Overview

In this project, a database and SQL queries are created to support a retail operation that sells goods to clients. The system's goal is to keep tabs on inventory levels, sales activity, outlays, and earnings.

Database Design

We have designed the database using four tables: products, sales, expenses, and remaining_items.

Products Table

The products table stores information about the products sold by the business. It has the following columns:

- product_id: Unique identifier for each product
- product_name: Name of the product
- unit_price: The cost price of the product
- stock_available: The quantity of the product available in the inventory

Sales Table

The sales table stores information about the sales transactions made by the business. It has the following columns:

- sale_id: Unique identifier for each sales transaction
- product_id: The product that was sold
- salesperson_name: The seller.
- sale_price: The selling price of the product
- sale_date: The selling date of the product
- quantity_sold: The quantity of the product sold

Expenses Table

The expenses table stores information about the expenses incurred by the business. It has the following columns:

- expense_id: Unique identifier for each expense transaction
- expense_name: The type of expense (e.g. rent, utilities, salaries)
- expense_date: The date of the expense
- expense_amount: The amount of the expense

Profit Table

The profit table stores information about the profit after a sale is made. It has the following columns:

- profit_id: Unique identifier for each profit made.
- sale_id: The sale that the profit belongs to
- profit_amount: The amount of the item profit made

SQL Queries

We have created several SQL queries to support the functionality of the retail sales and inventory management system. The queries are as follows:

1. **Query to create the products table:** This query creates a table named products with four columns: product_id, product_name, unit_price, and stock_quantity. The product_id column is the primary key for the table.

```
CREATE TABLE Products (
  ->    product_id INT PRIMARY KEY AUTO_INCREMENT,
  ->    product_name VARCHAR(255) NOT NULL,
  ->    unit_price DECIMAL(10,2) NOT NULL,
  ->    stock_quantity INT NOT NULL
  -> );
```

2. **Query to create the sales table:** This query creates a table named sales with six columns: sale_id, product_id, salesperson_name, sale_price, sale_date, and quantity. The sale_id column is the primary key for the table, and product_id is a foreign key that references the product_id column in the products table.

```
CREATE TABLE Sales (
  ->    sale_id INT PRIMARY KEY AUTO_INCREMENT,
  ->    product_id INT NOT NULL,
  ->    salesperson_name VARCHAR(255) NOT NULL,
  ->    sale_price DECIMAL(10,2) NOT NULL,
  ->    sale_date DATE NOT NULL,
  ->    quantity INT NOT NULL,
  ->    FOREIGN KEY (product_id) REFERENCES Products(product_id)
```

3. **Query to insert data into the products table:** This query inserts data into the products table. It adds six rows of data, with each row representing a different product.

```
INSERT INTO Products (product_name, unit_price, stock_quantity)

  -> VALUES

  ->  ('Keychain', 75, 30),

  -> ('Resin Art', 50, 25),

  -> ('Scrunchies',20,50),
```

-> ('Chocolates',30,40),

-> ('Action Figures',150,15),

-> ('Rings',40,50);

```
mysql> SELECT * from Products;
+------------+----------------+------------+----------------+
| product_id | product_name   | unit_price | stock_quantity |
+------------+----------------+------------+----------------+
|          1 | Keychain       |      75.00 |             30 |
|          2 | Resin Art      |      50.00 |             25 |
|          3 | Scrunchies     |      20.00 |             50 |
|          4 | Chocolates     |      30.00 |             40 |
|          5 | Action Figures |     150.00 |             15 |
|          6 | Rings          |      40.00 |             50 |
+------------+----------------+------------+----------------+
```

4. **Query to insert data into the sales table:** This query inserts data into the sales table. It adds six rows of data, with each row representing a different sale made by a salesperson.

INSERT INTO Sales (product_id, salesperson_name, sale_price, sale_date, quantity)

-> VALUES

-> (4,'Aditya',75,'22/10/23',2),

-> (2,'Aditya',100,'22/10/23',1),

-> (1,'Manisha',120,'27/10/23',1),

-> (5,'Dhairya',500,'22/10/23',1),

-> (4,'Mridul',50,'22/11/24',3),

-> (3,'Abhishek',40,'22/11/24',5);

```
mysql> SELECT * from Sales;
+---------+------------+------------------+------------+------------+----------+
| sale_id | product_id | salesperson_name | sale_price | sale_date  | quantity |
+---------+------------+------------------+------------+------------+----------+
|       1 |          4 | Aditya           |      75.00 | 2022-10-23 |        2 |
|       2 |          2 | Aditya           |     100.00 | 2022-10-23 |        1 |
|       3 |          1 | Manisha          |     120.00 | 2027-10-23 |        1 |
|       4 |          5 | Dhairya          |     500.00 | 2022-10-23 |        1 |
|       5 |          4 | Mridul           |      50.00 | 2022-11-24 |        3 |
|       6 |          3 | Abhishek         |      40.00 | 2022-11-24 |        5 |
+---------+------------+------------------+------------+------------+----------+
```

5. **Query to update data is sales table:** This query updates the incorrect entered data.
   mysql> UPDATE Sales
   -> SET sale_date = '22/10/23'
   -> WHERE sale_id = 3;

```
mysql> SELECT * from Sales;
+---------+------------+------------------+------------+------------+----------+
| sale_id | product_id | salesperson_name | sale_price | sale_date  | quantity |
+---------+------------+------------------+------------+------------+----------+
|       1 |          4 | Aditya           |      75.00 | 2022-10-23 |        2 |
|       2 |          2 | Aditya           |     100.00 | 2022-10-23 |        1 |
|       3 |          1 | Manisha          |     120.00 | 2022-10-23 |        1 |
|       4 |          5 | Dhairya          |     500.00 | 2022-10-23 |        1 |
|       5 |          4 | Mridul           |      50.00 | 2022-11-24 |        3 |
|       6 |          3 | Abhishek         |      40.00 | 2022-11-24 |        5 |
+---------+------------+------------------+------------+------------+----------+
```

6. **Query to create the expenses table:** This shows the expenses made on any date.

   CREATE TABLE Expenses (
   -> expense_id INT PRIMARY KEY AUTO_INCREMENT,
   -> expense_name VARCHAR(255) NOT NULL,
   -> expense_date DATE NOT NULL,
   -> expense_amount DECIMAL(10,2) NOT NULL
   ->);

7. **Query to enter data is expense table:** This is used the enter the miscellaneous data.

   INSERT INTO Expenses (expense_name, expense_date, expense_amount)
   -> VALUES
   -> ('Refreshments', '22/10/23', 100.00),
   -> ('Refreshments', '22/11/24', 80.00);

```
mysql> SELECT * from Expenses;
+------------+--------------+--------------+----------------+
| expense_id | expense_name | expense_date | expense_amount |
+------------+--------------+--------------+----------------+
|          1 | Refreshments | 2022-10-23   |         100.00 |
|          2 | Refreshments | 2022-11-24   |          80.00 |
+------------+--------------+--------------+----------------+
```

8. **Query to create the profit table:** This is to create the table carrying the profit of individual sales.

   CREATE TABLE Profit (
   -> profit_id INT PRIMARY KEY AUTO_INCREMENT,
   -> sale_id INT NOT NULL,
   -> profit_amount DECIMAL(10,2) NOT NULL,
   -> FOREIGN KEY (sale_id) REFERENCES Sales(sale_id)
   ->);

9. **Query to insert the profit:** This is to insert the data int the profit table.

   INSERT INTO Profit (sale_id, profit_amount)
   -> VALUES
   -> (1,90),

-> (2,20),
-> (3,55),
-> (4,350),
-> (5,60),
-> (6,100);

```
mysql> SELECT * from Profit;
+-----------+---------+---------------+
| profit_id | sale_id | profit_amount |
+-----------+---------+---------------+
|         1 |       1 |         90.00 |
|         2 |       2 |         20.00 |
|         3 |       3 |         55.00 |
|         4 |       4 |        350.00 |
|         5 |       5 |         60.00 |
|         6 |       6 |        100.00 |
+-----------+---------+---------------+
```

10. **Query to find the maximum sales by salesperson:** This query uses the GROUP BY clause to group the sales by salesperson, and the SUM function to calculate the total sales for each salesperson. It then uses the ORDER BY and LIMIT clauses to retrieve the salesperson with the maximum sales.

    SELECT salesperson_name, SUM(quantity) as total_sales
    -> FROM Sales
    -> GROUP BY salesperson_name
    -> ORDER BY total_sales DESC
    -> LIMIT 3;

```
+------------------+-------------+
| salesperson_name | total_sales |
+------------------+-------------+
| Abhishek         |           5 |
| Aditya           |           3 |
| Mridul           |           3 |
+------------------+-------------+
```

11. **Query to find the salesperson with maximum revenue:** This query uses the GROUP BY clause to group the sales by salesperson, and the SUM function to calculate the total revenue for each salesperson. It then uses the ORDER BY and LIMIT clauses to retrieve the salesperson with the maximum revenue.

```
mysql> SELECT salesperson_name, SUM(sale_price * quantity) as total_revenue
    -> FROM Sales
    -> GROUP BY salesperson_name
    -> ORDER BY total_revenue DESC
    -> LIMIT 3;
+------------------+---------------+
| salesperson_name | total_revenue |
+------------------+---------------+
| Dhairya          |        500.00 |
| Aditya           |        250.00 |
| Abhishek         |        200.00 |
+------------------+---------------+
```

12. **Query to find the salesperson with maximum profit:** This query uses the GROUP BY clause to group the sales by salesperson, and calculates the profit for each sale by subtracting the unit_price from the sale_price. It then uses the SUM function to calculate the total profit for each salesperson, and the ORDER BY and LIMIT clauses to retrieve the salesperson with the maximum profit.

    SELECT salesperson_name, SUM(profit_amount) as total_profit
      -> FROM Profit
      -> JOIN Sales ON Profit.sale_id = Sales.sale_id
      -> GROUP BY salesperson_name
      -> ORDER BY total_profit DESC
      -> LIMIT 3;

```
+------------------+--------------+
| salesperson_name | total_profit |
+------------------+--------------+
| Dhairya          |       350.00 |
| Aditya           |       110.00 |
| Abhishek         |       100.00 |
+------------------+--------------+
```

13. **Query to find the most profitable products:** This query uses the GROUP BY clause to group the sales by product, and calculates the profit for each sale by subtracting the unit_price from the sale_price. It then uses the SUM function to calculate the total profit for each product, and the ORDER BY and LIMIT clauses to retrieve the most profitable products.

    SELECT Products.product_name, SUM(profit_amount) as total_profit
      -> FROM Profit
      -> JOIN Sales ON Profit.sale_id = Sales.sale_id
      -> JOIN Products ON Sales.product_id = Products.product_id
      -> GROUP BY Products.product_name
      -> ORDER BY total_profit DESC
      -> LIMIT 3;

product_name	total_profit
Action Figures	350.00
Chocolates	150.00
Scrunchies	100.00

14. **Query to find the most sold products:** This query uses the GROUP BY clause to group the sales by product, and the SUM function to calculate the total quantity sold for each product. It then uses the ORDER BY and LIMIT clauses to retrieve the most sold products.

    ```
    SELECT Products.product_name, SUM(quantity) as total_sales
       -> FROM Sales
       -> JOIN Products ON Sales.product_id = Products.product_id
       -> GROUP BY Products.product_name
       -> ORDER BY total_sales DESC
       -> LIMIT 3;
    ```

product_name	total_sales
Chocolates	5
Scrunchies	5
Resin Art	1

15. **Query to find the reminder of overstocked and understocked products:** This query uses a CASE statement to categorize each product as overstocked or understocked based on a predetermined threshold value. It then uses the GROUP BY clause to group the products by category, and the SUM function to calculate the total stock quantity for each category.

    ```
    mysql> SELECT product_name, stock_quantity
       -> FROM Products
       -> WHERE stock_quantity > 30;
    ```

product_name	stock_quantity
Scrunchies	50
Chocolates	40
Rings	50

    ```
    SELECT product_name, stock_quantity
       -> FROM Products
       -> WHERE stock_quantity < 20;
    ```

product_name	stock_quantity
Action Figures	15

16. **Query to find the most revenue generating products:** This query uses the GROUP BY clause to group the sales by product, and the SUM function to calculate the total revenue for each product. It then uses the ORDER BY and LIMIT clauses to retrieve the products that generate the most revenue.

```
SELECT Products.product_name, SUM(sale_price * quantity) as total_revenue
    -> FROM Sales
    -> JOIN Products ON Sales.product_id = Products.product_id
    -> GROUP BY Products.product_name
    -> ORDER BY total_revenue DESC
    -> LIMIT 3;
```

product_name	total_revenue
Action Figures	500.00
Chocolates	300.00
Scrunchies	200.00

17. **Query to find the most profitable products:** This query uses the GROUP BY clause to group the sales by product, and the Join operation to calculate the total profit for each product. It then uses the ORDER BY and LIMIT clauses to retrieve the products that generate the most profit.

```
mysql> SELECT Products.product_name, SUM(profit_amount) as total_profit
    -> FROM Profit
    -> JOIN Sales ON Profit.sale_id = Sales.sale_id
    -> JOIN Products ON Sales.product_id = Products.product_id
    -> GROUP BY Products.product_name
    -> ORDER BY total_profit ASC
    -> LIMIT 3;
```

product_name	total_profit
Resin Art	20.00
Keychain	55.00
Scrunchies	100.00

18. **Query to find the least sold products:** This query uses the GROUP BY clause to group the sales by product, and the SUM function to calculate the total quantity sold for each product. It then uses the ORDER BY and LIMIT clauses to retrieve the least sold products.

```
SELECT Products.product_name, SUM(quantity) as total_sales
   -> FROM Sales
   -> JOIN Products ON Sales.product_id = Products.product_id
   -> GROUP BY Products.product_name
   -> ORDER BY total_sales ASC
   -> LIMIT 3;
+----------------+-------------+
| product_name   | total_sales |
+----------------+-------------+
| Resin Art      |           1 |
| Keychain       |           1 |
| Action Figures |           1 |
+----------------+-------------+
```

19. **Query to find the unsold products:** This query checks for the availability of the product id of a product in the sales table to find if the product has ever been sold.

```
mysql> SELECT product_name
   -> FROM Products
   -> WHERE product_id NOT IN (SELECT product_id FROM Sales);
+--------------+
| product_name |
+--------------+
| Rings        |
+--------------+
```

20. **Query to find the profit or loss at a particular date:** This query is used to find the individual profit or loss on a date by subtracting the expenses from the sum of total revenue is sales column.

```
SELECT sale_date, SUM(s.sale_price * s.quantity) - SUM(e.expense_amount) AS profit_loss
   -> FROM sales s
   -> LEFT JOIN expenses e ON s.sale_date = e.expense_date
   -> GROUP BY sale_date;
+------------+-------------+
| sale_date  | profit_loss |
+------------+-------------+
| 2022-10-23 |      470.00 |
| 2022-11-24 |      190.00 |
+------------+-------------+
```

21. **Query to update the inventory list:** This query is used to update the inventory list using GROUP BY function on product_id. It sets the new details by subtracting the quantity sold.
UPDATE products p

```
-> JOIN (
->   SELECT product_id, SUM(quantity) AS total_quantity_sold
->   FROM sales
->   GROUP BY product_id
-> ) s ON p.product_id = s.product_id
-> SET p.stock_quantity = p.stock_quantity - s.total_quantity_sold;
```

product_id	product_name	unit_price	stock_quantity
1	Keychain	75.00	29
2	Resin Art	50.00	24
3	Scrunchies	20.00	45
4	Chocolates	30.00	35
5	Action Figures	150.00	14
6	Rings	40.00	50

Conclusion

In this project, we developed a sales and inventory management system using SQL. The system includes four tables: products, sales, expenses, and profit. We used various SQL queries to analyze the data stored in these tables and generate useful insights for managing the business.

Using the products table, we were able to keep track of the products in stock, their prices, and the quantity available. The sales table helped us record the sales made by each salesperson, the product sold, the sale price, sale date, and quantity. The expenses table helped us keep track of the expenses incurred in the business, including refreshments and other facilities for the employees. Finally, the profit table was used to track the profitability of the business based on the sales and expenses data.

With the help of the various SQL queries we developed, we were able to extract useful insights such as the most profitable products, most sold products, understocked and overstocked products, maximum sale date, and individual profit of different dates. We also identified the salesperson with the maximum revenue, maximum profit, and maximum sales.

The system is designed to automatically update the inventory and profit tables after every sale, eliminating the need for manual data entry. This helps to save time and reduce the chances of errors.

In conclusion, this sales and inventory management system provides valuable insights into the performance of the business, enabling the owner to make informed decisions on product stocking, pricing, and sales strategies. The system is easy to use and can be scaled up to accommodate the needs of a growing business.