CS 5200
Final Project
By Tuan Nguyen

Functionality:

For this project, we designed the functions for two kinds of users: one is for customers who can create their information and place orders on the site, and another one is for a store manager who can update a new product, inventory, price.

For the customer's side, the customer should register their information like Name, Address, City, State, Country, and Postal first to the database. The system will validate the inputs' format and save this information into the database. If the inputs' format is not correct, it will throw error messages. After registration, the customer can now shop on the site by placing single or multiple items per order placement. Besides that, the customer also has an option for their ordered item(s) to be shipped wholly or partially in case of inventory scarcity.

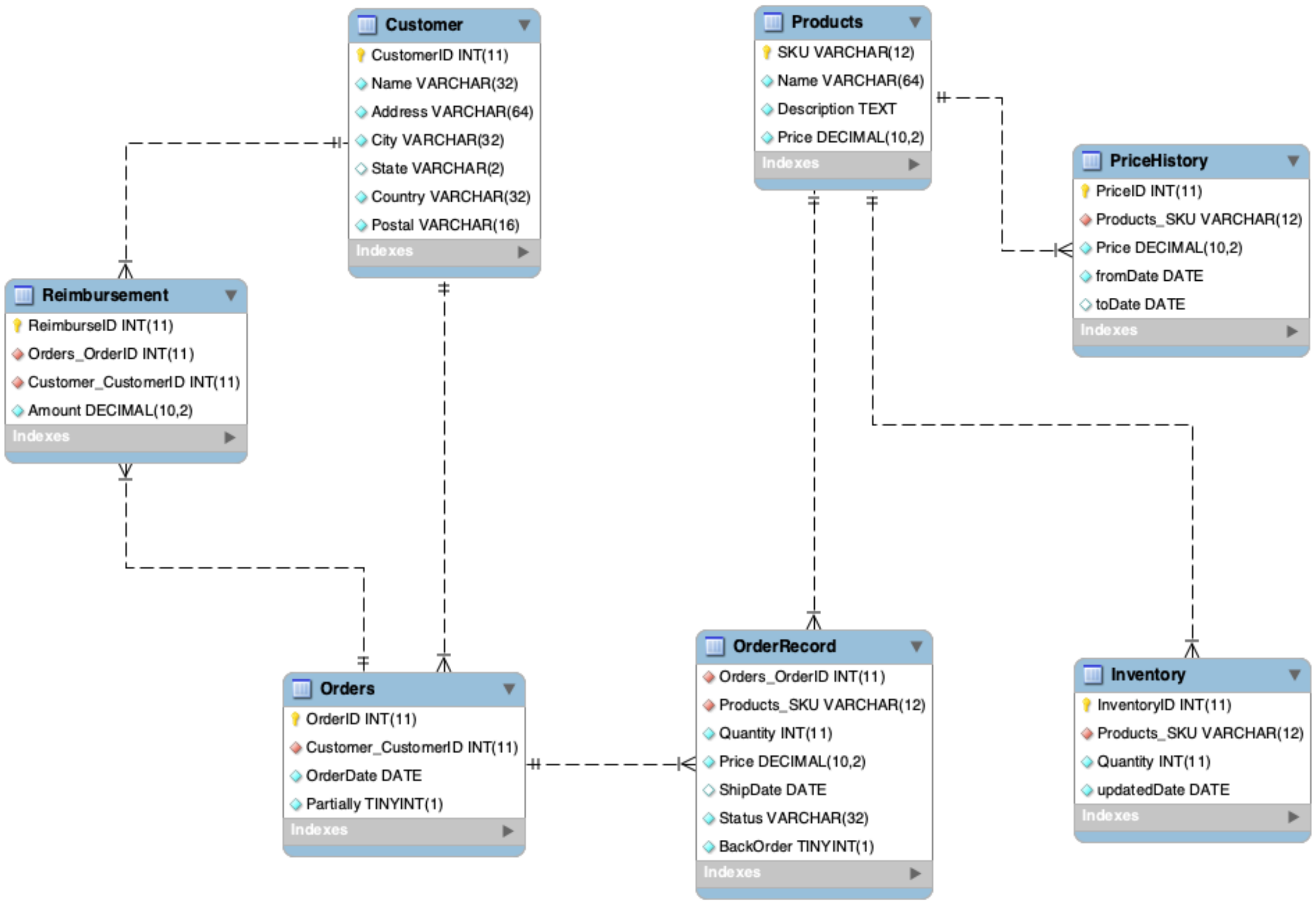
For the manager's side, the manager can add new products' information like SKU number, name of the product, description, and price. The system will also validate the inputs' format and save this information into the database. If the inputs' format is not correct, it will throw error messages. Besides adding a new product(s), the manager also can add or update the inventory quantity of these products. The system will validate the inputs' format and check if the product exists or not before saving it to the database. Furthermore, the manager also has the power to update the price of the product for new promotions. For this function, the manager only needs to enter the product's SKU, its new price, the date that he wants the new price promotion to kick in. If the manager wants to check for the price history, he even has the option to check all the prices of the products.

Smart system functionality:
Pre Order or backorder scenario: The customer is allowed to place a backorder if there is not enough inventory of that product. The customer is also allowed to preorder the product that is just posted on the site even if there is no inventory of that product in the database. When the manager updates the quantity of inventory, the pre order or backorder will be automatically fulfilled.
Reimbursement scenario: When the customer has a backorder, and there is a new lower price promotion. When the fulfillment date of the backorder after the new lower price promotion date, the system will automatically issue a price match reimbursement to the order with the back-ordered item. The reimbursement amount is equal to the prices' difference times the quantity, and automatically subtract from the order's total through trigger.
Recommended Products scenario: Throughout all of the order records, the system will automatically check if there are groups of products that the customers often buy them together. If there are 2 or more orders that have these group of products goes together, the system will automatically recommend these group of products. If not, the Recommended Products is NULL

Assumption:
Once an order is placed, it cannot be canceled in the future.
In the order placement function, Partially Input is either 1 or 0
For all the functions that require date input, the date format inputs are "yyyy-mm-dd" as MySQL Date type

DESIGN:

EER Diagram



🔑 is a Yellow (non Red) Key so it's only a Primary Key
◆ is a blue lined filled diamond so it's a NOT NULL simple attribute
◆ is a red colored filled diamond so it's a NOT NULL Foreign Key
◇ is a blue lined not filled diamond so it's a simple attribute which can be NULL

TABLES:

Customer table:
Customer ID is the primary key: An integer of up to 11 digits. Not NULL
Name: A string of up to 32 characters. Not NULL
Address: A string of up to 32 characters. Not NULL
City: A string of up to 32 characters. Not NULL
State: A string of up to 2 characters. Can be NULL, since it can be shipped internationally
Country: A string of up to 32 characters. Not NULL
Postal: A string of up to 16 characters. Not NULL

Products table:
SKU is the primary key: A string of up to 12 characters. Not NULL
Name: A string of up to 64 characters. Not NULL
*Description: A text in String characters. Not NULL
Price: Integers up to 10 digits with 2 decimal places. Not NULL

Orders table:
OrderID is the primary key: An integer of up to 11 digits. Not NULL
CustomerID is a foreign key (references Customer table): An integer of up to 11 digits. Not NULL
OrderDate: A string input in the format of ‘yyyy-mm-dd’. Not NULL
Partially: A boolean value of 1 or 0, 1 signifying True and 0 False. Not NULL

OrderRecord table:
OrderID is a foreign key (references Orders table): An integer of up to 11 digits. Not NULL
SKU is a foreign key (references Products table): A string of up to 12 characters. Not NULL
Quantity: An integer of up to 11 digits. Not NULL
Price: Integers up to 10 digits with 2 decimal places. Not NULL
ShipDate: A string input in the format of ‘yyyy-mm-dd’. Can be NULL if an item is not shipped.
Status: A string of up to 32 characters. Not NULL
BackOrder: A boolean value of 1 or 0, 1 signifying True and 0 False. Not NULL
Order Record Triggers:
Inventory Update Trigger: this trigger checks if there is an order in OrderRecord that the same SKU as SKU insert is in backorder. If that SKU is not in backorder, then it deducts the order quantity from the inventory quantity of that SKU.

```
BEFORE INSERT ON `Orderrecord`
FOR EACH ROW
BEGIN
SET @PrevBackOrder:= (SELECT MIN(OrderID) FROM OrderRecord WHERE BackOrder = 1
AND SKU = NEW.SKU);
IF (@PrevBackOrder IS NULL) THEN
        UPDATE Inventory SET Quantity = Quantity - NEW.Quantity WHERE SKU = NEW.SKU;
END IF;
END
```

Price Update trigger: this trigger checks if the new price of the SKU in the price history table is less than the price when the customer bought that product. If it is less than the price when the customer bought, it calculates the amount difference by subtracting the price in OrderRecord to the price from price history and then time Quantity. Finally, it inserts that amount difference, customerID, orderID into the Reimbursement table.

```
BEFORE UPDATE ON `Orderrecord` FOR EACH ROW BEGIN
SET @NewPrice:= (SELECT Price FROM PriceHistory WHERE fromDate IN (SELECT
MAX(fromDate) FROM PriceHistory WHERE SKU = NEW.SKU) AND SKU = NEW.SKU);
SET @CustomerID:= (SELECT CustomerID FROM Orders WHERE OrderID = NEW.OrderID);
IF (NEW.Price > @NewPrice) THEN
        SET @Amount = (NEW.Price - @NewPrice) * NEW.Quantity;
   INSERT INTO Reimbursement(CustomerID, OrderID, Amount) VALUES (@CustomerID,
NEW.OrderID, @Amount);
END IF;
END
```

Inventory table:
InventoryID is a primary key: An integer of up to 11 digits. Not NULL
SKU is a foreign key (references Products table): A string of up to 12 characters. Not NULL
Quantity: An integer of up to 11 digits. Not NULL
updatedDate: A string input in the format of 'yyyy-mm-dd'. Can be NULL.

PriceHistory table:
PriceID is a primary key: An integer of up to 11 digits. Not NULL
SKU is a foreign key (references Products table): A string of up to 12 characters. Not NULL
Price: Integers up to 10 digits with 2 decimal places. Not NULL
fromDate: A string input in the format of 'yyyy-mm-dd'. Not NULL.
toDate: A string input in the format of 'yyyy-mm-dd'. Can be NULL.

Reimbursement table:
ReimburseID is a primary key: An integer of up to 11 digits. Not NULL
OrderID is a foreign key (references Orders table): An integer of up to 11 digits. Not NULL
CustomerID is a foreign key (references Customer table): An integer of up to 11 digits.
Amount: Integers up to 10 digits with 2 decimal places. Not NULL

STORED PROCEDURES:

CustomerInfo:
The procedure takes in customerID input and matches it to Customer table to display customer information.
Parameter: customerIDInput INT
Return: none
Error : none

CustomerOrder:
The Procedure takes in customerID input and displays customer orders.
Parameter: customerIDInput INT
Return: none
Error: none

InsertCustomer:
The Procedure takes in all customer information and inserts information into Customer table.
Parameter: 'NameInput' VARCHAR(32), 'AddressInput' VARCHAR(64), 'CityInput' VARCHAR(32), 'StateInput' VARCHAR(2), 'CountryInput' VARCHAR(32), 'PostalInput' VARCHAR(16).
Return: none
Error: none

InsertInventory:
The procedure takes in all product information and inserts information into Inventory table.
Parameter: `SKUInput` VARCHAR(12), `QuantityInput` INT, `DateInput` DATE)
Return: none
Error: if product does not exist, throw 'Product does not exist' message error.

InsertOrder:
The procedure takes in all order information and inserts information into the Order table.

Parameter: `CustomerIDInput` INT, `OrderDateInput` DATE, `PartiallyInput` BOOLEAN, `SKUList` TEXT, `QuantityList` TEXT)
Return: none
Error: none

InsertOrderRecord:
The procedure takes in all order information and inserts information into the OrderRecord table.
Parameter: `OrderID1` INT, `SKUInput` VARCHAR(12), `QuantityInput` INT, `OrderDate` DATE, `Part` BOOLEAN)
Return: none
Error: if product does not exist, throw 'Product does not exist' message error.

InsertPrice:
The procedure takes in all product information such as SKU, Price, and date product was ordered and inserts information into PriceHistory table.
Parameter: `SKUInput` VARCHAR(12), `PriceInput` DECIMAL(10,2), IN `fromDateInput` DATE)
Return: none
Error: if product does not exist, throw 'Product does not exist' message error.

InsertProduct:
The procedure takes in all product information and inserts information into Product table.
Parameter: `SKUInput` VARCHAR(12), `NameInput` VARCHAR(64), `DescriptionInput` TEXT, `PriceInput` DECIMAL(10,2), `DateInput` DATE)
Return: none
Error: if product exists, throw 'Product already exists' message error.

InventoryDisplay:
The procedure displays inventory attributes: SKU, Quantity, updatedDate
Parameter: none
Return: none
Error: none

ProductInfo:
The procedure takes in product SKU and displays appropriate product information
Parameter: `SKUInput` VARCHAR(12)
Return: none
Error: if the product exists, throw 'Product already exists' message error.

ValidateCustomerAddress:
The procedure checks customer address is a valid entry in the form of alphabets and numbers
Parameter: `address` VARCHAR(64))
Return: none
Error: If address entry not valid, throw 'Invalid address' message error

ValidateCustomerCity:
The procedure check’s customer’s city is a valid entry in the form of alphabets
Parameter: `City` VARCHAR(32)
Return: none
Error: If city entry not valid, throw 'Invalid city name' message error

ValidateCustomerCountry:
The procedure check's customer's country is a valid entry in the form of alphabets
Parameter: \`Country\` VARCHAR(32)
Return: none
Error: If country entry not valid, throw 'Invalid country' message error

ValidateCustomerName:
The procedure check's customer's name is a valid entry in the form of alphabets
Parameter: \`Name\` VARCHAR(32
Return: none
Error: if name entry not valid, throw 'Invalid Name' message error

ValidateCustomerPostal:
The procedure check's customer's postal code is a valid entry in the form of alphabets, numbers or "-"
Parameter: \`Postal\` VARCHAR(16
Return: none
Error: If postal code entry not valid, throw 'Invalid Postal' message error

ValidateCustomerState:
The procedure check's customer's state is a valid entry in the form of alphabets
Parameter: \`State\` VARCHAR(2)
Return: none
Error: If state entry not valid, throw 'Invalid state' message error

ValidatePrice:
The procedure check's price is a positive entry
Parameter: \`Price\` DECIMAL(10,2)
Return: none
Error: If price entry not valid, throw 'Invalid Price' message error

ValidateQuantity:
The procedure check's quantity is a positive entry
Parameter: Quantity\` INT
Return: none
Error: If quantity entry not valid, throw 'Invalid Quantity' message error

ValidateSKU:
The procedure check's SKU is a valid entry in the form of AA-NNNNNN-CC where A is an upper-case letter, N is a digit from 0-9, and C is either a digit or an uppercase letter.
Parameter: \`SKU\` VARCHAR(12)
Return: none
Error: If SKU entry not valid, throw 'Invalid SKU' message error

Application Programming Interface (API):

**Note: Because of the system design and constraints, on the manager side, the manager should run InsertProduct API first, and on the customer side, the customer should run InsertCustomer API first. For the test cases, please run the InsertProduct API first, then InsertCustomer API second, after that you can run other APIs.

InsertProduct (SKU VARCHAR(12), Name VARCHAR(64), Description TEXT, Price DECIMAL(10,2), Date DATE)
Example Code:
CALL InsertProduct('AN-111111-II', 'Macbook Air', 'Apple Mac Line', 1600, '2019-11-01');
CALL InsertProduct('AN-22222-II', 'Magic Keyboard', 'Apple keyboard', 600, '2019-11-01');

Parameters:
The API requires 5 inputs which are SKU, Name, Description, Price, and Date.

Function:
It validates the SKU and Price first and checks the SKU input exists in the Product table or not if exists it throws an error message. Then, it takes SKU, Name, Description, and Price to create new product information into the Products table, also take the SKU, Price, and Date to create new product price entry into the PriceHistory table.

Error Messages:
“Invalid SKU”: if the SKU format does not correct as in the requirement
“Invalid Price”: if the Price input is not a positive number
“Product already exists”: if the product is in the Products table already, since the SKU is the primary key, it should be a unique value.

InsertCustomer(NameInput VARCHAR(32), AddressInput VARCHAR(64), CityInput VARCHAR(32), StateInput VARCHAR(2), CountryInput VARCHAR(32), PostalInput VARCHAR(16));
Example Code:
CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States','94088');

Parameters:
The API requires 6 inputs which are Name, Address, City, State, Country, and Postal.

Function:
It validates the Name Input that only takes alphabets.
It validates the Address Input that takes numeric and alphabet.
It validates the City Input that only takes alphabets.
It validates the State Input that only takes alphabets.
It validates the Country Input that only takes the alphabet.
It validates the Postal Input that takes numeric and alphabet. Since it can be shipped internationally, some countries’ zip codes are not numeric only.
After all the validations, it inserts new the customer’s information into the Customer table.

Error Messages:
“Invalid Name”: if there are special characters or numbers in Name Input

“Invalid Address”: if there are special characters (except “/” character since there are some streets’ names with “/”) in Address Input
“Invalid City”: if there are special characters or numbers in City Input
“Invalid State”: if there are special characters or numbers in State Input
“Invalid Country”: if there are special characters or numbers in Country Input
“Invalid Postal”: if there are special characters (except “-” character since there are some postal in other countries with “-”) in Postal Input

InsertInventory (SKUInput VARCHAR(12), QuantityInput INT, DateInput DATE)
Example Code:

```
CALL InsertInventory('AN-111111-II', 100, '2019-11-01');
CALL InsertInventory('AN-222222-II', 100, '2019-11-01');
```

Parameters:
The API requires 3 inputs which are SKU, Quantity, and Date.

Function & Business Logic:
When there are new inventories coming, the user can use this API to insert a quantity of a new item or update a quantity of an existing item. This API validates the SKU and Quantity input first, then it checks if the product exists in the Product table or not, if not it throws an error message. After that, it checks all the BackOrder flag = 1 from oldest to newest in OrderRecord table, and if there is enough inventory, it will update the Status and switch Backorder = 0 which means the order is ready to be shipped. The inventory quantity is also deducted after the order is fulfilled. Here are the implementations of each input:

1) “SKU” Input
The SKU is required for this API, so the system can keep a record of the inventory of an item. After InsertInventory API is called, it validates the SKU Input format first, then checks on the Products table to make sure if the SKU Input exists in the Product table or not, if not it will throw error messages.

2) “Quantity” Input
The SKU is required for this API, so the system can keep a record of how many units left in stock. After InsertInventory API is called, it validates the Quantity Input to make sure the input is a positive integer. Negative Quantity is not allowed in the table.

3) “Date” Input
It is important to keep track of the Date when the inventory is available. Therefore, if the new Date is inserted or updated, the items with backorder can be fulfilled and shipped the next day after the available date of the inventory.

Error Messages:
“Invalid SKU”: if the SKU format does not correct as in the requirement
“Invalid Price”: if the Price input is not a positive number
“Product does not exist”: since the SKU of the Inventory table is foreign key from Products table, the SKU input should exist in the Products table.

InsertOrder (CustomerID INT, OrderDate DATE, Partially BOOLEAN, SKUList TEXT, QuantityList TEXT)
Example Code:

```
CALL InsertOrder(1, '2019-12-01', 1, 'AN-111111-II, AN-333333-II, AN-222222-II', '10, 10, 10');
CALL InsertOrder(1, '2019-12-01', 1, 'AN-111111-II', '10');
CALL InsertOrder(1, '2019-12-01', 0, 'AN-111111-II, AN-333333-II, AN-222222-II', '10, 10, 10');
CALL InsertOrder(1, '2019-12-01', 0, 'AN-111111-II', '10');
```

Parameters:
The API requires 5 inputs which are CustomerID, OrderDate, Partially, SKUList and QuantityList.

Function and Business Logic:
The customers can use this API to make new orders. When the API is called, at first, it inserts a new record to the Orders table including new OrderID, CustomerID, OrderDate, and Partially inputs. Then it will separate SKUList and QuantityList by comma and call a stored procedure InsertOrderRecord. Here are the implementations of each input:

1) “CustomerID” and “OrderDate” Input
When the customer makes a new order, this API requires to put the CustomerID and Order Date, so the system can keep track of who orders and on what date.

2) “Partially” Input
With this extra parameter, while making an order, the customer can choose the item that they are ordering to ship partially or not.

Scenario 1: If the customer chooses to ship the item partially, the customer can enter 1 for this parameter. When the “Partially” parameter is 1, if the inventory does not have enough quantity of the ordered item, it ships the “Partially” of the item quantity first. When it has more inventory for that item in the future, it will ship the rest of the item quantity later.

Scenario 2: If the customer chooses to ship the item as a whole, the customer can enter a 0 for this parameter. When the “Partially” parameter is 0, if the inventory does not have enough quantity of the ordered item, it keeps a hold of the whole item quantity. When it has enough inventory for that item in the future, it will ship all of the item quantity.

3) “SKUList” and “QuantityList” Input
With these parameters, the customer can order one or multiple items as a list in one order. Since we only use MySQL syntax for this system, so the type of inputs of these two lists are text in MySQL. To make sure the valid SKUList and QuantityList inputs, the API breaks down all elements from the text list and validate each element in the list one by one.

Scenario 1: If the customer just ordered one item in an order, the customer can input SKU of the item as a text to “SKUList”, as well as the quantity as a string to “QuantityList”. For the quantity input as a text, MySQL automatically converts from digit string to numeric value.

Scenario 2: If the customer orders multiple items in an order, the customer can input the SKU of multiple items separated by commas as a string to “SKUList”, as well as multiple quantities coordinate to the items separated by commas as a string to “QuantityList”. This API will handle and separate all of the items SKU and Quantity in the string list by commas and make multiples insert to the OrderRecord table.

Error Messages:
“Invalid SKU”: if the SKU format does not correct as in the requirement
“Invalid Quantity”: if the Quantity input is not a positive number
“Product does not exist”: since the SKU of the Inventory table is foreign key from Products table, the SKU input should exist in the Products table.

InsertPrice (SKU VARCHAR(12), Price DECIMAL(10,2), fromDate DATE)
Example Code:

```
CALL InsertPrice('AN-111111-II', 1500, '2019-11-30');
CALL InsertPrice('AN-222222-II', 500, '2019-11-30');
```

Parameters:
The API requires 3 inputs which are SKU, Price, and fromDate.

Function and Business Logic:
It validates the SKU and Price first and checks the SKU input exists in the Product table, if not it throws an error message. Before it creates a new product’s price entry, it also checks if there is any SKU as same as the new SKU Input which means that the user updates the price of the same product. The API will automatically update the toDate of the old price of the product to date before the fromDate Input of the new price. Since we want to keep all records of the price history of the product, the PriceHistory table keeps all the old prices of the product. Then, it inserts the new price of the product to the table and also updates the new price of that product in the Products table.

Error Messages:
“Product does not exist”: since the SKU of the PriceHistory table is foreign key from Products table, the SKU input should exist in the Products table.

ProductInfo(SKU VARCHAR(12))
Example Code:

```
CALL ProductInfo('AN-111111-II');
```

Parameters:
The API requires 1 input which is SKU

Function and Business Logic:
The API displays the Product’s SKU, Name, Description, and Price.
It also displays up to 3 Recommended Products which were ordered with the display product most from all orders history.

Error Messages:
“Product does not exist”: since the SKU input is the primary key of Products table, the product should exist.

CustomerInfo(CustomerIDInput INT)
Example Code:
CALL CustomerInfo(1);

Parameters:
The API requires 1 input which is CustomerID

Function:
The API displays the Customer's Name, Address, City, State, Country, and Postal where CustomerID is CustomerID Input.

Error Messages:
"Customer does not exist": since the CustomerID input is the primary key of the Customer table, the CustomerID should exist.

InventoryDisplay()
Example Code:
CALL InventoryDisplay()

Function:
The API displays SKU, Quantity, and updatedDate of all inventory items.

CustomerOrder(CustomerID INT)
Example Code:
CALL CustomerOrder(1);

Parameters:
The API requires 1 input which is CustomerID

Function and Business Logic:
The API displays product Name, Quantity, Price, Total, OrderDate, ShipDate, and Status where CustomerID is CustomerID Input. At the end of each order, it also shows the grand total of the order and reimbursement amount if there is any reimbursement.

Error Messages:
"Customer does not exist": since the CustomerID input is the foreign key of the Orders, Order Record and Reimbursement tables, the CustomerID should exist.

TEST PLAN:

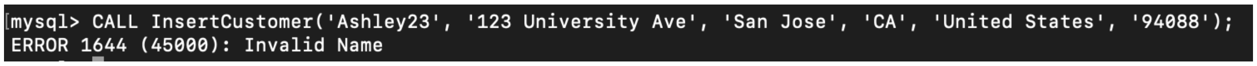
**Note: Because of the system design and constraints, on the manager side, the manager should run InsertProduct API first, and on the customer side, the customer should run InsertCustomer API first. For the test cases, please run the InsertProduct API first, then InsertCustomer API second, after that you can run other APIs.

INPUTS' VALIDATIONS

1. **Invalid Inputs (use invalidInputs.sql to test):**

Customer Name validation:

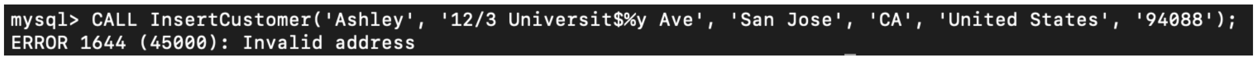
CALL InsertCustomer('Ashley23', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');



```
[mysql> CALL InsertCustomer('Ashley23', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
ERROR 1644 (45000): Invalid Name
```

Customer Address validation:

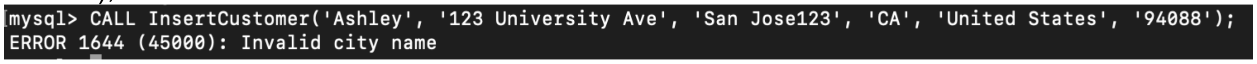
CALL InsertCustomer('Ashley', '12/3 Universit$%y Ave', 'San Jose', 'CA', 'United States', '94088');



```
mysql> CALL InsertCustomer('Ashley', '12/3 Universit$%y Ave', 'San Jose', 'CA', 'United States', '94088');
ERROR 1644 (45000): Invalid address
```

Customer City validation:

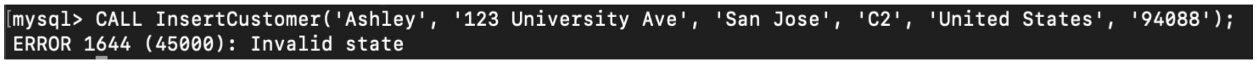
CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose123', 'CA', 'United States', '94088');



```
[mysql> CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose123', 'CA', 'United States', '94088');
ERROR 1644 (45000): Invalid city name
```

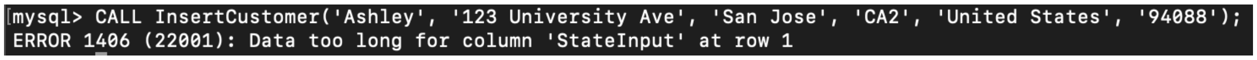
Customer State validation:

CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'C2', 'United States', '94088');



```
[mysql> CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'C2', 'United States', '94088');
ERROR 1644 (45000): Invalid state
```

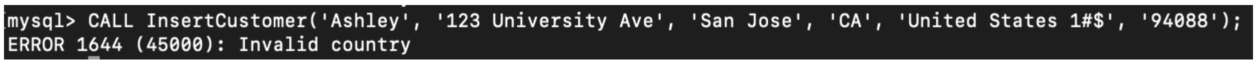
CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA2', 'United States', '94088');



```
[mysql> CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA2', 'United States', '94088');
ERROR 1406 (22001): Data too long for column 'StateInput' at row 1
```

Customer Country validation:

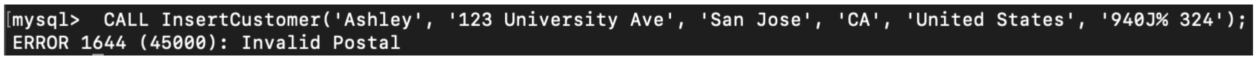
CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States 1#$', '94088');



```
mysql> CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States 1#$', '94088');
ERROR 1644 (45000): Invalid country
```

Customer Postal validation:

CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '940J% 324');

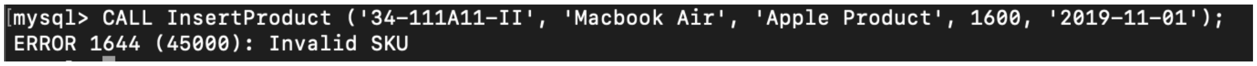


```
[mysql>  CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '940J% 324');
ERROR 1644 (45000): Invalid Postal
```

SKU format validation:

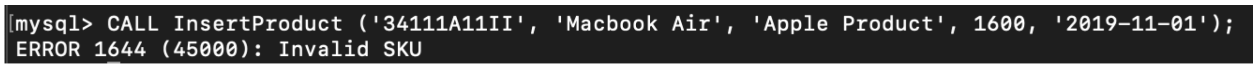
Insert Product API:

CALL InsertProduct ('34-111A11-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');



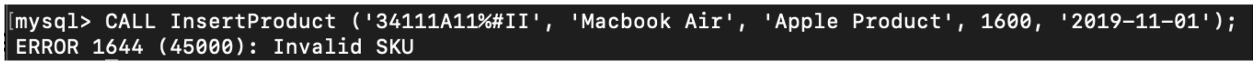
```
[mysql> CALL InsertProduct ('34-111A11-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
ERROR 1644 (45000): Invalid SKU
```

CALL InsertProduct ('34111A11II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');



```
[mysql> CALL InsertProduct ('34111A11II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
ERROR 1644 (45000): Invalid SKU
```

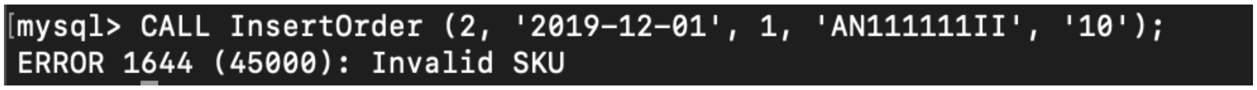
CALL InsertProduct ('34111A11%#II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');



```
[mysql> CALL InsertProduct ('34111A11%#II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
ERROR 1644 (45000): Invalid SKU
```

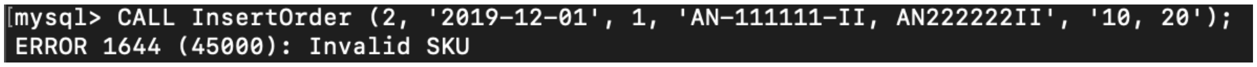
Insert Order API:

(Single item)
CALL InsertOrder (2, '2019-12-01', 1, 'AN111111II', '10');



```
[mysql> CALL InsertOrder (2, '2019-12-01', 1, 'AN111111II', '10');
ERROR 1644 (45000): Invalid SKU
```

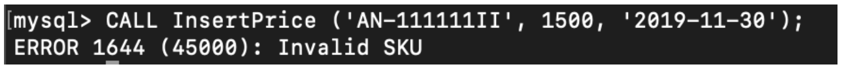
(Multiple items)
CALL InsertOrder (2, '2019-12-01', 1, 'AN-111111-II, AN222222II', '10, 20');



```
[mysql> CALL InsertOrder (2, '2019-12-01', 1, 'AN-111111-II, AN222222II', '10, 20');
ERROR 1644 (45000): Invalid SKU
```

Insert Price API:

CALL InsertPrice ('AN-111111II', 1500, '2019-11-30');

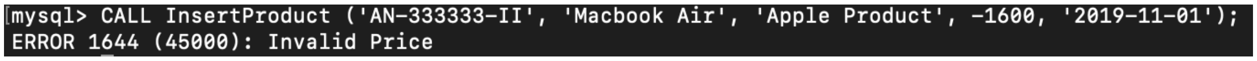


```
[mysql> CALL InsertPrice ('AN-111111II', 1500, '2019-11-30');
ERROR 1644 (45000): Invalid SKU
```

Price Positive Check validation:

Insert Product API:

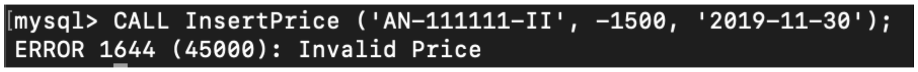
CALL InsertProduct ('AN-333333-II', 'Macbook Air', 'Apple Product', -1600, '2019-11-01');



```
mysql> CALL InsertProduct ('AN-333333-II', 'Macbook Air', 'Apple Product', -1600, '2019-11-01');
ERROR 1644 (45000): Invalid Price
```

Insert Price API:

CALL InsertPrice ('AN-111111-II', -1500, '2019-11-30');

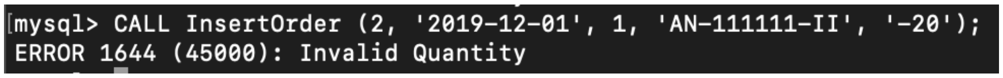


```
mysql> CALL InsertPrice ('AN-111111-II', -1500, '2019-11-30');
ERROR 1644 (45000): Invalid Price
```

Quantity positive Check validation:

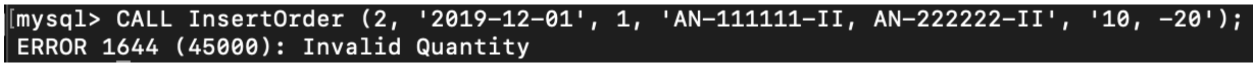
Insert Order API:

CALL InsertOrder (2, '2019-12-01', 1, 'AN-111111-II', '-20'); (Single item)



```
mysql> CALL InsertOrder (2, '2019-12-01', 1, 'AN-111111-II', '-20');
ERROR 1644 (45000): Invalid Quantity
```

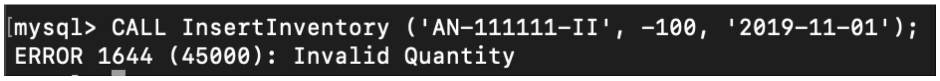
CALL InsertOrder (2, '2019-12-01', 1, 'AN-111111-II, AN-222222-II', '10, -20'); (Multiple items)



```
mysql> CALL InsertOrder (2, '2019-12-01', 1, 'AN-111111-II, AN-222222-II', '10, -20');
ERROR 1644 (45000): Invalid Quantity
```

Insert Inventory API:

CALL InsertInventory ('AN-111111-II', -100, '2019-11-01');

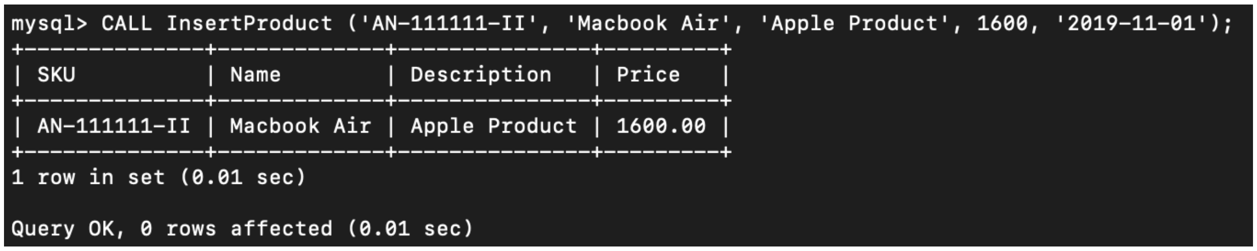


```
mysql> CALL InsertInventory ('AN-111111-II', -100, '2019-11-01');
ERROR 1644 (45000): Invalid Quantity
```

2. Valid Inputs (use validInputs.sql to test):

Insert Product API

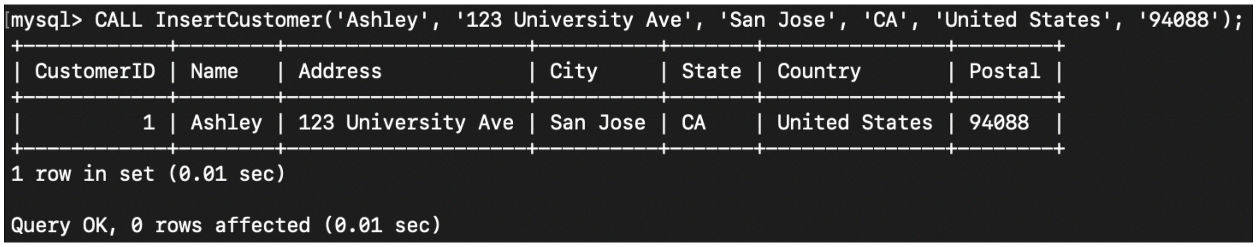
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');



```
mysql> CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1600.00 |
+--------------+-------------+---------------+---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Insert Customer API

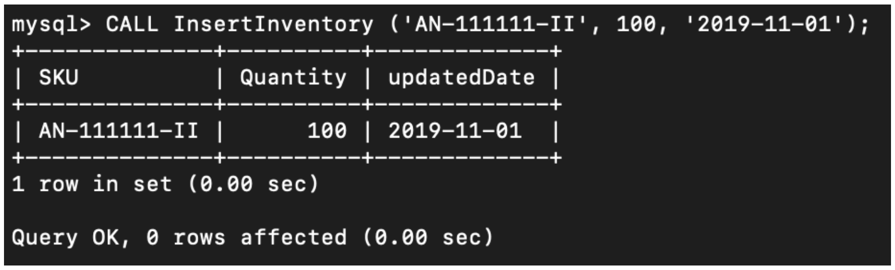
CALL InsertCustomer ('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');



```
[mysql> CALL InsertCustomer('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
+------------+--------+--------------------+----------+-------+---------------+--------+
| CustomerID | Name   | Address            | City     | State | Country       | Postal |
+------------+--------+--------------------+----------+-------+---------------+--------+
|          1 | Ashley | 123 University Ave | San Jose | CA    | United States | 94088  |
+------------+--------+--------------------+----------+-------+---------------+--------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Insert Inventory API

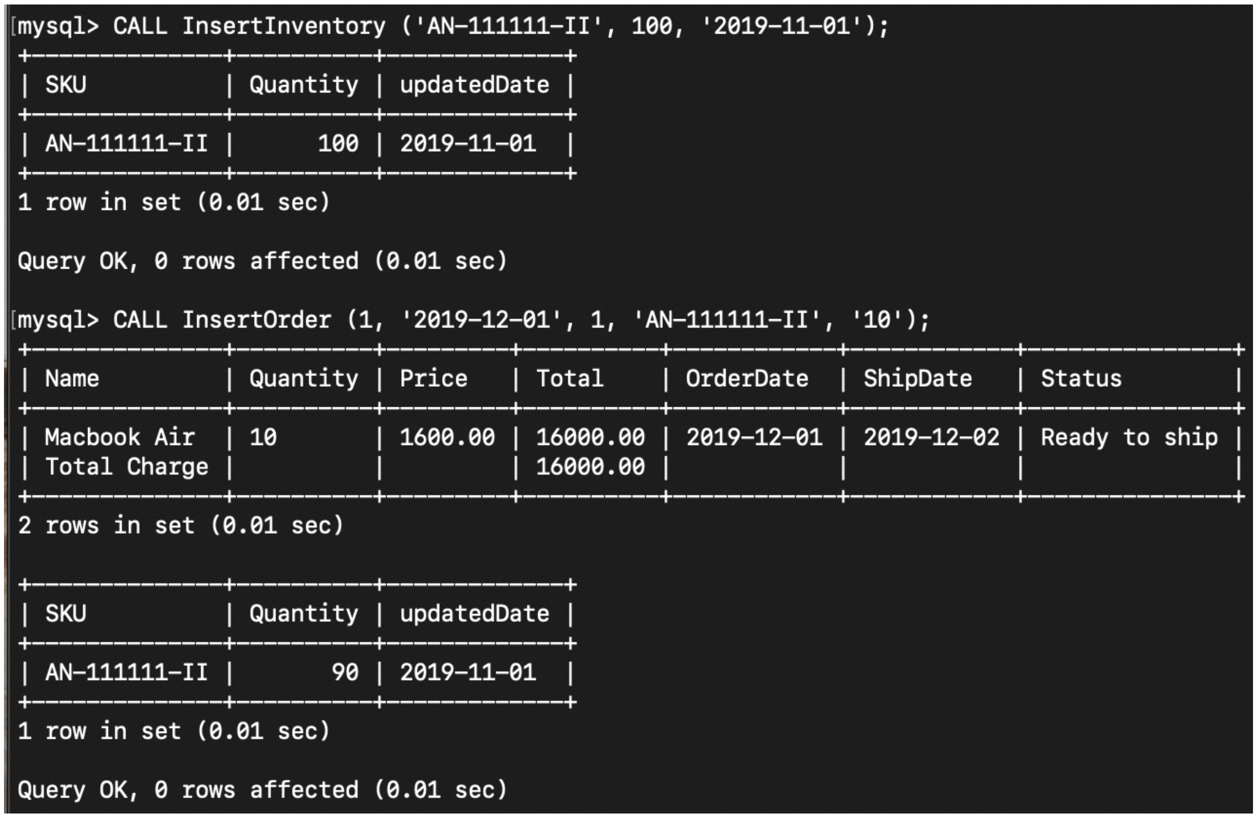
CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');



```
mysql> CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Insert Order API

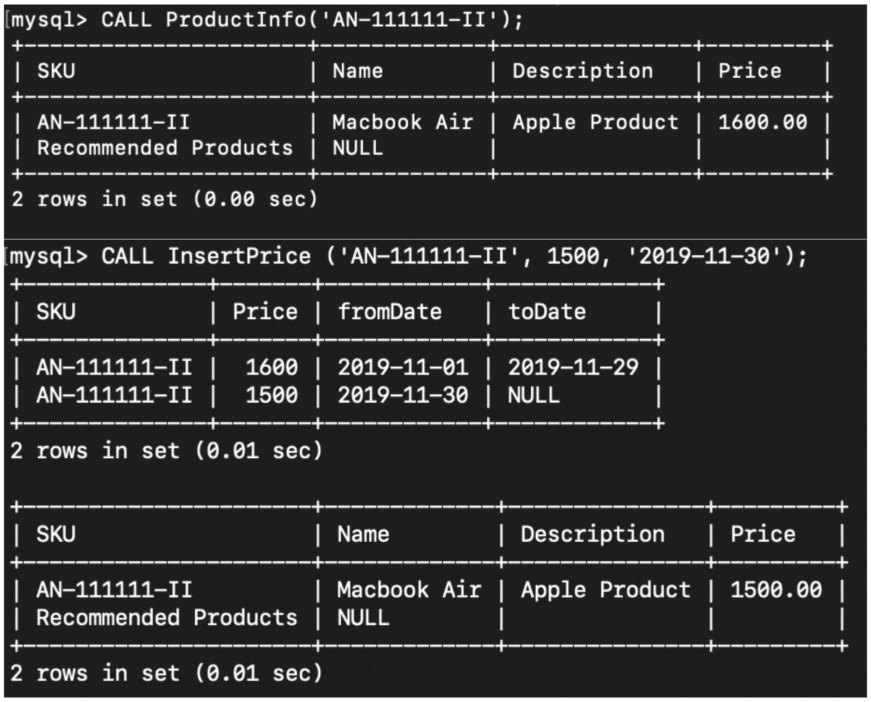
CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II', '10');



```
[mysql> CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

[mysql> CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II', '10');
+--------------+----------+---------+----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total    | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+----------+------------+------------+---------------+
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-01 | 2019-12-02 | Ready to ship |
| Total Charge |          |         | 16000.00 |            |            |               |
+--------------+----------+---------+----------+------------+------------+---------------+
2 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |       90 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Insert Price API

CALL InsertPrice ('AN-111111-II', 1500, '2019-11-30');

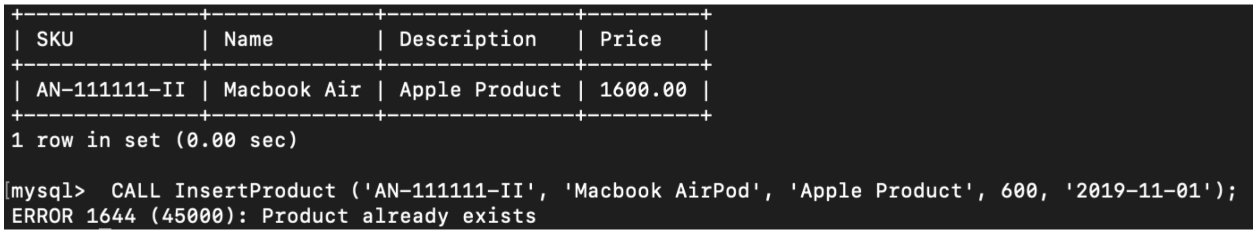


```
mysql> CALL ProductInfo('AN-111111-II');
+----------------------+-------------+----------------+---------+
| SKU                  | Name        | Description    | Price   |
+----------------------+-------------+----------------+---------+
| AN-111111-II         | Macbook Air | Apple Product  | 1600.00 |
| Recommended Products | NULL        |                |         |
+----------------------+-------------+----------------+---------+
2 rows in set (0.00 sec)

mysql> CALL InsertPrice ('AN-111111-II', 1500, '2019-11-30');
+--------------+-------+------------+------------+
| SKU          | Price | fromDate   | toDate     |
+--------------+-------+------------+------------+
| AN-111111-II |  1600 | 2019-11-01 | 2019-11-29 |
| AN-111111-II |  1500 | 2019-11-30 | NULL       |
+--------------+-------+------------+------------+
2 rows in set (0.01 sec)

+----------------------+-------------+----------------+---------+
| SKU                  | Name        | Description    | Price   |
+----------------------+-------------+----------------+---------+
| AN-111111-II         | Macbook Air | Apple Product  | 1500.00 |
| Recommended Products | NULL        |                |         |
+----------------------+-------------+----------------+---------+
2 rows in set (0.01 sec)
```

CHECK IF PRODUCT INPUT EXISTS (use checkExists.sql to test)

Insert Product API

If there is product’s SKU in Products table already, “Insert Product” API throws “Product already exists”
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL InsertProduct ('AN-111111-II', 'Macbook Airpod', 'Apple Product', 600, '2019-11-01');

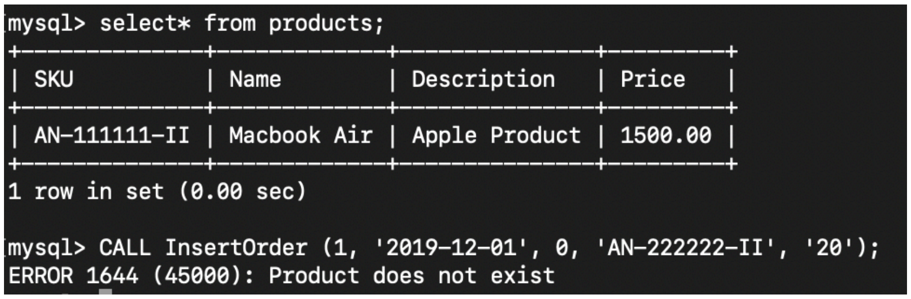


```
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1600.00 |
+--------------+-------------+---------------+---------+
1 row in set (0.00 sec)

mysql>  CALL InsertProduct ('AN-111111-II', 'Macbook AirPod', 'Apple Product', 600, '2019-11-01');
ERROR 1644 (45000): Product already exists
```

CHECK IF THE PRODUCT INPUT DOES NOT EXIST (use checkExists.sql to test)

Insert Order API

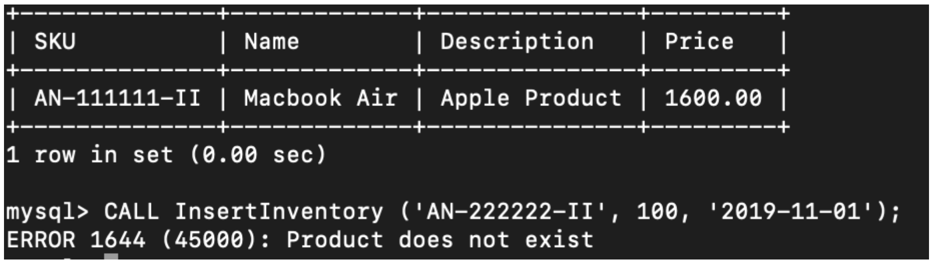
CALL InsertOrder (1, '2019-12-01', 0, 'AN-222222-II', '20');



```
mysql> select* from products;
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1500.00 |
+--------------+-------------+---------------+---------+
1 row in set (0.00 sec)

mysql> CALL InsertOrder (1, '2019-12-01', 0, 'AN-222222-II', '20');
ERROR 1644 (45000): Product does not exist
```

Insert Inventory API

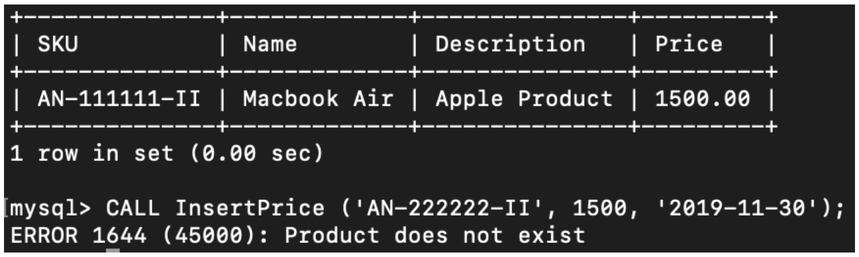
CALL InsertInventory ('AN-222222-II', 100, '2019-11-01');



```
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1600.00 |
+--------------+-------------+---------------+---------+
1 row in set (0.00 sec)

mysql> CALL InsertInventory ('AN-222222-II', 100, '2019-11-01');
ERROR 1644 (45000): Product does not exist
```

Insert Price API

CALL InsertPrice ('AN-222222-II', 1500, '2019-11-30');



```
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1500.00 |
+--------------+-------------+---------------+---------+
1 row in set (0.00 sec)

mysql> CALL InsertPrice ('AN-222222-II', 1500, '2019-11-30');
ERROR 1644 (45000): Product does not exist
```

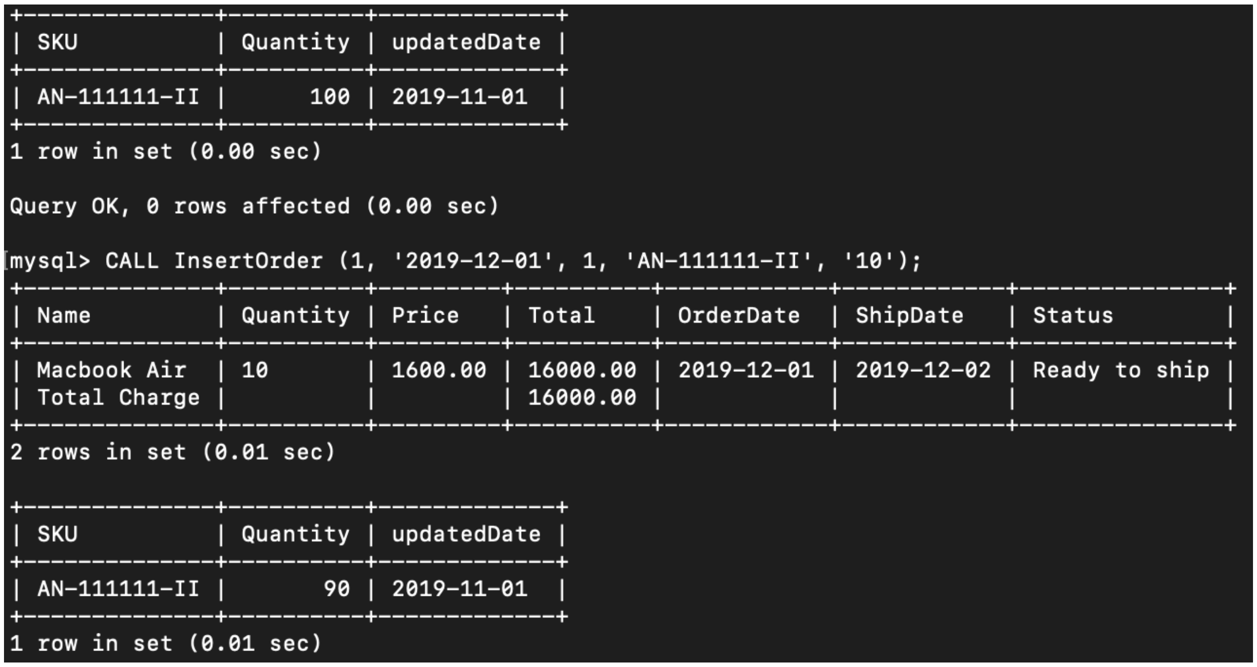
SCENARIOS TESTS

Regular single item order scenario test (use itemOrder.sql to test)

In this scenario, assuming that we have SKU “AN-111111-II” in the Products table and in Inventory of 100 quantity already. The Inventory table is the first table before the customer makes an order, and the second table is the customer orders table, and the third table is the inventory table after a customer order.

Example test run:

source createDB.txt
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL InsertCustomer ('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II', '10');



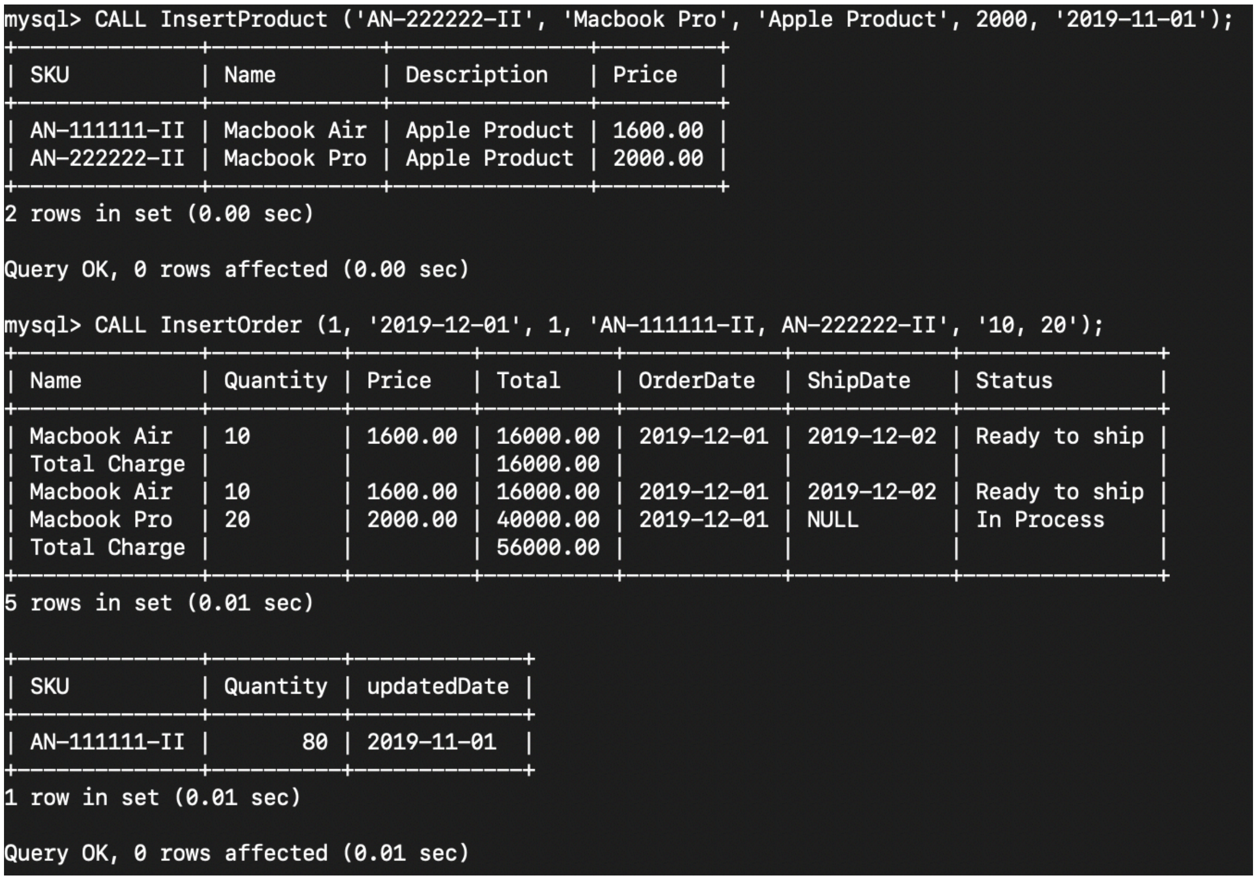
```
+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II', '10');
+--------------+----------+---------+----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total    | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+----------+------------+------------+---------------+
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-01 | 2019-12-02 | Ready to ship |
| Total Charge |          |         | 16000.00 |            |            |               |
+--------------+----------+---------+----------+------------+------------+---------------+
2 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |       90 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.01 sec)
```

Pre-order with Multiple Items in an order Scenario Test (use preorder.sql to test)

In this scenario, assuming that we have both SKU “AN-111111-II” and “AN-222222-II” in the Product table already. Then you insert the inventory for SKU “AN-111111-II”, so the SKU “AN-222222-II” is not the inventory table (first table), but the customer can make an order as a preorder. When the “AN-222222-II” is ready in the inventory, it will be shipped later.

Example test run:
continue from the test code above…

CALL InsertProduct ('AN-222222-II', 'Macbook Pro, 'Apple Product', 2000, '2019-11-01');
CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II, AN-222222-II', '10, 20');



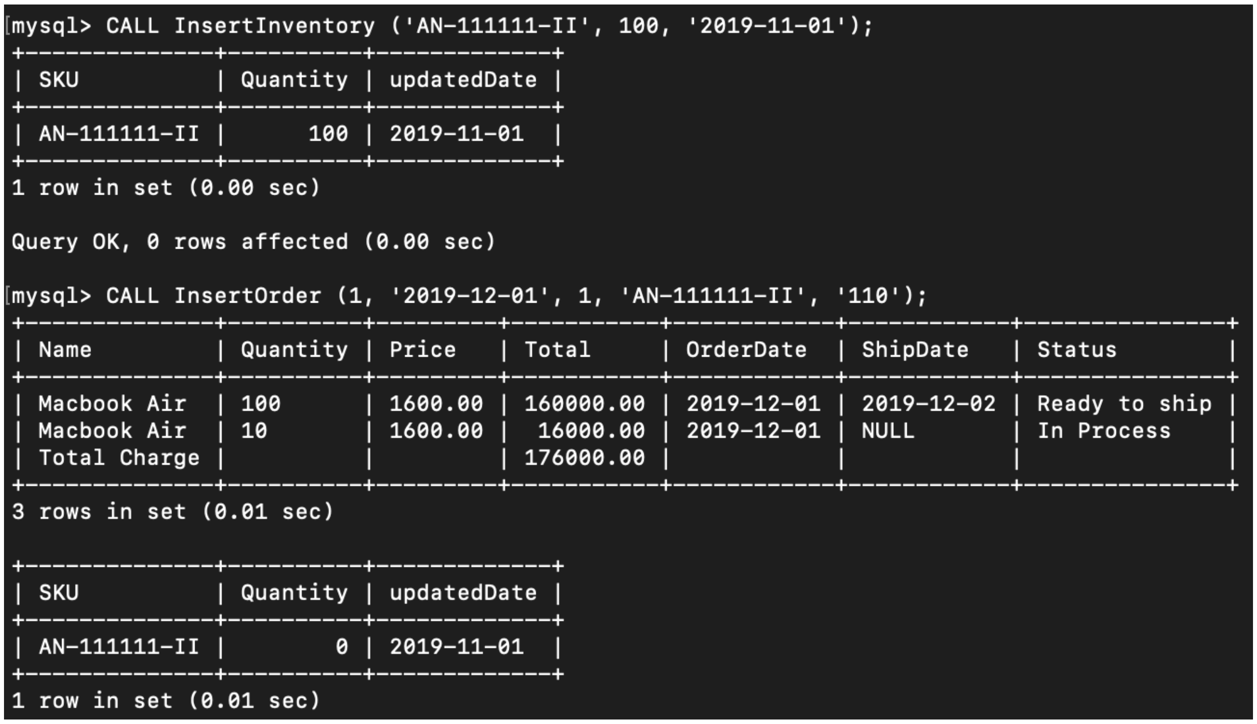
```
mysql> CALL InsertProduct ('AN-222222-II', 'Macbook Pro', 'Apple Product', 2000, '2019-11-01');
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1600.00 |
| AN-222222-II | Macbook Pro | Apple Product | 2000.00 |
+--------------+-------------+---------------+---------+
2 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II, AN-222222-II', '10, 20');
+--------------+----------+---------+----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total    | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+----------+------------+------------+---------------+
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-01 | 2019-12-02 | Ready to ship |
| Total Charge |          |         | 16000.00 |            |            |               |
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-01 | 2019-12-02 | Ready to ship |
| Macbook Pro  | 20       | 2000.00 | 40000.00 | 2019-12-01 | NULL       | In Process    |
| Total Charge |          |         | 56000.00 |            |            |               |
+--------------+----------+---------+----------+------------+------------+---------------+
5 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |       80 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

Back Order and Partial Shipment per item order scenario test (use partialShipment1.sql and partialShipment0.sql to test)

In this scenario, assuming that the quantity of the SKU “AN-111111-II” in the inventory table (first table) is 100, and the customer makes a new order of 110 quantity of that SKU. Therefore, if the customer chooses partial shipment (1), the part of the order with a quantity of 100 will ship first since there is only 100 left in the inventory. The quantity of 10 becomes a new back order record. When we have enough updated inventory quantity, the rest of the order will be shipped later.

Example test run:

source createDB.txt
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL InsertCustomer ('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II', '110');

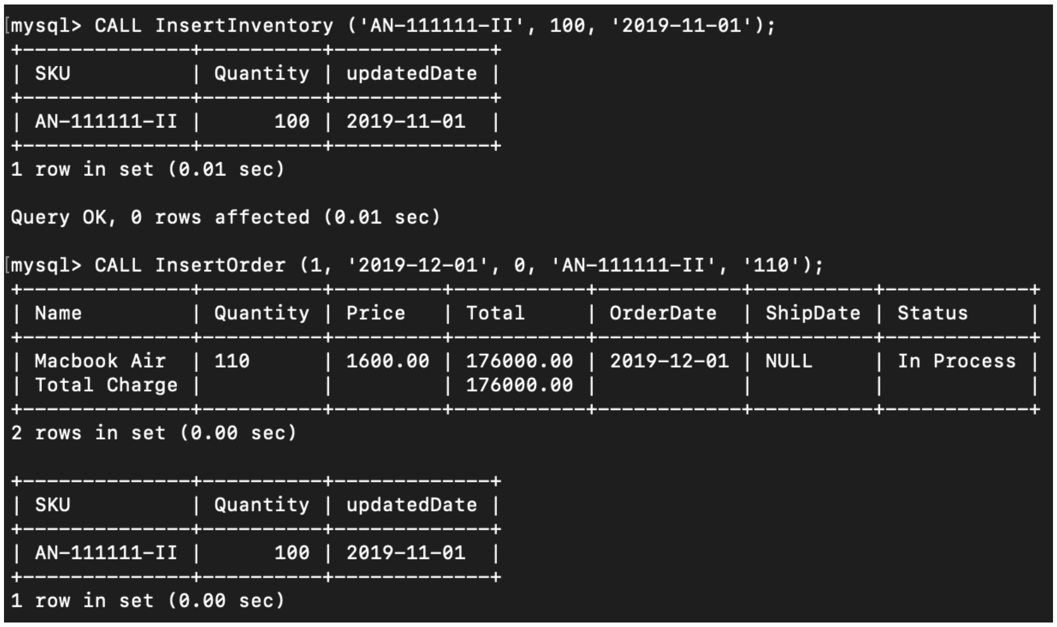


```
mysql> CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL InsertOrder (1, '2019-12-01', 1, 'AN-111111-II', '110');
+--------------+----------+---------+-----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total     | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+-----------+------------+------------+---------------+
| Macbook Air  | 100      | 1600.00 | 160000.00 | 2019-12-01 | 2019-12-02 | Ready to ship |
| Macbook Air  | 10       | 1600.00 |  16000.00 | 2019-12-01 | NULL       | In Process    |
| Total Charge |          |         | 176000.00 |            |            |               |
+--------------+----------+---------+-----------+------------+------------+---------------+
3 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |        0 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.01 sec)
```

In this scenario, assuming that the quantity of the SKU “AN-111111-II” in the inventory table (first table) is 100, and the customer makes a new order of 110 quantity of that SKU. Therefore, if the customer chooses partial shipment (0), the whole order of 110 quantity will be shipped later since there are only 100 left in the inventory. In this case, the inventory quantity still remains 100, not back to 0. However, if another customer orders this “temporarily out of stock” product, it will not take the part of 100 products in the inventory and ship them to another customer. Since these 100 products is on hold from the initial customer. When we have enough updated inventory quantity, the whole order will be shipped later.

Example test run:

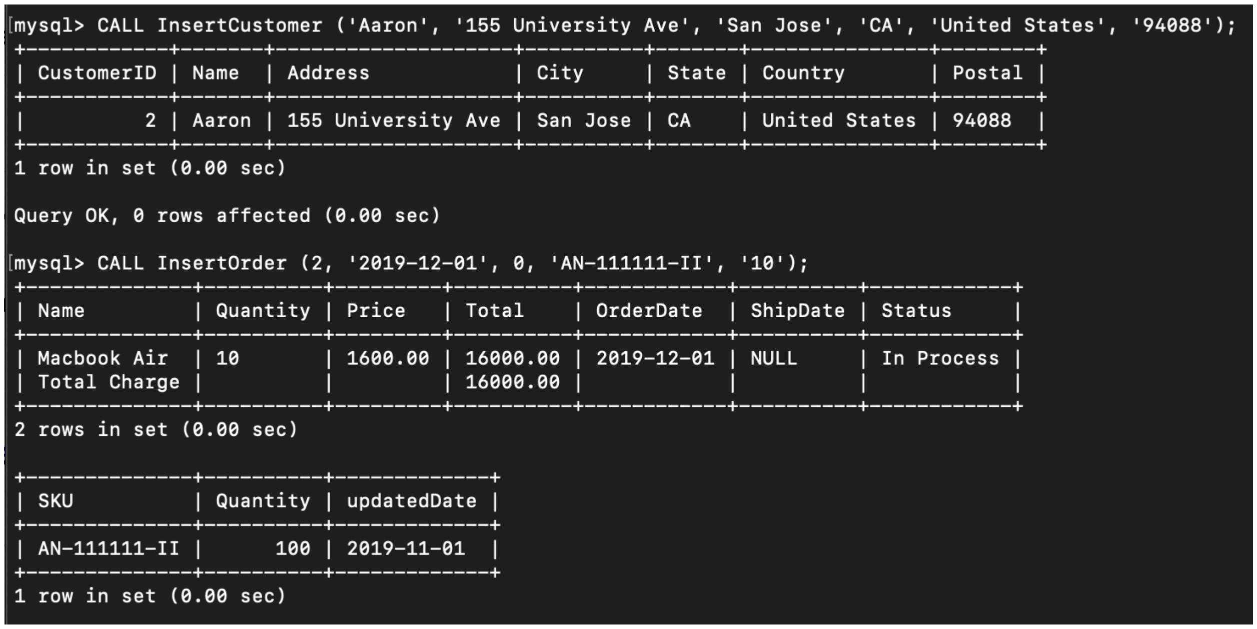
```
source createDB.txt
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL InsertCustomer ('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
CALL InsertOrder (1, '2019-12-01', 0, 'AN-111111-II', '110');
```



```
mysql> CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> CALL InsertOrder (1, '2019-12-01', 0, 'AN-111111-II', '110');
+--------------+----------+---------+-----------+------------+----------+------------+
| Name         | Quantity | Price   | Total     | OrderDate  | ShipDate | Status     |
+--------------+----------+---------+-----------+------------+----------+------------+
| Macbook Air  | 110      | 1600.00 | 176000.00 | 2019-12-01 | NULL     | In Process |
| Total Charge |          |         | 176000.00 |            |          |            |
+--------------+----------+---------+-----------+------------+----------+------------+
2 rows in set (0.00 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.00 sec)
```

If another customer orders the same product case. Continue from the code above

```
CALL InsertCustomer ('Aaron', '155 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertOrder (2, '2019-12-01', 0, 'AN-111111-II', '10');
```



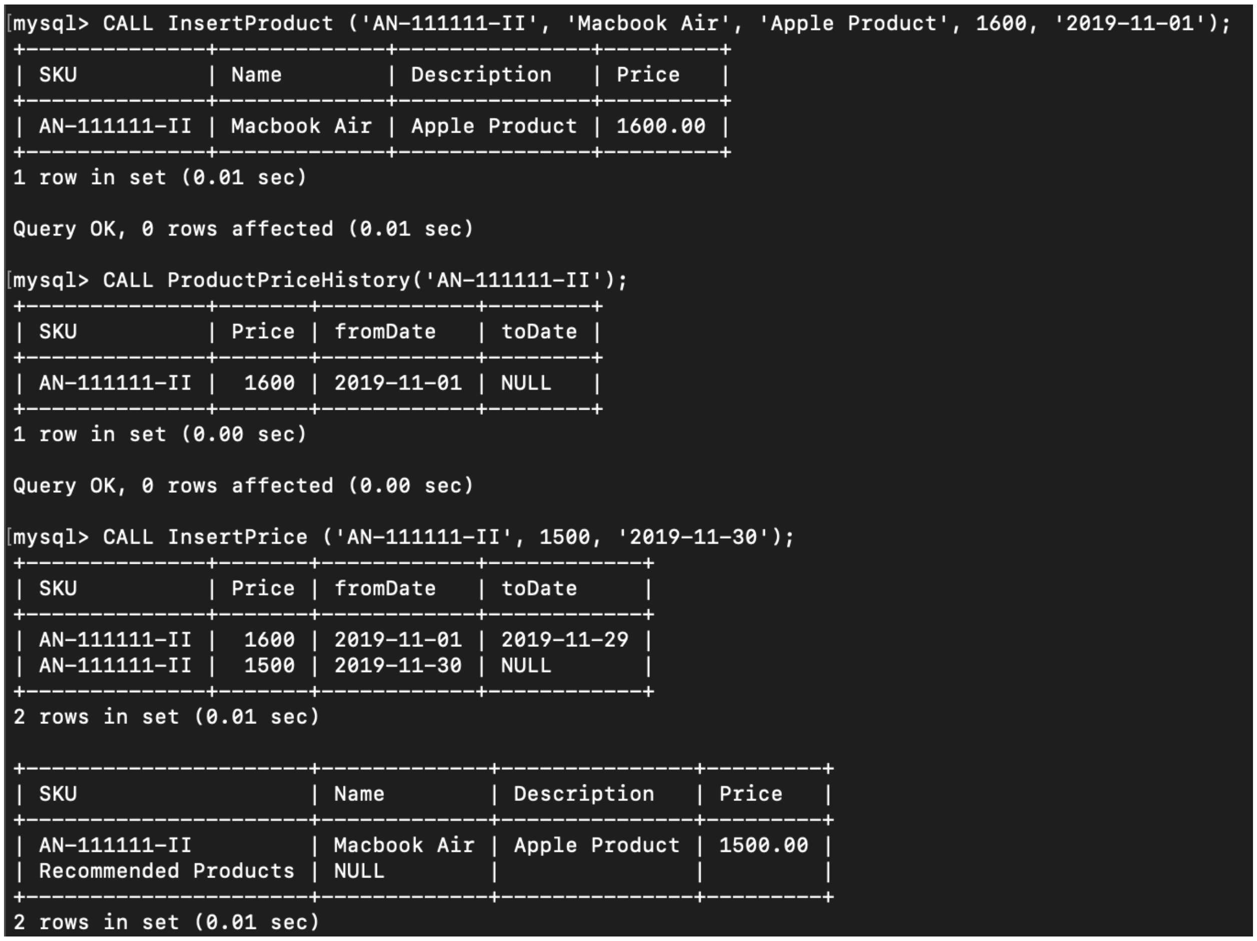
```
[mysql> CALL InsertCustomer ('Aaron', '155 University Ave', 'San Jose', 'CA', 'United States', '94088');
+------------+-------+--------------------+----------+-------+---------------+--------+
| CustomerID | Name  | Address            | City     | State | Country       | Postal |
+------------+-------+--------------------+----------+-------+---------------+--------+
|          2 | Aaron | 155 University Ave | San Jose | CA    | United States | 94088  |
+------------+-------+--------------------+----------+-------+---------------+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

[mysql> CALL InsertOrder (2, '2019-12-01', 0, 'AN-111111-II', '10');
+--------------+----------+---------+----------+------------+----------+------------+
| Name         | Quantity | Price   | Total    | OrderDate  | ShipDate | Status     |
+--------------+----------+---------+----------+------------+----------+------------+
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-01 | NULL     | In Process |
| Total Charge |          |         | 16000.00 |            |          |            |
+--------------+----------+---------+----------+------------+----------+------------+
2 rows in set (0.00 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |      100 | 2019-11-01  |
+--------------+----------+-------------+
1 row in set (0.00 sec)
```

Price Update of Product scenario test (use priceUpdate.sql to test)

In this scenario, the original price of the product is shown in the first and second tables. The first table is the Products table, and the second table is the Price History table. After using InsertPrice API below, the new price is updated in the Products table (fourth table), and the new entry of the new price is added in the Price History table (third table).

Example test run:

source createDB.txt
CALL InsertProduct('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL ProductPriceHistory('AN-111111-II');
CALL InsertPrice ('AN-111111-II', 1500, '2019-11-30');



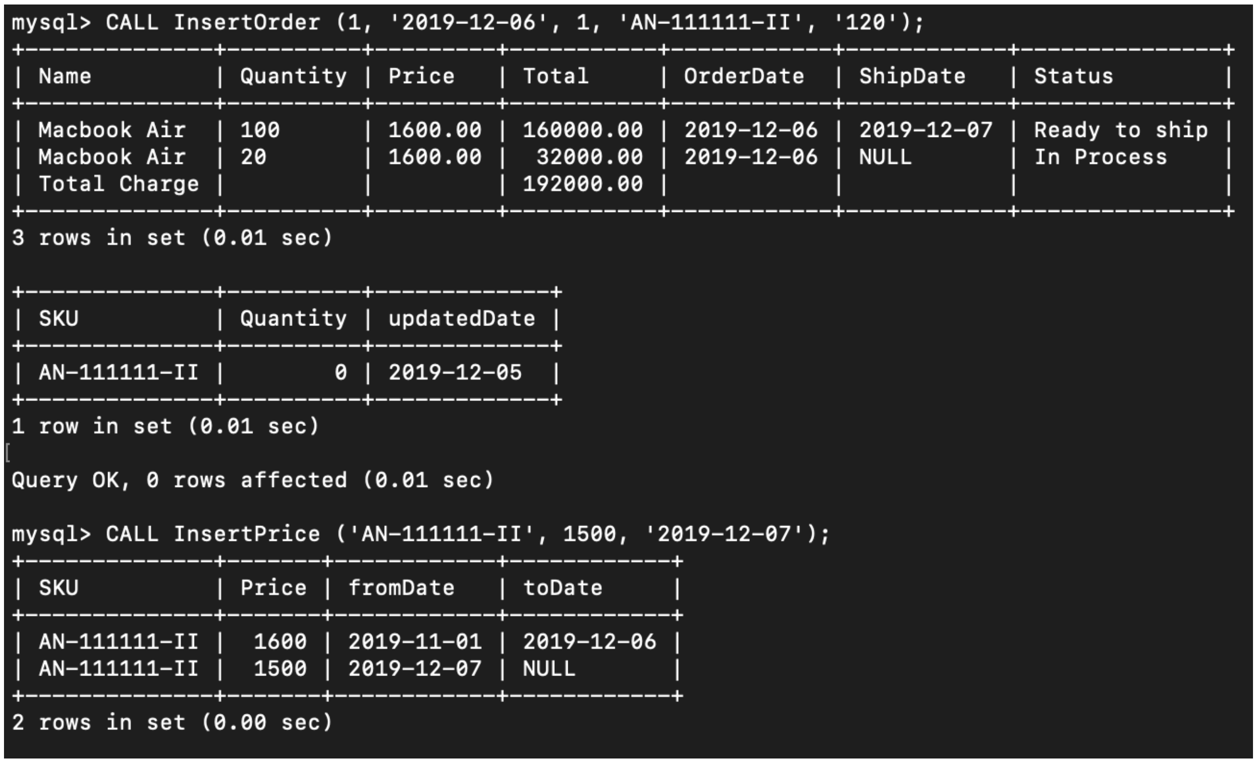
```
mysql> CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
+--------------+-------------+---------------+---------+
| SKU          | Name        | Description   | Price   |
+--------------+-------------+---------------+---------+
| AN-111111-II | Macbook Air | Apple Product | 1600.00 |
+--------------+-------------+---------------+---------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> CALL ProductPriceHistory('AN-111111-II');
+--------------+-------+------------+--------+
| SKU          | Price | fromDate   | toDate |
+--------------+-------+------------+--------+
| AN-111111-II |  1600 | 2019-11-01 | NULL   |
+--------------+-------+------------+--------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL InsertPrice ('AN-111111-II', 1500, '2019-11-30');
+--------------+-------+------------+------------+
| SKU          | Price | fromDate   | toDate     |
+--------------+-------+------------+------------+
| AN-111111-II |  1600 | 2019-11-01 | 2019-11-29 |
| AN-111111-II |  1500 | 2019-11-30 | NULL       |
+--------------+-------+------------+------------+
2 rows in set (0.01 sec)

+----------------------+-------------+---------------+---------+
| SKU                  | Name        | Description   | Price   |
+----------------------+-------------+---------------+---------+
| AN-111111-II         | Macbook Air | Apple Product | 1500.00 |
| Recommended Products | NULL        |               |         |
+----------------------+-------------+---------------+---------+
2 rows in set (0.01 sec)
```

Reimbursement Scenario test (use reimburse.sql to test)

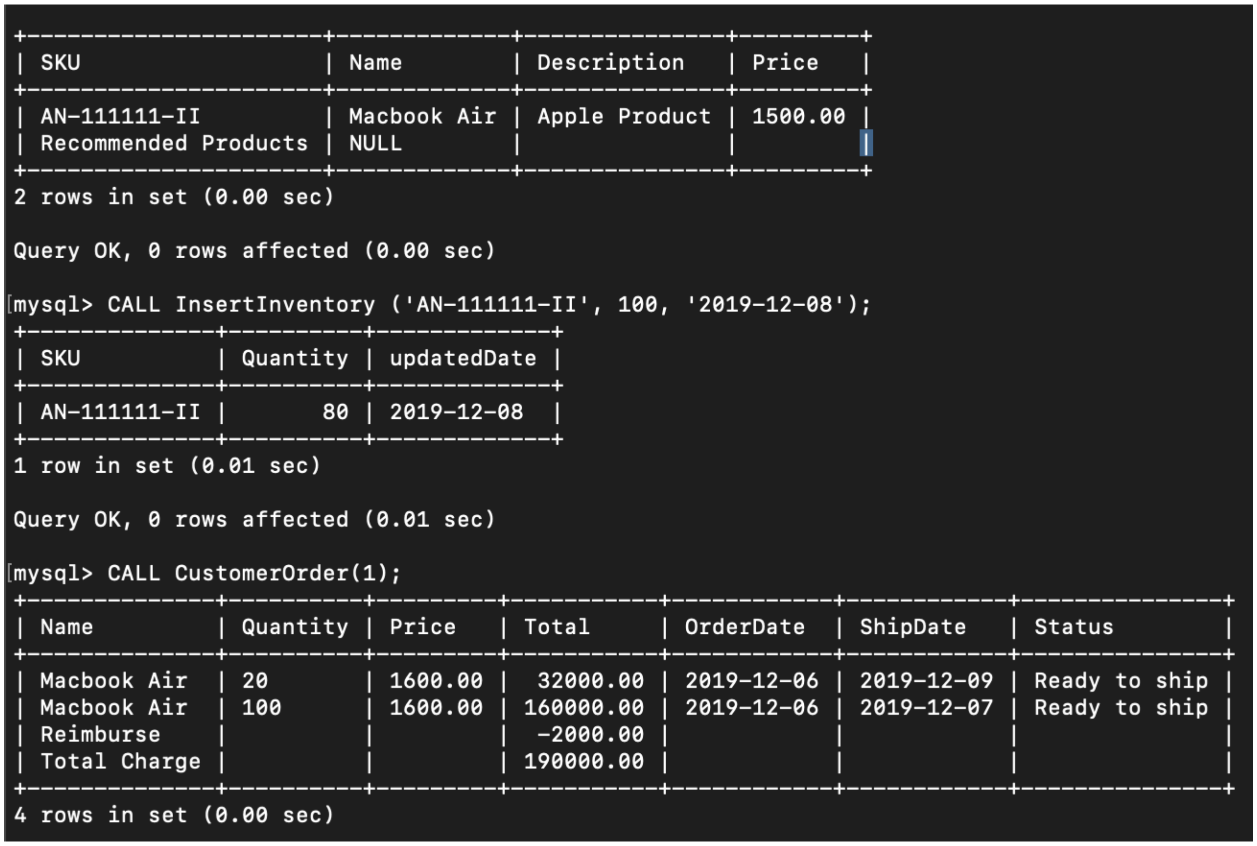
In this scenario, assuming that we have SKU “AN-111111-II” in the Products and in Inventory of 100 quantity already. When the customer has a backorder, then a few days later the price of the product of back order drops, and the backorder is fulfilled. The reimbursement query will be triggered in the insert inventory API call and insert new reimbursed amount (= (Old price - New Price) * BackOrder Quantity) in the Reimbursement table which displays in the CustomerOrder API call below.

Example test run:

source createDB.txt
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL InsertCustomer ('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
CALL InsertOrder (1, '2019-12-06', 1, 'AN-111111-II', '120');
CALL InsertPrice ('AN-111111-II', 1500, '2019-12-07');
CALL InsertInventory ('AN-111111-II', 100, '2019-12-08');
CALL CustomerOrder(1);



```
mysql> CALL InsertOrder (1, '2019-12-06', 1, 'AN-111111-II', '120');
+--------------+----------+---------+-----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total     | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+-----------+------------+------------+---------------+
| Macbook Air  | 100      | 1600.00 | 160000.00 | 2019-12-06 | 2019-12-07 | Ready to ship |
| Macbook Air  | 20       | 1600.00 |  32000.00 | 2019-12-06 | NULL       | In Process    |
| Total Charge |          |         | 192000.00 |            |            |               |
+--------------+----------+---------+-----------+------------+------------+---------------+
3 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |        0 | 2019-12-05  |
+--------------+----------+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> CALL InsertPrice ('AN-111111-II', 1500, '2019-12-07');
+--------------+-------+------------+------------+
| SKU          | Price | fromDate   | toDate     |
+--------------+-------+------------+------------+
| AN-111111-II |  1600 | 2019-11-01 | 2019-12-06 |
| AN-111111-II |  1500 | 2019-12-07 | NULL       |
+--------------+-------+------------+------------+
2 rows in set (0.00 sec)
```



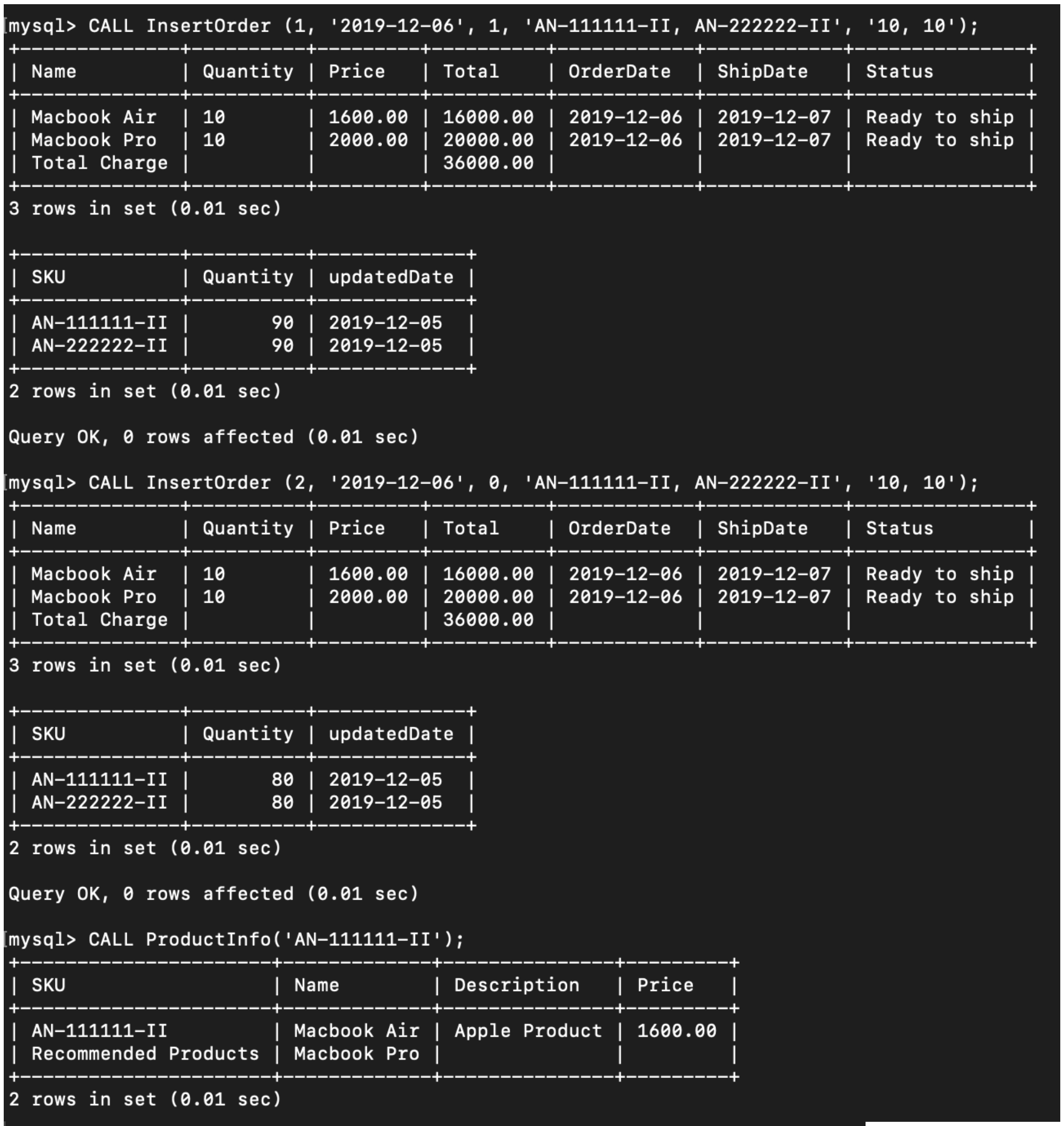
```
+----------------------+--------------+-----------------+---------+
| SKU                  | Name         | Description     | Price   |
+----------------------+--------------+-----------------+---------+
| AN-111111-II         | Macbook Air  | Apple Product   | 1500.00 |
| Recommended Products | NULL         |                 |         |
+----------------------+--------------+-----------------+---------+
2 rows in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALL InsertInventory ('AN-111111-II', 100, '2019-12-08');
+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |       80 | 2019-12-08  |
+--------------+----------+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> CALL CustomerOrder(1);
+--------------+----------+---------+-----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total     | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+-----------+------------+------------+---------------+
| Macbook Air  | 20       | 1600.00 |  32000.00 | 2019-12-06 | 2019-12-09 | Ready to ship |
| Macbook Air  | 100      | 1600.00 | 160000.00 | 2019-12-06 | 2019-12-07 | Ready to ship |
| Reimburse    |          |         |  -2000.00 |            |            |               |
| Total Charge |          |         | 190000.00 |            |            |               |
+--------------+----------+---------+-----------+------------+------------+---------------+
4 rows in set (0.00 sec)
```

Recommended Products Scenario test (use recommended,sql to test)

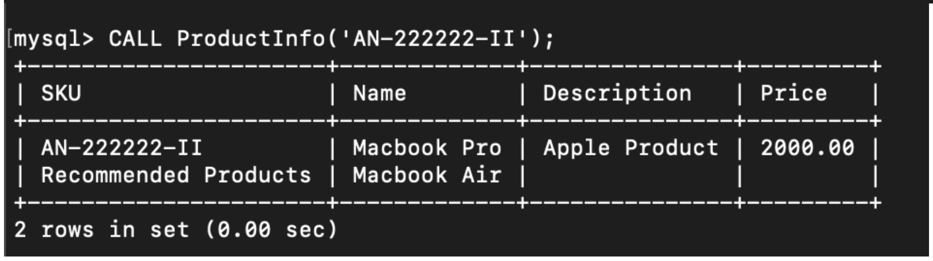
In this scenario, assuming that we have SKU “AN-111111-II” and “AN-222222-II” in the Products and in Inventory of 100 quantity each already. The recommended products feature in the ProductInfo API tracks in all the order records and see if the SKU input is used to be ordered most with what other SKUs. If it sees that product is used to be ordered with other products more than 2 (just an arbitrary number, we can adjust this number in the future) times, the other product(s) will be recommended product(s) for the product that is displaying. There are 3 maximum recommended products for each product.

Example test run:

```
source createDB.txt
CALL InsertProduct ('AN-111111-II', 'Macbook Air', 'Apple Product', 1600, '2019-11-01');
CALL InsertProduct ('AN-222222-II', 'Macbook Pro', 'Apple Product', 2000, '2019-11-01');
CALL InsertCustomer ('Ashley', '123 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertCustomer ('Tom', '155 University Ave', 'San Jose', 'CA', 'United States', '94088');
CALL InsertInventory ('AN-111111-II', 100, '2019-11-01');
CALL InsertInventory ('AN-222222-II', 100, '2019-11-01');
CALL InsertOrder (1, '2019-12-06', 0, 'AN-111111-II, AN-222222-II', '10, 10');
CALL InsertOrder (2, '2019-12-06', 0, 'AN-111111-II, AN-222222-II', '10, 10');
CALL ProductInfo('AN-111111-II');
CALL ProductInfo('AN-222222-II');
```



```
[mysql> CALL InsertOrder (1, '2019-12-06', 1, 'AN-111111-II, AN-222222-II', '10, 10');
+--------------+----------+---------+----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total    | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+----------+------------+------------+---------------+
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-06 | 2019-12-07 | Ready to ship |
| Macbook Pro  | 10       | 2000.00 | 20000.00 | 2019-12-06 | 2019-12-07 | Ready to ship |
| Total Charge |          |         | 36000.00 |            |            |               |
+--------------+----------+---------+----------+------------+------------+---------------+
3 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |       90 | 2019-12-05  |
| AN-222222-II |       90 | 2019-12-05  |
+--------------+----------+-------------+
2 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

[mysql> CALL InsertOrder (2, '2019-12-06', 0, 'AN-111111-II, AN-222222-II', '10, 10');
+--------------+----------+---------+----------+------------+------------+---------------+
| Name         | Quantity | Price   | Total    | OrderDate  | ShipDate   | Status        |
+--------------+----------+---------+----------+------------+------------+---------------+
| Macbook Air  | 10       | 1600.00 | 16000.00 | 2019-12-06 | 2019-12-07 | Ready to ship |
| Macbook Pro  | 10       | 2000.00 | 20000.00 | 2019-12-06 | 2019-12-07 | Ready to ship |
| Total Charge |          |         | 36000.00 |            |            |               |
+--------------+----------+---------+----------+------------+------------+---------------+
3 rows in set (0.01 sec)

+--------------+----------+-------------+
| SKU          | Quantity | updatedDate |
+--------------+----------+-------------+
| AN-111111-II |       80 | 2019-12-05  |
| AN-222222-II |       80 | 2019-12-05  |
+--------------+----------+-------------+
2 rows in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

[mysql> CALL ProductInfo('AN-111111-II');
+----------------------+-------------+---------------+---------+
| SKU                  | Name        | Description   | Price   |
+----------------------+-------------+---------------+---------+
| AN-111111-II         | Macbook Air | Apple Product | 1600.00 |
| Recommended Products | Macbook Pro |               |         |
+----------------------+-------------+---------------+---------+
2 rows in set (0.01 sec)
```



```
[mysql> CALL ProductInfo('AN-222222-II');
+----------------------+-------------+---------------+---------+
| SKU                  | Name        | Description   | Price   |
+----------------------+-------------+---------------+---------+
| AN-222222-II         | Macbook Pro | Apple Product | 2000.00 |
| Recommended Products | Macbook Air |               |         |
+----------------------+-------------+---------------+---------+
2 rows in set (0.00 sec)
```