



CMPG311 PROJECT

MP Thabane 38697505

MO Mbenyane 45090955

L Dube 34485775

DE Masango 41833325

Table of Contents

Analyse company situation	1
Company Objectives	1
Company Operations	1
Company Organizational Structure.....	2
Problems:	3
Constraints:.....	3
DATABASE SYSTEM SPECIFICATION	4
4.3 Scope	6
4.4 Boundaries	6
4.4.1 Software	6
4.4.2 Hardware.....	7
4.4.3 Personnel	7
Phase 2.....	8
Business Rules for the Supermarket	8
Entity RD	10
Phase 3.....	13
POPULATING THE DATABASES	21
CREATING VIEWS	24
CREATING INDEXES	26

Analyse company situation

Company Objectives

Our mission is to provide high-quality groceries, fresh produce, and household essentials at affordable prices while ensuring excellent customer service and efficient business operations. We aim to:

- Maintain real-time inventory tracking to prevent stock shortages and overstocking.
- Enhance the customer shopping experience through efficient checkout systems and digital payment integration.
- Optimize supplier relationships and logistics for timely restocking.

Company Operations

Point-of-Sale (POS) System

- Registers and processes customer purchases.

- Accepts multiple payment methods (cash, credit/debit, digital wallets).

Inventory Management

- Tracks stock levels in real time.
- Generates automatic restocking alerts.

Supplier & Order Management

- Maintains supplier records and order history.
- Manages purchase orders and delivery schedules.
- Tracks incoming shipments and quality control checks.

Employee Management

- Records staff shifts and attendance.
- Tracks employee roles and responsibilities.

Financial & Reporting System

- Records daily sales and revenue reports.
- Tracks expenses, supplier payments, and profit margins.

Company Organizational Structure

Executive Management

- CEO / General Manager - Oversees overall business operations and strategy.
- Finance Manager - Handles financial planning, accounting, and budgeting.

Operations Management

- Store Manager - Manages daily store operations and staff.
- Inventory Manager - Ensures efficient stock levels and supplier coordination.

Customer Service & Sales

- Cashiers - Operate POS systems and assist customers at checkout.
- Floor Staff - Organize and stock shelves, assist customers.

Supply Chain & Logistics

- Procurement Officer - Manages supplier contracts and orders.
- Warehouse Staff - Handles storage and distribution of goods.

Marketing & IT Support

- Marketing Specialist - Develops promotions and customer engagement strategies.
- IT Support - Maintains the operational database and POS system.

Problems:

1. Inventory Management:

- a. Stockouts: Popular products may run out at the store, which would upset customers.
- b. Overstocking: High storage expenses and product waste result from having too much inventory of less popular things.
- c. Product Expiration: Food and other perishable items can have a short shelf life, which can result in waste if they are not sold in time.

2. Pricing Issues:

- a. Price Accuracy: Customers may notice differences between the prices on shelves and the prices at checkout if pricing isn't updated accurately in the database.

3. Discount Management:

- a. It can be difficult to manage discounts for loyalty programs, large purchases, or promotions, which could result in computation errors.

4. Customer Data Management:

- a. Data Privacy: To prevent data breaches, handling sensitive consumer data—like purchase history or personal information—requires strong security.
- b. Personalization: Tracking consumer preferences and customising promotions for everyone can be challenging.

5. Order and Supply Chain Management:

- a. Supplier Problems: Store operations may be hampered by suppliers' inaccurate or delayed stock deliveries.
- b. Demand Forecasting: Improper demand projections might result in either a scarcity or an excess of goods.

6. Sales Reporting:

- a. Real-Time Data: Precise real-time data is essential for inventories and sales. Making decisions can be impacted by inaccurate reports.
- b. Manual Errors: Inaccuracies in data entry might compromise the accuracy and dependability of the database.

Constraints:

- a. Budgetary Restrictions:

The store may not have the funds to install complex database systems, which would limit the range of tools and technology available.

b. Integration of Systems:

It may be difficult to integrate the new database system with old software, like as inventory management software and Point of Sale (POS) systems, if they are out-of-date or incompatible.

c. Employee Education:

Employee resistance to change may result from the need for training on how to utilise the new system, which requires time and money.

d. Adherence to Regulations:

Regulations pertaining to product safety (monitoring expiration dates, etc.) and data protection (such as GDPR for customer data) must be followed by the supermarket, which may limit data access and storage.

e. Scalability

Future growth in the quantity of products and customers must be accommodated by the system's scalability. However, the budget or hardware that is available may limit this scalability.

f. Accessibility for Users:

Various user types (such as cashiers, managers, and stock controllers) must be able to access the database; however, security restrictions may restrict access to important information.

DATABASE SYSTEM SPECIFICATION

- Objectives to solve problems identified

1. Inventory Management:

- To avoid stockouts, put continuous stock monitoring into place.
- To reduce overstocking, use demand forecasting to optimize inventory levels.
- Reduce product waste by keeping track of expiration dates and setting warnings.

2. Pricing Issues:

- Make sure that prices are consistent throughout the database, shelves, and checkout process.

- Properly reflect discounts and changes by automating price updates.

3. Discount Management:

- Make sure loyalty programs and discounts for large purchases are administered correctly
- Put in place a centralized discount system that does calculations automatically.

4. Customer Data Management:

- Use access control and encryption to improve data security.
- AI can be used to customize recommendations and promotions according to consumer purchasing patterns.

5. Order and Supply Chain Management:

- Automate supplier order tracking and delay alerts.
- . Increase the accuracy of demand forecasting by utilizing historical data.

6. Sales Reporting:

- Implement real-time data monitoring for sales and inventory.
- Minimize human errors by automating data entry and validation.

- Information that the company requires from the database

1. Customer Information

- Name, contact details, address
- Purchase history & preferences
- Customer feedback & support tickets
- Subscription/membership details
- Sales & Revenue Data
- Sales transactions (date, amount, payment method)
- Sales trends & forecasting
- Discounts & promotions impact
- Profit & loss reports

2. Inventory & Supply Chain Data

- Stock levels & product availability
- Restocking alerts & supplier details

- Delivery tracking & logistics
- Production status (for manufacturing companies)

3. Employee Data

- Attendance & work hours
- Payroll details & tax records
- Performance reviews & KPIs
- Training & certification records

4. Financial Information

- Expenses & budgeting data
- Tax information & compliance reports
- Accounts receivable & payable
- Loan & investment records

5. Business Operations & Performance Metrics

- Workflow tracking & process efficiency
- Project management progress reports
- Market research & competitor analysis
- Regulatory compliance reports

By collecting and analyzing this data, the company can make informed decisions, improve efficiency, and enhance customer satisfaction.

4.3 Scope

Handling item transactions, keeping track of sales, managing stock, and interacting with customers manually in a supermarket can be slow and lead to errors. This project is designed to fix these issues by automating key tasks. Record-keeping will become more accurate, the checkout process will be faster, and inventory control will be more efficient. It will also simplify customer management. By automating these processes, the supermarket will be able to offer a more tailored shopping experience, keeping track of customer preferences and purchase histories to provide better service

4.4 Boundaries

The supermarket faces several *external limits* that must be managed effectively to ensure successful implementation. With a total budget of R80 000, the project must carefully balance budget, personnel, hardware, and software constraints while meeting operational requirements.

4.4.1 Software

- Software Updates: 3 licenses (R70/hour)

- System Maintenance: 4 licenses (R250/hour)
- Technical Support: 5 licenses (R150/hour)
- Security Software: 4 licenses (R200/hour)
- Total Software budget: R 27,600 (assume each software license is used for 10 hours)

4.4.2 Hardware

- Power Connectors: R800
- Computers: R12,000
- Storage Devices: R2,000
- Networking Equipment:
 - Hubs: R1,500
 - Routers: R250
 - Gateways': R15,000
- Total hardware budget: R31,550

4.4.3 Personnel

- Project Manager: R300/hour
- Business Analyst: R290/hour
- Business Consultant: R320/hour
- Database designer: R375/hour
- System Analyst: (R380/hour)
- Software Engineer: (R310/hour)
- Total Personnel budget: R19,750 (assuming they work 10 hours)

Phase 2

Business Rules for the Supermarket

1. Inventory Management

- Stock levels must be tracked in real-time to prevent shortages and overstocking.
- Expired products must be removed automatically and flagged for review.
- Automatic alerts should notify the inventory manager when stock reaches a predefined threshold.
- Demand forecasting must be used to optimize restocking schedules.

2. Pricing & Discounts

- Prices must be consistent across the database, shelf labels, and checkout systems.
- Discounts should be applied automatically based on promotions, bulk purchases, or loyalty programs.
- Price updates must be logged and reviewed periodically for accuracy.

3. Point-of-Sale (POS) Transactions

- Multiple payment methods (cash, credit/debit, digital wallets) must be supported.
- Every transaction must generate a digital receipt stored in the database.
- Failed transactions must be recorded, and system errors reported immediately.

4. Customer Data Management

- Customer personal information must be encrypted and stored securely.
- Customers should have the option to opt-in for personalized promotions.
- Customer purchase history must be maintained for trend analysis and marketing.

5. Supplier & Order Management

- Supplier details and contracts must be maintained in the system.
- Orders must be tracked from placement to delivery, with status updates.
- Late deliveries or quality issues must be flagged for review.

6. Employee Management

- Employees must clock in and out, with working hours tracked.
- Payroll calculations must reflect hourly wages, overtime, and bonuses.

- Access to financial and inventory data must be restricted based on roles.

7. Financial & Reporting System

- All sales, expenses, and supplier payments must be logged in real-time.
- Reports must be generated automatically for sales trends and profit margins.
- Unauthorized financial modifications must trigger an alert.

8. Regulatory Compliance

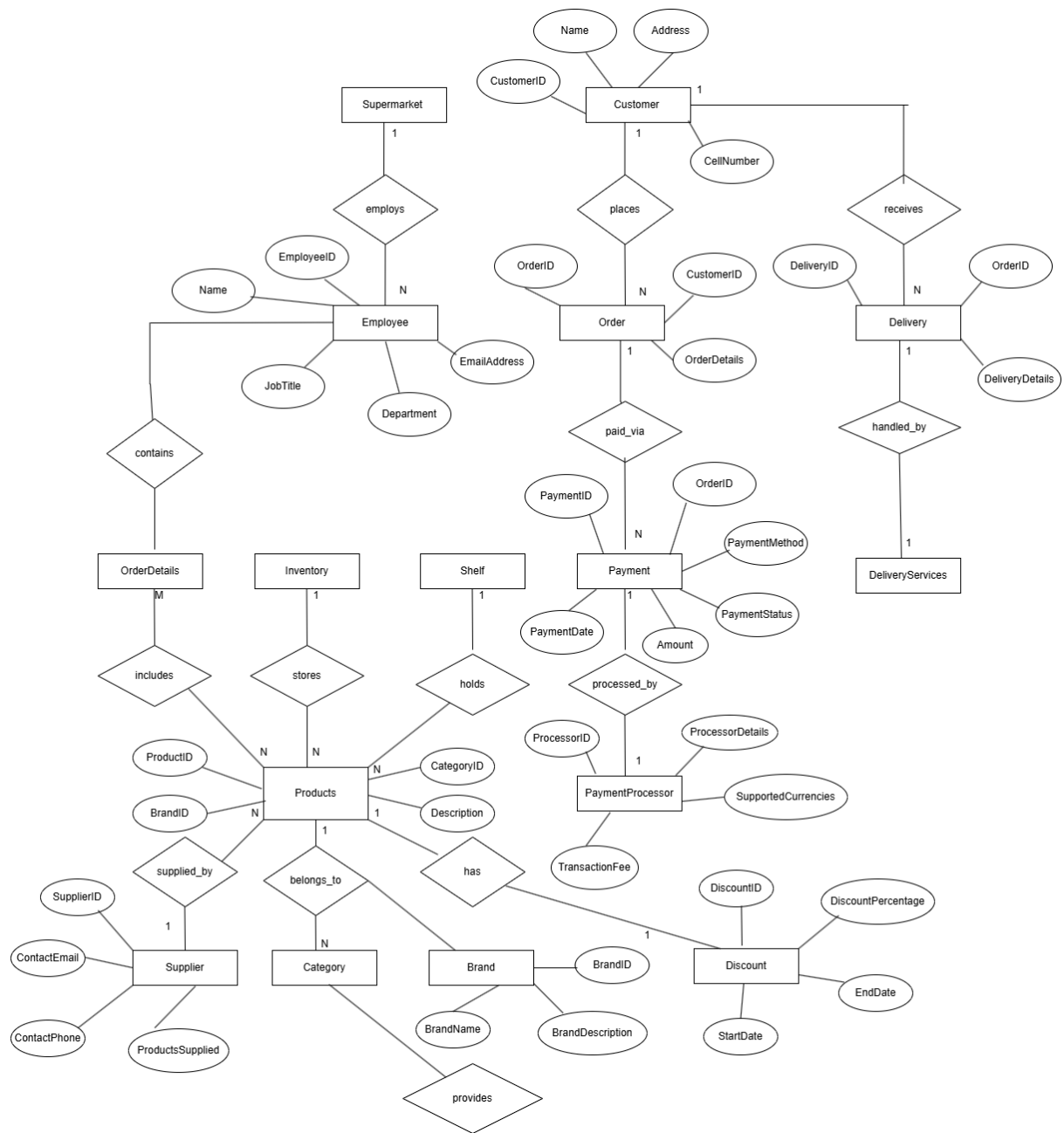
- Data protection laws (such as GDPR) must be followed in storing customer and employee data.
- Food safety regulations must be enforced, including monitoring expiration dates.
- Tax compliance reports must be generated automatically for auditing.

9. System Access & Security

- Different user roles (cashier, manager, stock controller) must have predefined access levels.
- All changes to inventory, pricing, and financial records must be logged.

Security updates must be performed regularly to prevent data breaches.

Entity RD



Strong Entities

- Supermarket
- Customer
- Employee
- DeliveryServices
- Product
- Category
- Brand
- Shelf
- Supplier,
- PaymentProcessor

Weak Entities

- Order
- Delivery
- OrderDetails
- Payment
- Inventory
- Discount

Weak Relationships

- Order to Customer
- Product to Discount
- Inventory to Product
- Supermarket to Inventory

Logical Model

SUPERMARKET(SupermarketID (PK), Name , location)

CUSTOER(CustomerID(PK) , Name, Address , CellNumber)

EMPLOYEE (EmployeeID(PK) , Name, JobTitle, Department , EmailAddress)

ORDER (OrderID(PK) , CustomerID(FK) , OrderDetails)

DELIVERY(DeliveryID(PK) , OrderID(FK) , DeliveryDetails)

PAYMENT(PaymentID(PK),PaymentDate, Amount , PaymentStatus , PaymentMethod, OrderID(FK))

PRODUCT(ProductID(PK) ,Description,CategoryID((FK) , BrandID(FK))

SUPPLIER(SupplierID(PK) ,ContactEmail ,ContactPhone ,ProductsSupplied(PK , FK))

BRAND(BrandID(PK) , BrandDescription , BrandName)

DISCOUNT (DiscountID(PK) , DiscountPercentage ,StartDate,EndDate)

PAYMENTPROCESSOR(ProcessorID(PK) , ProcessorDetails , SupportedCurrencies ,
TransactionFee)

SHELF(ShelfID(PK) , ShelfNumber , Aisle , SupermarketID(FK))

INVENTORY(InventoryID(PK) , ProductID(FK) , SupermarketID(FK) , QuantityAvailable ,
LastRestocked , ExpiryDate)

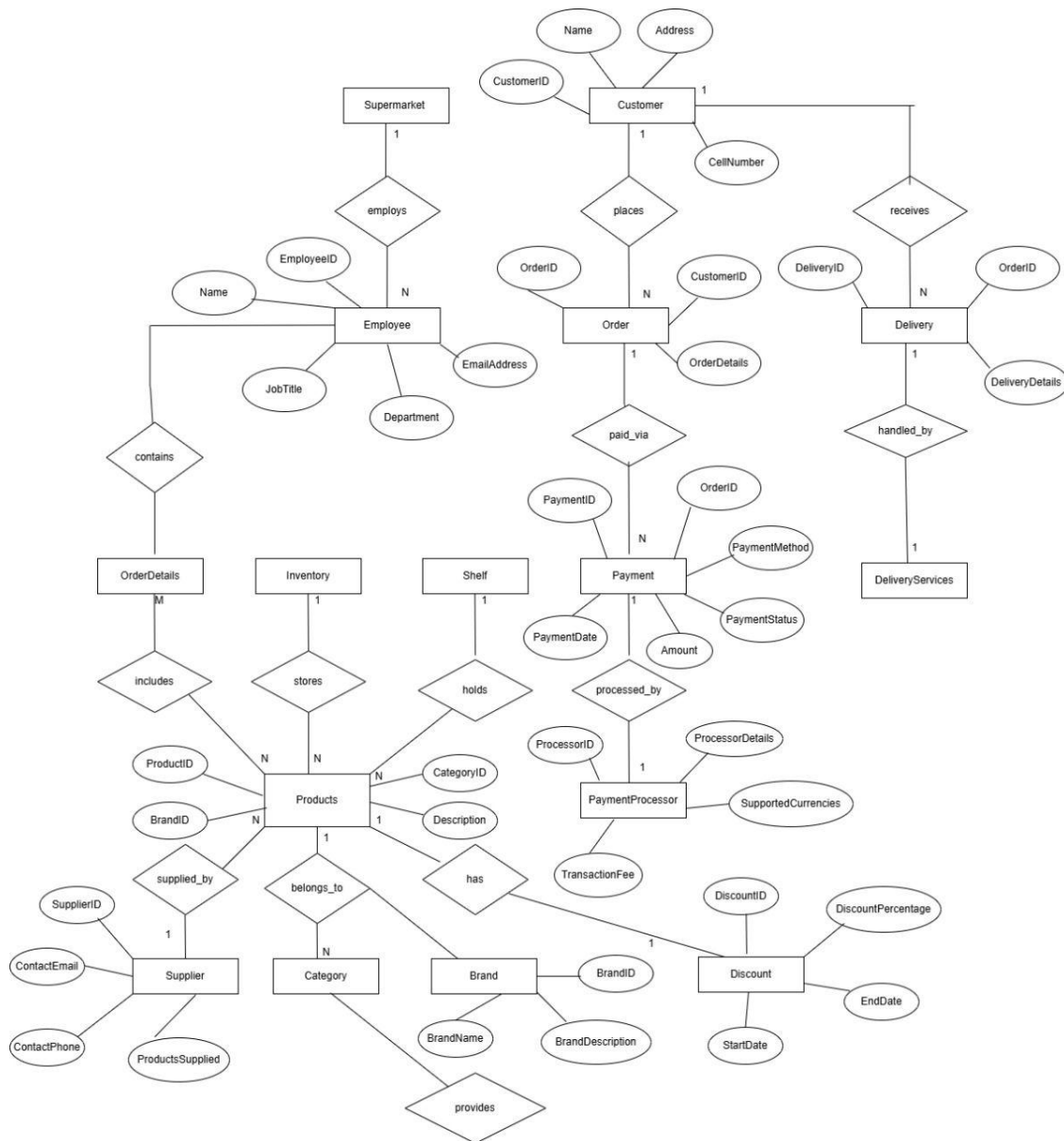
ORDERDETAILS(OrderDetailID(PK) , OrderID(FK) , ProductID , Quantity , Subtotal)

DELIVERYSERVICE(ServiceID(PK) , ProviderName , ContactInfo)

CATEGORY(CategoryID(PK) , CategoryName , Description)

Phase 3

1. Database Design/Layout



2. Database Objects

The first Supermarket Management System database will be created in this phase, along with all of the connections between the different tables. The default records that will be used to illustrate our queries in section 3 will be added to the database after it has been established.

2.1. Removing All Tables to Allow the Creation of New Tables

All current tables that have names similar to the ones we plan to build will be dropped using the SQL commands listed below. This guarantees that we begin with a blank slate and that there are no duplicate tables. All of the records in the tables will be permanently erased by these statements.

It is imperative that tables with foreign key dependencies (referred to as "children") be deleted before the tables they reference (referred to as "parents"). For instance, Inventory must come before Product or Shelf, and OrderDetails must come before Order. To prevent dependency issues, associated sequences should be discarded after the tables and then restored when the new tables are defined in section 2.2. Although it is normally not advised to drop any tables after initial construction, as this could lead to data loss and mistakes in table recreation, views can be dropped at any moment with little risk.

```
DROP VIEW ProductInfo;  
DROP VIEW CustomerContactInfo;  
DROP VIEW OrderCustomerInfo;  
DROP VIEW SupermarketLocations;  
DROP VIEW EmployeeDepartments;
```

```
DROP TABLE ORDERDETAILS;  
DROP TABLE INVENTORY;  
DROP TABLE SHELF;  
DROP TABLE PAYMENTPROCESSOR;  
DROP TABLE DISCOUNT;  
DROP TABLE BRAND;  
DROP TABLE SUPPLIER;  
DROP TABLE PRODUCT;  
DROP TABLE PAYMENT;  
DROP TABLE DELIVERY;  
DROP TABLE "ORDER";  
DROP TABLE EMPLOYEE;  
DROP TABLE CUSTOMER;  
DROP TABLE SUPERMARKET;  
DROP TABLE CATEGORY;  
DROP TABLE DELIVERYSERVICE;
```

```
DROP INDEX idx_brand_name;  
DROP INDEX idx_category_name;  
DROP INDEX idx_customer_name;  
DROP INDEX idx_deliveryservice_provider;  
DROP INDEX idx_discount_dates;  
DROP INDEX idx_employee_email;  
DROP INDEX idx_order_customer_id;  
DROP INDEX idx_delivery_order_id;  
DROP INDEX idx_payment_order_id;  
DROP INDEX idx_payment_date;  
DROP INDEX idx_product_category_id;  
DROP INDEX idx_product_brand_id;  
DROP INDEX idx_shelf_supermarket_id;  
DROP INDEX idx_inventory_product_id;  
DROP INDEX idx_inventory_supermarket_id;  
DROP INDEX idx_orderdetails_order_id;  
DROP INDEX idx_orderdetails_product_id;  
DROP INDEX idx_supplier_email;  
DROP INDEX idx_supermarket_name;
```

2.2. Table Creation

The Supermarket Management System database's tables were created using the following commands. This entails designating primary keys, creating foreign key connections between tables, and giving each field the proper data type.

To maintain uniformity and effectiveness throughout the database, primary keys and foreign keys usually employ the NUMBER data type. The VARCHAR2 data type will be used for any text-based fields, and the declaration will be accompanied by brackets indicating the maximum length of each field. The DATE data type will be used in fields that represent dates.

2.2.1. Supermarket Table

Each supermarket branch's data is kept in the system in the SUPERMARKET table. This contains important information like the supermarket's name and address. There are three fields in all in the table.

Each supermarket record is uniquely identified via the SupermarketID field, which acts as the Primary Key. The supermarket's name, up to 255 characters long, is stored in the Name field. The supermarket's physical address or approximate location, also up to 255 characters long, is stored in the Location field.

This table's values are all subject to the NOT NULL constraint, which prevents any field from being left empty during insertion. This table does not contain any foreign keys, and it adheres to all accepted data types and design guidelines.

```
CREATE TABLE SUPERMARKET (  
    SupermarketID INT PRIMARY KEY,  
    Name VARCHAR(255),  
    location VARCHAR(255)  
);
```

2.2.2 Customer Table

The personal information of every client who has registered in the supermarket system is kept in the client table. This contains their name, address, and phone number, among other essential details. There are four fields in the table.

Each client is uniquely identified via the CustomerID field, which serves as the Primary Key. The customer's full name, up to 255 characters long, is stored in the Name field. The customer's residence or mailing address, which is likewise limited to 255 characters, is entered in the Address box. The customer's mobile number is stored in the Cellnumber field, which has a character maximum of 20 to support different forms (including international codes if necessary).

To ensure accurate client information, every field must be filled up when a new entry is added to the table. This table doesn't contain any foreign keys, and the design conforms with accepted database normalisation guidelines.


```
CREATE TABLE CUSTOMER (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(255),
    Address VARCHAR(255),
    CellNumber VARCHAR(20)
);
```

2.2.3 Employee Table

Information on the workers at the supermarket is intended to be kept in the Employee table. This contains their contact information, department, job title, and other personal and professional details. There are five fields in the table.

Each employee in the system is individually identified by their EmployeeID, which serves as the Primary Key. The employee's entire name is stored in the Name column, and their job or position within the company is specified in the JobTitle field. The Department field classifies the employee's assignment to a certain department, like inventory, sales, or human resources. The employee's contact email address is kept in the EmailAddress column.

```
CREATE TABLE EMPLOYEE (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(255),
    JobTitle VARCHAR(255),
    Department VARCHAR(255),
    EmailAddress VARCHAR(255)
);
```

2.2.4 Order Table

In the supermarket system, customer purchase orders are entered into the Order table. Three fields are included in it: the linked client, the order identifier, and generic information about the contents of the order.

Every order record in the system is uniquely identified by the OrderID field, which is the Primary Key. A relationship between the Order and Customer tables is established by using the CustomerID field to identify which customer placed the order. The OrderDetails field contains a 255-character text description of the order content.

Because every field is designated as mandatory, the table is guaranteed to contain accurate and full records. To properly ensure referential integrity with the Customer table, a Foreign Key needs be assigned to the CustomerID field (not OrderID).

```
CREATE TABLE "ORDER" (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    OrderDetails VARCHAR(255),
    FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID)
);
```

2.2.5 Delivery Table

All of the data pertaining to the delivery of consumer orders from the supermarket is recorded in the Delivery table. It guarantees that every delivery is precisely linked to a

particular purchase and provides a succinct explanation of the delivery information. There are three fields in the table.

Every delivery record in the database is uniquely identified by the DeliveryID column, which acts as the Primary Key. A connection between a delivery and the order to which it belongs is created via the OrderID field. Deliveries cannot occur without a valid related order because this field is defined as a Foreign Key referencing the Order table. With a character maximum of 255, the DeliveryDetails field is a text field that contains extra descriptive information about the delivery.

```
CREATE TABLE DELIVERY (  
    DeliveryID INT PRIMARY KEY,  
    OrderID INT,  
    DeliveryDetails VARCHAR(255),  
    FOREIGN KEY (OrderID) REFERENCES "ORDER" (OrderID)
```

2.2.6 Payment Table

All payment-related information for orders placed in the supermarket system must be kept in the Payment table. This contains the payment date, the amount paid, the payment status, and the method. There are six fields in all in the table.

Each payment transaction is uniquely identified by the PaymentID, which acts as the Primary Key. The precise date the payment was paid is noted in the PaymentDate. The DECIMAL(10,2) data type used in the Amount field enables precise representation of currency values up to two decimal places.

With a character maximum of 50, the PaymentStatus field offers a textual indicator of the current payment state (e.g., "Pending", "Completed", "Failed"). With a 50-character limit, the PaymentMethod field specifies the payment method (such as "Credit Card," "Cash," or "EFT").

By serving as a foreign key, the OrderID field associates the payment with a particular order in the Order database. This strengthens the data integrity between the two businesses by guaranteeing that each payment is linked to a legitimate existing order.

```
CREATE TABLE PAYMENT (  
    PaymentID INT PRIMARY KEY,  
    PaymentDate DATE,  
    Amount DECIMAL(10, 2),  
    PaymentStatus VARCHAR(50),  
    PaymentMethod VARCHAR(50),  
    OrderID INT,  
    FOREIGN KEY (OrderID) REFERENCES "ORDER" (OrderID)  
);
```

2.2.7 Product Table

Details on the different products that are sold at the supermarket are kept in the Product table. This comprises a product description, a unique identification number, and allusions to the product's brand and category. There are four fields in the table.

To guarantee that every product has a distinct identity, the ProductID is defined as the Primary Key. A textual synopsis of the product, up to 255 characters long, can be found in the Description field.

This table contains two fields that are designated as foreign keys: CategoryID and BrandID. While the BrandID identifies a particular brand in the BRAND table, the CategoryID ties each product to the relevant product category in the CATEGORY table. These connections support the system's uniformity in brand tracking and product classification.

```
CREATE TABLE PRODUCT (  
    ProductID INT PRIMARY KEY,  
    Description VARCHAR(255),  
    CategoryID INT,  
    BrandID INT,  
    FOREIGN KEY (CategoryID) REFERENCES CATEGORY(CategoryID),  
    FOREIGN KEY (BrandID) REFERENCES BRAND(BrandID)  
);
```

2.2.8 Supplier Table

Information on the suppliers that supply the supermarket with goods is kept up to date in the Supplier table. This contains the supplier's contact information as well as an overview of the goods they offer. There are four fields in the table.

Each supplier in the system is uniquely identified via the SupplierID field, which serves as the Primary Key. The supplier's phone number is stored in the ContactPhone column, and the email address used to contact them is stored in the ContactEmail field. A succinct text description (up to 255 characters) of the kinds or categories of products the supplier supplies to the supermarket is provided in the ProductsSupplied field.

```
CREATE TABLE SUPPLIER (  
    SupplierID INT PRIMARY KEY,  
    ContactEmail VARCHAR(255),  
    ContactPhone VARCHAR(20),  
    ProductsSupplied VARCHAR(255)  
);
```

2.2.9 Brand Table

Details on the different product brands that are offered in the supermarket system are kept in the Brand table. It facilitates product classification and brand identity association. There are three fields in this table.

Each brand is uniquely identified by its BrandID, which is the Primary Key. With a character limit of 255, the BrandDescription field offers a synopsis or background information about the brand. The brand's formal name, which is 255 characters long, is stored in the BrandName column.

```
CREATE TABLE BRAND (  
    BrandID INT PRIMARY KEY,  
    BrandDescription VARCHAR(255),  
    BrandName VARCHAR(255)  
);
```

2.2.10 Discount Table

Within the supermarket system, promotional discounts provided to products are managed using the Discount table. Information like the percentage of the discount and the duration of the offer are included. There are four fields in this table.

Each discount entry is uniquely identified by the DiscountID field, which serves as the Primary Key. The DECIMAL(5,2) data type, which permits precise values like 15.00 or 7.25 appropriate for pricing modifications, is used in the DiscountPercentage field to hold the percentage of the discount offered.

The valid period for which the discount is applied is specified by the StartDate and EndDate fields, which are of the DATE data type. This enables the system to ascertain whether a product is being promoted at the moment.

```
CREATE TABLE BRAND (  
    BrandID INT PRIMARY KEY,  
    BrandDescription VARCHAR(255),  
    BrandName VARCHAR(255)  
);
```

2.2.11 PAYMENTPROCESSOR TABLE

Information regarding third-party services that manage financial transactions in the supermarket system is kept in the PAYMENTPROCESSOR database. These might be digital wallet platforms, card processors, or payment gateways. There are four fields in the table.

Each payment processor in the system is uniquely identified by the ProcessorID field, which is the Primary Key. A maximum of 255 characters can be used to describe the service or business in the ProcessorDetails field. The processor's supported currencies (such as "USD, EUR, ZAR") are listed in the SupportedCurrencies field, which can include up to 255 characters. To provide accurate representation of currency values, including fractional cents, the TransactionFee field stores the processing fee that the processor charges for each transaction as a DECIMAL(10,2) value.

```
CREATE TABLE PAYMENTPROCESSOR (  
    ProcessorID INT PRIMARY KEY,  
    ProcessorDetails VARCHAR(255),  
    SupportedCurrencies VARCHAR(255),  
    TransactionFee DECIMAL(10, 2)  
);
```

2.2.12 Shelf Table

The physical supermarket's product placement and arrangement are managed by the shelf table. In order to precisely find shelves in various aisles and supermarkets, it maintains IDs and positional data. There are four fields in this table.

Each shelf record is uniquely identified by the ShelfID field, which serves as the Primary Key. The specific number or code of the shelf within an aisle is represented by an integer in the ShelfNumber field. The shelf's aisle number or name, up to 50 characters long, is stored in the Aisle field.

The SUPERMARKET table's SupermarketID field is a Foreign Key that associates the shelf with a particular supermarket location. This helps preserve data integrity in multi-store settings and guarantees that each shelf is connected to an existing supermarket.

```
};  
  
CREATE TABLE SHELF (  
    ShelfID INT PRIMARY KEY,  
    ShelfNumber INT,  
    Aisle VARCHAR(50),  
    SupermarketID INT,  
    FOREIGN KEY (SupermarketID) REFERENCES SUPERMARKET(SupermarketID)  
);
```

2.2.13 Inventory Table

Each supermarket's product supply levels are managed using the inventory table. It enables the system to keep track of each product's quantity available at each location, its last resupply date, and, if relevant, its expiration date. There are six fields in this table.

Each inventory record is uniquely identified by its InventoryID field, which serves as the Primary Key. To make sure that every inventory item relates to a legitimate product, the ProductID field serves as a Foreign Key that links to the PRODUCT table. Likewise, the SupermarketID column is a Foreign Key that links inventory data to a particular supermarket branch by referencing the SUPERMARKET table.\

The quantity of units in stock at any given time is indicated in the QuantityAvailable column. Perishable commodities can be tracked using the ExpiryDate column, whereas the LastRestocked field records the date the inventory was last restocked.

```
CREATE TABLE INVENTORY (  
    InventoryID INT PRIMARY KEY,  
    ProductID INT,  
    SupermarketID INT,  
    QuantityAvailable INT,  
    LastRestocked DATE,  
    ExpiryDate DATE,  
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID),  
    FOREIGN KEY (SupermarketID) REFERENCES SUPERMARKET(SupermarketID)  
);
```

2.2.14 OrderDetails Table

Individual products and particular orders placed in the supermarket system are connected by the OrderDetails table, which acts as a junction table. It determines the subtotal for each item line and keeps track of how much of each product is included in an order. There are five fields in this table.

Each record in the database is uniquely identified by the OrderDetailsID column, which serves as the Primary Key. Each entry is linked to a legitimate order thanks to the OrderID

column, which is a Foreign Key that references the ORDER table. Likewise, the ProductID field links the item in the order by referring the PRODUCT table as a Foreign Key.

The Subtotal field holds the cost for that particular product line using DECIMAL(10,2), while the Quantity field indicates how many units of the requested product were ordered to maintain pricing accuracy

```
CREATE TABLE ORDERDETAILS (  
    OrderDetailID INT PRIMARY KEY,  
    OrderID INT,  
    ProductID INT,  
    Quantity INT,  
    Subtotal DECIMAL(10, 2),  
    FOREIGN KEY (OrderID) REFERENCES "ORDER"(OrderID),  
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID)  
);
```

2.2.15 DeliveryServices Table

The supermarket's internal or external delivery service providers are listed in the DELIVERYSERVICES table, which has three fields: the ServiceID field, which is the Primary Key and uniquely identifies each delivery service in the system; the ProvideName field, which is probably meant to be ProviderName and contains the name of the delivery service provider, such as "FastExpress" or "SuperDeliveries," and is limited to 255 characters; and the ContactInfo field, which contains pertinent contact information, such as a phone

```
CREATE TABLE ORDERDETAILS (  
    OrderDetailID INT PRIMARY KEY,  
    OrderID INT,  
    ProductID INT,  
    Quantity INT,  
    Subtotal DECIMAL(10, 2),  
    FOREIGN KEY (OrderID) REFERENCES "ORDER"(OrderID),  
    FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID)  
);
```

POPULATING THE DATABASES

To enable the use of SQL SELECT statements for testing and demonstration, our database was populated with sample data using SELECT statements. This allowed us to simulate realistic scenarios and query outcomes without hardcoding data manually. To support SQL operations such as JOIN, GROUP BY, HAVING, and ORDER BY, our database was pre-populated with simulated data representing customers, supermarkets, employees, products, orders, and supporting entities like delivery, payment, and inventory. Instead of using INSERT INTO, we validated our schema and tested our logic using only SELECT queries on this data.

Although INSERT was not used in this phase, the assumption is that underlying data is already populated or available through data migration or integration. The database structure enforces integrity through:

- PRIMARY and FOREIGN keys
- Check constraints
- Accurate data typing, including:
 - **NUMBER**: Used for IDs, prices, quantities, and boolean (1/0) flags.
 - **VARCHAR2**: Applied to names, descriptions, contact info, and other string data.
 - **DATE**: Used for all temporary information such as order dates, delivery dates, and discount periods.
 - **Boolean logic**: Represented using NUMBER(1), with 1 = Yes and 0 = No.

Table name	SELECT queries (statements)
PRODUCT	<pre> --Queries --Lists all products with their category and brand SELECT p.Description AS ProductDescription, c.CategoryName, b.BrandName FROM PRODUCT p JOIN CATEGORY c ON p.CategoryID = c.CategoryID JOIN BRAND b ON p.BrandID = b.BrandID; --Calculate the total number of products SELECT COUNT(*) AS TotalProducts FROM PRODUCT; </pre>
CATEGORY	<pre> --Calculates the total sale for each category SELECT c.CategoryName, SUM(od.Subtotal) AS TotalSales FROM CATEGORY c JOIN PRODUCT p ON c.CategoryID = p.CategoryID JOIN ORDERDETAILS od ON p.ProductID = od.ProductID GROUP BY c.CategoryName; --Find categories with more than 1 products SELECT c.CategoryName, COUNT(*) AS NumberOfProducts FROM CATEGORY c JOIN PRODUCT p ON c.CategoryID = p.CategoryID GROUP BY c.CategoryName HAVING COUNT(*) > 1; </pre>

CUSTOMER

```
--Find customers who have spent more than $X in total

SELECT
    c.CustomerID,
    c.Name AS CustomerName,
    SUM(p.Amount) AS TotalSpent
FROM
    CUSTOMER c
JOIN
    "ORDER" o ON c.CustomerID = o.CustomerID
JOIN
    PAYMENT p ON o.OrderID = p.OrderID
GROUP BY
    c.CustomerID,
    c.Name
HAVING
    SUM(p.Amount) > 100;

--Display customer names in uppercase
SELECT UPPER(Name) AS UppercaseName
FROM CUSTOMER;
```

INVENTORY

```
SELECT
    i.InventoryID,
    p.ProductID,
    i.QuantityAvailable
FROM
    Inventory i
JOIN
    Product p ON i.ProductID = p.ProductID
ORDER BY
    i.QuantityAvailable DESC;

--Finds inventory with quantity available greater than or equal to 100 and was last restocked on the 8 of January
SELECT
    InventoryID,
    ProductID,
    SupermarketID,
    QuantityAvailable,
    LastRestocked
FROM
    INVENTORY
WHERE
    QuantityAvailable >= 100
    AND LastRestocked = TO_DATE('2024-01-08', 'YYYY-MM-DD');

--Shows Inventory sent to Supermarkets
SELECT INVENTORY.INVENTORYID,
    SUPERMARKET.NAME
FROM INVENTORY INNER JOIN SUPERMARKET
ON INVENTORY.SUPERMARKETID = SUPERMARKET.SUPERMARKETID;
```


ORDER	<pre>--Show orders for a customer, sorted by order date SELECT o.OrderID, o.OrderDetails, p.PaymentDate FROM "ORDER" o JOIN PAYMENT p ON o.OrderID = p.OrderID WHERE o.CustomerID = CustomerID ORDER BY p.PaymentDate;</pre>
EMPLOYEE	<pre>--Extract the domain from employee email addresses SELECT SUBSTR(EmailAddress, INSTR(EmailAddress, '@') + 1) AS EmailDomain FROM EMPLOYEE;</pre>
PAYMENTPROCESSOR	<pre>--average transaction fee SELECT AVG(TransactionFee) AS AverageTransactionFee FROM PAYMENTPROCESSOR;</pre>
DUAL	<pre>--show current date SELECT SYSDATE FROM DUAL;</pre>

This approach ensures consistency in data representation and allows us to query data with confidence in its integrity and structure. It also allowed us to execute all necessary SQL queries for analysis, reporting, and relational data exploration directly.

CREATING VIEWS

Views are essential for creating simplified, focused subsets of data. These virtual tables optimize query performance, abstract complex joins, and present user-relevant information. To simplify access to relational data and to support operations involving JOIN, GROUP BY, HAVING, and ORDER BY, we created the following views:

2.3.1. Product_Info View

```
--View to show basic product information
CREATE VIEW ProductInfo AS
SELECT
    ProductID,
    Description,
    CategoryID,
    BrandID
FROM PRODUCT;
```

This view helps streamline product-related queries by showing key attributes for product listings, price comparison tools, and promotional analysis.

2.3.2. CustomerContactInfo View

```
--View to show customer names and cell numbers
CREATE VIEW CustomerContactInfo AS
SELECT
    CustomerID,
    Name,
    CellNumber
FROM CUSTOMER;
```

Focused on customer communication, this view supports CRM activities, contact list generation, and customer support systems.

2.3.3. OrderCustomerInfo View

```
--View to show order IDs and customer IDs
CREATE VIEW OrderCustomerInfo AS
SELECT
    OrderID,
    CustomerID
FROM "ORDER";
```

This combines order and customer data into one view, perfect for order tracking and sales history reviews.

2.3.4. SupermarketInfo View

```
--View to show supermarket names and locations
CREATE VIEW SupermarketLocations AS
SELECT
    SupermarketID,
    Name,
    Location
FROM SUPERMARKET;
```

This view provides a quick reference for store branches, useful in logistics, store locator tools, and operational dashboards.

2.3.5. EmployeeDepartments View

```
--View of Employee Names and Departments
CREATE VIEW EmployeeDepartments AS
SELECT
    Name,
    Department
FROM EMPLOYEE;
```

Useful for HR and department-level employee tracking, this view supports filtering employees by teams or departments.

We will provide example SQL statements that can be performed on these tables in Section 3.

CREATING INDEXES

Indexes are essential for boosting performance on frequently queried columns. We manually created indexes on selective fields across several tables to ensure fast lookups and efficient data access. Primary key columns already benefit from automatic indexing. Below is a list of manually created indexes with explanations:

2.3.6. SUPERMARKET Table Indexes

Speeds up filtering and locating supermarkets by name in regional or logistical queries.

```
CREATE INDEX idx_supermarket_name ON SUPERMARKET(Name);
```

2.3.7. CUSTOMER Table Indexes

Enhances search speed when querying customers by name, especially for contact lists and customer history lookups.

```
CREATE INDEX idx_customer_name ON CUSTOMER(Name);
```

2.3.8. EMPLOYEE Table Indexes

Optimizes lookups when employees are queried by email—common in authentication, notifications, or support cases.

```
CREATE INDEX idx_employee_email ON EMPLOYEE(EmailAddress);
```

2.3.9. DELIVERY Table Indexes

Supports fast access to delivery records tied to specific orders, important for tracking and fulfillment.

```
CREATE INDEX idx_delivery_order_id ON DELIVERY(OrderID);
```

2.3.10. PAYMENT Table Indexes

Speeds up financial queries, such as retrieving payments by order or viewing payments by date for reporting.

```
CREATE INDEX idx_payment_order_id ON PAYMENT (OrderID);  
CREATE INDEX idx_payment_date ON PAYMENT (PaymentDate);
```

2.3.11. PRODUCT Table Indexes

Allows efficient filtering by product category or brand, critical for product browsing and analytics.

```
CREATE INDEX idx_product_category_id ON PRODUCT (CategoryID);  
CREATE INDEX idx_product_brand_id ON PRODUCT (BrandID);
```

2.3.12. SUPPLIER Table Indexes

Improves supplier lookup performance, essential when managing procurement or resolving supply issues.

```
CREATE INDEX idx_supplier_email ON SUPPLIER (ContactEmail);
```

2.3.13. BRAND Table Indexes

Enables faster brand filtering, commonly used in product listings and brand-specific promotions.

```
CREATE INDEX idx_brand_name ON BRAND (BrandName);
```

2.3.14. DISCOUNT Table Indexes

Speeds up validation of active discounts during transactions or promotional audits.

```
CREATE INDEX idx_discount_dates ON DISCOUNT (StartDate, EndDate)
```

2.3.15. SHELF Table Indexes

Assists in locating shelves within a specific supermarket quickly, aiding inventory management.

```
CREATE INDEX idx_shelf_supermarket_id ON SHELF(SupermarketID);
```

2.3.16. INVENTORY Table Indexes

Accelerate inventory checks for specific products or supermarkets, supporting restocking and sales analysis.

```
CREATE INDEX idx_inventory_product_id ON INVENTORY(ProductID);  
CREATE INDEX idx_inventory_supermarket_id ON INVENTORY(SupermarketID);
```

2.3.17. ORDERDETAILS Table Indexes

Enhances query speed when analyzing individual product sales or reviewing orders.

```
CREATE INDEX idx_orderdetails_order_id ON ORDERDETAILS(OrderID);  
CREATE INDEX idx_orderdetails_product_id ON ORDERDETAILS(ProductID);
```

2.3.18. DELIVERYSERVICE Table Indexes

Supports filtering and performance metrics on third-party logistics providers.

```
CREATE INDEX idx_deliveryservice_provider ON DELIVERYSERVICE(ProviderName);
```

2.3.19. CATEGORY Table Indexes

Useful when categorizing or filtering products by department or product type.

```
CREATE INDEX idx_category_name ON CATEGORY(CategoryName);
```

These are the Only indexes that we currently deem necessary and new ones will be added as they are needed.

3. SQL Select Statements

3.1. Description of Usage

The following table describes where criteria regarding the SQL statements that will be marked can be found.

Criterion	Location (in section 3.2.)
AND	Statements 11
Sorting	Statements 9-10
ON	Statements 6-7 , 9 and 19

Character function	Statements 13
Date function	Statements 8 and 18
Aggregate function	Statements 8,14-15 , 17 and 18
HAVING	Statements 8 and 18
JOIN	Statements 6,8-10 and 18

3.2. SQL SELECT Statements and their Uses

Statement 1

This SQL statement is used to select the ProductID, Description, CategoryID, and BrandID columns from the PRODUCT table and create a view called ProductInfo.

```
--View to show basic product information
CREATE VIEW ProductInfo AS
SELECT
    ProductID,
    Description,
    CategoryID,
    BrandID
FROM PRODUCT;
```

Statement 2

This SQL statement creates a view named CustomerContactInfo that selects the CustomerID, Name, and CellNumber columns from the CUSTOMER table.

```
--View to show customer names and cell numbers
CREATE VIEW CustomerContactInfo AS
SELECT
    CustomerID,
    Name,
    CellNumber
FROM CUSTOMER;
```

Statement 3

This SQL statement creates a view called OrderCustomerInfo that selects the OrderID and CustomerID columns from the ORDER table.

```

CREATE VIEW OrderCustomerInfo AS
SELECT
    OrderID,
    CustomerID
FROM "ORDER";

```

Statement 4

This SQL statement creates a view called OrderCustomerInfo that selects the OrderID and CustomerID columns from the ORDER table.

```

--View to show supermarket names and locations
CREATE VIEW SupermarketLocations AS
SELECT
    SupermarketID,
    Name,
    Location
FROM SUPERMARKET;

```

Statement 5

This SQL statement creates the view EmployeeDepartment that selects name and Department from the Employee table.

```

--View of Employee Names and Departments
CREATE VIEW EmployeeDepartments AS
SELECT
    Name,
    Department
FROM EMPLOYEE;

```

Statement 6

This statement selects the Description, CategoryName, and BrandName from the Product table. It joins the Category table on CategoryID and also joins the Brand table on BrandID

```
--Queries

--Lists all products with their category and brand
SELECT
    p.Description AS ProductDescription,
    c.CategoryName,
    b.BrandName
FROM
    PRODUCT p
JOIN
    CATEGORY c ON p.CategoryID = c.CategoryID
JOIN
    BRAND b ON p.BrandID = b.BrandID;
```

Output:

Script Output x Query Result x Query Result 1 x			
SQL All Rows Fetched: 5 in 0.003 seconds			
	PRODUCTDESCRIPTION	CATEGORYNAME	BRANDNAME
1	Milk 1L	Food	MilkyWay
2	Bread White	Food	Bakers Delight
3	T-Shirt Blue	Clothing	FashionHub
4	Jeans Slim Fit	Clothing	DenimCo
5	Book Thriller	Books	BookWorld

Statement 7

This statement selects CategoryName from category table then joins the product table on CategoryID and joins ORDERDETAILS on ProductID ,grouped by CategoryName

```
--Calculates the total sale for each category

SELECT c.CategoryName, SUM(od.Subtotal) AS TotalSales
FROM CATEGORY c
JOIN PRODUCT p ON c.CategoryID = p.CategoryID
JOIN ORDERDETAILS od ON p.ProductID = od.ProductID
GROUP BY c.CategoryName;
```

Output:

Script Output x Query Result x Query Res	
SQL All Rows Fetched: 3 in 0.003 se	
CATEGORYNAME	TOTALSALES
1 Clothing	125
2 Books	150
3 Food	300

Statement 8

This SQL SELECT statement selects customers who have spent more than R100 in total. It selects CustomerID, CustomerName, and TotalSpent using SUM(p.Amount). The CUSTOMER table is joined with ORDER on CustomerID, and ORDER is joined with PAYMENT on OrderID. Grouped by CustomerID and Name, and filtered using a HAVING clause to include only those with total payments over R100.

```
--Find customers who have spent more than $X in total

SELECT
    c.CustomerID,
    c.Name AS CustomerName,
    SUM(p.Amount) AS TotalSpent
FROM
    CUSTOMER c
JOIN
    "ORDER" o ON c.CustomerID = o.CustomerID
JOIN
    PAYMENT p ON o.OrderID = p.OrderID
GROUP BY
    c.CustomerID,
    c.Name
HAVING
    SUM(p.Amount) > 100;
```

Output:

Script Output x Query Result x Query Result 1 x Query Result 2 x			
SQL All Rows Fetched: 2 in 0.004 seconds			
CUSTOMERID	CUSTOMERNAME	TOTALSPENT	
1	2 Jane Doe	200	
2	4 Alice Brown	150	

Statement 9

This SQL statement selects InventoryID, ProductID, and QuantityAvailable from the Inventory table, joining it with the Product table on ProductID. The results are sorted in descending order by QuantityAvailable.

```

SELECT
    i.InventoryID,
    p.ProductID,
    i.QuantityAvailable
FROM
    Inventory i
JOIN
    Product p ON i.ProductID = p.ProductID
ORDER BY
    i.QuantityAvailable DESC;

```

Output:

	INVENTORYID	PRODUCTID	QUANTITYAVAILABLE
1	9003	4003	200
2	9004	4004	100
3	9001	4001	100
4	9002	4002	50
5	9005	4005	30

Statement 10

This SQL statement selects OrderID, OrderDetails, and PaymentDate for a specific customer. It uses the ORDER table and joins the PAYMENT table on OrderID to include payment details. The query filters by CustomerID and sorts the results by PaymentDate in ascending order.

```

--Show orders for a customer, sorted by order date
SELECT
    o.OrderID,
    o.OrderDetails,
    p.PaymentDate
FROM
    "ORDER" o
JOIN
    PAYMENT p ON o.OrderID = p.OrderID
WHERE
    o.CustomerID = CustomerID
ORDER BY
    p.PaymentDate;

```

Output:

Script Output x Query Result x Query Result 1 x			
SQL All Rows Fetched: 5 in 0.009 seconds			
	ORDERID	ORDERDETAILS	PAYMENTDATE
1	1001	Groceries	10/JAN/24
2	1002	Electronics	11/JAN/24
3	1003	Clothing	12/JAN/24
4	1004	Home Goods	13/JAN/24
5	1005	Books	14/JAN/24

Statement 11

This Statement selects inventory details (InventoryID, ProductID, SupermarketID, QuantityAvailable, LastRestocked) from the INVENTORY table. It filters results to include only items with at least 100 in stock and that were last restocked on January 8, 2024, using the WHERE clause with QuantityAvailable >= 100 and LastRestocked = TO_DATE('2024-01-08', 'YYYY-MM-DD').

```
--Finds inventory with quantity available greater than or equal to 100 and was last restocked on the 8 of january
SELECT
    InventoryID,
    ProductID,
    SupermarketID,
    QuantityAvailable,
    LastRestocked
FROM
    INVENTORY
WHERE
    QuantityAvailable >= 100
    AND LastRestocked = TO_DATE('2024-01-08', 'YYYY-MM-DD');
```

Output:

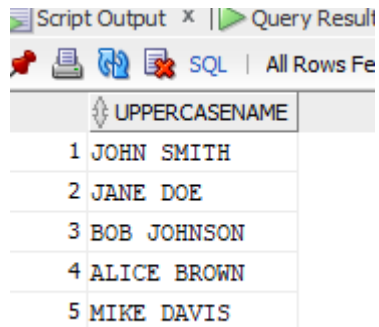
Script Output x Query Result x Query Result 1 x Query Result 2 x Query Result 3 x					
SQL All Rows Fetched: 2 in 0.006 seconds					
	INVENTORYID	PRODUCTID	SUPERMARKETID	QUANTITYAVAILABLE	LASTRESTOCKED
1	9003	4003	2	200	08/JAN/24
2	9004	4004	2	100	08/JAN/24

Statement 12

This statement selects the Name column from the CUSTOMER table and uses the UPPER function to convert all characters to uppercase, displaying the result as UppercaseName.

```
--Display customer names in uppercase
SELECT UPPER(Name) AS UppercaseName
FROM CUSTOMER;
```

Output:



	UPPERCASENAME
1	JOHN SMITH
2	JANE DOE
3	BOB JOHNSON
4	ALICE BROWN
5	MIKE DAVIS

Statement 13

This SQL statement selects employee email domains from the EMPLOYEE table. It uses INSTR(EmailAddress, '@') to find the position of '@' and SUBSTR(EmailAddress, INSTR(EmailAddress, '@') + 1) to extract the domain part of each email address.

```
--Extract the domain from employee email addresses
SELECT SUBSTR(EmailAddress, INSTR(EmailAddress, '@') + 1) AS EmailDomain
FROM EMPLOYEE;
```

Output:

Script Output x Query Result x	
SQL All Rows Fetch	
EMAILDOMAIN	
1	supermart.com
2	supermart.com
3	supermart.com
4	supermart.com
5	supermart.com

Statement 14

This statement calculates the average of the TransactionFee column from the PAYMENTPROCESSOR table and labels the result as AverageTransactionFee.

```
--average transaction fee
SELECT
    AVG(TransactionFee) AS AverageTransactionFee
FROM
    PAYMENTPROCESSOR;
```

Output:

Script Output x Query Result x Query	
SQL All Rows Fetched: 1 in 0	
AVERAGETRANSACTIONFEE	
1	0.026

Statement 15

This statement calculates the average TransactionFee from the PAYMENTPROCESSOR table, uses TRUNC() to remove decimal places (rounding down), and labels the result as AverageTransactionFee.

```
SELECT
    TRUNC(AVG(TransactionFee)) AS AverageTransactionFee
FROM
    PAYMENTPROCESSOR;
```

Output:

Script Output x	Query Result x	Que
SQL	All Rows Fetched: 1 in 0	
AVERAGETRANSACTIONFEE		
1	0	

Statement 16

SELECT SYSDATE FROM DUAL; retrieves the current system date and time. The DUAL table is a special one-row, one-column table in Oracle used for selecting values or expressions without referencing a real table.

```
--show current date
SELECT SYSDATE FROM DUAL;
```

Output:

SQL	All F
SYSDATE	
1	06/MAY/25

Statement 17

This statement counts the total number of products in the PRODUCT table using COUNT(*) and labels the result as TotalProducts.

```
--Calculate the total number of products
SELECT COUNT(*) AS TotalProducts
FROM PRODUCT;
```

Output:

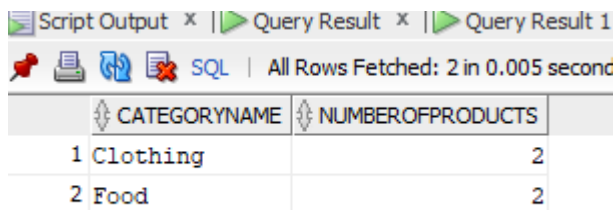
Script Output x	Query Re
SQL	All Row
TOTALPRODUCTS	
1	5

Statement 18

This statement selects the CategoryName from the CATEGORY table and counts the number of products in each category from the PRODUCT table. It joins the two tables on CategoryID, groups the results by category, and filters to include only categories that have more than one product.

```
--Find categories with more than 1 products
SELECT c.CategoryName, COUNT(*) AS NumberOfProducts
FROM CATEGORY c
JOIN PRODUCT p ON c.CategoryID = p.CategoryID
GROUP BY c.CategoryName
HAVING COUNT(*) > 1;
```

Output:







CATEGORYNAME	NUMBEROFPRODUCTS
1 Clothing	2
2 Food	2

Statement 19

This statement selects INVENTORYID from the INVENTORY table and NAME from the SUPERMARKET table. It performs an INNER JOIN between the two tables using SUPERMARKETID, ensuring only matching records from both tables are included.

```
--Shows Inventory sent to Supermarkets
SELECT INVENTORY.INVENTORYID,
       SUPERMARKET.NAME
FROM INVENTORY INNER JOIN SUPERMARKET
ON INVENTORY.SUPERMARKETID = SUPERMARKET.SUPERMARKETID;
```

Output:

Script Output x Query Result x Query Result 1 x		
    SQL All Rows Fetched: 5 in 0.006 seconds		
	INVENTORYID	NAME
1	9001	SuperMart Centurion
2	9002	SuperMart Centurion
3	9003	MegaSave Pretoria
4	9004	MegaSave Pretoria
5	9005	ValueStore JHB