Order Entry Database – Working with Views and Procedures

The simplified Order Entry database was designed to support transactions related to customer purchases of computer and electronics products either over the phone or online. The SQL code for all 12 problems should be entered into a single `MySQL4_OrdEntry2.sql` file, separated by comments, but each problem should be run in Workbench independently. For views, including form and report views, copy/paste the view data first, followed by the results of the query (or queries) based on that view into a single Excel file `Assign4_OrdEntry.xlsx`, one result per sheet. For the stored procedures and triggers, copy/paste the results of all the required test runs, including the changes to tables highlighted in yellow. For several problems you will need to use MS Access database in `AccDB4_OrdEntry.accdb` file.

Each of the questions is worth **0.25** points.

1. Create a single table view, named `WA_Cust`, of Washington state customers only. Create a query using this view to show names and balance of only those WA customers with balance over $500.
2. Create a multiple table view, named `Comm_Emp_Cust_Ord`, showing customer name, balance, order dates, and employee names for employees with commission of 0.04 or higher. Create a query using this view to show customer names, balance and order dates for orders taken by Johnson.
3. Create a grouping view, named `Product_Summary`, to summarize the total sales by product and manufacturer. Include the product manufacturer, and make sure to name the product `ProductName`, manufacturer `ManufName`, and total sales as `TotalSales`. Create a grouping query on this view to summarize the number of products and total sales by manufacturer, sorted descending on sales.
4. Create a single table updatable view, named `WA_Cust_Update`, that will allow you to update Washington state customers only. Make sure to include all the customer columns in the view. Create an insert query using this view that will add a new Washington customer with only the required data, followed by an update query that will add some of the optional data of your choice. Use C9998888 for the CustNo, and make up the rest.
5. Create a multiple table updatable view in MySQL, named `Order_Update`, with OrderTbl as a parent and OrderLine as a child table. The view must show only those orders where the customer state is not the same as the state where the order is going to. The sole purpose of the view is the ability to update where an order is going to and any order lines associated with a given order. The view should not be able to change anything else about the order or insert new orders or order lines. It should not be able to make changes to any table beyond the two discussed here. This does not mean, however, that there should not be any supporting info coming from other tables. You must use INNER JOIN style. Create a query using this view that will modify order (O6565656) to go to Helen Sibley and the quantity of the product (P0036566) from 10 to 1. What message do you get when you run this query in Workbench? Redo the whole process in MS Access in `AccDB4_OrdEntry.accdb` file, saving the view as `AccDB4_05_Order_Update_View`, and the query as `AccDB4_05_OrdName_Qty_Update`.
6. Create a hierarchical form, named `AccDB4_Main_06_Main_Order_Form`, with OrderTbl as parent and OrderLine as the child table in MS Access in `AccDB4_OrdEntry.accdb` file, saving the main form query as `AccDB4_06_Main_Ord_Update_View`, and the query for the subform as `AccDB4_06_Subform_OrdLine_Update_View`. The main form query must have all the columns from OrdTbl, and must include the customer and employee first and last names. The subform query must include all the columns from OrderLine table, and must include product name and price. The form and subform should look presentable. Both queries must use INNER JOIN style. Use the form to insert a new order (O8887777) for an existing customer (C9943201), taken by an existing employee (E9954302) and for two units of an existing product (P0036577). Modify the quantity of the product ordered from 2 to 1.
7. Create a hierarchical report, named `AccDB4_07_Summary_Report`, based on a query `AccDB4_07_Report_Summary_View`, that will count the number of orders and find the total sales by employee and customer. You must use INNER JOIN style. The report must have a single grouping level on employee last name with subtotals, and it should look presentable.

8. Create a stored function named `GetCustName` that accepts customer ID as its input and returns customer's full name, first-space-last name. Then create a stored procedure `DisplayCustomerInfo` that accepts customer ID as the only input variable and returns, using a single SELECT statement, the customer's full name, obtained with `GetCustName` function, as well as the city, balance and estimated delivery time based on the state the customer resides in. For Washington state customer, the delivery is within 3 business days, Colorado customers can expect their orders to come between 3 and 5 business days. You must use `Case When` statement to define the delivery time, and this statement must have a case for customers from other states (or missing state info) where the delivery time is not defined. You must test the code by calling the stored procedure (and function) with a customer C3340959 from WA and then with C9128574 from Colorado.
9. Create two stored procedures used for inserting a new customer order into the database. The first procedure, named `InsertCustOrder` must use the four input parameters for order, customer and employee numbers, as well as the order date. The procedure should assume that the order is going to the customer that placed the order, so you need to retrieve the relevant information from the `Customer` table, use `GetCustName` function to get the full name of the customer, and then insert all the required information into `OrderTbl`. The second procedure is used to insert order lines and should be named `InsertOrderLine`. The testing procedure, named `TestAddingCustOrder`, should insert a single order number O8888777 for customer C9943201, taken by employee E9954302, on March 1, 2030, for a single unit of P1114590 and two units of P1445671.
10. Create a procedure, named `GenerateEmpEmailList` that will generate employee email list, where each email is separated from the next by a semicolon. The procedure should retrieve all the emails, and then use a cursor to loop through those emails, concatenating them one to another, with semicolons in between. The procedure should use a single input/output string variable of sufficiently large size to accommodate a relatively large string. A testing procedure `TestEmailCursor` should be used to declare a blank email list string variable, call the procedure, and then display the returned email list.
11. Create a `Customer_Audit` table with auto-incrementing ID that will store the customer number, full name, balance, timestamp and description of the audit. Then create an after-insert trigger named `After CustomerInsert` that will record the required information into the audit table after a new customer record is inserted into the Customer table. Continue by creating an after-update and a before-update trigger named `AfterCustomerUpdate` and `BeforeCustomerUpdate` that will record the balance changes to the Customer table, with description depending on whether the balance went up or down. Finally, create an after-delete trigger named `AfterCustomerDelete` that will record any delete operation on the Customer table.
12. Create a single testing procedure, named `TestCustomerTriggers`, that will test all four triggers at once. Start with an insert into operation that will add a new employee C8888777, named Jane Doe. Then perform two update operations on the same customer. The first one should increase the balance from zero to 500, the second one should decrease the balance by back to zero. Finish by deleting the newly added customer.

Submission: You must submit `MySQL4_OrdEntry2.sql` SQL script, the `Assgn4_OrdEntry.xlsx` Excel file, and `AccDB4_OrdEntry.accdb` file on Canvas by the designated due date.