Assignment Answers

|  |
| --- |
|  |
|  |
| **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:**    **ANSWER:**  **use classicmodels;**  **select employeeNumber,firstName,lastName**  **from employees**  **where reportsTo = 1102;**   1. Show the unique productline values containing the word cars at the end from the **products table**.   **Expected output:**    **ANSWER:**  **select productLine**  **from productlines**  **where productLine like "%Cars";**  **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:**    **ANSWER:**  **select customerNumber,customerName,**  **case**  **when country in ("USA" ,"Canada") then "North America"**  **when country in ("UK", "France", "Germany") then "Europe"**  **else "others"**  **end as CustomerSegment**  **from customers;** |
| **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:**    **ANSWER:**  **select productCode, sum(quantityOrdered) as total\_ordered**  **from orderdetails**  **group by productCode**  **order by total\_ordered desc**  **limit 10;**   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: |
| **ANSWER:**  **select monthname(paymentDate) as payment\_month, count(customerNumber) as num\_payment**  **from payments**  **group by payment\_month**  **having num\_payment > 20**  **order by num\_payment desc;** |
|  |

**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.

**ANSWER:**

**create database Customers\_Orders;**

**use Customers\_Orders;**

**create table customers(**

**customer\_id int primary key auto\_increment,**

**first\_name varchar(50),**

**last\_name varchar(50),**

**email varchar(255) unique,**

**phone\_number varchar(20) unique,**

**check(first\_name is not null),**

**check(last\_name is not null)**

**);**

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.

**ANSWER:**

**create table orders(**

**order\_id int primary key auto\_increment,**

**customer\_id int,**

**order\_date date,**

**total\_amount decimal(10,2),**

**constraint for\_key foreign key (customer\_id) references customers(customer\_id),**

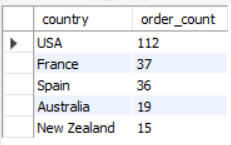
**check (total\_amount > 0)**

**);**

**Q5. JOINS**

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

**Expected output:**

****

**ANSWER:**

**use classicmodels;**

**select country, count(orderNumber) as order\_count**

**from customers inner join orders using (customerNumber)**

**group by country**

**order by order\_count desc**

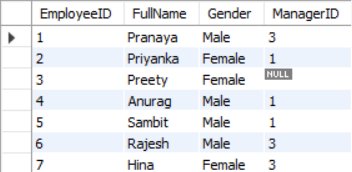
**limit 5;**

**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:**



**ANSWER:**

**create table project(**

**EmployeeID int primary key auto\_increment,**

**FUllName varchar(50) not null,**

**Gender enum("Male","Female"),**

**ManagerID int**

**);**

**insert into project**

**(FUllName,Gender,ManagerID)**

**values**

**("Pranaya", "Male", 3),**

**("Priyanka", "Female", 1),**

**("Preety", "Female", null),**

**("Anurag", "Male", 1),**

**("Sambit", "Male", 1),**

**("Rajesh", "Male", 3),**

**("Hina", "Female", 3);**

**select \* from project;**

**select M.FUllName as ManagerName,**

**E.FUllName as EmployeeName**

**from project M join project E on M.EmployeeID = E.ManagerID;**

**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.



**ANSWER:**

**create table facility(**

**Facility\_ID int,**

**Name varchar(100),**

**State varchar(100),**

**Country varchar(100)**

**);**

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

**alter table facility**

**modify column Facility\_ID int primary key auto\_increment;**

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

**alter table facility**

**add City varchar(100) not null;**

**explain facility;**

**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:

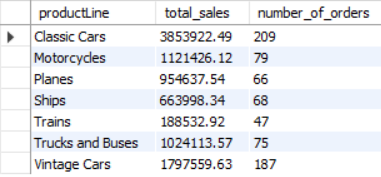
**productLine**: The category name of the product (from the ProductLines table).

**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:



**ANSWER:**

**create view product\_category\_sales as**

**select pl.productLine, SUM(od.quantityOrdered \* od.priceEach) AS total\_sales, COUNT(DISTINCT o.orderNumber) AS number\_of\_orders**

**from productlines as pl**

**join products as p on pl.productLine = p.productLine**

**join orderdetails as od on p.productCode = od.productCode**

**join orders as o on od.orderNumber = o.orderNumber**

**group by pl.productLine;**

**Q9. Stored Procedures in SQL with parameters**

a. 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:**



**ANSWER:**

**delimiter //**

**create procedure Get\_country\_payments(in in\_year int, in in\_country varchar(50))**

**begin**

**select in\_year as year,in\_country as country, concat(round(sum(p.amount) / 1000), "K") as total\_amount**

**from customers as c join payments as p using (customerNumber)**

**where year(p.paymentDate) = in\_year and lower(c.country) = lower(in\_country)**

**group by in\_year,in\_country;**

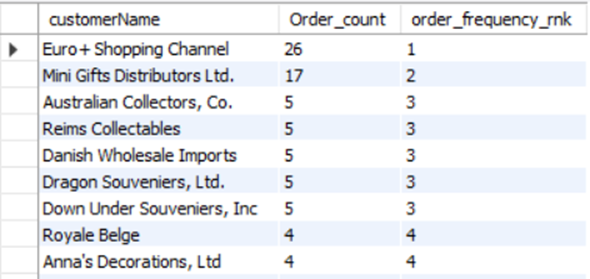
**end //**

**delimiter ;**

**call Get\_country\_payments(2003, "france");**

**Q10. Window functions - Rank, dense\_rank, lead and lag**

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



**ANSWER:**

**select customerName, count(o.orderNumber) as order\_count, dense\_rank() over(order by count(o.orderNumber) desc) as order\_frequency\_rank**

**from customers join orders as o using (customerNumber)**

**group by customerName**

**order by order\_frequency\_rank;**

b) 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 in % sign.

Table: Orders

**Expected output:**



**ANSWER:**

**select year(orderDate) as year, monthname(orderDate) as month,**

**count(orderNumber) as OrderCount,**

**round((count(orderNumber) - lag(count(orderNumber)) over (order by year(orderDate)))**

**/ lag(COUNT(orderNumber)) over (order by year(orderDate)) \* 100, 0) as YoYChange**

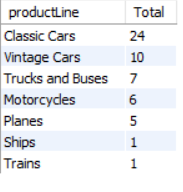
**from orders**

**group by Year, Month;**

**Q11.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:**



**ANSWER:**

**select productLine, count(productLine) as total**

**from products**

**where buyPrice > (select avg(buyPrice) from products)**

**group by productLine;**

**Q12. ERROR HANDLING in SQL**

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.

**ANSWER:**

**-- Creating employee Table**

**create table Emp\_EH(**

**EmpID int primary key,**

**EmpName varchar(50) not null,**

**EmailAddress varchar(50) not null**

**);**

**-- creating Procedure**

**delimiter //**

**create procedure in\_emp(in p\_EmpID int, in p\_EmpName varchar(50), in p\_EmailAddress varchar(50))**

**begin**

**declare exit handler for sqlexception**

**begin**

**select "Error occurred!!!" as message;**

**end;**

**insert into Emp\_EH (EmpID,EmpName,EmailAddress)**

**values (p\_EmpID,p\_EmpName,p\_EmailAddress);**

**select "Record inserted successfully" as message;**

**end //**

**delimiter ;**

**-- Record successfully inserted**

**CALL in\_emp(1, "Rutikesh", "rutikeshpawar@227");**

**-- Record Error massage showing due to duplicate value**

**CALL in\_emp(1, "Rutikesh", "rutikeshpawar@227");**

**-- Again successfully inserted**

**CALL in\_emp(2, "Rutikesh", "rutikeshpawar@227");**

**Q13. TRIGGERS**

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

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

Insert the data as shown in 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.

**ANSWER:**

**-- Creating Table Emp\_BIT**

**create table Emp\_BIT(**

**Name varchar(50) not null,**

**Occupation varchar(50) not null,**

**Working\_date date,**

**Working\_hours int**

**);**

**-- Inserting Data**

**insert into Emp\_BIT (Name,Occupation,Working\_date,Working\_hours)**

**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);**

**select \* from Emp\_BIT;**

**-- Creating Before insert trigger**

**delimiter //**

**create trigger befo\_ins**

**before insert on Emp\_BIT**

**for each row**

**begin**

**if new.Working\_hours < 0 then**

**set new.Working\_hours = abs(new.Working\_hours);**

**end if;**

**end //**

**delimiter ;**

**-- checking trigger works**

**insert into Emp\_BIT (Name,Occupation,Working\_date,Working\_hours)**

**values**

**('Rutikesh', 'Data Analytics', '2025-10-04', 40),**

**('Kohli', 'Crickter', '2011-10-04', -20);**

**select \* from emp\_bit;**