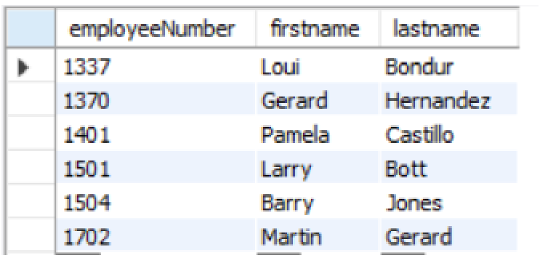
Q1. SELECT clause with WHERE, AND, DISTINCT, Wild Card (LIKE)

a. 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)

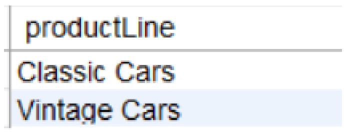
Expected output:



employeeNumber	firstname	lastname
1337	Loui	Bondur
1370	Gerard	Hernandez
1401	Pamela	Castillo
1501	Larry	Bott
1504	Barry	Jones
1702	Martin	Gerard

Query:
```
select employeeNumber,firstname,lastname
        from employees
  where reportsTo=1102;
```

b. Show the unique productline values containing the word cars at the end from the products table.
Expected output:



productLine
Classic Cars
Vintage Cars

Query:
```
select distinct(productLine)
from products
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)**

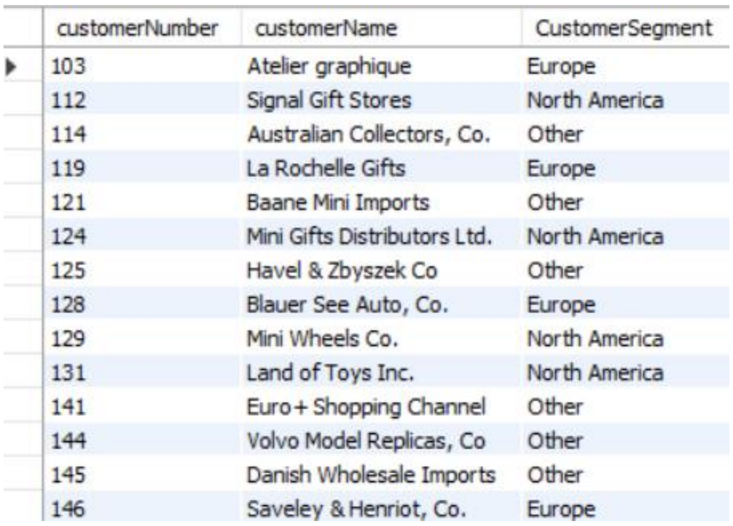
"North America" for customers from USA or Canada

"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:



customerNumber	customerName	CustomerSegment
103	Atelier graphique	Europe
112	Signal Gift Stores	North America
114	Australian Collectors, Co.	Other
119	La Rochelle Gifts	Europe
121	Baane Mini Imports	Other
124	Mini Gifts Distributors Ltd.	North America
125	Havel & Zbyszek Co	Other
128	Blauer See Auto, Co.	Europe
129	Mini Wheels Co.	North America
131	Land of Toys Inc.	North America
141	Euro+ Shopping Channel	Other
144	Volvo Model Replicas, Co	Other
145	Danish Wholesale Imports	Other
146	Saveley & Henriot, Co.	Europe

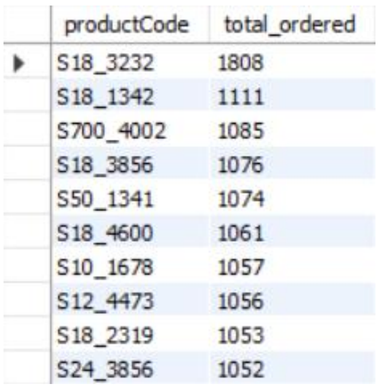
Query:

```
Select customerNumber, customerName,
case
when country="USA" or country="Canada" then "North America"
when country="UK"or country="France" or country="Germany" then "Europe"
else "Other"
end as CustomerSegment
from customers;
```

Q3. Group By with Aggregation functions and Having clause, Date and Time functions

a. Using the **OrderDetails table**, identify the top 10 products (by productCode) with the highest total order quantity across all orders.

Expected output:



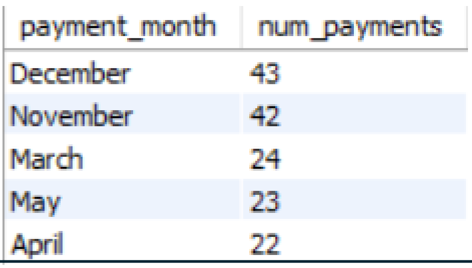
productCode	total_ordered
S18_3232	1808
S18_1342	1111
S700_4002	1085
S18_3856	1076
S50_1341	1074
S18_4600	1061
S10_1678	1057
S12_4473	1056
S18_2319	1053
S24_3856	1052

Query:

```
select productcode,sum(quantityOrdered) as total_ordered from orderdetails
group by productcode
order by total_ordered desc
limit 10;
```

b. 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:



payment_month	num_payments
December	43
November	42
March	24
May	23
April	22

Query:

```
select monthname(paymentdate) as Payment_month,count(*) as num_payments
from payments
group by Payment_month
having num_payments>20
order by num_payments 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

a. 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.

Query:

```
Create database Customers_Orders;
use Customers_Orders;
create table customers(
customer_id int primary key auto_increment,
first_name varchar(50) not null,
last_name varchar(50) not null,
email varchar(255) unique,
phone_number varchar(20)
);
```

b. 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:

a) Set a FOREIGN KEY constraint on customer_id to reference the Customers table.
b) Add a CHECK constraint to ensure the total_amount is always a positive value.

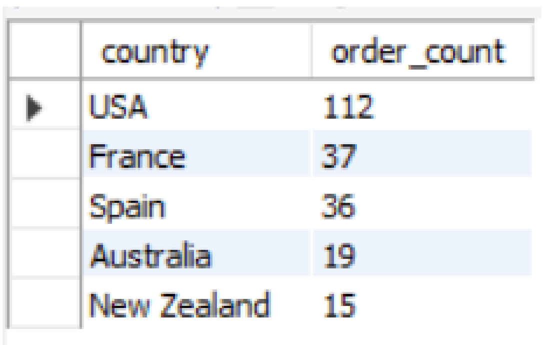
Query:

```
Create table Orders(
order_id int primary key auto_increment,
customer_id int,
foreign key(customer_id) references customers(customer_id),
order_date date,
total_amount decimal 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:



	country	order_count
▶	USA	112
	France	37
	Spain	36
	Australia	19
	New Zealand	15

Query:

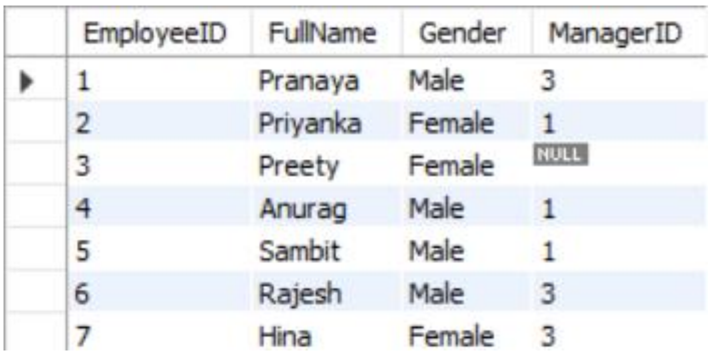
```
select c.country,count(o.ordernumber) as order_count

from customers c join orders o

on c.customernumber=o.customernumber

group by c.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

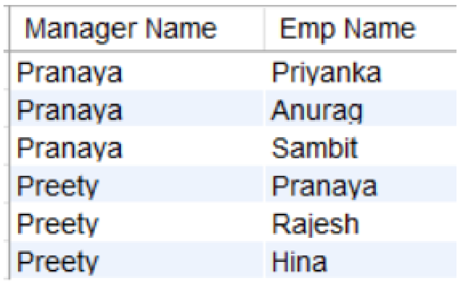
Add below data into it.



	EmployeeID	FullName	Gender	ManagerID
▶	1	Pranaya	Male	3
	2	Priyanka	Female	1
	3	Preety	Female	NULL
	4	Anurag	Male	1
	5	Sambit	Male	1
	6	Rajesh	Male	3
	7	Hina	Female	3

Find out the names of employees and their related managers.

Expected output:



Manager Name	Emp Name
Pranaya	Priyanka
Pranaya	Anurag
Pranaya	Sambit
Preety	Pranaya
Preety	Rajesh
Preety	Hina

Query:

```
Create table projects(

EmployeeID int PRIMARY KEY AUTO_INCREMENT,

FullName varchar(50) NOT NULL,

Gender varchar(10) check (gender in('male','female')),

ManagerID int

);

insert into projects values

(1,'Pranaya','Male',3),(2,'Priyanka','Female',1),

(3,'Preety','Female',null),(4,'Anurag','Male',1),

(5,'Sambit','Male',1),(6,'Rajesh','Male',3),(7,'Hina','Female',3);


select  e.fullname as manager_name,m.fullname as employee_name

from projects e join projects m

on e.employeeID=m.managerID

order by manager_name;
```

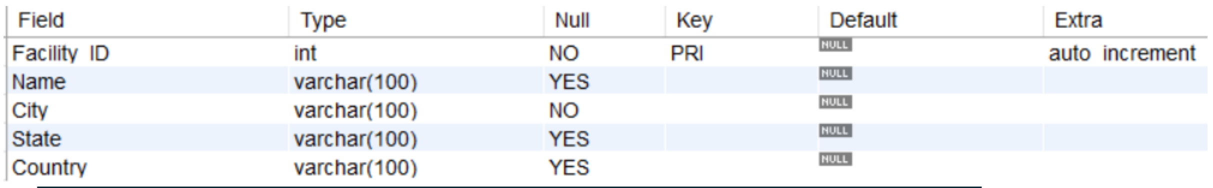
Q7. DDL Commands: Create, Alter, Rename

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.



Field	Type	Null	Key	Default	Extra
Facility_ID	int	NO	PRI	NULL	auto_increment
Name	varchar(100)	YES		NULL	
City	varchar(100)	NO		NULL	
State	varchar(100)	YES		NULL	
Country	varchar(100)	YES		NULL	

Query:

```
create table facility(
Facility_ID int,
Name varchar(100),
State varchar(100),
Country varchar(100)
);
alter table facility
modify facility_id int primary key auto_increment;
alter table facility
add column city varchar(100) not null
after name;
```

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:

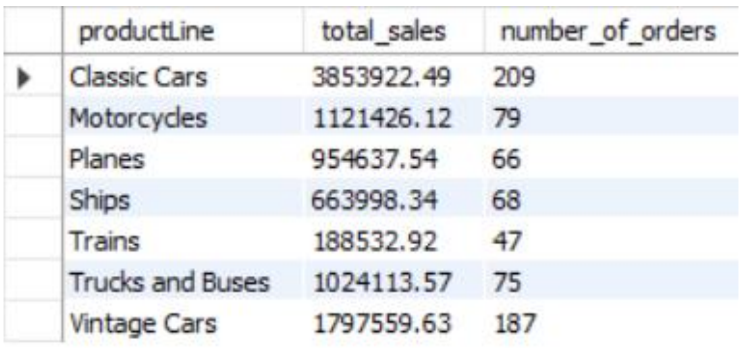
productLine: The category name of the product (from the ProductLines table).

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:



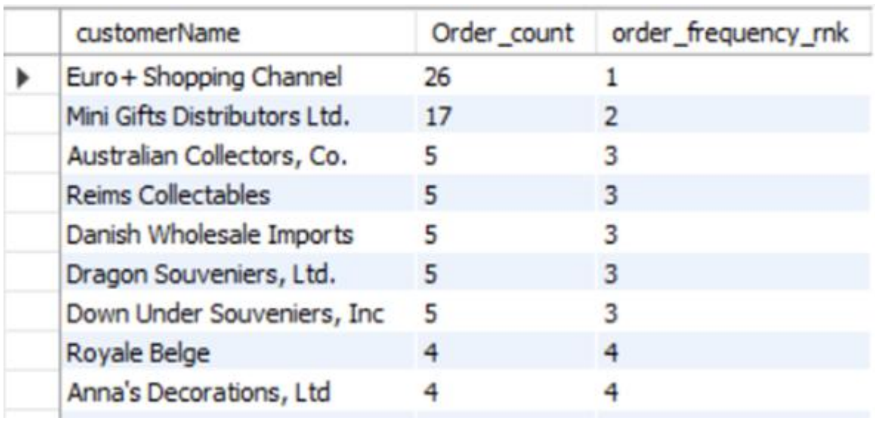
	productLine	total_sales	number_of_orders
▶	Classic Cars	3853922.49	209
	Motorcycles	1121426.12	79
	Planes	954637.54	66
	Ships	663998.34	68
	Trains	188532.92	47
	Trucks and Buses	1024113.57	75
	Vintage Cars	1797559.63	187

Query:

```
create view product_category_sales as

select pl.productLine,SUM(od.priceeach * od.quantityordered) as
total_sales,COUNT(DISTINCT(od.ordernumber)) as number_of_orders

from products pd join orderdetails od on od.productcode=pd.productcode

join productlines pl on pd.productline=pl.productline

group by productline;
```

Q09. Window functions - Rank, dense_rank, lead and lag

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



customerName	Order_count	order_frequency_rnk
Euro+ Shopping Channel	26	1
Mini Gifts Distributors Ltd.	17	2
Australian Collectors, Co.	5	3
Reims Collectables	5	3
Danish Wholesale Imports	5	3
Dragon Souveniers, Ltd.	5	3
Down Under Souveniers, Inc	5	3
Royale Belge	4	4
Anna's Decorations, Ltd	4	4

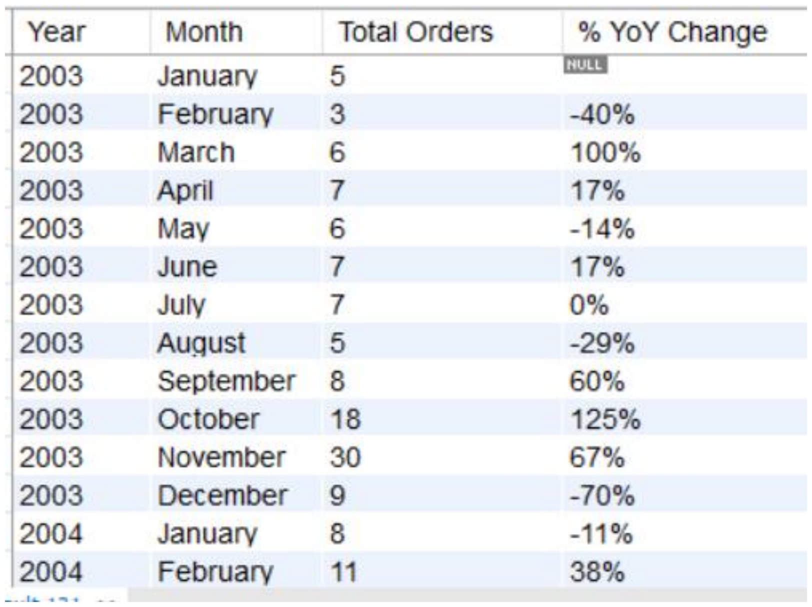
Query:

```
select c.customername,count(o.ordernumber) as order_count,

dense_rank() over (order by count(o.ordernumber) desc) as order_frequency_rnk

from orders o join customers c

on o.customernumber=c.customernumber

group by o.customernumber;
```

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:



Year	Month	Total Orders	% YoY Change
2003	January	5	NULL
2003	February	3	-40%
2003	March	6	100%
2003	April	7	17%
2003	May	6	-14%
2003	June	7	17%
2003	July	7	0%
2003	August	5	-29%
2003	September	8	60%
2003	October	18	125%
2003	November	30	67%
2003	December	9	-70%
2004	January	8	-11%
2004	February	11	38%

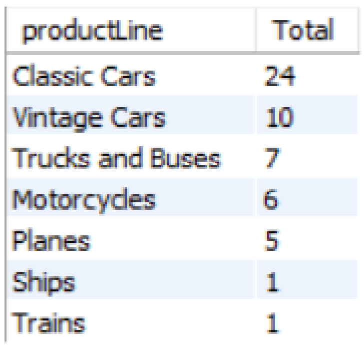
Query:

```
select year(orderdate) as year,monthname(orderdate) as month,count(ordernumber) as total_orders,

concat(

round(((count(ordernumber)-lag(count(ordernumber)) over (order by year(orderdate)))

/lag(count(ordernumber)) over (order by year(orderdate)) *100),0),

'%')

as YoY

from orders

group by 1,2;
```

Q10.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:



productLine	Total
Classic Cars	24
Vintage Cars	10
Trucks and Buses	7
Motorcycles	6
Planes	5
Ships	1
Trains	1

Query:

```
select productline, count(*) as total from products where buyprice>
(select avg(buyprice) from products)
group by productline order by total desc;
```

Q11. TRIGGERS

Create the table Emp_BIT. Add below fields in it.

- Name
- Occupation
- Working_date
- Working_hours

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

Query:

```
Create table Emp_BIT(
Name varchar(20),
Occupation varchar(25),
Working_date date,
Working_hours float);


/*How to create trigger:
Go to triggers. Execute and save below code:
CREATE DEFINER=`ARDRA`@`%` TRIGGER `emp_bit_BEFORE_INSERT` 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
/*
```