Big Bazaar (E-commerce shopping system)

Updated Scope and Relational Schema

Logical Database Design :-

- **Customer**(Customer_ID, Name, Address, Email-ID, Username, Password, {Coupon Code}, {Telephone number})
- **Category**(Category_ID, Category Name)
- **Product**(Product_ID, Item name, Item_info, Quantity, Reviews)
- **Cart**(Customer_ID, Item_info, Total_price)
- **Payment**(Payment_ID, Customer_ID, Amount, Payment_method, Coupon_code)
- **Order**(Order_ID,Payment_ID, Customer_ID, Amount, Item_info, Date)
- **Supplier**(Supplier_ID, Name, Address, Rating)
- **Delivery**(Delivery_ID, Customer_ID, Order_ID, Status, Helpline)
- **Order history**(Order_ID, Customer_ID, Payment_ID, Date)
- **Return Order**(Return_ID, Order_ID)

Mapping Constraints :-

- **Customer to Supplier (I : M)**
- **Customer to Category (I : M)**
- **Customer to Order_history (I : M)**
- **Category to Product (I : M)**
- **Product to Cart (M : I)**
- **Cart to Payment (I : I)**
- **Payment to Order (I : I)**
- **Order to Supplier (I : I)**
- **Order to Order_history (I : I)**
- **Order_history to Return_Order (I : M)**
- **Supplier to Delivery (I : M)**

Integrity Constraints :-

- **Customer**
    - Customer_ID (integer(5), primary key)
    - Name (char(25), not null)
    - Address (varchar(255), not null)
    - Email_ID (varchar(255), default null)
    - Username (varchar(30), not null, unique)
    - Password (varchar(20), not null, unique

- **Telephone_number**
    - Phone_number (varchar(13), not null)
    - Customer_ID (integer(5), referential integrity constraint Customer(Customer_ID))

- **Category**
    - Category_ID (integer(3) primary key)
    - Category_name (char(25) ,not null ,unique)

- **Product**
    - Product_ID (integer(5), primary key)
    - Item_name (char(25), not null)
    - Quantity (integer(3), not null)
    - Item_info (json, not null)
    - Reviews (json)

- **Cart**
    - Customer_ID (integer(5), not null)
    - Item_info (json)
    - Total_price (float(7),not null)

- **Payment**
    - Payment_ID (integer(10), primary key)
    - Customer_ID (integer(5), not null)
    - Amount (float(7), not null)
    - Payment_method (varchar(10), default null)
    - Coupon_code (varchar(7), default null)

- **Coupon Code**
    - Coupon_name varchar(20) not null)
    - Coupon_discount integer(5) not null)

- **_Order_**
    - Order_ID (int (5) ,primary key)
    - Payment_ID (integer(10), not null, unique, referential integrity constraint Payment (Payment_ID))
    - Customer_ID (int(5) ,not null, unique)
    - Amount (float(7) ,not null)
    - Item_info (json, not null)
    - Order_Date (DATE)

- **Delivery**
    - Delivery_ID (integer(5), primary key)
    - Customer_ID (integer(5), not null)
    - Order_ID (integer(5), not null, unique, referential integrity constraint _Order_(Order_ID))
    - Status (char(20), not null)
    - Helpline (integer(13), not null)

- **Supplier**
    - Supplier_ID (integer(5), primary key)
    - Supplier_name (char(25), not null)
    - Address (varchar(255), not null)
    - Rating (integer(5), not null)

- **Order_history**
    - Order_ID (integer(5), primary key)
    - Customer_ID (integer(5), not null, unique, referential integrity constraint Customer(Customer_ID))
    - Payment_ID (integer(10), not null. unique, referential integrity constraint Payment (Payment_ID))
    - Order_Date (DATE, not null)

- **Return_order**
    - Return_ID (integer(5), primary key)
    - Order_ID (integer(5), not null, unique, referential integrity constraint _Order_(Order_ID))

Views and Grants

View1:
SQL query 3

View2:
SQL query 4

View 1 is a view created for the suppliers so that a supplier is able to access a customer's contact info and deliver an order to the customer's address successfully, this view doesn't show everything that is included, that is the username and password details of the customer. Therefore this view only shows the required information and hides the rest of it.

View 2 is a view created so that it becomes easy for the suppliers to see the frequency of orders placed by customers throughout or see the frequency of the orders placed in one single day. This query also doesn't include those orders which are being returned. This view hides the payment details of a customer and shows only the required information

Similar to this many views exist so that the customers or suppliers are only able to see only some particular things whereas the admin has access to everything.

Grants are given according to the assumptions made while designing the database for the respective ER diagram,
Only the admin is allowed to insert/delete/update the coupon codes and no one else can use some coupon code which will not be valid.
Only the admin has access to removing or inserting a new supplier to the E-commerce system.
On the request from either of the suppliers the admin can add new products.
Once the products are sold out their availability will be updated by the admin.

The following grants are given to the different suppliers that exist so that they can see when a product is getting out of stock or the statistics from the view2 that we created above or just use the information from the consumer_details view to deliver the orders and contact the customer.

```
grant Select on Product TO 'S1';
grant Select on Product TO 'S2';
grant Select on Product TO 'S3';
```

```
grant Select on consumer_details TO 'S1';
grant Select on consumer_details TO 'S2';
grant Select on consumer_details TO 'S3';

grant Select on order_dates TO 'S1';
grant Select on order_dates TO 'S2';
grant Select on order_dates TO 'S3';
```

SQL Queries

1) Customers who have availed coupon code for their purchase

```
Select Customer_ID, Name
from Customer
where Customer_Id IN
        (Select  C.Customer_ID
         From  Payment P, Cart C
         Where P.Couponcode IS NOT NULL AND P.Customer_ID = C.Customer_ID)
```

2) Customers who have placed orders uptil now.

```
Select C.Customer_ID, Name, Order_Date
From Customer C
        INNER JOIN _Order_ O
                On C.Customer_ID = O.Customer_ID;
```

3) Create a view where login credentials of customers are not required

```
create view consumer_details as
        select Customer_ID, Name, Address, Email_ID
        from customer;
```

4) Create a view where the customer's ID and the date the customer placed order is visible, and the view should not create those order's which were returned.

```
create view order_dates as
        select Customer_ID, Order_Date
         from _Order_ O1, Return_order O2
        where O1.Order_ID != O2.Order_ID;
```

5) Return count details of orders returned by customer

```
Select  O.Customer_ID ,count(O.Order_ID) as Count_
from Order_history O, Return_order R
where O.Order_ID = R.Return_ID
GROUP BY(O.Customer_ID);
```

6) ID's and name of suppliers who have 2+ rating AND are located at Najafgarh.

```
Select S.Supplier_ID,S.Supplier_name
From Supplier S
Where S.Address = 'Najafgarh' AND
        S.Supplier_ID IN (Select s1.Supplier_ID
                          From Supplier s1
                          Where s1.Rating > 2);
```

7) Name of the coupon code used for the orders whose status is either packed or delivered.

```
Select P.Coupon_code, O1.Order_ID
From Payment P , _Order_ O1
Where P.Payment_ID = O1.Payment_ID and P.Payment_ID IN
                (Select O.Payment_ID
                 From _Order_ O
                 Where O.Order_ID IN
                        (Select Order_ID
                         From Delivery
                         Where Order_status = 'Packed' or Order_status = 'Delivered'));
```

8) Name and price of those items present in Category having ID 3 or 5.

```
Select json_value(Item_info,'$.Price') as Price_, Item_name
from Product P
where json_value(Item_info,'$.Category_ID') = 3 OR  json_value(Item_info,'$.Category_ID') = 5 ;
```

9) Rank the orders which were not returned with respect to their amounts and their payment method was cash.

```
Select O.Order_ID, O.Item_Info, O.Amount
Rank() OVER (Order BY O.Amount Desc ) As Index_no
From _Order_ O
        INNER JOIN Payment P
                On P.Payment_ID = O.Payment_ID
        Where P.Payment_method = 'Cash' ;
```

10) List the item details from category with ID 1 having price greater than the maximum price form category with ID 3

```
Select P.Product_ID, P.Item_name
From Product P
Where json_value(P.Item_info,'$.Category_ID') = 1 AND
        json_value(P.Item_info,'$.Price') > (Select max(json_value(P2.Item_info,'$.Price'))
                                             From Product P2
                                             Where json_value(P2.Item_info,'$.Category_ID') = 3);
```

Embedded SQL Queries

1) Select Product_ID, Quantity, Item_name,
   RANK () OVER (Order BY Quantity DESC) As Index_no
   From Product;

2) Select Coupon_name, Coupon_Discount,
   RANK () OVER (Order BY Coupon_Discount Desc) As Index_no
   From Coupon_code;

3) (Select json_value(Item_info,'$.Price') as state from Product where Item_name='%s'"%atc_variable)

4) ("INSERT INTO _Order_ (Order_ID,Payment_ID, Customer_ID,Amount,Item_info,Order_Date)
   VALUES(%d,%d,%d,'%s','{}','2022-04-27')"%(int(orderid),int(paymentid),int(c_id),str(fprice[c_id])))

Indexing

1) create index Item_names on Product(Item_name);
Here we have created an index Item_names so that when we are searching for a particular item it becomes efficient and the other tuples of the relation are not scanned.

2) create index suppliers on Supplier(Supplier_name);
suppliers is an index here to list down the names of the available suppliers for the orders being placed.

3) create index Stock on Product(Item_name, Quantity);
Stock is the index created here because it would allow us to efficiently look at the availability of the products.

4) create index C_orders on _Order_(Order_ID, Customer_ID);
Order_ID is frequently used with Customer_ID to access a particular customer order, to make the process efficient the index C_orders is created.

5) create index Delivery_updates on Delivery(Order_ID, Order_Status);
Similar to the above index, here also Order_ID and Order_Status are used together to get details about an order being delivered or not, for making the process efficient we have created the index Delivery_updates.