CSE202: Fundamentals of Database Systems

Project Final Evaluation

Group 5

Day 'n' Nite Store

An Online Retail Store

Anuj Yadav	Bhagesh Gaur	Shivansh Choudhary	Shreyas Gupta
2020284	2020558	2020127	2020131

Description of the Problem

Day 'n' Nite Store has several branches all around India. Each branch stores many products that are supplied by various suppliers, such as wholesalers or producers. Customers can arrive at these branches physically and purchase one or more products. Day 'n' Nite Store has employees in each branch. Each employee works at exactly one branch, while each branch can have one or more employees. At each branch, exactly one of the employees works as the manager.

Apart from arriving at the store physically, customers can also order products using Day 'n' Nite Store’s App or Web portal. To order products, it is necessary for the customer to have an account registered on the app. The ordered items are delivered to the customers’ address.

To allow home delivery of products, the company makes use of many delivery services located throughout the country. Each courier service comprises various Courier Employees, who receive the product from a branch or partner store and deliver the products to the customers.

Day 'n' Nite Store has various types of stores. Some stores allow customers to arrive physically to purchase items, while also having inventory items that can be delivered from online orders. Some warehouse stores only stock products for online delivery and not for physical purchase. Day 'n' Nite Store also works with come partner stores, whose items can be delivered under the branding of Day 'n' Nite Store.

Products have many different categories, such as groceries, books, electronics, and clothing. For each product, its name and price are stored, along with its product description and manufacturer. Every product has 2 types of taxes: State GST and Central GST, which must be paid along with the MRP of the product.

Accounts can store multiple payment options. These can be one of three types: Credit/Debit cards, Net banking, and UPI. While making an order, the customer can select any of these options

Account holders can opt for a ‘Premium membership’, a subscription which gives them benefits such as discounts and faster shipping. The company also offers various coupon codes, which customers can redeem to get discounts on products purchased.

Items purchased physically, as well as items ordered online, have a return policy, and can be returned upto a few days after being delivered in case of any defect. In case of a return, the product is picked up by a delivery employee.

Account holders can write a review about a product. A review consists of a rating from 1 to 5 stars, a title, and a description. Account holders can also ask a question about a product, which can be answered by any of the sellers of the product. All questions/answers and reviews can be viewed by all customers on the app.

Scope of the Project

This project aims to create a database storing all the necessary details for a retail store system, to allow incorporation of data and features seen in larger retail store chains such as Big Bazaar and JioMart.

Such retail store systems consist of multiple stores or branches across the country, and each branch receives their products from various suppliers. Storing multiple attributes of products is important for the management of the store structure, finances, and communication of the store with all external stakeholders including customers. Furthermore, storing information about various customers and their purchases allows making the purchases and transactions easier, and allows many more features to the customers, such as availing discounts.

Stores such as JioMart have also allowed purchasing items online, further increasing its consumer base. An online interface further allows features such as home delivery, writing reviews, and asking questions.

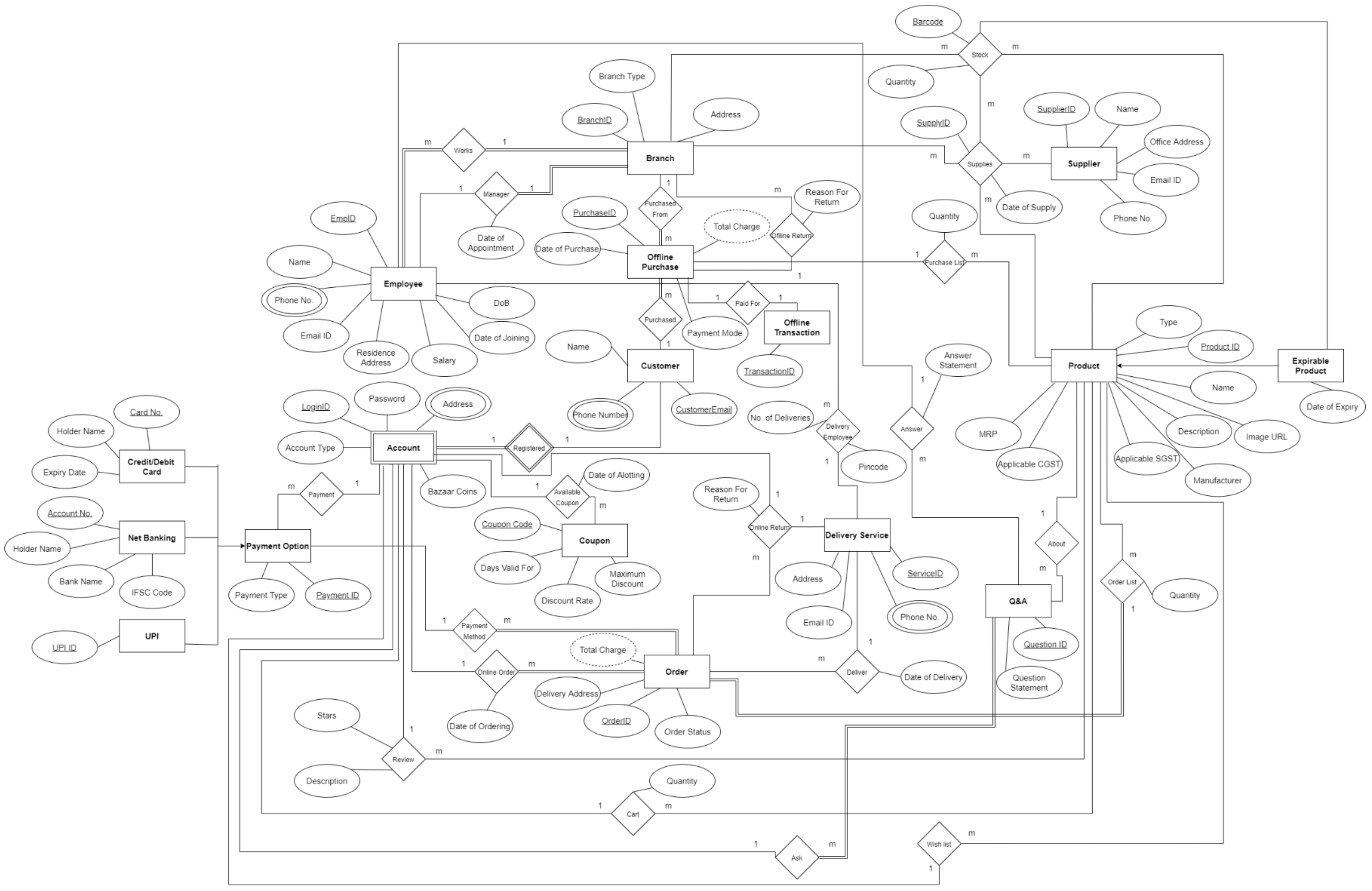
This project keeps these points in mind. For the mid-evaluation of the project, the project demonstrates the implementation of the back-end database in MySQL, including the implementation of various tables and examples of how to query the database.

Stakeholders Identified

The retail store system design incorporates the following identified stakeholders:

- Suppliers, such as wholesalers and producers
- Employees, including store employees and delivery employees
- Customers
- Delivery Services
- Net Banking Services, providing payment options to the customers

ER Diagram



Barcode
Stock
Quantity
Branch Type
Address
BranchID
Branch
Works
Manager
Date of Appointment
Purchased From
PurchaseID
Date of Purchase
Offline Purchase
Total Charge
Offline Return
Reason For Return
SupplyID
Supplies
SupplierID
Name
Supplier
Office Address
Email ID
Phone No.
Date of Supply
Quantity
Purchase List
EmpID
Name
Employee
Phone No.
Email ID
Residence Address
Salary
DoB
Date of Joining
Purchased
Payment Mode
Paid For
Offline Transaction
TransactionID
Name
Customer
Phone Number
CustomerEmail
No. of Deliveries
Delivery Employee
Pincode
Answer Statement
Answer
Type
Product ID
Product
Expirable Product
Name
Date of Expiry
Description
Image URL
MRP
Applicable CGST
Applicable SGST
Manufacturer
LoginID
Password
Address
Account Type
Account
Registered
Bazaar Coins
Available Coupon
Date of Alotting
Card No.
Holder Name
Credit/Debit Card
Expiry Date
Account No.
Net Banking
Holder Name
Bank Name
IFSC Code
UPI
UPI ID
Payment
Payment Option
Payment Type
Payment ID
Coupon Code
Coupon
Days Valid For
Maximum Discount
Discount Rate
Reason For Return
Online Return
Delivery Service
Address
ServiceID
Email ID
Phone No.
About
Q&A
Question ID
Question Statement
Order List
Quantity
Payment Method
Total Charge
Online Order
Order
Delivery Address
OrderID
Order Status
Date of Ordering
Deliver
Date of Delivery
Stars
Review
Description
Quantity
Cart
Ask
Wish list

Logical DataBase Design

Following is the relational schema derived from the ER diagram, and followed in the creation of the database tables:

- **Employee** (EmpID#, Name, {Phone No.#}, Email ID, ResidenceAddress(Street, Address Line, District, City, PinCode, Country), Date of Joining, Salary,BranchID#)
- **Branch** (BranchID#,Branch Type, Address(Street, Address Line, District, City, PinCode, Country))
- **Customer** (Name, {Phone No.#},EmailID)
- **Supplier** (SupplierID#, Name, OfficeAddress(Street, Address Line, District, City, PinCode, Country), Email ID, {Phone No.#}))
- **Product** (ProductID#,Name, Description, Applicable SGST, Applicable CGST, MRP, Manufacturer, Type, Product Link)
- **ExpirableProduct** (ProductID#, BarCode#, Date of Expiry)
- **Order** (OrderID#, Order Status, PaymentID#, TotalCost, DeliveryAddress(Street, Address Line, District, City, PinCode, Country))
- **QnA**(QuestionID#, ProductID#, LoginID#, Question Statement)
- **Offline Purchase** (PurchaseID#, BranchID#, CustomerID#, Date of Purchase, Payment Mode, Total Charge)
- **Account** (LoginID#,CustomerID#, Account Type, Bazaar Coins, {Address(Street, Address Line, District, City, PinCode, Country)})
- **Payment Options** (PaymentID#, LoginID#, Payment_type)
- **CreditDebitCard** (CardNo#, Holder Name, Expiry Date, PaymentID#)
- **Net Banking** (Account No#,Holder Name, Bank Name, IFSC Code, PaymentID#)
- **UPI** (UPIID, PaymentID#)
- **DeliveryService** (ServiceID#, Address(Street, Address Line, District, City, PinCode, Country), Email ID, {Phone No.#})
- **Coupon** (CouponCode#, DaysValidFor, Discount Rate, Maximum Discount)
- **Offline Transaction** (TransactionID#, PurchaseID)

- **Manager** (EmpID#, BranchID#, Date of Appointment)
- **DeliveryEmployee** (EmpID#, ServiceID#, No of Deliveries, PinCode)
- **Supplies** (SupplyID#, SupplierID#, ProductID#, BranchID#, Date of Supply)
- **Stock** (BarCode#, SupplyID#, BranchID#, ProductID#, Quantity)
- **Answer** (EmpID#, QuestionID#, Answer Statement)
- **OnlineOrder** (LoginID#, OrderID#, Date of Ordering)
- **Order List** (OrderID#, ProductID#, Quantity)
- **Offline Return** (PurchaseID#, BranchID#, Reason of Return)
- **Purchase List** (PurchaseID#, ProductID#, Quantity)
- **Online Return** (LoginID#, OrderID#, ServiceID#, Reason of Return)
- **Deliver** (OrderID#, ServiceID#, Date of Delivery)
- **AvailableCoupon** (LoginID#, CouponCode#, Date of Allotting)
- **Review** (ReviewID#, LoginID#, ProductID#, Stars, Description)
- **Cart**(LoginID#, ProductID#, Quantity)
- **Wishlist** (Wishlist#, LoginID#)

Views and Grants

Views

View for displaying Questions asked about products, along with answers for the same

```
create view Question_and_answer as
select QnA.QuestionID, ProductID, LoginID, Question_Statement,
Answer, Employee_Name
from QnA
left join Answer on QnA.QuestionID = Answer.QuestionID
left join Employee on Answer.EmployeeID = Employee.EmployeeID;

select * from Question_and_answer;
```

View for displaying all the details of an online order

```
CREATE VIEW ORDER_PAGE AS
     SELECT Order_Table.OrderID, LoginID, Date_of_Ordering,
PaymentID, Total_cost, Order_Status, Street, Address_Line, District,
City, Pincode, Country
    FROM Order_Table
    LEFT JOIN OnlineOrder ON Order_Table.OrderID=OnlineOrder.OrderID;
```

```
select * from ORDER_PAGE;
```

View for showing the names of all products in an order

```
create view Product_order as
select OrderID, Product_Name, Quantity
from Order_List
left join Product on Order_List.ProductID = Product.ProductID;
select * from Product_order;
```

Grants

```
const isAuthenticated = (req, res, next) => {
     if (req.session.auth) {
          next();
     } else {
          res.redirect('/signin');
     }
};

const isemployee = (req, res, next) => {
     if (req.session.isemployee) {
          next();
     } else {
          res.redirect('/signin');
     }
};

const isdiv = (req, res, next) => {
     if (req.session.isdiv) {
          next();
     } else {
          res.redirect('/signin');
     }
};

const issupplier = (req, res, next) => {
     if (req.session.issupplier) {
          next();
     } else {
          res.redirect('/signin');
     }
};
```

SQL Queries

For the database created, following are a list of **16 queries**, along with their corresponding relational algebraic expressions, supporting various application features:

Query 1. List all employees whose salary is less than the price of some item sold (inclusive of taxes) in their store in September 2021.

```
SELECT E.EmployeeID
FROM Employee E
WHERE E.Salary < SOME (
    SELECT P.Product_MRP + P.Applicable_CGST + P.Applicable_SGST
    FROM Product P
    WHERE P.ProductID IN (
        SELECT PL.ProductID
        FROM Purchase_List PL, Offline_Purchase OP
        WHERE E.BranchID = OP.BranchID
            AND OP.PurchaseID = PL.PurchaseID
            AND OP.Date_Of_Purchase >= '2021-09-01' AND
OP.Date_Of_Purchase <= '2021-09-30'
    )
);
```

Query 2. List all customers along with their reviews, who wrote a 1 star review of a given product and returned an order or purchase containing the same product

```
SELECT P.ProductID, P.Product_Name, R.Product_Preview
FROM Review R, Product P
WHERE P.ProductID = R.ProductID AND R.Stars = 1 AND R.ProductID IN
     (SELECT OL.ProductID
      FROM Order_List OL, OnlineReturn OnR
      WHERE OL.OrderID = OnR.OrderID AND R.LoginID = OnR.LoginID
      );
```

Query 3. Update the salary of the manager of a store by 5% in all stores where the total cost of items sold in that branch - total salary of all employees that are not non-manager employees is more than 100000. Else, decrease it by 2%.

```
UPDATE Employee
SET Employee.Salary = CASE
WHEN ((SELECT SUM(P.Product_MRP)
     FROM Product P
     WHERE P.ProductID IN (SELECT PL.ProductID
          FROM Purchase_List PL, Offline_Purchase OP
```

```
        WHERE PL.ProductID = P.ProductID AND PL.PurchaseID =
OP.PurchaseID AND OP.BranchID = Employee.BranchID))
     - (SELECT * FROM (SELECT SUM(E2.Salary)
           FROM Employee E2
        WHERE E2.BranchID = Employee.BranchID) AS X)) > 100000 THEN
salary * 1.05
ELSE salary * 0.98
END;
```

Query 4. List the coupon code(s) currently valid that offer the maximum discount for a purchase with a total cost of 1000.

```
SELECT C.CouponCode, LEAST(C.Discount_Rate * 1000/100,
C.Maximum_Discount) AS Discount
FROM Coupon C
WHERE LEAST(C.Discount_Rate * 1000/100, C.Maximum_Discount) =
     (SELECT MAX(LEAST(C2.Discount_Rate * 1000/100,
C2.Maximum_Discount))
     FROM Coupon C2
     WHERE C2.Valid_Till > CURDATE());
```

Query 5. List all questions and their answers that are answered by employees that are not manager employees.

```
SELECT Q.QuestionID, Q.Question_Statement, A.Answer
FROM QnA Q, Answer A
WHERE Q.QuestionID = A.QuestionID AND A.EmployeeID NOT IN (
    SELECT M.EmployeeID
    FROM Manager M
    GROUP BY M.BranchID
    HAVING MAX(Date_of_Appointment)
);
```

Query 6. Display the number of distinct products in offline purchases that were returned, having the word ‘damaged’ in their reason for return.

```
SELECT COUNT(*) AS Damaged_Or_Used
FROM (
     SELECT DISTINCT PL.ProductID
     FROM Purchase_List PL
     WHERE PL.PurchaseID IN (
          SELECT DISTINCT PurchaseID
          FROM Offline_Return OfR
          WHERE OfR.Reason_for_Return LIKE '%damaged%' OR
OfR.Reason_for_Return LIKE '%used%'
```

```
     )
) AS T;
```

Query 7. List all unanswered questions about the products of type 'Clothing'

```
SELECT Q.QuestionID, Q.Question_Statement
FROM QnA Q, Product P
WHERE Q.ProductID = P.ProductID AND P.Product_type = 'Clothing' AND
NOT EXISTS
      (SELECT A.QuestionID
      FROM Answer A
      WHERE A.QuestionID = Q.QuestionID);
```

Query 8. List the IDs and names all suppliers who delivered products to Branch ID 18 in January 2019

```
SELECT S.SupplierID, Sp.Supplier_Name
FROM Supplies S, Supplier Sp
WHERE S.SupplierId = Sp.SupplierID AND S.BranchID = 18 AND
S.Date_of_Supply >= '2019-01-01' AND S.Date_of_Supply <=
'2019-01-31';
```

Query 9. List all products that expired on or before August 1, 2021, along with branch stocking them, stored in the stock of each branch.

```
SELECT B.BranchID, S.Barcode
FROM Branch B, Stock S
WHERE B.BranchID = S.BranchID AND EXISTS
      (SELECT EB.Barcode
      FROM Expirable_Barcodes EB
      WHERE EB.Barcode = S.Barcode AND EB.Expiry_date <=
'2021-08-01');
```

Query 10. List the total salary of all non-manager employees in each branch.

```
SELECT B.BranchID, SUM(E.Salary) AS Total_Salary
FROM Branch B, Employee E
WHERE E.BranchID = B.BranchID AND NOT EXISTS
      (SELECT M.EmployeeID
      FROM Manager M
      WHERE M.EmployeeID = E.EmployeeID)
GROUP BY B.BranchID;
```

Query 11. List all customers who have either purchased books offline or have ordered books online.

```
(SELECT OP.CustomerID
FROM Offline_Purchase OP, Purchase_List PL
WHERE OP.PurchaseID = PL.PurchaseID AND EXISTS
      (SELECT P.ProductID
      FROM Product P
      WHERE P.ProductID = PL.ProductID AND P.Product_type = 'Books'))
UNION
(SELECT A.CustomerID
FROM Account A, OnlineOrder OO, Order_List OL
WHERE A.LoginID = OO.LoginID AND OO.OrderId = Ol.OrderID AND EXISTS
      (SELECT P.ProductID
      FROM Product P
      WHERE P.ProductID = OL.ProductID AND P.Product_type = 'Books'))
```

Query 12. List the Login IDs of all accounts having fewer Bazaar Coins than the sum of MRPs of all their cart items

```
SELECT A.LoginID
FROM Account A
Where A.Bazaar_coins <
      (SELECT SUM(P.Product_MRP * C.Quantity)
      FROM Product P, Cart C
      WHERE C.LoginID = A.LoginID AND C.ProductID = P.ProductID);
```

Query 13. List the description of all 1-start reviews of the product ‘Lettuce - Sea / Sea Asparagus’

```
SELECT DISTINCT R.Product_Preview
FROM Review R, Product P
WHERE P.ProductID = R.ProductID AND R.Stars = 1 AND P.Product_Name =
'Lettuce - Sea / Sea Asparagus';
```

Query 14: List the IDs and names of all products having ‘Bread’ in their name.

```
SELECT P.ProductID, P.Product_Name
FROM Product P
WHERE P.Product_Name LIKE '%Bread%';
```

Query 15. Create a view of product details which are in wishlist of all customers.

```
CREATE VIEW wishlist_Product_details AS
SELECT wishlist.loginID, wishlist.ProductID, Product.Product_Name,
Product_type, Product_MRP
FROM wishlist, Product
WHERE wishlist.ProductID = Product.ProductID;

select * from wishlist_Product_details;
```

Query 16. Create a view of login id's with available coupon details.

```
CREATE VIEW List_ids_with_coupon_details AS
SELECT available_coupon.LoginID, coupon.CouponCode,
coupon.Valid_Till, coupon.Discount_Rate, coupon.Maximum_Discount
FROM coupon, available_coupon
WHERE coupon.CouponCode = available_coupon.CouponCode;

select * from List_ids_with_coupon_details;
```

Embedded SQL Queries

Queries for Delivery Status and Return Management

Status of deliveries are managed by various delivery services, and they need to log in to be able to access the same.

The login credentials for a delivery service are their ServiceID and their email address. These are verified as follows:

```
SELECT COUNT(*) AS 'count'
FROM DeliveryService D
WHERE D.ServiceID = '${service_id}' AND D.Email_ID = '${email_id}'
```

A count of 0 means that no such delivery service exists. A count value of 1 indicates a valid login.

A delivery service is only allowed to manage the delivery status of orders of the same city. The city of the delivery service is obtained, and is used to obtain the order details from the same city.

```
SELECT D.City
FROM DeliveryService D
WHERE D.ServiceID = ${req.query.service_id}

SELECT OT.OrderID, OT.Total_cost, OO.Date_of_Ordering,
OT.Order_Status, OT.Street, OT.Address_Line, OT.District, OT.City,
OT.PinCode
FROM Order_Table OT, OnlineOrder OO
WHERE OT.OrderID = OO.OrderID AND OT.City = '${city}'
```

The delivery service can choose an order ID and can change its order status.

If an order is delivered, it must be recorded in the 'Deliver' table with the date of delivery.

```
INSERT INTO Deliver (orderid, serviceid, date_of_delivery)
VALUES (${req.query.order_id}, ${req.query.service_id},
'${req.body.delivery_date_field})'
```

If an order is returned, it must be recorded in the 'Returned' table with the reason.

```
INSERT INTO OnlineReturn (orderid, loginid, serviceid,
reason_for_return)
VALUES (${req.query.order_id}, '${loginID}', ${req.query.service_id},
'${req.body.reason_field}')
```

Queries for Offline Purchase Registration

A customer purchases items from a particular branch. To login, an employee enters their employee ID and date of joining as credentials. They are then logged in to their particular branch.

```
SELECT E.BranchID
FROM Employee E
WHERE E.EmployeeID = ${employee_id} AND E.Date_of_join =
'${employee_dob}';
```

For a particular branch, all items in stock are displayed.

```
SELECT *
FROM Stock S
WHERE S.BranchID = ${req.query.branch_id};
```

Items are added to a customer's purchase list by entering the barcode of the given product (Similar to scanning a barcode at a retail store). Each scan adds the item with quantity 1.

```
SELECT S.ProductID
FROM Stock S
WHERE S.BranchID = ${req.query.branch_id} AND S.barcode =
'${item_list[i]}';

SELECT P.Product_Name AS ProductName, (P.Product_MRP +
P.Applicable_CGST + P.Applicable_SGST) * ${quantity_list[i]} AS Cost
FROM Product P
WHERE P.ProductID = ${result[0].ProductID};
```

On checkout, the customer's details are taken. The purchase is registered in 'Offline_Order' and the corresponding items are recorded in 'Purchase_List'

```
INSERT INTO Offline_Purchase (date_of_purchase, payment_mode,
total_charge, branchid, customerid)
```

```
VALUES (CURDATE(), '${req.body.payment_mode_field}',
${req.query.total_charge}, ${req.query.branch_id},
${req.body.customer_id_field});

INSERT INTO Purchase_List
VALUES (${purchaseID}, '${item_list[i]}', ${quantity_list[i]},
${productID_list[i]});
```

Some Additional Embedded Queries:

```
"UPDATE Order_Table SET Order_Table = 'C' WHERE OrderID = " + ID + "
and Order_Status = 'ND'";
"Select Available_coupon.CouponCode, Valid_Till, Date_of_allocating,
Date_of_allocating, Discount_Rate, Maximum_Discount from
Available_coupon left join Coupon on Available_coupon.CouponCode =
Coupon.CouponCode where LoginID = '" + req.session.user + "'";

"INSERT INTO Supplies
(SupplyID,SupplierID,ProductID,BranchID,Date_of_Supply) VALUES ('" +
supplyid + "', '" + supplierid + "', '" + productid + "', '" +
branchid + "', CURDATE())";

('SELECT * FROM product WHERE Product_Name LIKE ?', ['%' + search +
'%'], (err, products) => {
        if (!err) {
        res.render('index', { products });
        } else {
        console.log(err);
           }


"DELETE FROM Account_address WHERE AddressId = " + [address_index] +
";";
"Insert into Review (ProductID, LoginID, Product_Preview, Stars)
VALUES (" + [search.ProductID] + ", '" + req.session.user + "', '" +
Review + "', " + rating + ");";

"SELECT WishList.LoginID, WishList.ProductID, Product.Product_Name,
Product.Product_MRP, Product.Applicable_SGST,
Product.Applicable_CGST, Product.Product_link FROM WishList, Product
WHERE WishList.LoginID = '"+req.session.user+"' AND
WishList.ProductID = Product.ProductID";
"SELECT * FROM Cart WHERE LoginID='"+req.session.user +"' AND
ProductID=" + search.ProductID +";";
```

```
"INSERT INTO Cart(LoginID, ProductID) VALUES ('"+req.session.user
+"'," + search.ProductID +");"


"SELECT * FROM WishList WHERE LoginID='"+req.session.user +"' AND
ProductID=" + search.ProductID +";";
"SELECT * FROM WishList WHERE LoginID='"+req.session.user +"' AND
ProductID=" + search.ProductID +";";

"SELECT Cart.Quantity, Cart.LoginID, Cart.ProductID,
Product.ProductID, Product.Product_Name, Product.Product_MRP,
Product.Applicable_SGST, Product.Applicable_CGST,
Product.Product_link FROM Cart, Product WHERE Cart.LoginID =
'"+userid+"' AND Cart.ProductID = Product.ProductID";

"SELECT AC.CouponCode, AC.Date_of_allocating, C.Valid_Till,
C.Discount_Rate, C.Maximum_Discount FROM Available_coupon AC, Coupon
C WHERE AC.LoginID = '"+userid+"' AND AC.CouponCode = C.CouponCode
AND C.Valid_Till >= CURDATE()";

"SELECT Discount_Rate, Maximum_Discount FROM Coupon WHERE
CouponCode='"+ couponCode +"'";
"SELECT AddressId, Street, Address_Line, District, City, Pincode,
Country FROM Account_address WHERE AddressId="+ address;

"INSERT INTO Order_List(OrderID, ProductID, Quantity) VALUES ("+
last_order_id +"," + ans[i].ProductID +","+ ans[i].Quantity+");"
"DELETE FROM Cart WHERE LoginID = '"+req.session.user +"' AND
ProductID = "+ ans[i].ProductID +";";
"INSERT INTO Order_Table(OrderID, PaymentID, Total_cost, Street,
Address_Line, District, City, Pincode, Country) VALUES ("+
last_order_id +"," + payment +","+
total+",'"+ans[0].Street+"','"+ans[0].Address_Line
+"','"+ans[0].District+"','"+
ans[0].City+"','"+ans[0].Pincode+"','"+ans[0].Country+"');"
"INSERT INTO OnlineOrder(OrderID, LoginID, Date_of_Ordering) VALUES
("+ last_order_id +",'" + req.session.user +"',CURDATE());"
"SELECT * FROM Cart WHERE LoginID = '"+req.session.user +"'";


'Delete from ' + table + ' where PaymentID = ' +
req.params.ID.substring(1);
"SELECT Quantity FROM Cart WHERE ProductID = " +
req.params.ID.substring(1) + " AND LoginID = '"+ req.session.user
+"'";
```

```
"UPDATE Cart SET Quantity = Quantity - 1 WHERE ProductID = " +
req.params.ID.substring(1) + " AND LoginID = '"+req.session.user+"'";

"DELETE FROM Cart WHERE ProductID = " + req.params.ID.substring(1) +
" AND LoginID = '"+req.session.user+"'";

"SELECT Quantity FROM Cart WHERE ProductID = " +
req.params.ID.substring(1) + " AND LoginID = '"+req.session.user+"'";

"SELECT SUM(Product_Quantity) AS Quantity FROM Stock WHERE ProductID
= "+ req.params.ID.substring(1);

"UPDATE Cart SET Quantity = Quantity + 1 WHERE ProductID = " +
req.params.ID.substring(1) + " AND LoginID = '"+req.session.user+"'";
```

Query Optimization

The queries listed above have been optimized to ensure their execution is efficient in their worst case. This has been done in the following ways:

1. Tables that are frequently searched have been **indexed** with appropriate parameters. Tables containing login information of various users and stakeholders are queried frequently, as well as tables containing information about products, questions, and reviews, that are frequently searched by the app.

2. Joining two tables using **natural join** is expensive, and queries that could use natural join have instead been written more optimally.

3. In case of multiple selection or projection operations, they are ordered to ensure a minimum number of seek operations. For instance, for finding products in Stock with a particular BranchID and Barcode, selection operation is performed on BranchID, followed by on Barcode. Stock is indexed on BranchID, and this order is faster in the average case.

4. **Views** have been created for certain tables and attributes. Querying them is a faster process for tables/attributes that are searched frequently and not updated often.

Indexing

To facilitate fast retrieval of data, various indexes have been defined in various tables of the database. The following points were kept in mind when defining indexes:

- Logging in for various stakeholders requires fast retrieval of the person's record. Logging in is done using some defined primary key (such as ServiceID for a Delivery Service) and so the same is made as an index
- For a particular entity, the app often either searches for it through its primary, or with reference to some foreign key (eg: A customer's order list is fetched using its orderID). Such attributes are indexed as well to support such queries.

The following list shows the relational tables, along with the attribute(s) that have been indexed

Table	**Index Attribute(s)**
Branch	BranchID
Customer	CustomerID
Customer_Phone	CustomerID
Employee	EmployeeID
Employee_Phone	EmployeeID
Manager	EmployeeID, BranchID
Supplier	SupplierID, City
Supplier_Phone	SupplierID
Product	ProductID
Stock	BranchID, Barcode
Coupon	CouponCode
Account	LoginID
Payment_Options	LoginID
QnA	ProductID
Answer	QuestionID
DeliveryService	ServiceID
DeliveryService_Phone	ServiceID
Deliver	ServiceID, OrderID
Review	ProductID

OnlineReturn	OrderID, ServiceID
Purchase_List	PurchaseID
Cart	LoginID
Wishlist	LoginID

<u>Indexing Queries</u>

```
CREATE UNIQUE INDEX Branch_index ON Branch (BranchID);
CREATE UNIQUE INDEX Customer_index ON Customer (Email_ID);
CREATE INDEX Customer_Phone_index ON Customer_Phone (CustomerID);
CREATE UNIQUE INDEX Employee_index ON Employee (EmployeeID);
CREATE INDEX Employee_Phone_index ON Employee_Phone (EmployeeID);
CREATE UNIQUE INDEX Manager_Employee_index ON Manager (EmployeeID);
CREATE UNIQUE INDEX Manager_Branch_index ON Manager (BranchID);
CREATE UNIQUE INDEX Supplier_index ON Supplier (SupplierID);
CREATE INDEX Supplier_Phone_index ON Supplier_Phone (SupplierID);
CREATE UNIQUE INDEX Product_index ON Product (ProductID);
CREATE INDEX Stock_Branch_index ON Stock (BranchID);
CREATE INDEX Stock_Barcode_index ON Stock (Barcode);
CREATE UNIQUE INDEX Coupon_index ON Coupon (CouponCode);
CREATE UNIQUE INDEX Account_index ON Account (LoginID);
CREATE INDEX Payment_Options_index ON Payment_Options (LoginID);
CREATE INDEX QnA_Product_index ON QnA (ProductID);
CREATE INDEX Answer_Question_index ON Answer (QuestionID);
CREATE UNIQUE INDEX DeliveryService_index ON DeliveryService
(ServiceID);
CREATE INDEX DeliveryService_Phone_index ON DeliveryService_Phone
(ServiceID);
CREATE INDEX Deliver_Service_index ON Deliver (ServiceID);
CREATE INDEX Deliver_Order_index ON Deliver (OrderID);
CREATE INDEX Review_Product_index ON Review (ProductID);
CREATE INDEX OnlineReturn_Order_index ON OnlineReturn (OrderID);
CREATE INDEX OnlineReturn_Service_index ON OnlineReturn (ServiceID);
CREATE INDEX Purchase_List_index ON Purchase_List (PurchaseID);
CREATE INDEX Cart_Login_index ON Cart (LoginID);
CREATE INDEX Wishlist_Login_index ON Wishlist (LoginID);
```

<u>Triggers</u>

In the database, the following triggers have been implemented to support data management:

- Temporal Trigger to remove all expired products from stock everyday at 12 a.m.

- After Insert Trigger to update the quantity of each product in stock after a purchase is made.

<u>Remove all expired products from stock everyday at 12 a.m.</u>

To support temporal triggers, an event is created in the MySQL database that performs the task of deleting all expired products everyday at 12 a.m.

```
CREATE EVENT remove_expired_products
ON SCHEDULE EVERY 1 DAY
STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY)
DO
      DELETE FROM Stock
      WHERE Stock.Barcode IN (
            SELECT EB.Barcode
            FROM Expirable_Barcodes EB
            WHERE EB.Expiry_date < CURDATE()
      );
```

<u>Update the quantity of each product in stock after a offline purchase is made</u>

This is an ‘after insert’ trigger. Once items are added to the purchase list, the same items must be removed from stock. This trigger ensures that addition to the purchase list automatically updates the stock.

```
CREATE TRIGGER update_stock_after_purchase
AFTER INSERT
ON Purchase_List FOR EACH ROW
      UPDATE Stock
      SET Stock.Quantity = Stock.Quantity - NEW.Quantity
      WHERE Stock.BranchID IN (
            SELECT OP.BranchID
            FROM Offline_Purchase OP
            WHERE OP.PurchaseID = NEW.PurchaseID
      );
```

<u>Remove all expired coupons from the database everyday at 12 a.m.</u>

To support temporal triggers, an event is created in the MySQL database that performs the task of deleting all expired coupons everyday at 12 a.m.

```
CREATE EVENT remove_expired_coupons
ON SCHEDULE EVERY 1 DAY
STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY)
DO
      DELETE FROM Coupon, Available_coupon
      WHERE Coupon.CouponCode IN (
          SELECT C.CouponCode
          FROM Coupon C
          WHERE C.Valid_Till < CURDATE()
     );
```

<u>Remove all expired credit/debit cards from the database everyday at 12 a.m.</u>

To support temporal triggers, an event is created in the MySQL database that performs the task of deleting all expired credit/debit cards everyday at 12 a.m.

```
CREATE EVENT remove_expired_creditDebitCards
ON SCHEDULE EVERY 1 DAY
STARTS (TIMESTAMP(CURRENT_DATE) + INTERVAL 1 DAY)
DO
     DELETE FROM CreditDebitCard
     WHERE Expiry_Date < CURDATE()
     );
```