aProgramming Lab 3

In this programming lab, we consider a Mail Order Database with 8 tables.

ZIPCODES(zip, city)
EMPLOYEES(eno, ename, zip, hdate)
PARTS(pno, pname, qoh, price, olevel)
CUSTOMERS(cno, cname, street, zip, phone)
ORDERS(ono, cno, eno, received, shipped)
ODETAILS(ono, pno, qty)
RESTOCK(res_date, pno)
ORDERS_ERRORS(Error_Date, ono, Error_Msg)

The **eno, pno, cno, ono** are the employee number, part number, customer number, and order number respectively.

hdate of in **EMPLOYEES** table means the hiring date of an employee.

qoh in the **PARTS** table is the quantity the part on hold.

olevel in the **PARTS** table is the number that determines when to restock a part. For example, given **olevel**=20, we need to restock this part to 2*olevel when its qoh < 20.

The **received** and **shipped** are the dates that the order is received and shipped.

qty in the **ODETAILS** table is the quantity for the product for an order.

In the **RESTOCK** table, **res_date** is the date when a restock is requested.

ORDERS_ERRORS is a table to log order related error messages.

Programming Tasks

Task1(5 points**):** Create tables with scripts provided in 'Create-Table.sql' and load data into these tables with scripts provided in 'Load-Data.sql'. Please read these provided scripts carefully before executing them.

What to submit: No submission is required

Task2(15 points**)**: There is a procedure "**add_order**" in the file **p2_task2.sql**. However, this procedure has syntax errors. In this task, you need to find and fix these syntax errors.

What to submit:

1. Point out where are errors for the procedure, and your SQL scripts for the correct procedure.
2. Call the "**add_order**" procedure with the following parameters (666,1,3,null,null) as input. Submit your screenshot for query select * from ORDERS after executing this procedure call.

```
Delimiter $$
create procedure add_order(
        IN onum int,
        IN cnum int,
   IN receive date
)
begin
DECLARE employee_name varchar(255);
if (receive is null)
        insert into ORDERS values (onum,cnum,enum,CURDATE(),null);
else then
        insert into ORDERS values (onum,cnum,enum,receive,null);
end if;
end;
$$
```

```
delimiter ||
CREATE PROCEDURE add_order(
        IN onum INT,
   IN cnum INT,
   IN enum INT,
   IN receive DATE
)
BEGIN
        DECLARE employee_name VARCHAR(255);
        IF (receive IS NULL) THEN
                INSERT INTO ORDERS VALUES (onum, cnum, enum, CURDATE(), null);
        ELSE
                INSERT INTO ORDERS VALUES (onum, cnum, enum, CURDATE(), null);
   END IF;
END;
||
```

ono	cno	eno	received	shipped
313	3	1	2016-11-02	2016-11-09
315	3	1	2016-11-03	2016-11-09
438	3	1	2017-03-12	NULL
666	1	3	2019-04-09	NULL
NULL	NULL	NULL	NULL	NULL

cTask3(10 points**)**: Create a procedure "**ship_order**" with order number and shipping date as input. If the shipping date in the procedure call is null, use the current date as the shipping date.

What to submit: Your scripts for the "**ship_order**" procedure. The screen of select * from ORDERS after executing the procedure with (438, null) as input.

```
1   delimiter ||
2   CREATE PROCEDURE ship_order (
3       IN order_num INT,
4       IN ship DATE
5   )
6   BEGIN
7       IF (ship IS NULL) THEN
8           UPDATE ORDERS SET shipped = CURDATE() WHERE ono = order_num
9       ELSEIF (ship IS NOT NULL) THEN
10          UPDATE ORDERS SET shipped = ship WHERE ono = order_num;
11      END IF;
12  END;
```

ono	cno	eno	received	shipped
313	3	1	2016-11-02	2016-11-09
315	3	1	2016-11-03	2016-11-09
438	3	1	2017-03-12	2019-04-10
666	1	3	2019-04-09	NULL
NULL	NULL	NULL	NULL	NULL

Task4(35 points**)**: Create a new **function** "**cancel_order**". The **cancel_order** procedure will take the order number as input. In this function, we will delete rows in tables **ORDERS** and **ODETAILS** with ono equal to the order number input by the function caller. The function should return a message to indicate the order cancellation status as follows:

1. The order is removed, but order details do not exist, return "Order Details Do Not Exist! Order Is Cancelled Successfully!"
2. Both order and order details do not exist, return "Order Details Do Not Exist! Order Does Not Exist!"
3. Both order and order details are removed, return "Order Details Removed Successfully! Order Is Cancelled Successfully!"

Note that: You can use CONCAT() function for the concatenation of multiple strings. Pay attention to the Foreign key restriction exist between table **ORDERS** and **ODETIALS**.

What to submit:

1. Your scripts of "**cancel_order**" function.
2. Screenshot of function calls for "**cancel_order**" as
   SET @x=cancel_order(315);
   SELECT @x;

   SET @x=cancel_order(666);
   SELECT @x;

3. The screenshot for the following query after the above two function calls.

SELECT * FROM ORDERS;

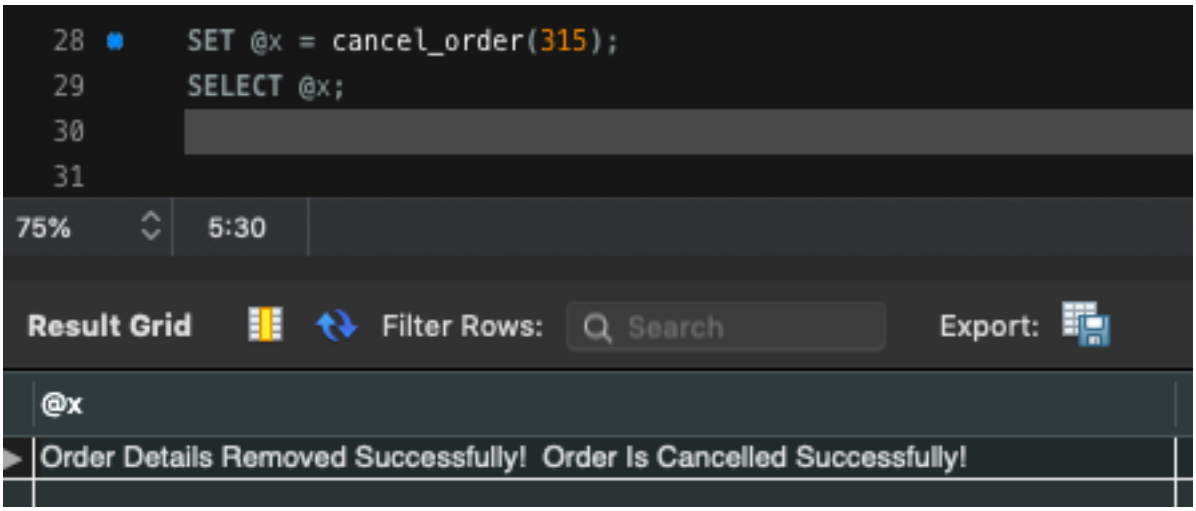
SELECT * FROM ODETAILS;

```
delimiter ||
CREATE FUNCTION cancel_order(order_number INT) RETURNS VARCHAR(255)
DETERMINISTIC
BEGIN
    DECLARE result VARCHAR(255);
    DECLARE orderCount INT;
    DECLARE orderDetCount INT;

    SELECT COUNT(ono) FROM ORDERS WHERE ono = order_number INTO orderCount;
    SELECT COUNT(ono) FROM ODETAILS WHERE ono = order_number INTO orderDetCount;

    IF (orderCount = 1 AND orderDetCount = 1) THEN
        DELETE FROM ODETAILS WHERE ono = order_number;
        DELETE FROM ORDERS WHERE ono = order_number;
        SET result = 'Order Details Removed Successfully!  Order Is Cancelled Successfully!';
    ELSEIF (orderCount = 1 AND orderDetCount = 0) THEN
        DELETE FROM ORDERS WHERE ono = order_number;
        SET result = 'Order Details Do Not Exist!  Order Is Cancelled Successfully!';
    ELSEIF (orderCount = 0 AND orderDetCount = 0) THEN
        SET result = 'Order Details Do Not Exist!  Order Does Not Exist!';
    END IF;

    return(result);
END;
||
```



```
31      SET @y = cancel_order(666);
32      SELECT @y;
33
```

75% 5:26

Result Grid | Filter Rows: Search

@y
Order Details Do Not Exist! Order Is Cancelled Successfully!

ono	cno	eno	received	shipped
313	3	1	2016-11-02	2016-11-09
438	3	1	2017-03-12	2019-04-10
NULL	NULL	NULL	NULL	NULL

ono	pno	qty
313	10701	5
438	10201	6
NULL	NULL	NULL

Task5(35 points) Create a procedure "**add_order_details**" with order number, part number, and quantity as input. The procedure shall satisfy the following requirements:

1. If the quantity required by the order is greater than quantity on hold, insert an error message into the **ORDERS_ERRORS** table with an error message as "Do not have enough quantity to sell!". At the same time, the order shall be cancelled by calling the "**cancel_order**" you created in task 2. Note that, "**cancel_order**" shall be invoked directly inside the body of "**add_order_details**".
2. If the quantity required by the order is less than quantity on hold, reduce the quantity on hold for the part in the PARTS table, and insert a record into the **ODETAILS** table. Then, if the remained quantity on hold for the part is less than the **olevel**, insert a restock record into the **RESTOCK** table, and update the quantity of the product to the double of the value of olevel this product.

What to submit:

Your scripts of "**add_order_details**" procedure

```
delimiter ||
CREATE PROCEDURE add_order_details (
    IN order_number INT,
    IN part_number INT,
    IN quant INT
)
BEGIN
    DECLARE quantAvailable INT;
    DECLARE quantRemain INT;
    DECLARE oLevel INT;
    SELECT qoh FROM PARTS WHERE pno = part_number INTO quantAvailable;
    SELECT olevel FROM PARTS WHERE pno = part_number INTO oLevel;
    SET quantRemain = quantAvailable - quant;
    IF (quantRemain < 0) THEN
            INSERT INTO ORDERS_ERRORS VALUES (CURDATE(), order_number, 'Do not have enough quantity to sell!');
            SET @x = cancel_order(order_number);
    ELSEIF (quantRemain >= 0) THEN
        UPDATE PARTS SET qoh = qoh - quant;
        IF (quantRemain < oLevel) THEN
            INSERT INTO RESTOCK VALUES (CURDATE(), part_number);
        END IF;
    END IF;
END;
||
```