SW', -20);
SELECT * FROM InventoryRecord;
```

[Invalid Cases] Finally, we have invalid test cases.

```
# Cannot find the input SKU
CALL changePreOrderUnits('LE-857342-AS', 10);
# MySQLException: Invalid input units
CALL changePreOrderUnits('LE-420422-JE',   -500);
CALL changePreOrderUnits('LE-420492-SW', -800);
```

(17)createOrderRecord

[Base Situation] First we check the 20 rows of record in the InventoryRecord table.

```
CALL testData();
SELECT * FROM OrderRecord;
```

[Valid Cases] Then we try two order records. We will find 22 rows of record in the InventoryRecord table.

```
CALL createOrderRecord('LE-420422-JE','197359093', 10);
CALL createOrderRecord('LE-420422-JE','906743588', 5);
SELECT * FROM OrderRecord;
```

[Invalid Cases] Finally, we try to test 3 invalid invalid cases.

```
#orderUnits can not be negative
CALL createOrderRecord('LE-594382-HA', '197359093', -10);
#orderUnits can not exceed inventory
CALL createOrderRecord('LE-594382-HA', '197359093', 1000);
#product does not exist sqlException
CALL createOrderRecord('LE-594382-45', '197359093', 5);
```

(18)calculateOrderPrice and Trigger applyPromotion

[Base Situation] First we check the status of the Orders table. No totalPrice in Order table has been calculated yet.

```
CALL testData();
SELECT * FROM Orders;
```

[Valid Cases] Then we calculate totalPrice for two orders. We can check that the column of totalPrice of the two orders has been updated. We also have test cases for Trigger applyPromotion.

```
CALL calculateOrderPrice('197359093');
CALL calculateOrderPrice('197354610');
SELECT * FROM Orders;
## PLUS* Trigger applyPromotion works for the 3 orders above, 2 orders exceeds $300 can apply promotion
```

```
SELECT * FROM Promotion;
```

[Invalid Cases] Finally, we try to test 3 invalid invalid cases.

```
# Cannot find the input order ID.
CALL calculateOrderPrice('435254123'); #enter an orderID that does not exist
```

(19)FinalizeOrderPrice

[Base Situation] First we check the status of the OrderRecord table.

```
CALL testData();
SELECT * FROM OrderRecord;
```

[Valid Cases] Then we need to run two test cases. In our design, we should first calculateOrderPrice then use calculateOrderPrice to get the final price for payment.

```
CALL calculateOrderPrice('256354690');
CALL calculateOrderPrice('345543099');
# this order is from customer 'Later983' from California, tax rate 7.25%, order exceeds $300, got $50 off from promotion. We get 1029.6.
CALL FinalizeOrderPrice('256354690');
#this order is from customer 'Lisa312' from PEI, Canada, tax rate 10%, order less than $300, no promotion, we get 220.
CALL FinalizeOrderPrice('345543099');
```

[Invalid Cases] Finally, we try to test 3 invalid invalid cases.

```
#cannot find the input order ID.
CALL FinalizeOrderPrice('764783289');# inout an order ID that does not exists
```

(20)createOneOrderWithManyItems

[Base Situation] First we check the status of the Orders and OrderRecord. We find 12 orders and 20 order records.

```
CALL testData();
SELECT * FROM Orders;#show 12 orders in Orders table
SELECT * FROM OrderRecord; #show 20 order records in OrderRecord table
```

[Valid Cases] Then we call the procedure. As the procedure creates one order and three order records. We will find 13 orders in Orders table and 23 order records in OrderRecord table.

```
CALL createOneOrderWithManyItems(577435, '826493587', '2019-10-19',
'LE-420422-JE', 5, 'LE-420492-SW', 30, 'LE-497892-PA', 70);
SELECT * FROM Orders; #show 13 orders in Orders table
SELECT * FROM OrderRecord; #show 23 order records in OrderRecord table
```

(21)updateShipmentDate

[Base Situation] First we check the status of the orders.

```
CALL testData();
SELECT * FROM Orders;
```

[Valid Cases] Then we update the shipment date of two orders.

```
CALL updateShipmentDate('197359093','2019-11-10 17:02:55');# Valid shipment date.
CALL updateShipmentDate('256354690', NOW());
SELECT * FROM Orders;
```

[Invalid Cases] Finally, we have invalid cases. In our design, we cannot set shipment date before the order date.

```
# Invalid shipment date, precedes order date
CALL updateShipmentDate('197354610','2015-12-07 00:00:00');
#Cannot find the input order ID.
CALL updateShipmentDate('236435075', '2019-10-10 00:00:00');# input a nonexistent orderID
```

(22)cancelOrder

[Base Situation] First we check the status of the orders. Only one order out of 12 orders are cancelled.

```
CALL testData();
SELECT * FROM Orders;
```

[Valid Cases] Then we cancel two orders.

```
CALL cancelOrder('197354610');
CALL cancelOrder('197359093');
SELECT * FROM Orders;
```

[Invalid Cases] Finally, we try to cancel order with shipment date. In our design, we can not cancel order with shipment date. So we cannot cancel order 256354690.

```
# MysqlException: Can not cancel a shipped order
update Orders set ShipmentDate = '2019-12-03' where OrderID = 256354690;
CALL cancelOrder('256354690');
# Cannot find the input order ID.
CALL cancelOrder('468679123'); # input a nonexistent orderID
```

(23)calculateTotalRevenue

[Base Situation] First we need to check the status of the orders. We do not calculate total price for the orders in the testData for the reason that we can test calculateOrderPrice and FinalizeOrderPrice. In order to test calculateTotalRevenue, we need to call calculateOrderPrice and FinalizeOrderPrice for 3 orders first.

```
CALL testData();
CALL calculateOrderPrice(197354610);
CALL FinalizeOrderPrice(197354610);
CALL calculateOrderPrice(197359093);
CALL FinalizeOrderPrice(197359093);
CALL calculateOrderPrice(256354501);
CALL FinalizeOrderPrice(256354501);
SELECT * FROM Orders;
```

[Valid Cases] Then we can have two test cases for calculateTotalRevenue.

```
# Only one order, total revenue = 1280.
CALL calculateTotalRevenue('2019-10-10','2019-12-20');
# Three orders, total revenue = 2060.
CALL calculateTotalRevenue('2019-01-10','2019-12-20');
```

[Invalid Cases] Finally, we try one invalid case.

```
#Invalid time range input.
CALL calculateTotalRevenue('2019-10-18', '2019-04-27'); #end time precedes start time
```

(24)readCustomerOrders

[Base Situation] First we can check 12 orders in the table of Orders.

```
CALL testData();
SELECT * FROM Orders;
```

[Valid Cases] Then we check the orders of customer 577435 and 123457. They have 8 and 3 orders separately.

```
# Find eight customerOrders.
CALL readCustomerOrders(577435);
# Find three customerOrders.
CALL readCustomerOrders(123457);
```

[Invalid Cases] Finally, we try one invalid case.

```
#No such customer exists
CALL readCustomerOrders(546235); # input a nonexistent customer ID
```

(25)readCustomerOrdersSpecifyingTime

[Base Situation] First we can check 12 orders in the table of Orders.

```
CALL testData();
SELECT * FROM Orders;
```

[Valid Cases] Then we can check two valid cases.

```
# Find five customerOrders.
CALL readCustomerOrdersSpecifyingTime(577435, '2019-10-01', '2019-12-11');
# Find one customerOrder.
CALL readCustomerOrdersSpecifyingTime(123457, '2019-09-01', '2019-10-01');
# No order during this time.
CALL readCustomerOrdersSpecifyingTime(123457, '2015-09-01', '2015-10-01');
```

[Invalid Cases] Finally, we check one invalid case.

```
#No such customer exists
CALL readCustomerOrdersSpecifyingTime(546235, '2019-01-01', '2019-03-03'); # input a nonexistent customer ID
```

(26)readOrderDetail

[Base Situation] First we can check 20 rows of order records in the table of OrderRecord.

```
CALL testData();
SELECT * FROM OrderRecord;
```

[Valid Cases] Then we can check two valid cases with the order ID. The order record information of the same order ID is shown.

```
# Find two items for orderID 197354610.
CALL readOrderDetail(197354610);
# Find two items for orderID 687312546.
CALL readOrderDetail(687312546);
```

[Invalid Cases] Finally, we test one invalid order ID. It does not show anything.

```
# No such orderId as 451000000.
CALL readOrderDetail(451000000); # input a nonexistent order ID
```

(27)readSpecificProductInventory

[Base Situation] First we can check the table of InventoryRecord.

```
CALL testData();
SELECT * FROM InventoryRecord;
```

[Valid Cases] We have two valid cases to check the inventory of two products.

```
CALL readSpecificProuductInventory('LE-420422-JE');
CALL readSpecificProuductInventory('LE-420423-JE');
```

[Invalid Cases] We have invalid cases with invalid SKU.

```
CALL readSpecificProuductInventory('LE-432658-LE'); #cannot find input SKU
CALL readSpecificProuductInventory('LE-432659-JD'); #cannot find input SKU
```

(28)applyPromotion Trigger
The trigger applyPromotion is tested along with part 3-18 Procedure calculateOrderPrice.

(29)unlistProductIfNotAvailable Trigger
The trigger unlistProductIfNotAvailable is tested along with Part3-10 Procedure changeInventoryUnits.