AM05 Data Mgmt – Assignment 3

This assignment has three parts in which you will use the my_guitar_shop database you also used to complete the previous assignments. **There are 15** SQL challenges. When you think you have the query working correctly copy the code to a Word document that will be your assignment submission along with a cropped screen shot that shows the results of the query.

Note: If you think you have messed up your database just rerun the create_my_guitar_shop.sql script again. That should restore the original data to the database.

Part 1. Working with multiple tables

.

1. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first_name, last_name, line1, city, state, zip_code.

   Return one row for each address for the customer with an email address of allan.sherwood@yahoo.com.

2. Write a SELECT statement that joins the Customers table to the Addresses table and returns these columns: first_name, last_name, line1, city, state, zip_code.

   Return one row for each customer, but only return addresses that are the shipping address for a customer.

3. Write a SELECT statement that joins the Customers, Orders, Order_Items, and Products tables. This statement should return these columns: last_name, first_name, order_date, product_name, item_price, discount_amount, and quantity.

   Use aliases for the tables.

   Sort the final result set by the last_name, order_date, and product_name columns.

4. Write a SELECT statement that returns the product_name and list_price columns from the Products table.

   Return one row for each product that has the same list price as another product.
   Hint: Use a self-join to check that the product_id columns aren't equal but the list_price columns are equal.

   Sort the result set by the product_name column.

5. Write a SELECT statement that returns these two columns:

category_name	The category_name column from the Categories table
product_id	The product_id column from the Products table

   Return one row for each category that has never been used. *Hint: Use an outer join and only return rows where the product_id column contains a null value.*

Part 2. Aggregation and GROUP BY

6. Write a SELECT statement that returns one row for each customer that has orders with these columns:

   The email_address column from the Customers table

   The sum of the item price in the Order_Items table multiplied by the quantity in the Order_Items table

   The sum of the discount amount column in the Order_Items table multiplied by the quantity in the Order_Items table

   Sort the result set in descending sequence by the item price total for each customer.

7. Write a SELECT statement that returns one row for each customer that has orders with these columns:

   The email_address column from the Customers table

   A count of the number of orders

   The total amount for each order (*Hint: First, subtract the discount amount from the price. Then, multiply by the quantity.*)

   Return only those rows where the customer has more than 1 order.

   Sort the result set in descending sequence by the sum of the line item amounts.

8. Write a SELECT statement that answers this question: What is the total amount ordered for each product? Return these columns:

   The product_name column from the Products table

   The total amount for each product in the Order_Items table (*Hint: You can calculate the total amount by subtracting the discount amount from the item price and then multiplying it by the quantity*)

   Use the WITH ROLLUP operator to include a row that gives the grand total.

   Note: Once you add the WITH ROLLUP operator, you may need to use MySQL Workbench's Execute SQL Script button instead of its Execute Current Statement button to execute this statement.

9. Write a SELECT statement that answers this question: Which customers have ordered more than one product? Return these columns:

   The email_address column from the Customers table

   The count of distinct products from the customer's orders

   Sort the result set in ascending sequence by the email_address column.

10. Write a SELECT statement that answers this question: What is the total quantity purchased for each product within each category? Return these columns:

    The category_name column from the category table

    The product_name column from the products table

    The total quantity purchased for each product with orders in the Order_Items table

    Use the WITH ROLLUP operator to include rows that give a summary for each category name as well as a row that gives the grand total. Use the IF and GROUPING functions to replace null values in the category_name and product_name columns with literal values if they're for summary rows.

Part 3. Subqueries and VIEWS

11. Write a SELECT statement that answers this question: Which products have a list price that’s greater than the average list price for all products?

    Return the product_name and list_price columns for each product.

    Sort the result set by the list_price column in descending sequence.

12. Write a SELECT statement that returns the category_name column from the Categories table.

    Return one row for each category that has never been assigned to any product in the Products table. To do that, use a subquery introduced with the NOT EXISTS operator.

13. Write a SELECT statement that returns three columns: email_address, order_id, and the order total for each customer. To do this, you can group the result set by the email_address and order_id columns. In addition, you must calculate the order total from the columns in the Order_Items table.

    Write a second SELECT statement that uses the first SELECT statement in its FROM clause. The main query should return two columns: the customer’s email address and the largest order for that customer. To do this, you can group the result set by the email_address. Sort the result set by the largest order in descending sequence.

14. Use a correlated subquery to return one row per customer, representing the customer’s oldest order (the one with the earliest date). Each row should include these three columns: email_address, order_id, and order_date.

    Sort the result set by the order_date and order_id columns.

15. Create a view named customer_addresses that shows the shipping and billing addresses for each customer.

    This view should return these columns from the Customers table: customer_id, email_address, last_name and first_name.

    This view should return these columns from the Addresses table: bill_line1, bill_line2, bill_city, bill_state, bill_zip, ship_line1, ship_line2, ship_city, ship_state, and ship_zip.

    Write a SELECT statement that returns these columns from the customer_addresses view: customer_id, last_name, first_name, bill_line1. The rows in the result should be sorted by the last_name and then first_name columns.