Department of Computer Science and Engineering, University of Jeddah

---

LET'S KEEP IT LOCAL

DataBase Project

Student's Names:

Aseel Almehmadi - 2211696

Fatima Almaashi - 2210392

Hadeel Abdulhadi - 2210937

Second Semester 1445 A.H-2024 A.D.

Table of contents

Narrative Description:

Leaf & Bean *is a specialty store committed to promoting locally-made Saudi products, specialising in a diverse selection of matcha, coffee, and brewing equipment. Dedicated to showcasing Saudi craftsmanship and enhancing the local coffee industry, the store faces challenges in efficiently managing inventory, processing orders, and maintaining customer relations. As the product range expands and the customer base grows, there is a pressing need to optimise operations. A proposed solution is the implementation of a database. This database is designed to streamline the tracking of inventory levels, manage employee roles, and process orders more efficiently. By creating a centralised information system,* ***Leaf & Bean*** *aims to improve the elaborate relationships between products, employees, and customers, thereby boosting operational efficiency and enhancing customer satisfaction*

Identification of Information Needs:

For the purpose of resolving the problem, it is essential for us to have the following information:

- ***Inventory Levels****: Real-time data on the stock levels of different products (matcha, coffees, equipment) to manage supply effectively.*
- ***Employee Roles and Responsibilities****: Information on each employee's position, responsibilities, and store assignments to optimise workforce management.*
- ***Order Management****: Details on order statuses, customer information, and invoice management for efficient order processing and fulfilment.*
- ***Customer Data****: Insights into customers, and an accurate up-to-date contact information to facilitate communication and personalised service*

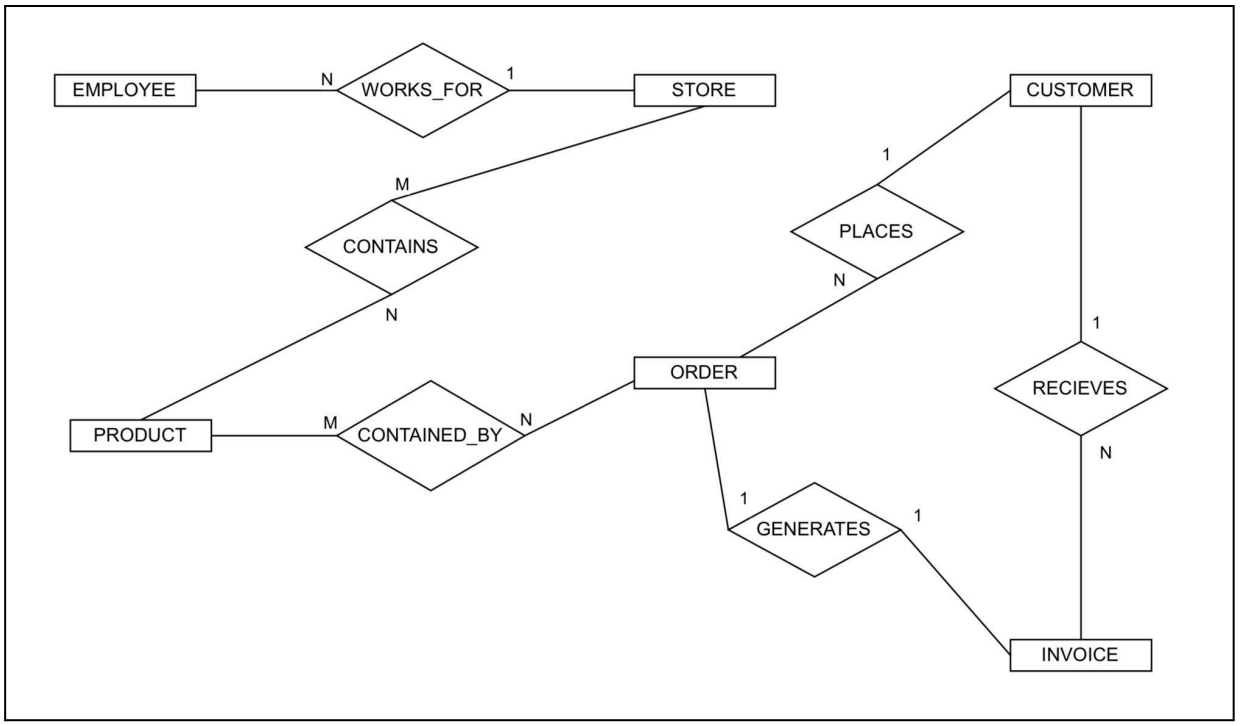
Initial List of Entities (Tables) Identified:

The entities that would be integral to the database management system of the leaf and bean store:

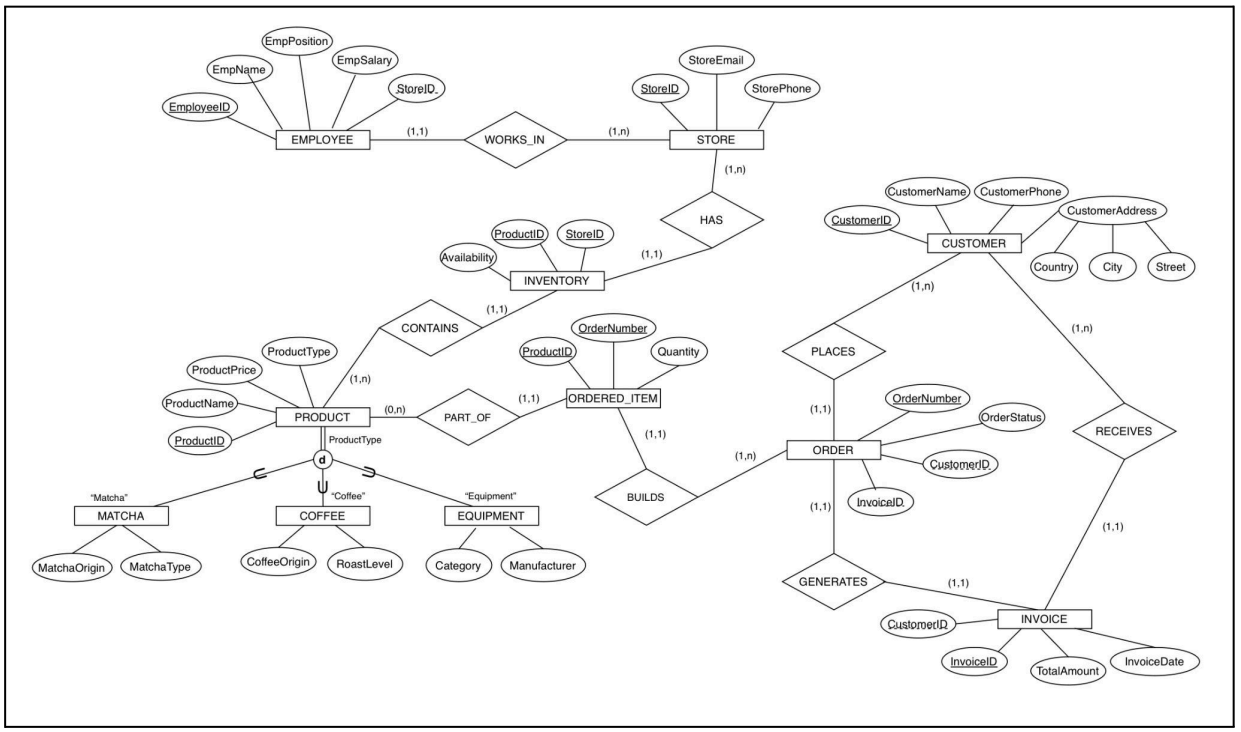
1. ***EMPLOYEE***
    - ***Attributes****: EmployeeID, EmpName, EmpPosition, EmpSalary*
    - ***Relationships****: Works in STOREs.*
2. ***STORE***
    - ***Attributes****: StoreID, StoreEmail, StorePhone*
    - ***Relationships****: Contains multiple EMPLOYEES, manages INVENTORY, and is associated with ORDERs through its products.*
3. ***INVENTORY***
    - ***Attributes****: Availability*
    - ***Relationships****: Contains PRODUCTs, linked to a STORE.*
4. ***PRODUCT***
    - ***Attributes****: ProductID, ProductName, ProductPrice, ProductType*
    - ***Sub-categories****:*
        - ***Matcha*** *(MatchaOrigin, MatchaType)*
        - ***Coffee*** *(CoffeeOrigin, RoastLevel)*
        - ***Equipment*** *(Category, Manufacturer)*
    - ***Relationships****: Part of INVENTORY, and ORDERED_ITEM.*
5. ***ORDERED_ITEM***
    - ***Attributes****: ProductID, OrderNumber, Quantity*
    - ***Relationships****: Part of an ORDER, associated with a PRODUCT.*
6. ***ORDER***
    - ***Attributes****: OrderNumber, OrderStatus*
    - ***Relationships****: Placed by a CUSTOMER, generates an INVOICE, contains ORDERED_ITEMs.*
7. ***INVOICE***
    - ***Attributes****: InvoiceID, TotalAmount, InvoiceDate*
    - ***Relationships****: Generated by an ORDER, related to a CUSTOMER.*
8. ***CUSTOMER***
    - ***Attributes****: CustomerID, CustomerName, CustomerPhone, CustomerAddress(Country,City,Street)*
    - ***Relationships****: Places ORDERs, receives INVOICEs.*

ER-Diagram

- Conceptual:

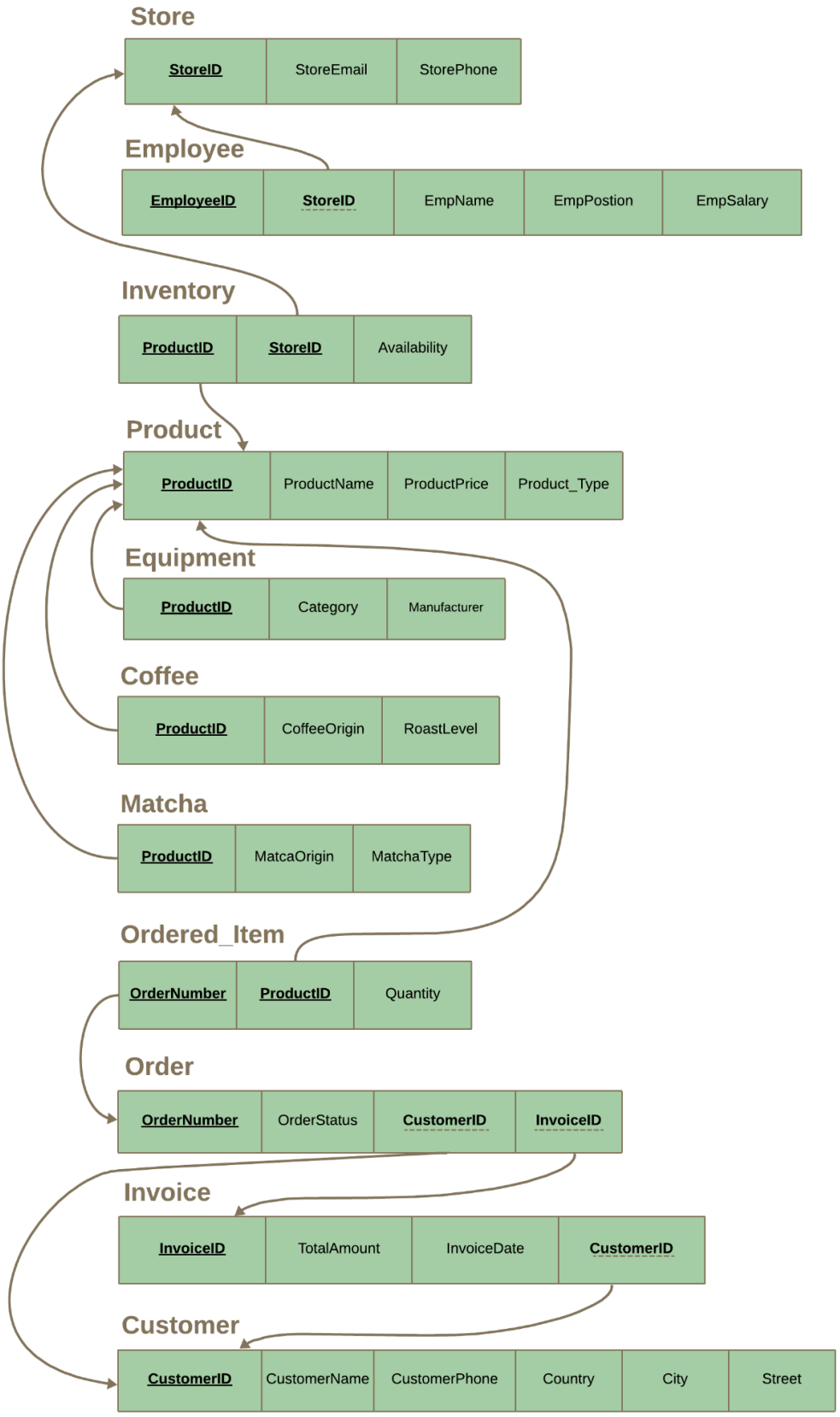


- Logical:



Logical Modeling & Normalization:

-Relational Schema:



Functional Dependencies

- ***StoreID*** ⇒ *StoreEmail, StorePhone*
- ***EmployeeID*** ⇒ *EmpName, EmpPosition, EmpSalary*
- ***CustomerID*** ⇒ *CustomerName, CustomerPhone, Country, City, Street*
- ***InvoiceID*** ⇒ *TotalAmount, InvoiceDate*
- ***OrderNumber*** ⇒ *OrderStatus*
- ***ProductID*** ⇒ *ProductName, ProductPrice, MatchaOrigin, MatchaType, CoffeeOrigin, RoastLevel, Category, Manufacturer*
- ***ProductID, StoreID*** ⇒ *Availability*
- ***ProductID, OrderNumber*** ⇒ *Quantity*

Normalization

➢ ***First Normal Form (1NF):***

Already in First Normal Form because:

- *No multi-value attributes (repeating groups) exist, and a unique key has been identified for each relation. This ensures each record is unique and data is stored in a clear, tabular format without any groups of data.*

➢ ***Second Normal Form (2NF):***

Already in Second Normal Form because:

- *No partial dependencies are present; all non-key attributes are fully dependent on the entire primary key, not just parts of it. This enhances data integrity and reduces redundancy.*

➢ ***Third Normal Form (3NF):***

Already in Third Normal Form because:

- *No transitive dependencies are found; all non-key attributes depend directly on the primary key and not on other non-key attributes. This simplifies data maintenance and improves data consistency.*

Physical Database implementation:

➢ *Creating tables:*

```
-- STORE Table --
CREATE TABLE STORE(
StoreID VARCHAR2(7) NOT NULL,
StoreEmail VARCHAR2(30) NOT NULL,
StorePhone NUMBER(10) NOT NULL,
CONSTRAINT StoreID_PK PRIMARY KEY(StoreID));

-- PRODUCT Table --
CREATE TABLE PRODUCT(
ProductID NUMBER(6) NOT NULL,
ProductName VARCHAR2(50) NOT NULL,
ProductType VARCHAR2(25) NOT NULL,
ProductPrice NUMBER(10,2) NOT NULL,
CONSTRAINT ProductID_PK PRIMARY KEY(ProductID));

-- COFFEE Table --
CREATE TABLE COFFEE (
ProductID NUMBER(6) PRIMARY KEY,
CoffeeOrigin VARCHAR2(50) NOT NULL,
RoastLevel VARCHAR2(50) NOT NULL,
FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID));

-- MATCHA Table --
CREATE TABLE MATCHA (
ProductID NUMBER(6) PRIMARY KEY,
MatchaOrigin VARCHAR2(50) NOT NULL,
MatchaType VARCHAR2(50) NOT NULL,
FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID));

-- EQUIPMENT Table --
CREATE TABLE EQUIPMENT (
ProductID NUMBER(6) PRIMARY KEY,
Category VARCHAR2(50) NOT NULL,
Manufacturer VARCHAR2(50),
FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID));

-- EMPLOYEE Table --
CREATE TABLE EMPLOYEE(
EmployeeID NUMBER(7) NOT NULL,
EmpName VARCHAR2(25) NOT NULL,
EmpPosition VARCHAR2(20) NOT NULL,
EmpSalary NUMBER(5) NOT NULL,
StoreID VARCHAR2(7) NOT NULL,
CONSTRAINT EmployeeID_PK PRIMARY KEY(EmployeeID),
CONSTRAINT StoreID_FK FOREIGN KEY(StoreID) REFERENCES STORE(StoreID));
```

```
-- INVENTORY Table --
CREATE TABLE INVENTORY(
ProductID NUMBER(6) NOT NULL,
StoreID VARCHAR2(7) NOT NULL,
Availability NUMBER(5) NOT NULL,
CONSTRAINT Inventory_PK PRIMARY KEY (ProductID, StoreID),
CONSTRAINT Inventory_FK1 FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID),
CONSTRAINT Inventory_FK2 FOREIGN KEY (StoreID) REFERENCES STORE(StoreID));

-- CUSTOMER Table --
CREATE TABLE CUSTOMER(
CustomerID VARCHAR2(7) NOT NULL,
CustomerName VARCHAR2(30) NOT NULL,
CustomerPhone NUMBER(10) NOT NULL,
Country VARCHAR2(20) NOT NULL,
City VARCHAR2(20) NOT NULL,
Street VARCHAR2(50) NOT NULL,
CONSTRAINT CustomerID_PK PRIMARY KEY(CustomerID));

-- INVOICE Table --
CREATE TABLE INVOICE(
InvoiceID VARCHAR2(7) NOT NULL,
TotalAmount NUMBER(10,2),
InvoiceDate DATE NOT NULL,
CustomerID VARCHAR2(7) NOT NULL,
CONSTRAINT InvoiceID_PK PRIMARY KEY(InvoiceID),
CONSTRAINT CustomerID_FK FOREIGN KEY(CustomerID) REFERENCES CUSTOMER(CustomerID));

-- ORDER Table --
CREATE TABLE ORDER_T(
OrderNumber VARCHAR2(20) NOT NULL,
OrderStatus VARCHAR2(30) NOT NULL,
CustomerID VARCHAR2(7) NOT NULL,
InvoiceID VARCHAR2(7) NOT NULL,
CONSTRAINT OrderNumber_PK PRIMARY KEY(OrderNumber),
CONSTRAINT OrderNumber_FK1 FOREIGN KEY(CustomerID) REFERENCES CUSTOMER(CustomerID),
CONSTRAINT OrderNumber_FK2 FOREIGN KEY(InvoiceID) REFERENCES INVOICE(InvoiceID));

-- ORDERED_ITEM Table --
CREATE TABLE ORDERED_ITEM(
ProductID NUMBER(6) NOT NULL,
OrderNumber VARCHAR2(20) NOT NULL,
Quantity NUMBER(3) NOT NULL,
CONSTRAINT OrderedItem_PK PRIMARY KEY (ProductID, OrderNumber),
CONSTRAINT OrderedItem_FK1 FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID),
CONSTRAINT OrderedItem_FK2 FOREIGN KEY (OrderNumber) REFERENCES ORDER_T(OrderNumber));
```

➢ *Insert Into tables:*

```
--STORE table--
INSERT INTO STORE VALUES ('A538C71', 'beanandleaf@gmail.com', 512345678);
INSERT INTO STORE VALUES ('A538C72', 'beanandleaf@gmail.com', 587453298);
INSERT INTO STORE VALUES ('A538C73', 'beanandleaf@gmail.com', 542642665);
INSERT INTO STORE VALUES ('A538C74', 'beanandleaf@gmail.com', 519846736);
INSERT INTO STORE VALUES ('A538C75', 'beanandleaf@gmail.com', 524685149);
```

```
--EMPLOYEE table--
INSERT INTO EMPLOYEE VALUES(2210937, 'Hadeel Abdulhadi', 'Manager', 10000, 'A538C71');
INSERT INTO EMPLOYEE VALUES(2215434, 'Ahmed Osama', 'Delivery man', 3000, 'A538C71');
INSERT INTO EMPLOYEE VALUES(2228749, 'Mona Hamza', 'Cashier', 5000, 'A538C71');
INSERT INTO EMPLOYEE VALUES(2234567, 'Yasmin Ali', 'Manager', 10000, 'A538C72');
INSERT INTO EMPLOYEE VALUES(2245678, 'Mohamed Ahmed', 'Delivery man', 3000,'A538C72');
INSERT INTO EMPLOYEE VALUES(2256789, 'Fatima Abdulrahman', 'Cashier', 5000, 'A538C72');
INSERT INTO EMPLOYEE VALUES(2267890, 'Ali Ramadan', 'Manager', 10000, 'A538C73');
INSERT INTO EMPLOYEE VALUES(2278901, 'Essam Saleh', 'Delivery man', 3000, 'A538C73');
INSERT INTO EMPLOYEE VALUES(2289012, 'Mahmoud Abdullah', 'Cashier', 5000, 'A538C73');
INSERT INTO EMPLOYEE VALUES(2290123, 'Nour Ali', 'Manager', 10000, 'A538C74');
INSERT INTO EMPLOYEE VALUES(2301234, 'Salem Khalid', 'Delivery man', 3000, 'A538C74');
INSERT INTO EMPLOYEE VALUES(2312345, 'Omar Hassan', 'Cashier', 5000, 'A538C74');
INSERT INTO EMPLOYEE VALUES(2323456, 'Layla Ahmed', 'Manager', 10000, 'A538C75');
INSERT INTO EMPLOYEE VALUES(2334567, 'Khaled Ibrahim', 'Delivery man', 3000, 'A538C75');
INSERT INTO EMPLOYEE VALUES(2345678, 'Mariam Ali', 'Cashier', 5000, 'A538C75');
```

```
--CUSTOMER table--
INSERT INTO CUSTOMER VALUES ('C001', 'Ahmed Ali', 0534567890, 'Saudi Arabia', 'Riyadh','King Fahd Road');
INSERT INTO CUSTOMER VALUES ('C002', 'Fatima abdulkarim', 0576543210, 'Saudi Arabia','Jeddah', 'Prince Sultan Street');
INSERT INTO CUSTOMER VALUES ('C003', 'Abdullah Hassan', 0576543219, 'Saudi Arabia','Dammam', 'Arabian Gulf Street');
INSERT INTO CUSTOMER VALUES ('C004', 'Aseel Ahmed', 0565432109, 'Saudi Arabia','Medina', 'Uthman Ibn Affan Street');
INSERT INTO CUSTOMER VALUES ('C005', 'Saud Mohammed', 0554321098, 'Saudi Arabia','Tabuk', 'King Abdulaziz Road');
INSERT INTO CUSTOMER VALUES ('C006', 'Hadeel Abdulhadi', 0543210987, 'Saudi Arabia','Mecca', 'Al Masjid Al Haram Street');
```

```
--INVOICE table--
INSERT INTO INVOICE VALUES ('I001', 100.00, TO_DATE('2024-04-01', 'YYYY-MM-DD'), 'C001');
INSERT INTO INVOICE VALUES ('I002', 250.00, TO_DATE('2024-04-15', 'YYYY-MM-DD'), 'C002');
INSERT INTO INVOICE VALUES ('I003', 150.00, TO_DATE('2024-05-02', 'YYYY-MM-DD'), 'C003');
INSERT INTO INVOICE VALUES ('I004', 300.00, TO_DATE('2024-05-15', 'YYYY-MM-DD'), 'C004');
INSERT INTO INVOICE VALUES ('I005', 200.00, TO_DATE('2024-05-20', 'YYYY-MM-DD'), 'C005');
INSERT INTO INVOICE VALUES ('I006', 400.00, TO_DATE('2024-05-30', 'YYYY-MM-DD'), 'C006');
```

```
--ORDER table--
INSERT INTO ORDER_T VALUES ('O001', 'Pending', 'C001', 'I001');
INSERT INTO ORDER_T VALUES ('O002', 'Completed', 'C002', 'I002');
INSERT INTO ORDER_T VALUES ('O003', 'Pending', 'C003', 'I003');
INSERT INTO ORDER_T VALUES ('O004', 'Completed', 'C004', 'I004');
INSERT INTO ORDER_T VALUES ('O005', 'Pending', 'C005', 'I005');
INSERT INTO ORDER_T VALUES ('O006', 'Completed', 'C006', 'I006');
```

```
--COFFEE table--
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (101,'Shan', 'Coffee', 19.99);
INSERT INTO COFFEE (ProductID, CoffeeOrigin, RoastLevel) VALUES (101, 'Ethiopia', 'Medium');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (102,'Arcila', 'Coffee', 18.99);
INSERT INTO COFFEE (ProductID, CoffeeOrigin, RoastLevel) VALUES (102, 'Colombia', 'Light');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (103,'Divisadero', 'Coffee', 20.99);
INSERT INTO COFFEE (ProductID, CoffeeOrigin, RoastLevel) VALUES (103, 'Brazil', 'Medium');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (104,'Gaid Blend', 'Coffee' ,22.99);
INSERT INTO COFFEE (ProductID, CoffeeOrigin, RoastLevel) VALUES (104, 'Guatemala', 'Dark');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (105,'BrewBliss', 'Coffee' ,17.99);
INSERT INTO COFFEE (ProductID, CoffeeOrigin, RoastLevel) VALUES (105, 'Costa Rica', 'Light');
```

```
--MATCHA table--
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (201,'SoMatcha', 'Matcha' ,29.99);
INSERT INTO MATCHA (ProductID, MatchaOrigin, MatchaType) VALUES (201, 'Japan','Ceremonial');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (202,'jpDose', 'Matcha' ,27.99);
INSERT INTO MATCHA (ProductID, MatchaOrigin, MatchaType) VALUES (202, 'Japan','Premium');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (203,'Tora', 'Matcha' ,26.99);
INSERT INTO MATCHA (ProductID, MatchaOrigin, MatchaType) VALUES (203, 'Korea','Premium');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (204,'MatchaYa', 'Matcha' ,30.99);
INSERT INTO MATCHA (ProductID, MatchaOrigin, MatchaType) VALUES (204, 'Japan','Ceremonial');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (205,'MaChii', 'Matcha' ,31.99);
INSERT INTO MATCHA (ProductID, MatchaOrigin, MatchaType) VALUES (205, 'Korea', 'Koicha');
```

```
--EQUIPMENT table--
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (301,'Baratza', 'Manufacturer' ,120.00);
INSERT INTO EQUIPMENT (ProductID, Category, Manufacturer) VALUES (301, 'Coffee Grinder','USA');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (302,'Breville', 'Manufacturer' , 599.00);
INSERT INTO EQUIPMENT (ProductID, Category, Manufacturer) VALUES (302, 'EspressoMachine', 'Australia');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (303,'Mr. Coffee', 'Manufacturer' , 85.00);
INSERT INTO EQUIPMENT (ProductID, Category, Manufacturer) VALUES (303, 'Drip CoffeeMaker', 'USA');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (304,'Bodum', 'Manufacturer' , 35.00);
INSERT INTO EQUIPMENT (ProductID, Category, Manufacturer) VALUES (304, 'French Press','Denmark');
INSERT INTO PRODUCT (ProductID, ProductName, ProductType, ProductPrice) VALUES (305,'Chemex', 'Manufacturer' , 45.00);
INSERT INTO EQUIPMENT (ProductID, Category, Manufacturer) VALUES (305, 'Pour Over CoffeeMaker', 'USA');
```

```
--ORDERED_ITEM table--
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (101, 'O001', 3);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (102, 'O001', 5);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (302, 'O001', 1);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (204, 'O002', 4);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (105, 'O002', 2);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (104, 'O003', 1);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (101, 'O003', 6);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (303, 'O003', 2);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (201, 'O003', 5);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (105, 'O004', 4);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (102, 'O004', 1);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (202, 'O004', 2);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (304, 'O004', 3);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (103, 'O004', 7);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (104, 'O005', 2);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (205, 'O005', 1);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (301, 'O005', 4);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (101, 'O006', 4);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (203, 'O006', 3);
INSERT INTO ORDERED_ITEM (ProductID, OrderNumber, Quantity) VALUES (305, 'O006', 1);
```

```sql
--INVENTORY table (stocks of branch 1)--
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (101, 'A538C71', 50);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (102, 'A538C71', 25);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (103, 'A538C71', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (104, 'A538C71', 15);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (105, 'A538C71', 30);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (201, 'A538C71', 70);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (202, 'A538C71', 65);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (203, 'A538C71', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (204, 'A538C71', 12);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (205, 'A538C71', 78);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (301, 'A538C71', 19);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (302, 'A538C71', 45);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (303, 'A538C71', 34);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (304, 'A538C71', 39);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (305, 'A538C71', 0);
```

```sql
--INVENTORY table (stocks of branch 2)--
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (101, 'A538C72', 42);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (102, 'A538C72', 17);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (103, 'A538C72', 33);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (104, 'A538C72', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (105, 'A538C72', 28);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (201, 'A538C72', 63);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (202, 'A538C72', 82);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (203, 'A538C72', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (204, 'A538C72', 91);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (205, 'A538C72', 48);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (301, 'A538C72', 76);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (302, 'A538C72', 22);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (303, 'A538C72', 43);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (304, 'A538C72', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (305, 'A538C72', 34);
```

```sql
--INVENTORY table (stocks of branch 3)--
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (101, 'A538C73', 60);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (102, 'A538C73', 32);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (103, 'A538C73', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (104, 'A538C73', 20);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (105, 'A538C73', 42);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (201, 'A538C73', 80);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (202, 'A538C73', 71);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (203, 'A538C73', 28);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (204, 'A538C73', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (205, 'A538C73', 39);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (301, 'A538C73', 50);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (302, 'A538C73', 35);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (303, 'A538C73', 45);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (304, 'A538C73', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (305, 'A538C73', 27);
```

```sql
--INVENTORY table (stocks of branch 4)--
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (101, 'A538C74', 55);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (102, 'A538C74', 29);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (103, 'A538C74', 18);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (104, 'A538C74', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (105, 'A538C74', 37);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (201, 'A538C74', 64);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (202, 'A538C74', 77);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (203, 'A538C74', 41);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (204, 'A538C74', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (205, 'A538C74', 52);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (301, 'A538C74', 42);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (302, 'A538C74', 29);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (303, 'A538C74', 48);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (304, 'A538C74', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (305, 'A538C74', 31);
```

```sql
--INVENTORY table (stocks of branch 5)--
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (101, 'A538C75', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (102, 'A538C75', 21);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (103, 'A538C75', 12);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (104, 'A538C75', 18);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (105, 'A538C75', 31);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (201, 'A538C75', 59);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (202, 'A538C75', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (203, 'A538C75', 43);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (204, 'A538C75', 88);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (205, 'A538C75', 52);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (301, 'A538C75', 41);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (302, 'A538C75', 29);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (303, 'A538C75', 0);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (304, 'A538C75', 56);
INSERT INTO INVENTORY (ProductID, StoreID, Availability) VALUES (305, 'A538C75', 31);
```

➢ *Queries:*

- ***Using WHERE clause:***

With this query, the store can efficiently retrieve the names and prices of all products classified under the "Coffee" type from the database

```
--This query retrieves the names and prices of all products that belong to the 'Coffee' type,using WHERE clause.
SELECT ProductName, ProductPrice
FROM PRODUCT
WHERE ProductType = 'Coffee';
```

PRODUCTNAME	PRODUCTPRICE
Shan	19.99
Arcila	18.99
Divisadero	20.99
Gaid Blend	22.99
BrewBliss	17.99

- ***Using ORDER BY clause:***

With this query, the store can quickly identify its top five most expensive products, helping management adjust pricing strategies, optimize inventory, and enhance marketing efforts for these high-value items.

```
--Retrieve the products from the PRODUCT table ordered by their prices in descending order and display only the top 5 expensive products.
--using ORDER BY Clause.
SELECT ProductName, ProductPrice
FROM PRODUCT
ORDER BY ProductPrice DESC
FETCH FIRST 5 ROWS ONLY;
```

PRODUCTNAME	PRODUCTPRICE
Breville	599
Baratza	120
Mr. Coffee	85
Chemex	45
Bodum	35

- ***Using GROUP BY clause and AGGREGATE FUNCTION:***

With this query, the store can analyze inventory distribution and pricing, aiding in decisions on pricing strategies and inventory management to meet market demands and maximize profits

```
--Retrieve the product type, count of products in each type, and the maximum price of products for each type.
--using GROUP BY and aggregate function.
SELECT ProductType, COUNT(*) AS ProductCount, MAX(ProductPrice) AS MaxPrice
FROM PRODUCT
GROUP BY ProductType;
```

PRODUCTTYPE	PRODUCTCOUNT	MAXPRICE
Matcha	5	31.99
Manufacturer	5	599
Coffee	5	22.99

- ***Using JOIN , SUBQUERY and CASE statement :***

With this query, the store can assess salary fairness, aiding in HR decisions to ensure equitable pay, enhance employee satisfaction, and manage payroll budgeting efficiently

```
--Retrieve the names of employees along with a column indicating whether their salary is above the average salary of all employees.
--using JOIN, subquery, and CASE statement.
SELECT EmpName,
       CASE WHEN EmpSalary > (SELECT AVG(EmpSalary) FROM EMPLOYEE) THEN 'Above Average' ELSE 'Below Average' END AS SalaryStatus
FROM EMPLOYEE;
```

EMPNAME	SALARYSTATUS
Hadeel Abdulhadi	Above Average
Ahmed Osama	Below Average
Mona Hamza	Below Average
Yasmin Ali	Above Average
Mohamed Ahmed	Below Average
Fatima Abdulrahman	Below Average
Ali Ramadan	Above Average
Essam Saleh	Below Average
Mahmoud Abdullah	Below Average
Nour Ali	Above Average
Salem Khalid	Below Average
Omar Hassan	Below Average
Layla Ahmed	Above Average

- ***Using SUBQUERY and WHERE clause:***

With this query, the store can quickly identify out-of-stock products, streamline inventory replenishment, and improve customer service by providing accurate availability information

```
-- This query retrieve the product names and prices of products that are currently out of stock (have an availability of 0) in any store.
-- Using Subquery and WHERE clause.
SELECT ProductName, ProductPrice
FROM PRODUCT
WHERE ProductID IN (
    SELECT ProductID
    FROM INVENTORY
    WHERE Availability = 0
);
```

PRODUCTNAME	PRODUCTPRICE
Divisadero	20.99
Tora	26.99
Chemex	45
Gaid Blend	22.99
Bodum	35
MatchaYa	30.99
Shan	19.99
jpDose	27.99
Mr. Coffee	85

- ***Using JOIN, GROUP BY , HAVING and AGGREGATE FUNCTION:***

With this query, the store can analyze multi-item orders to understand customer purchasing patterns and adjust inventory and marketing strategies accordingly

```
--This query groups orders by order number and calculates the number of items and total quantity for each order, only including orders with more than one item.
--Using JOIN, GROUP BY, and Aggregate functions
SELECT O.OrderNumber, COUNT(*) AS NumItems, SUM(OI.Quantity) AS TotalQuantity
FROM ORDER_T O
JOIN ORDERED_ITEM OI ON O.OrderNumber = OI.OrderNumber
GROUP BY O.OrderNumber
HAVING COUNT(*) > 1;
```

ORDERNUMBER	NUMITEMS	TOTALQUANTITY
O002	2	6
O004	5	17
O001	3	9
O003	4	14
O006	3	8
O005	3	7

- ***Using JOIN and WHERE clause:***

With this query, the store can efficiently follow up with customers who purchased coffee products, gathering feedback to enhance service quality and customer satisfaction

```
--Retrieve data using a join query with multiple tables and conditions.
SELECT C.CustomerName, I.InvoiceID, O.OrderStatus
FROM CUSTOMER C
JOIN INVOICE I ON C.CustomerID = I.CustomerID
JOIN ORDER_T O ON I.InvoiceID = O.InvoiceID
JOIN ORDERED_ITEM OI ON O.OrderNumber = OI.OrderNumber
JOIN PRODUCT P ON OI.ProductID = P.ProductID
WHERE P.ProductType = 'Coffee'
AND O.OrderStatus = 'Completed';
```

CUSTOMERNAME	INVOICEID	ORDERSTATUS
Fatima abdulkarim	I002	Completed
Aseel Ahmed	I004	Completed
Aseel Ahmed	I004	Completed
Aseel Ahmed	I004	Completed
Hadeel Abdulhadi	I006	Completed

With this query, the store can identify pending orders, enabling proactive follow-up to resolve any issues and expedite order completion. This targeted approach helps improve customer service by ensuring timely updates and resolutions for delayed orders

```
--Retrieve data using the JOIN clause and multiple conditions.
SELECT C.CustomerName, I.InvoiceID, O.OrderStatus
FROM CUSTOMER C
JOIN INVOICE I ON C.CustomerID = I.CustomerID
JOIN ORDER_T O ON I.InvoiceID = O.InvoiceID
JOIN ORDERED_ITEM OI ON O.OrderNumber = OI.OrderNumber
JOIN PRODUCT P ON OI.ProductID = P.ProductID
WHERE C.Country = 'Saudi Arabia'
AND P.ProductType = 'Coffee'
AND O.OrderStatus = 'Pending';
```

CUSTOMERNAME	INVOICEID	ORDERSTATUS
Ahmed Ali	I001	Pending
Ahmed Ali	I001	Pending
Abdullah Hassan	I003	Pending
Abdullah Hassan	I003	Pending
Saud Mohammed	I005	Pending

- ***Using JOIN, GROUP BY, ORDER BY clauses and AGGREGATE FUNCTION :***

With this query, the store can identify the three highest-selling products, providing insights into customer preferences and guiding inventory restocking decisions to ensure popular items are always available

```
-- Identify the top 3 best-selling products based on quantity sold
SELECT p.ProductName, p.ProductType, SUM(oi.Quantity) AS TotalSold
FROM PRODUCT p
JOIN ORDERED_ITEM oi ON p.ProductID = oi.ProductID
GROUP BY p.ProductName, p.ProductType
ORDER BY TotalSold DESC
FETCH FIRST 3 ROWS ONLY;
```

PRODUCTNAME	PRODUCTTYPE	TOTALSOLD
Shan	Coffee	13
Divisadero	Coffee	7
Arcila	Coffee	6

- ***Using JOIN , GROUP BY and ORDER BY clauses and AGGREGATE FUNCTION :***

With this query the store can Identify sales trends and seasonal variations in product popularity, helping in strategic product planning and promotional activities

```
-- Track product sales trends over time
SELECT p.ProductName, TO_CHAR(iv.InvoiceDate, 'YYYY-MM') AS Month, SUM(oi.Quantity) AS TotalSold
FROM PRODUCT p
JOIN ORDERED_ITEM oi ON p.ProductID = oi.ProductID
JOIN ORDER_T o ON oi.OrderNumber = o.OrderNumber
JOIN INVOICE iv ON o.InvoiceID = iv.InvoiceID
GROUP BY p.ProductName, TO_CHAR(iv.InvoiceDate, 'YYYY-MM')
ORDER BY p.ProductName, Month;
```

PRODUCTNAME	MONTH	TOTALSOLD
Arcila	2024-04	5
Arcila	2024-05	1
Baratza	2024-05	4
Bodum	2024-05	3
Breville	2024-04	1
BrewBliss	2024-04	2
BrewBliss	2024-05	4
Chemex	2024-05	1

- ***Using WHERE and ORDER BY clauses:***

With this query, the store can efficiently list all Matcha products priced below $30, allowing for strategic promotion of affordable options to attract price-sensitive customers

```sql
--Retrieve data using multiple conditions in the WHERE clause and the ORDER BY clause.
SELECT *
FROM PRODUCT
WHERE ProductType = 'Matcha' AND ProductPrice < 30
ORDER BY ProductPrice DESC;
```

PRODUCTID	PRODUCTNAME	PRODUCTTYPE	PRODUCTPRICE
201	SoMatcha	Matcha	29.99
202	jpDose	Matcha	27.99
203	Tora	Matcha	26.99

- ***Using JOIN, WHERE and ORDER BY clauses:***

With this query it will help the stores inventory management,ensure stock levels are adequate to meet demand, preventing stockouts

```sql
-- Display products with low inventory levels in any store, showing only store ID and product name
SELECT i.StoreID,  p.ProductName
FROM PRODUCT p
JOIN (SELECT ProductID, StoreID
     FROM INVENTORY
     WHERE Availability < 10) i
ON  p.ProductID = i.ProductID
ORDER BY  i.StoreID;
```

STOREID	PRODUCTNAME
A538C71	Divisadero
A538C71	Tora
A538C71	Chemex
A538C72	Gaid Blend
A538C72	Tora
A538C72	Bodum
A538C73	Divisadero

➢ *procedures :*

1) *With this procedure, the store can accurately manage inventory by updating the availability of an inventory, reducing it by one each time a transaction is processed. This systematic approach ensures that inventory records are kept up-to-date, preventing discrepancies and enabling efficient stock management*

```
-- this procedure control number of products in inventories of each store, by decreasing 1 as new product is sold
CREATE OR REPLACE PROCEDURE update_availability(
  v_PID INVENTORY.ProductID%TYPE,
  v_SID INVENTORY.StoreID%TYPE
) AS
  v_oldAvailabilty NUMBER;
BEGIN
  SELECT Availability INTO v_oldAvailabilty
  FROM INVENTORY
  WHERE ProductID = v_PID AND StoreID = v_SID;

  UPDATE INVENTORY
  SET Availability = v_oldAvailabilty - 1
  WHERE ProductID = v_PID AND StoreID = v_SID;
END;
```

```
EXEC update_availability(102, 'A538C75');
SELECT * FROM INVENTORY
WHERE StoreID = 'A538C75';
```

PRODUCTID	STOREID	AVAILABILITY
101	A538C75	0
102	A538C75	20
103	A538C75	12

2) *With this procedure, the store can efficiently update employee salaries, ensuring fair compensation based on performance or role changes, thereby maintaining staff satisfaction and loyalty*

```
--this procedure is designed to update the salary of an employee in the EMPLOYEE table based on the provided EmployeeID and newSalary values.
CREATE OR REPLACE PROCEDURE update_employee_salary(
v_empID EMPLOYEE.EmployeeID%TYPE,
v_newSalary EMPLOYEE.EmpSalary%Type
) AS
BEGIN
UPDATE EMPLOYEE
SET EmpSalary = v_newSalary
WHERE EmployeeID = v_empID;
END;
```

```
EXEC update_employee_salary (2210937, 50000);
SELECT * FROM EMPLOYEE;
```

Statement processed.

EMPLOYEEID	EMPNAME	EMPPOSITION	EMPSALARY	STOREID
2210937	Hadeel Abdulhadi	Manager	50000	A538C71
2215434	Ahmed Osama	Delivery man	3000	A538C71
2228749	Mona Hamza	Cashier	5000	A538C71

3) *With this procedure, the store can automatically calculate and update invoice totals based on ordered item quantities and prices, ensuring financial accuracy and enhancing operational efficiency*

```
--this procedure is designed to calculate the total amount for an invoice based on the quantities and prices of the ordered items,
--and then update the totalAmount column of the corresponding invoice in the INVOICE table.
CREATE OR REPLACE PROCEDURE calculate_invoice_amount(
v_InvoiceID INVOICE.InvoiceID%TYPE
) AS
v_TotalAmount INVOICE.totalAmount%TYPE := 0;
BEGIN
SELECT SUM(ORDERED_ITEM.Quantity * PRODUCT.ProductPrice)
INTO v_TotalAmount
FROM ORDER_T
JOIN ORDERED_ITEM ON ORDER_T.OrderNumber = ORDERED_ITEM.OrderNumber
JOIN PRODUCT ON ORDERED_ITEM.ProductID = PRODUCT.ProductID
WHERE ORDER_T.InvoiceID = v_InvoiceID;
IF v_TotalAmount IS NULL THEN
DBMS_OUTPUT.PUT_LINE('Invoice not found.');
ELSE
UPDATE INVOICE
SET totalAmount = v_TotalAmount
WHERE InvoiceID = v_InvoiceID;
DBMS_OUTPUT.PUT_LINE('Total amount updated for InvoiceID: ' || v_InvoiceID);
END IF;
END;
```

```
1  EXEC calculate_invoice_amount('I001');
2  SELECT * FROM INVOICE;
```

INVOICEID	TOTALAMOUNT	INVOICEDATE	CUSTOMERID
I001	733.93	01-APR-24	C001
I002	-	15-APR-24	C002
I003	-	02-MAY-24	C003
I004	-	15-MAY-24	C004
I005	-	20-MAY-24	C005
I006	-	30-MAY-24	C006

4) *With this procedure, the store can quickly check product availability, ensuring accurate inventory management and prompt customer communication about stock status*

```
1  --this procedure is designed to check the availability of a product in a specific store based on the provided ProductID and StoreID,
2  --and display an appropriate message regarding the availability status.
3  CREATE OR REPLACE PROCEDURE check_availability(
4  v_ProductID INVENTORY.ProductID%TYPE,
5  v_StoreID INVENTORY.StoreID%TYPE
6  ) AS
7  v_availability INVENTORY.Availability%TYPE;
8  BEGIN
9  SELECT Availability
10 INTO v_availability
11 FROM INVENTORY
12 WHERE ProductID = v_ProductID AND StoreID = v_StoreID;
13 IF v_availability = 0 THEN
14 DBMS_OUTPUT.PUT_LINE('Sorry, this product is out of stock');
15 ELSE
16 DBMS_OUTPUT.PUT_LINE('There are ' ||v_availability|| ' pieces in stock');
17 END IF;
18 END;
```

```
1  EXEC check_availability(101, 'A538C71');
2  EXEC check_availability(103, 'A538C71');
```

```
Statement processed.
There are 50 pieces in stock


Statement processed.
Sorry, this product is out of stock
```

➢ *Cursor procedure :*

With this procedure, the store can efficiently retrieve and display detailed information about employees at a specific store location, enabling effective management and oversight of staff. This enhanced visibility is crucial for strategic human resource planning and fostering a motivated workplace environment

```
--this cursor procedure is designed to retrieve and display employee information for a specific store.
CREATE OR REPLACE PROCEDURE PrintStoreEmployees(p_StoreID VARCHAR2)
AS
  CURSOR c_Employees (c_StoreID VARCHAR2) IS
    SELECT e.EmployeeID, e.EmpName, e.EmpPosition, e.EmpSalary
    FROM EMPLOYEE e
    WHERE e.StoreID = c_StoreID;

  v_EmployeeID EMPLOYEE.EmployeeID%TYPE;
  v_EmpName EMPLOYEE.EmpName%TYPE;
  v_EmpPosition EMPLOYEE.EmpPosition%TYPE;
  v_EmpSalary EMPLOYEE.EmpSalary%TYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Employees for Store ID: ' || p_StoreID);
  DBMS_OUTPUT.PUT_LINE('-------------------------------------');

  FOR emp IN c_Employees(p_StoreID) LOOP
    v_EmployeeID := emp.EmployeeID;
    v_EmpName := emp.EmpName;
    v_EmpPosition := emp.EmpPosition;
    v_EmpSalary := emp.EmpSalary;

    DBMS_OUTPUT.PUT_LINE('Employee ID     | ' || v_EmployeeID);
    DBMS_OUTPUT.PUT_LINE('Employee Name   | ' || v_EmpName);
    DBMS_OUTPUT.PUT_LINE('Position        | ' || v_EmpPosition);
    DBMS_OUTPUT.PUT_LINE('Salary          | ' || v_EmpSalary);
    DBMS_OUTPUT.PUT_LINE('-------------------------------------');
  END LOOP;
END;
```

```
1  EXEC printStoreEmployees('A538C71');
```

```
Statement processed.
Employees for Store ID: A538C71
-------------------------------------
Employee ID     | 2210937
Employee Name   | Hadeel Abdulhadi
Position        | Manager
Salary          | 50000
-------------------------------------
Employee ID     | 2215434
Employee Name   | Ahmed Osama
Position        | Delivery man
Salary          | 3000
-------------------------------------
Employee ID     | 2228749
Employee Name   | Mona Hamza
Position        | Cashier
Salary          | 5000
-------------------------------------
```

Table of Tasks

Stages	*Tasks*	*Names*
Stage 1	• ***Introduction*** • ***Problem Description*** • ***Information Needs*** • ***Entities*** • ***ER-Diagram*** *-Conceptual* *-Logical*	***All Members***
Stage 2	• ***Logical Modeling & Normalization*** - *Relation Schema* - *Normalization* • ***Functional Dependencies***	***All Members***
Stage 3	• ***Implementation of the relational DB*** - *Creating Tables* - *Inserting Data in Tables* - *Queries* - *Stored Procedures* - *Cursor procedure*	***All Members***
Final Submission	• ***Report Review & Editing***	***All Members***