Classic Model Requests

|  |
| --- |
|  |
|  |
| **Q1. 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:**    **Q2. CASE STATEMENTS for Segmentation**  . a. 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:** |
| **Q3. 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 analyse 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. Sort the results by total number of payments in descending order. **(Refer Payments table).**   Expected output: |
|  |
|  |

**Q4. CONSTRAINTS: Primary, key, foreign key, Unique, check, not null, default**

Create a new database named and **Customers\_Orders** and add the following tables as per the description

1. 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.

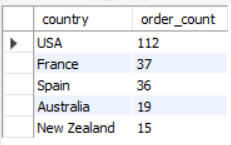
Constraints:

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.

**Q5. JOINS**

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

**Expected output:**

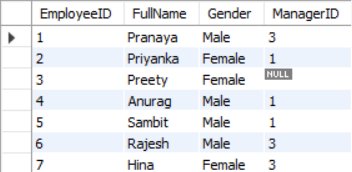
****

**Q6. SELF JOIN**

a. 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:**



**Q7. DDL Commands: Create, Alter, Rename**

a. Create 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.



**Q8. Views in SQL**

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

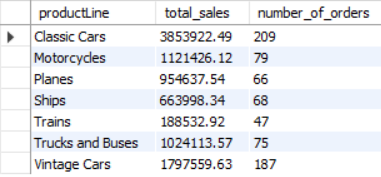
**productLine**: The category name of the product (from the ProductLines table).

**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 productlines)

The view when read should show the output as:



**Q9.Subqueries and their applications**

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

**Expected output:**

