Day wise Assignment Questions

**Note: -**

1. The tables mentioned in the questions for the references are available in the classic model database.

2. In the questions, if they specifically mention to create the tables, then you need to create the tables as per given specifications.

3. Solve all the assignments for each day in a single query tab only. Kindly submit your MySQL assignments in a single file, using a format that suits you best, such as a .sql file, Notepad, or Word document.

|  |
| --- |
|  |
|  |
| **SELECT clause with WHERE, AND,DISTINCT, Wild Card (LIKE)**   1. Fetch the employee number, first name and last name of those employees who are working as Sales Rep reporting to employee with employeenumber 1102 **(Refer employee table)**   **Expected output:**     1. Show the unique productline values containing the word cars at the end from the **products table**.   **Expected output:**    **CASE STATEMENTS for Segmentation**  . 1. Using a CASE statement, segment customers into three categories based on their country:**(Refer Customers table)**  "North America" for customers from USA or Canada  "Europe" for customers from UK, France, or Germany  "Other" for all remaining countries  Select the customerNumber, customerName, and the assigned region as "CustomerSegment".  **Expected output:** |
| **Group By with Aggregation functions and Having clause, Date and Time functions**   1. Using the **OrderDetails table**, identify the top 10 products (by productCode) with the highest total order quantity across all orders.   **Expected output:**     1. Company wants to analyze payment frequency by month. Extract the month name from the payment date to count the total number of payments for each month and include only those months with a payment count exceeding 20 **(Refer Payments table).**   Expected output: |
|  |
|  |

**CONSTRAINTS: Primary, key, foreign key, Unique, check, not null, default**

1. Create a new database named **Customers\_Orders** and add the following tables as per the description
2. Create a table named **Customers** to store customer information. Include the following columns:

customer\_id: This should be an integer set as the PRIMARY KEY and AUTO\_INCREMENT.

first\_name: This should be a VARCHAR(50) to store the customer's first name.

last\_name: This should be a VARCHAR(50) to store the customer's last name.

email: This should be a VARCHAR(255) set as UNIQUE to ensure no duplicate email addresses exist.

phone\_number: This can be a VARCHAR(20) to allow for different phone number formats.

Add a NOT NULL constraint to the first\_name and last\_name columns to ensure they always have a value.

1. Create a table named **Orders** to store information about customer orders. Include the following columns:

order\_id: This should be an integer set as the PRIMARY KEY and AUTO\_INCREMENT.

customer\_id: This should be an integer referencing the customer\_id in the Customers table (FOREIGN KEY).

order\_date: This should be a DATE data type to store the order date.

total\_amount: This should be a DECIMAL(10,2) to store the total order amount.

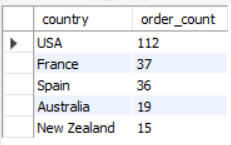
Constraints:

1. Set a FOREIGN KEY constraint on customer\_id to reference the Customers table.
2. Add a CHECK constraint to ensure the total\_amount is always a positive value.

**JOINS**

1. List the top 5 countries (by order count) that Classic Models ships to. (**Use the Customers and Orders tables**)

**Expected output:**

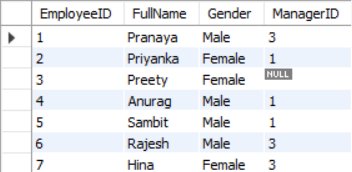
****

**SELF JOIN**

1. Create a table **project** with below fields.

* EmployeeID : integer set as the PRIMARY KEY and AUTO\_INCREMENT.
* FullName: varchar(50) with no null values
* Gender : Values should be only ‘Male’ or ‘Female’
* ManagerID: integer

Add below data into it.



Find out the names of employees and their related managers.

**Expected output:**



**DDL Commands: Create, Alter, Rename**

1. Create a table facility. Add the below fields into it.

* Facility\_ID
* Name
* State
* Country

i) Alter the table by adding the primary key and auto increment to Facility\_ID column.

ii) Add a new column city after name with data type as varchar which should not accept any null values.



**Views in SQL**

1. Create a view named product\_category\_sales that provides insights into sales performance by product category. This view should include the following information:

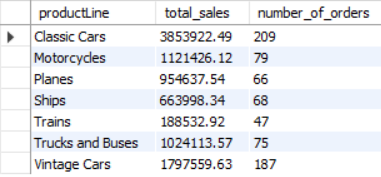
**productLine**: The category name of the product (from the ProductLines table).

**total\_sales**: The total revenue generated by products within that category (calculated by summing the orderDetails.quantity \* orderDetails.priceEach for each product in the category).

**number\_of\_orders**: The total number of orders containing products from that category.

(Hint: Tables to be used: Products, orders, orderdetails and product lines)

The view when read should show the output as:



**Stored Procedures in SQL with parameters**

1. Create a stored procedure Get\_country\_payments which takes in year and country as inputs and gives year wise, country wise total amount as an output. Format the total amount to nearest thousand unit (K)

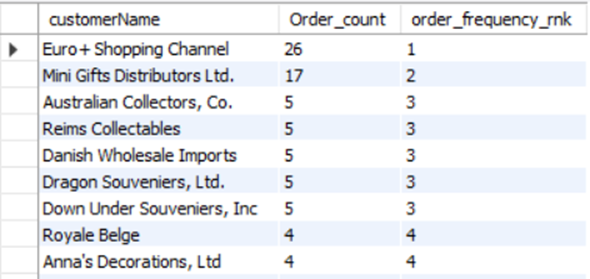
Tables: Customers, Payments

**Expected output:**



**Window functions - Rank, dense\_rank, lead and lag**

1a) Using customers and orders tables, rank the customers based on their order frequency



1b) Calculate year-wise, month-name-wise count of orders and year-over-year (YoY) percentage change. Format the YoY values in no decimals and show them in % sign.

Table: Orders

**Expected output:**

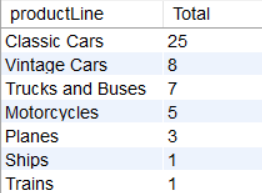


**Subqueries and their applications**

1. Find out how many product lines are there for which the buy price value is greater than the average of MSRP value. Show the output as a product line and its count.

Table: Products

**Expected output:**



**ERROR HANDLING in SQL**

1. Create the table Emp\_EH. Below are its fields.

* EmpID (Primary Key)
* EmpName
* EmailAddress

Create a procedure to accept the values for the columns in Emp\_EH. Handle the error using exception handling concept. Show the message as “Error occurred” in case of anything wrong.

**TRIGGERS**

1. Create the table Emp\_BIT. Add below fields in it.

* Name
* Occupation
* Working\_date
* Working\_hours

Insert the data as shown in the below query.

INSERT INTO Emp\_BIT VALUES

('Robin', 'Scientist', '2020-10-04', 12),

('Warner', 'Engineer', '2020-10-04', 10),

('Peter', 'Actor', '2020-10-04', 13),

('Marco', 'Doctor', '2020-10-04', 14),

('Brayden', 'Teacher', '2020-10-04', 12),

('Antonio', 'Business', '2020-10-04', 11);

Create before insert trigger to make sure any new value of Working\_hours, if it is negative, then it should be inserted as positive.