BHARAT STEEL MANUFACTURING COMPANY

create database BHARAT_STEEL_MANUFACTURING_COMPANY;

USE BHARAT_STEEL_MANUFACTURING_COMPANY;

create table product

(id int not null primary key, product_name varchar(200), stock_quantity int not null, unit_price varchar(200));

insert into product(id, product_name, stock_quantity, unit_price) values

(1, "plate A", 100, 51000),(2, "plate B", 150, 55000),

(3, "plate C", 200, 59000),(4, "plate D", 250, 61000),

(5, "plate E", 300, 66000),(6, "plate F", 350, 72000),

(7, "plate G", 400, 76000),(8, "plate H", 450, 81000),

(9, "plate I", 500, 83000),(10, "plate J", 550, 87000),

(11, "plate K", 600, 89000),(12, "plate L", 650, 93000);

select * from product;

Create table supplier

(supplier_id int not null primary key, supplier_name varchar(200), contact_person varchar(200), contact_number varchar(200));

insert into supplier(supplier_id, supplier_name, contact_person, contact_number) values

(1, "Ravi", "Ravi Hardware", 9712345678), (2, "Ram", "Ram Hardware", 9719145789),

(3, "Nikhil", "Nikhil Hardware", 9716845678), (4, "Nayan", "Nayan Hardware", 9947845678),

(5, "Chetan", "Chetan Hardware", 9712345325), (6, "Pankaj", "Pankaj Hardware", 9995445678),

(7, "Milind", "Milind Hardware", 9359845678), (8, "Amol", "Amol Hardware", 9719999678),

(9, "Akshay", "Akshay Hardware", 9713698578), (10, "Akhil", "Akhil Hardware", 9798564678),

(11, "Tushar", "Tushar Hardware", 9712345365), (12, "Mayur", "Mayur Hardware", 9712365478);

select * from supplier;

```
create table customer
(customer_id int not null primary key, customer_name varchar(200), contact_person varchar(200), contact_number varchar(200));
insert into customer(customer_id, customer_name, contact_person, contact_number) values
(1, "Mayur", "Mayur Hardware", 9712365478), (2, "Nikhil", "Nikhil Hardware", 9716845678),
(3, "Ram", "Ram Hardware", 9719145789), (4, "Tushar", "Tushar Hardware", 9712345365),
(5, "Akhil", "Akhil Hardware", 9798564678), (6, "Akshay", "Akshay Hardware", 9713698578),
(7, "Milind", "Milind Hardware", 9359845678), (8, "Amol", "Amol Hardware", 9719999678),
(9, "Pankaj", "Pankaj Hardware", 9995445678), (10, "Chetan", "Chetan Hardware", 9712345325),
(11, "Nayan", "Nayan Hardware", 9947845678), (12,"Ravi", "Ravi Hardware", 9712345678);
select * from customer;

create table orders
(orders_id int primary key, customer_id int, orders_date date, ship_date date,
foreign key(customer_id) references customer(customer_id) on delete cascade on update cascade);
Insert into orders(orders_id, customer_id, orders_date, ship_date) values
(1, 1, "2022-01-10", "2022-01-12"), (2, 2, "2022-01-15", "2022-01-18"),
(3, 3, "2022-01-22", "2022-01-25"), (4, 4, "2022-01-30", "2022-02-03"),
(5, 5, "2022-02-06", "2022-02-10"), (6, 6, "2022-02-14", "2022-02-16"),
(7, 7, "2022-02-20", "2022-02-22"), (8, 8, "2022-02-23", "2022-02-28"),
(9, 9, "2022-03-05", "2022-03-08"), (10, 10, "2022-03-11", "2022-03-17"),
(11, 11, "2022-03-15", "2022-03-20"), (12, 12, "2022-03-22", "2022-03-27");
select * from orders;

create table details
(orders_details_id int primary key, orders_id int, product_id int, quantity int not null,
foreign key(orders_id) references orders(orders_id) on delete cascade on update cascade,
foreign key(product_id) references product(id) on delete cascade on update cascade);
```

```
Insert into details(orders_details_id, orders_id, product_id, quantity) values
(1, 1, 1, 81), (2, 2, 2, 82), (3, 3, 3, 83), (4, 4, 4, 84),
(5, 5, 5, 85), (6, 6, 6, 86), (7, 7, 7, 87), (8, 8, 8, 88),
(9, 9, 9, 89), (10, 10, 10, 90), (11, 11, 11, 91), (12, 12, 12, 92);
select * from details;



#QUESTIONS---

#1. VIEW PRODUCTS INFO WITH ORDER DETAILS ?
create view detail as
select product.id, product.product_name, product.stock_quantity, product.unit_price,
details.orders_details_id, details.orders_id, details.quantity from product
join details on product.id = details.product_id;

#2. MANAGE SUPPLIERS AND CUSTOMER USING ADD, UPDATE, DELETE COMMANDS
        # ADD column location in customer table
alter table customer
add column location varchar (200);
select * from customer;
        # UPDATE customer name in customers table
update customer
set contact_number = 9999595954
where customer_id = 5;
select * from customer;
SET SQL_SAFE_UPDATES=0;
        #DELETE akshay row in supplier
delete from supplier
where supplier_id = 9;
```

#3. GENERATE REPORT- PRODUCT_SALES,ORDERS HISTORY, CURRENT INVENTORY

```
select sum (product.stock_quantity) as Total_available_stock,
sum(details.quantity) as Total_sales_stock from product
join details
on product.id = details.product_id;
```

#4. SUPPLIER MANAGEMENT- INFORMATION ABOUT SUPPLIERS INCLIDING CONTACT DETAILS.

#ANS- select contact_person, contact_number from supplier

#5. WRITE A SQL THEORY TO RETRIEVE ORDERS WITH AN ASSOCIATED CUSTOMER NAMES

```
select sum ( details.quantity) as total_stock_order, customer.customer_name from customer
join orders
on customer.customer_id = orders.customer_id
join details on orders.orders_id = details.order_id
group by customer.customer_name;
```

#6. CREATE A QUERY TO DISPLAY THE PRODUCTS ORDER IN A SPECIFIC ORDER INCLIDING THEIR NAMES AND QUANTITIES ?

```
select product.id, product.product_name, details.quantity from product
join details
on product.id = details.product_id
order by product.id ASC;
```

#7. PROVIDE SQL STATEMENT TO ADD A LIST OF ORDER WITH AN ASSOCIATED CUSTOMERS NAMES.

```
select  details.orders_id, product.product_name, details.quantity from product
join details
on product.id = details.product_id
order by details.order_id ASC;
```

#8. SHOW THE RELATIONSHIP BETWEEN ORDERS AND ORDERS DETAILS.

#ANS- The relationship shows the EER diagram which means Enhanced Entity Relationship.

Order table order_id is linked with order_details foreign key with same datatype and data.

Steps to show EER diagram (first click database on workbench after select reverse engineering and then click two times next and select

database and execute after that show EER diagram).

#9.HOW YOU HANDLE OR EXCEPTION DURING THE INSERTION OF NEW RECORDS INTO THE ORDERS AND ORDERS DETAILS TABLE

#ANS- Implement a try except block in code for inserting new records into the orders and orders details table on graceful

handle any exceptions that may arise during the process. Additionally, log or report the exception details for troubleshooting purposes.

#10. HOW WOULD YOU SECURE THE DATABASE FROM AN UNAUTORISE ACCESS.

#ANS--Enforce strict user authentication with strong passwords and implement role based access control to restrict

database access based on user roles, minimizing the risk of unauthorized access. Additionally, enable encryption for

data in transit and at rest to enhance overall database security.

#11. DESCRIBE THE STEPS TO RESTORE THE DATABASE TO A SPECIFIC POINTS IN TIME USING BACKUPS.

#ANS--first, ensure you have a valid backup file. Then, use the database management systems restoration process, specifying the

target point in time, to restore the database to a specific historical state.

#12. EXPLAIN THE CHALLENGES OF HANDLING CONCURRENT TRANSACTIONS IN A DATABASE WHERE MULTIPLE USERS MAY BE PLACING ORDERS SIMULATANIOUSLY.

#ANS-- Coordinating concurrent transactions in a database with simultaneous order placements requires addressing issues like data

consistency and implementing effective locking mechanisms for maintaining integrity.

#13. DESIGN A QUERY TO CALCULATE THE TOTAL QUANTITY OF SPECIFIC PRODUCT SOLD ACCROSS ALL PRODUCTS.

select sum (details.quantity) as total_quantity_product_sold, product.product_name from product

join details

```
on product.id = details.product_id

group by product.product_name;
```

#14. DESIGN A QUERY TO GENERATE THE REPORT SHOWING A TOTAL SALES FOR EACH PRODUCT.

```
select sum(details.quantity) as total_quantity_product_sold, product.product_name, product.id
from product

join details

on product.id = details.product_id

group by product.product_name, product.id;
```

#15. COMBINE THE ORDER AND ORDERS DETAILS TABLE USING INNER JOIN AND FILTER THE RESULT TO DISPLAY DETAILS FOR A SPECIFIC ORDERS.

```
SELECT * FROM orders

INNER JOIN details ON orders.orders_id = details.orders_id

order by orders ASC;
```

#16. DESIGN A QUERY TO GROUPS ORDERS BY CUSTOMER AND CALCULATE THE TOTAL QUANTITY OF PRODUCT ORDER BY EACH CUSTOMERS.

```
select sum(details.quantity)as total_quantity_order, customer.customer_id,
customer.customer_name, orders.customer_id from details

join customer on details.orders_details_id = customer.customer_id

join orders on orders.orders_id = details.orders_details_id

group by customer.customer_id

order by customer.customer_id ASC;
```

#17. UTILIZE SUBQUERY TO RETRIEVE THE NAMES OF CUSTOMER WHO HAVE PLACE ORDERS.

```
select customer_id, customer_name, contact_person from customer

where contact_person IN(select distinct(customer_name) from customer);
```

#18. COMBINE THE PRODUCT ORDER AND ORDERS DETAILS TABLE TO RETRIEVE THE LIST OF PRODUCTS ORDERS BY SPECIFIC CUSTOMER.

```
select product.id, product.product_name, orders.customer_id, details.orders_details_id,
details.product_id, details.orders_id,

customer.customer_name, customer.contact_person from product

join details on product.id = details.orders_details_id

join orders on details.orders_details_id = orders.orders_id

join customer on customer.customer_id = details.orders_details_id;
```

#19. IMPLEMENT A QUERY TO RETRIEVE THE FIRST 10 PRODUCT FROM THE PRODUCTS TABLE.

```
select * from product

limit 10;
```

#20. UTILIZE THE SUBQUERY TO FIND PRODUCT WITH A STOCK QUANTITY GREATER THAN THE AVG STOCK QUANTITY OF ALL PRODUCTS.

```
select * from product

where stock_quantity>(select avg(stock_quantity) from product);
```