MGMT 58200-001
Management of Organizational Data
MSBAIM

Final Project Report

Team Members:
Bansal, Samarth
Dasgupta, Avishek
Shahzad, Hummarah
Singh, Manat
Treasure, Siddhant
Tseng, Kai-Chieh

PURDUE UNIVERSITY
Mitchell E. Daniels, Jr. School of Business

Contents

1. Introduction and Background of the Client

The Original Frozen Custard stands as a cherished family-owned restaurant with a storied history that began in 1932. Founded by the current owner's grandparents, this establishment has become an emblem of tradition and pride. Unique recipes, including the signature frozen custard and special fruit drinks, have been passed down through generations, crafted by the founders themselves as a testament to a commitment to quality and tradition. The primary goal of Original Frozen Custard is to uphold the exceptional quality and taste that has distinguished the restaurant for over eight decades. There's a dedication to preserving cherished recipes and traditions while ensuring guests receive unparalleled service in a warm and welcoming environment. Guests are treated to a diverse menu of All-American classics, from perfectly cooked hamburgers and flavorful hot dogs to the local favorite, breaded tenderloins. To accompany these savory delights, the restaurant also serves old-fashioned sodas and other thirst-quenching beverages. Understanding the needs of modern patrons, Original Frozen Custard offers a swift and efficient drive-thru service, allowing customers to order, pay, and collect their meals without leaving the comfort of their vehicles. Located at 2319 Wallace Ave in Lafayette, IN, the restaurant stands prominently across from Loeb Stadium and Columbian Park, at the intersection of Main St and Wallace Ave. Whether patrons are spending a day at the nearby park or have a craving for nostalgic flavors, the restaurant is a go-to destination, inviting all to immerse themselves in its rich history, impeccable service, and delectable menu.

2. Business Problem Statement

The local business currently operates with multiple data sources, each capturing essential facets of its operations, yet there is no centralized system to cohesively bring this data together. Employees play a pivotal role in selling products, which are the offerings presented to customers. These products, in turn, drive the sales revenue for the business. On the marketing front, each channel incurs its own set of expenses. Furthermore, every product is crafted using distinct raw materials, supplied by a range of vendors. The business also meticulously records various expenses, encompassing manufacturing, selling, marketing, and other operational costs. Despite the intricate web of data points, the absence of a centralized database system poses challenges in harnessing this data effectively and drawing insights from it. Building a unified database is imperative to streamline data access, enhance analytical capabilities, and foster data-driven decision-making. By centralizing the data, management can generate pertinent insights, leading to informed business decisions. Such a system not only optimizes operational efficiency but also paves the way for strategic growth, cost savings, and improved customer engagement.

3. Objectives and Goals

- **Star Selling Products Identification:** One of the primary objectives is to pinpoint the star selling products within the business's portfolio. Recognizing these top-performing products is crucial as it allows the management to tailor marketing strategies effectively. By focusing on these star products, the business can allocate resources efficiently, ensuring that marketing efforts resonate with the target audience and yield optimal returns.

- **Employee Annual Bonus Allocation:** Employee motivation and retention are pivotal for any business's success. To this end, the objective is to make recommendations regarding an annual bonus structure that aligns with an employee's yearly performance matrix. This approach ensures that bonuses are merit-based, rewarding employees who have showcased exceptional performance throughout the year. Such a system not only incentivizes high performance but also fosters a culture of excellence and accountability within the organization.

- **Enhanced Inventory Management:** Efficient inventory management is the backbone of seamless business operations, especially for businesses dealing with tangible products. The objective here is to introduce multiple sorting parameters that can aid in better inventory oversight. By having the capability to sort and categorize inventory based on various criteria, the business can reduce overstocking or stockouts, optimize storage space, and ensure that products are readily available when needed, leading to improved customer satisfaction and operational efficiency.

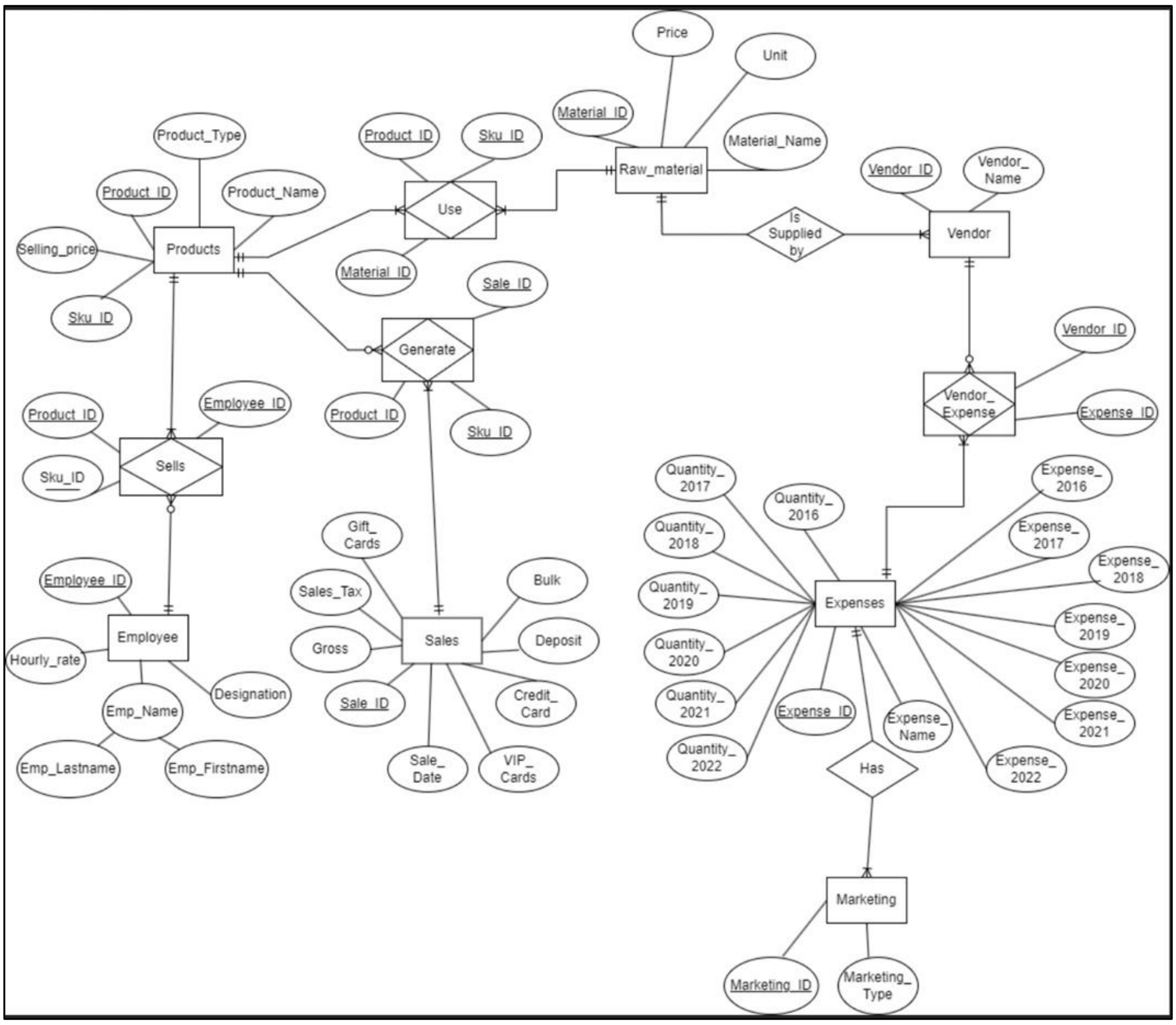
4. Conceptual Data Modelling: Present the ERD

After the initial modelling process has been taken care of and the requirements have been finalized, we better understand the main concepts surrounding our database. From here, to get a better conceptual understanding and model our database properly, we draw the Entity-Relationship diagram (aka. ERD). An entity-relationship diagram (ERD) displays the relationships of entity sets stored in our database. An entity in this context is an object or can be thought of as a component of data. These entities have attributes that define their properties. We have tables that are storing the following data:

- Products
- Raw Materials
- Vendor
- Expenses
- Sales
- Employee

The relationships between Employees > Product > Sales > Raw material & Vendor & Expenses are many-to-many relationships, so we create three associative entity tables.

Following is the ERD:



4.1. Representation of the entity Sets (including attributes and primary key choices)

A table named "employees" contains information about employees within an organization. It has several columns that store specific details about each employee. **EMPLOYEE_ID** column stores a unique identifier for each employee in the organization. **EMPLOYEE_FIRSTNAME** column stores the first name of the employee. It contains textual data representing the employee's given name. **EMPLOYEE_LASTNAME** column stores the last name of the employee. It contains textual data representing the employee's surname or family name. **DESIGNATION** column stores the job title or position of the employee within the organization. It contains textual data representing the specific role or job function of the employee. **HOURLY_RATE** column stores the hourly wage or rate at which the employee is paid for their work.

A table named "**PRODUCT**" contains information about various products offered by Frozen Custard. **Product_ID** column stores a unique identifier for each product in the table. It is typically a numeric or alphanumeric value assigned to uniquely identify each product. **Product_name** column stores the name of the product. Product_type column stores the category or type to which the product belongs.

A table named "**RAW_MATERIAL**" contains information about various raw materials used in inventory management. **MATERIAL_ID**: This column stores a unique identifier for each raw material in the table. **MATERIAL_NAME**: This column stores the name or description of the raw material. **PRICE**: This column stores the price per unit of the raw material. It contains numerical data representing the cost of purchasing one unit of the material. UNIT: This column stores the unit of measurement for the raw material's quantity. It specifies how the raw material is measured, such as kilograms, litres, meters, etc.

A table named "**Vendors**" contains information about vendors or suppliers providing raw materials to a company. **VENDOR_ID**: This column stores a unique identifier for each vendor in the table. It is typically a numeric or alphanumeric value assigned to uniquely identify each vendor. **VENDOR_NAME**: This column stores the name of the vendor or supplier. It contains textual data representing the specific name of the company or individual supplying the raw materials. **MATERIAL_ID**: This column stores the unique identifier of the raw material supplied by the vendor. It serves as a foreign key referencing the **MATERIAL_ID** column in the **RAW_MATERIAL** table, establishing a relationship between the vendors and the specific raw materials they supply.

The "**Expense**" table is a comprehensive database that tracks various expenses incurred by Frozen Custard over several years. **EXPENSE_ID**: This column stores a unique identifier for each expense entry in the table. It is typically a numeric or alphanumeric value assigned to uniquely identify each expense record. **EXPENSE_NAME**: This column stores the name or description of the expense. It contains textual data representing the specific purpose or nature of the expense, such as "Office Supplies," "Utilities," "Salaries," etc.**EXPENSE_2022**, **EXPENSE_2021**, **EXPENSE_2020**, **EXPENSE_2019**, **EXPENSE_2018**, **EXPENSE_2017**, **EXPENSE_2016**: These columns store the monetary values of expenses incurred in the respective years (2022, 2021, 2020, 2019, 2018, 2017, and 2016). Each column represents the total expense amount for the specific year mentioned in the column name. **QUANTITY_2022**, **QUANTITY_2021**, **QUANTITY_2020**, **QUANTITY_2019**, **QUANTITY_2018**,

QUANTITY_2017, QUANTITY_2016: These columns store the quantities or numerical values related to the expenses incurred in the respective years.

The "**Marketing**" table is a database that tracks marketing activities and related expenses within an organization. **MARKETING_ID**: This column stores a unique identifier for each marketing activity or campaign. This **ID** serves as the primary key for the table, ensuring each marketing activity is uniquely identified. **MARKETING_TYPE** column stores the type or category of the marketing activity. It contains textual data representing the specific marketing strategy or campaign category, such as "Online Advertising," "Social Media Campaign," "Print Media," etc. This column helps categorize and differentiate various marketing initiatives. **EXPENSE_ID** column stores the identifier of the expense related to the marketing activity. It serves as a foreign key referencing the **EXPENSE_ID** column in the Expense table.

4.2. Relationship Sets including Primary Key, cardinality, participation, attributes.

4.2.1. Sells Table

- Primary Key: The primary key of the "**Sells**" table can be a combination of the following columns: **SKU_ID**, **Product_ID**, and **Employee_id**. This composite key ensures uniqueness in the Sells table.
- Cardinality: The relationship between **SKU_ID** in the "**Sells**" table and **Product_ID** in the "**Product**" table is typically many-to-one because multiple SKUs (Stock Keeping Units) can correspond to a single product. However, this depends on the specific business context and how products are managed in the system.
- The relationship between **Employee_id** in the "**Sells**" table and **Employee_ID** in the "**Employees**" table is typically many-to-one because multiple sales can be made by the same employee.
- Participation: Assuming that every sale must be recorded, the participation of SKU_ID and **Employee_id** would be total. This means every record in the "Sells" table must have a corresponding **SKU_ID** and **Employee_id**, ensuring that all sales are accounted for.
- The participation of **Product_ID** might be partial, depending on the business rules. If every SKU must be associated with a product (which is common), then **Product_ID** would also have total participation.

4.2.2. Generates Table:

- **Primary Key:** The primary key of the "Generates" table can be a combination of the following columns: **PRODUCT_ID**, **SKU_ID**, and **SALE_ID**. This composite key ensures uniqueness in the Generates table.
- **Cardinality:** The relationship between **PRODUCT_ID** in the "Generates" table and **PRODUCT_ID** in the "Product" table is typically **one-to-many** because one product can have multiple SKUs associated with it, and each SKU can be linked to different sales
- The relationship between **SALE_ID** in the "Generates" table and **SALE_ID** in a "Sales" table is typically **many-to-one** because multiple SKUs can be sold in a single sale.
- **Participation:** Assuming that every sale must be recorded, and every product SKU must be associated with a sale, the participation of **PRODUCT_ID** and **SALE_ID** would be **total**. This means every record in the "Generates" table must have a corresponding **PRODUCT_ID** and **SALE_ID**, ensuring that all sales and products are accounted for.

- The participation of **SKU_ID** would also be **total** if each SKU must be associated with a specific sale.

4.2.3. Uses Table:

- **Primary Key:** The primary key of the "Uses" table can be a combination of the following columns: **Material_ID**, **Product_ID**, and **SKU_ID**. This composite key ensures uniqueness in the Uses table.
- **Cardinality:** The relationship between **Material_ID** in the "Uses" table and **Material_ID** in the "**Raw_Material**" table is typically **many-to-one** because multiple SKUs can be made using the same material.
- The relationship between **Product_ID** in the "Uses" table and **Product_ID** in the "Product" table is typically **many-to-one** because multiple SKUs can be associated with the same product.
- **Participation:** Assuming that every product SKU must be associated with a material, the participation of **Material_ID** and **Product_ID** would be **total**. This means every record in the "Uses" table must have a corresponding Material_ID and Product_ID, ensuring that all SKUs are properly linked to materials and products.
- The participation of **SKU_ID** would depend on the business rules. If every SKU must be associated with a specific inventory item, then the participation of **SKU_ID** would be **total**.

4.2.4. Vendor_Expenses Table:

- **Primary Key:** The primary key of the "vendor_expenses" table can be a combination of the following columns: **EXPENSE_ID** and **VENDOR_ID**. This composite key ensures uniqueness in the vendor_expenses table.
- **Cardinality:** The relationship between **EXPENSE_ID** in the "vendor_expenses" table and **EXPENSE_ID** in the "Expense" table is typically **one-to-many** because one expense can be associated with multiple vendors. However, this depends on the specific business context and how expenses are managed.
- The relationship between **VENDOR_ID** in the "vendor_expenses" table and **VENDOR_ID** in the "Vendors" table could also be **one-to-many** if one vendor can be associated with multiple expenses.
- **Participation:** Assuming that every expense must be associated with a vendor and every vendor must have incurred expenses, the participation of **EXPENSE_ID** and **VENDOR_ID** would be **total**. This means every record in the "vendor_expenses" table must have a corresponding EXPENSE**_ID** and **VENDOR_ID**, ensuring that all expenses and vendors are properly linked.

5. Relational Data Model: Display the Relational Schema.

The relational schema will allow us to set the framework for implementation of our DBMS. Once the E-R diagram is fully understood and structured, then we will convert the Entity Relationship Diagram to a set of Relational Schema, which will allow us to set the framework for the implementation of the database later. These Relation schemas will be converted into a visual representation using the Database Schema diagram, shown in the image at the end of this report.

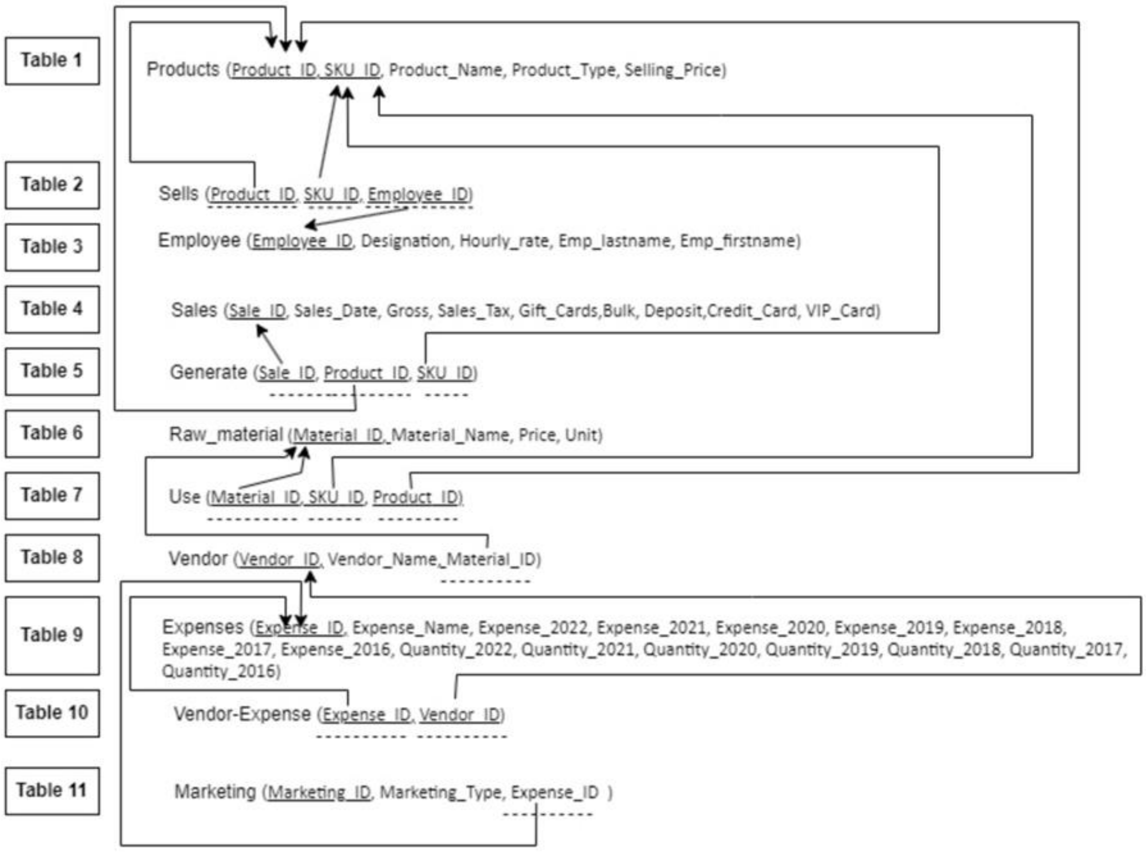


Table 1 Products (Product_ID, SKU_ID, Product_Name, Product_Type, Selling_Price)

Table 2 Sells (Product_ID, SKU_ID, Employee_ID)

Table 3 Employee (Employee_ID, Designation, Hourly_rate, Emp_lastname, Emp_firstname)

Table 4 Sales (Sale_ID, Sales_Date, Gross, Sales_Tax, Gift_Cards,Bulk, Deposit,Credit_Card, VIP_Card)

Table 5 Generate (Sale_ID, Product_ID, SKU_ID)

Table 6 Raw_material (Material_ID, Material_Name, Price, Unit)

Table 7 Use (Material_ID, SKU_ID, Product_ID)

Table 8 Vendor (Vendor_ID, Vendor_Name, Material_ID)

Table 9 Expenses (Expense_ID, Expense_Name, Expense_2022, Expense_2021, Expense_2020, Expense_2019, Expense_2018, Expense_2017, Expense_2016, Quantity_2022, Quantity_2021, Quantity_2020, Quantity_2019, Quantity_2018, Quantity_2017, Quantity_2016)

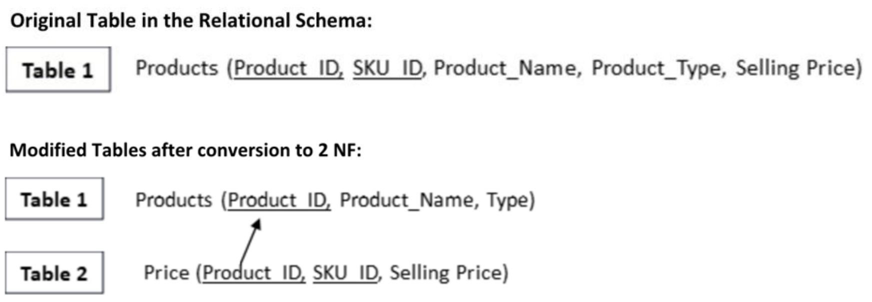
Table 10 Vendor-Expense (Expense_ID, Vendor_ID)

Table 11 Marketing (Marketing_ID, Marketing_Type, Expense_ID)

6. Normalization: Display all relations in 3NF by using the Relational Schema

At this juncture, it is important to check if the database is normalized. Normalization refers to a database design technique that reduces data redundancy and ensures logical storage. It is classified into levels or forms, each of which contain different requirements which a database must fulfil to be called normalized. Our objective is to ensure that the database is at least in the 3rd Normal Form (3NF).

6.1. **1NF**: Each table cell should contain a single value and each record must be unique.
6.2. **2NF**: Database must satisfy 1NF and each table should contain a single column primary key.
6.3. **3NF**: Database must satisfy 2NF and its tables should not evidence any functional transitive dependencies. Functional transitive dependencies occur when a change in a non-Primary Key column cause another non-Primary Key column to change.



Original Table in the Relational Schema:

Table 1 Products (Product_ID, SKU_ID, Product_Name, Product_Type, Selling Price)

Modified Tables after conversion to 2 NF:

Table 1 Products (Product_ID, Product_Name, Type)

Table 2 Price (Product_ID, SKU_ID, Selling Price)

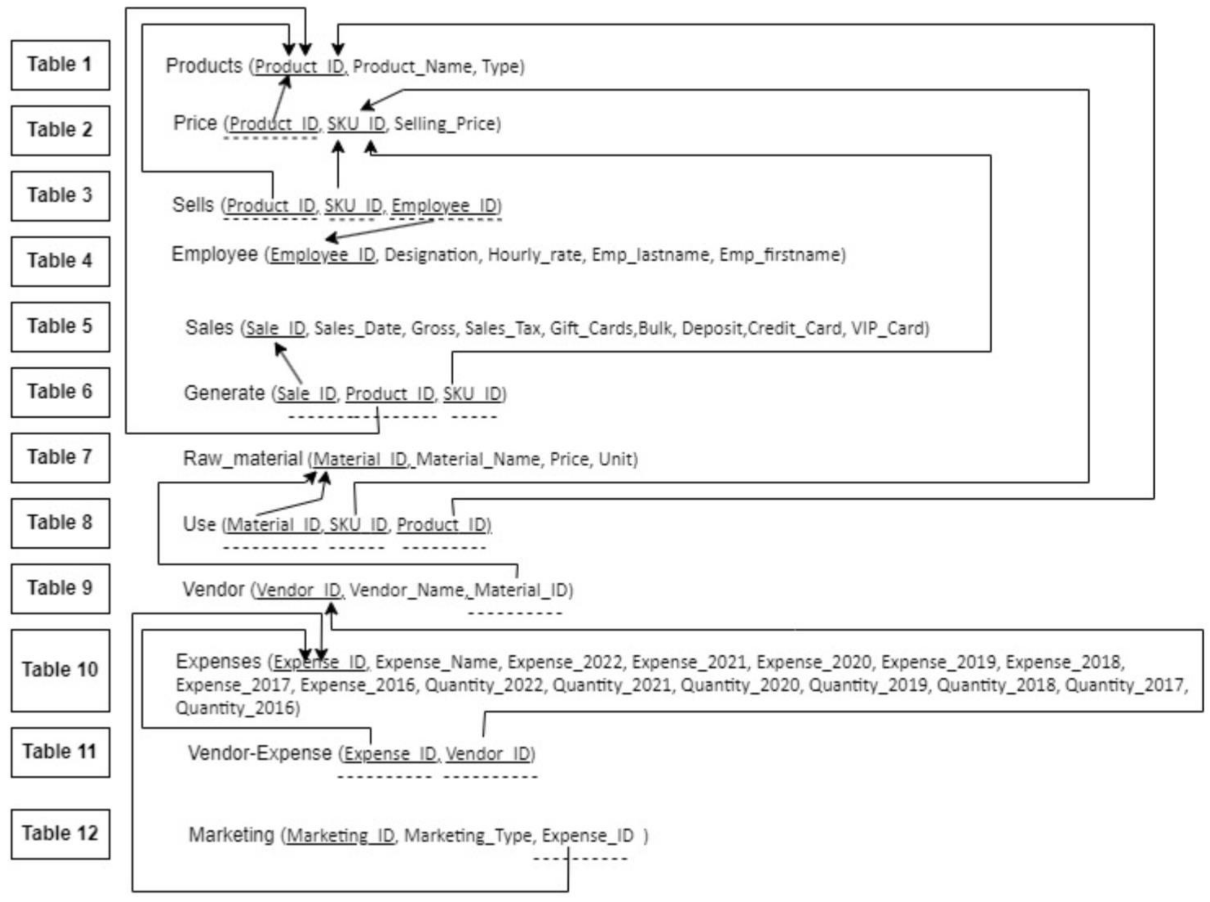


Table 1 Products (Product_ID, Product_Name, Type)

Table 2 Price (Product_ID, SKU_ID, Selling_Price)

Table 3 Sells (Product_ID, SKU_ID, Employee_ID)

Table 4 Employee (Employee_ID, Designation, Hourly_rate, Emp_lastname, Emp_firstname)

Table 5 Sales (Sale_ID, Sales_Date, Gross, Sales_Tax, Gift_Cards,Bulk, Deposit,Credit_Card, VIP_Card)

Table 6 Generate (Sale_ID, Product_ID, SKU_ID)

Table 7 Raw_material (Material_ID, Material_Name, Price, Unit)

Table 8 Use (Material_ID, SKU_ID, Product_ID)

Table 9 Vendor (Vendor_ID, Vendor_Name, Material_ID)

Table 10 Expenses (Expense_ID, Expense_Name, Expense_2022, Expense_2021, Expense_2020, Expense_2019, Expense_2018, Expense_2017, Expense_2016, Quantity_2022, Quantity_2021, Quantity_2020, Quantity_2019, Quantity_2018, Quantity_2017, Quantity_2016)

Table 11 Vendor-Expense (Expense_ID, Vendor_ID)

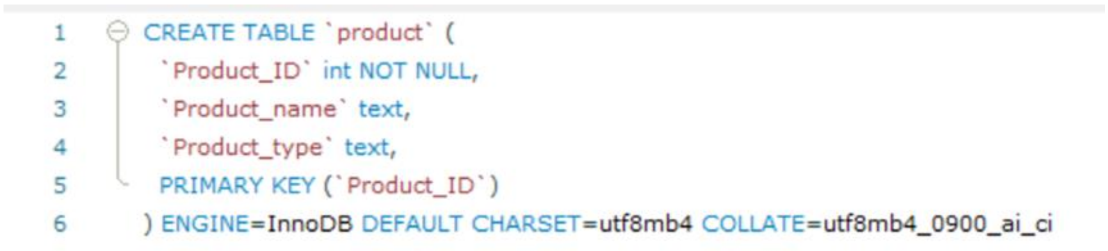
Table 12 Marketing (Marketing_ID, Marketing_Type, Expense_ID)

Sr. No.	Entity	1NF	2NF	3NF
1.	**Employees**	Satisfied as each column is atomic.	Satisfied as Primary Key: Employee_ID, consists of a single attribute.	Satisfied as no functional Transitive dependency can be seen. All the columns are dependent on only and only the Primary Key.
2.	**Expense**	Satisfied as each column is atomic. If the same book is in different languages, it is counted as separate books.	Satisfied as Primary Key: Expense_ID, Consists of a single attribute.	Satisfied as no functional transitive dependency can be seen. All of the columns are dependent on only and only the Primary Key.
3.	**Generate**	Satisfied as each column is single-valued.	Satisfied as Primary Key: Product_ID, SKU_ID, Sale_ID Consists of a single attribute.	Satisfied as no functional transitive dependency can be seen. All of the columns are dependent on only and only the Primary Key.
4.	**Marketing**	Satisfied as each column is single-valued.	Satisfied as Primary Key: Marketing_ID, Expenses_ID consists of a single attribute.	Satisfied as no functional transitive dependency can be seen.
5.	**Price**	Satisfied as each column is single-valued.	Satisfied as Primary Key: Product_ID, SKU_ID consists of a single attribute.	Satisfied as no functional transitive dependency can be seen.
6.	**Product**	Satisfied as each column is single-valued.	Satisfied as Primary Key: Product_ID, consists of a single attribute.	Satisfied as no functional transitive dependency can be seen.
7.	**Raw_material**	Satisfied as each column is single-valued.	Satisfied as Primary Key: Material_ID, consists of a single attribute.	Satisfied as no functional transitive dependency can be seen.
8.	**Sales**	Satisfied as each column is single-valued.	Satisfied as Primary Key: Sale_ID, consists of a single attribute.	Satisfied as no functional transitive dependency can be seen.
9.	**Sells**	Satisfied as each column is single-valued.	Satisfied as table is made up of Primary Keys: SKU_ID, Product_ID and Employee_ID.	Satisfied as no functional transitive dependency can be seen.
10.	**Uses**	Satisfied as each column is single-valued.	Satisfied as table is made up of Primary Keys: SKU_ID, Product_ID and Material_ID.	Satisfied as no functional transitive dependency can be seen.
11.	**Vendors**	Satisfied as each column is single-valued.	Satisfied as table is made up of Primary Key: Vendor_ID.	Satisfied as no functional transitive dependency can be seen.
12.	**Vendor_Expense**	Satisfied as each column is single-valued.	Satisfied as table is made up of Primary Keys: Vendor_ID and Expense_ID.	Satisfied as no functional transitive dependency can be seen.

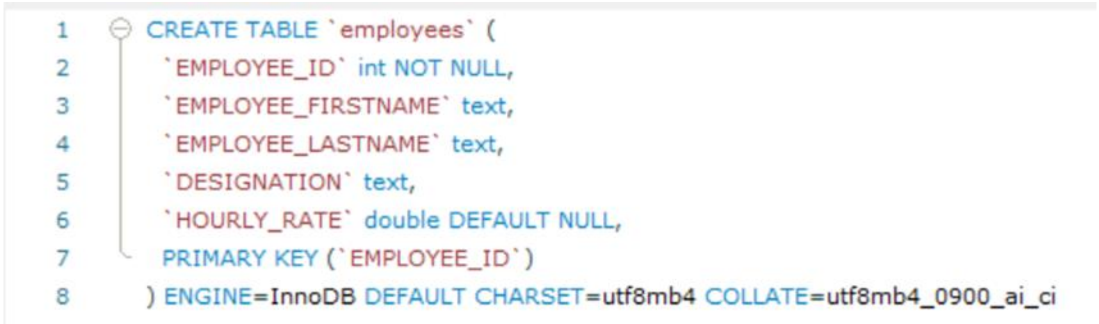
7. Implementation (Creating the database in SQL)

Once the database has been designed conceptually and logically, we create the relevant Tables and Views in SQL as per our ERD and Relational Schema. While creating the tables in SQL, data type for every attribute was selected so that it complies with the kind of data that must be inserted in a Library Database. Below are the examples of how we created entities, relationships, and their corresponding associative identities:

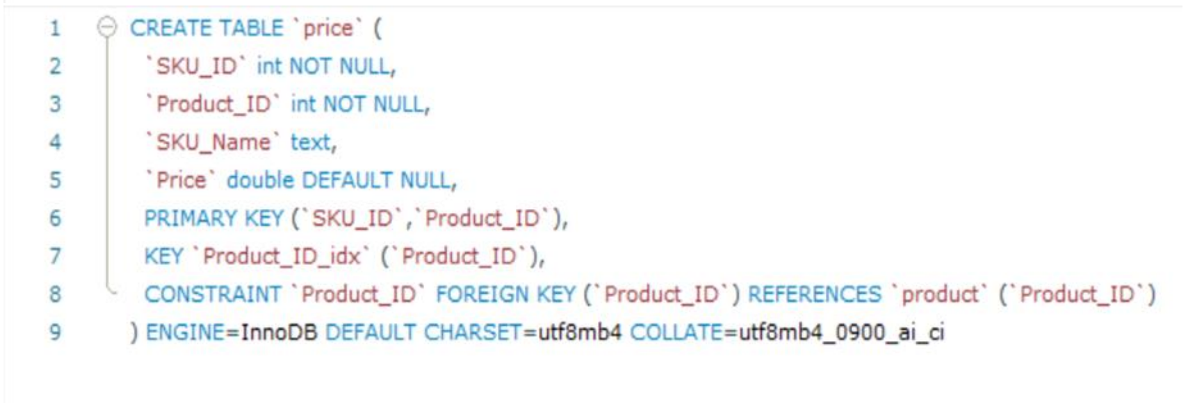
7.1. Using CREATE to generate entity tables.



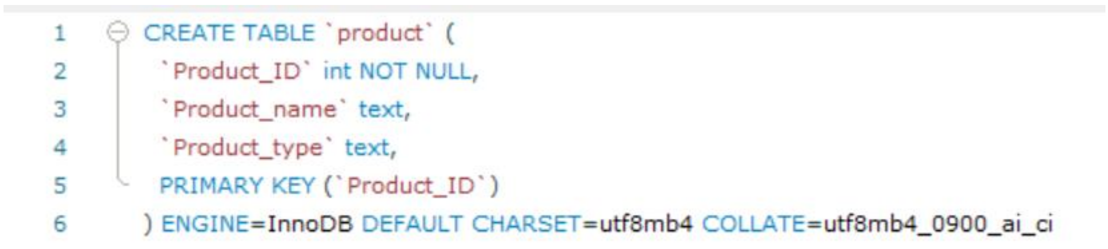
```sql
CREATE TABLE `product` (
  `Product_ID` int NOT NULL,
  `Product_name` text,
  `Product_type` text,
  PRIMARY KEY (`Product_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



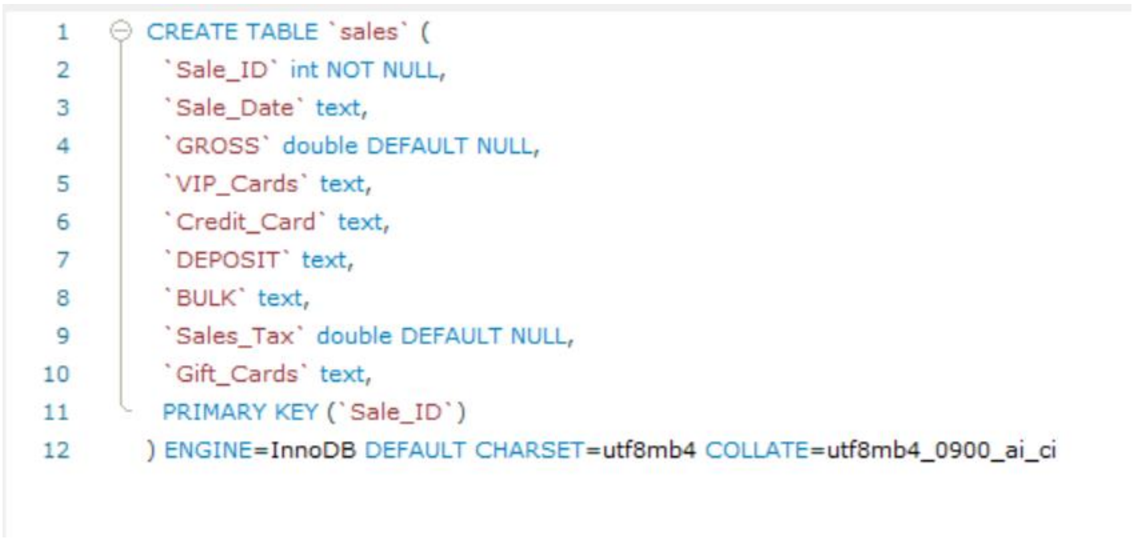
```sql
CREATE TABLE `employees` (
  `EMPLOYEE_ID` int NOT NULL,
  `EMPLOYEE_FIRSTNAME` text,
  `EMPLOYEE_LASTNAME` text,
  `DESIGNATION` text,
  `HOURLY_RATE` double DEFAULT NULL,
  PRIMARY KEY (`EMPLOYEE_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



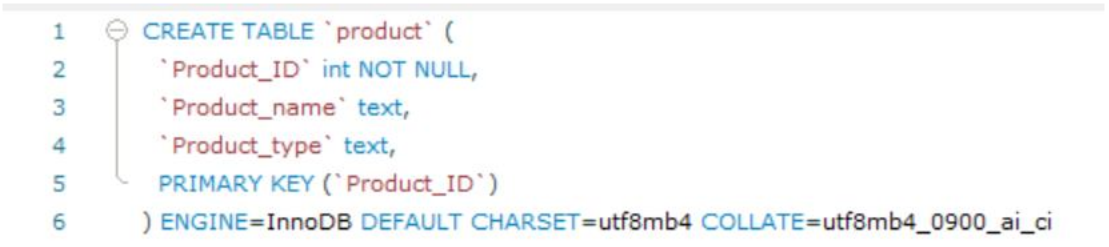
```sql
CREATE TABLE `price` (
  `SKU_ID` int NOT NULL,
  `Product_ID` int NOT NULL,
  `SKU_Name` text,
  `Price` double DEFAULT NULL,
  PRIMARY KEY (`SKU_ID`,`Product_ID`),
  KEY `Product_ID_idx` (`Product_ID`),
  CONSTRAINT `Product_ID` FOREIGN KEY (`Product_ID`) REFERENCES `product` (`Product_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



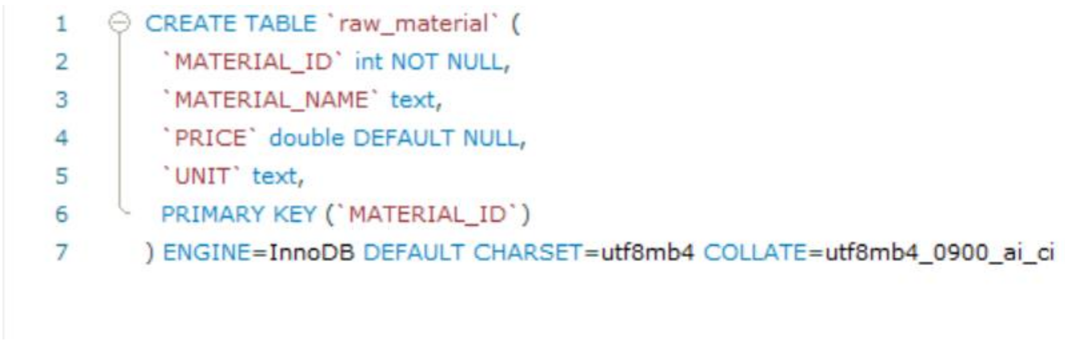
```sql
CREATE TABLE `product` (
  `Product_ID` int NOT NULL,
  `Product_name` text,
  `Product_type` text,
  PRIMARY KEY (`Product_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



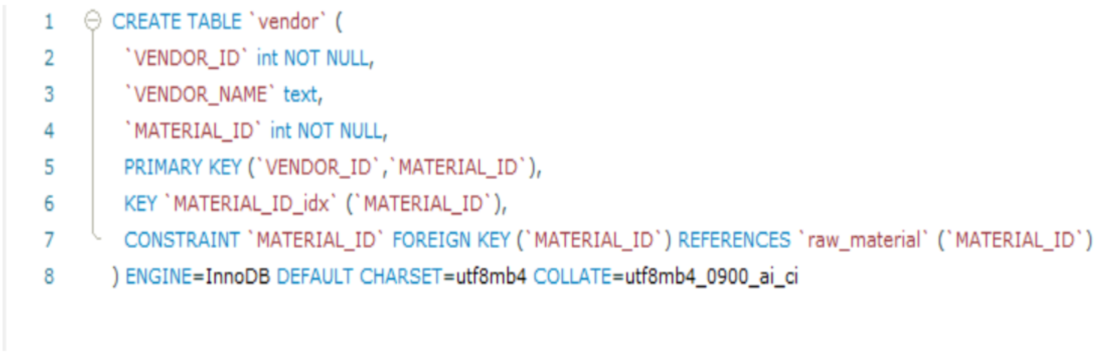
```sql
CREATE TABLE `sales` (
  `Sale_ID` int NOT NULL,
  `Sale_Date` text,
  `GROSS` double DEFAULT NULL,
  `VIP_Cards` text,
  `Credit_Card` text,
  `DEPOSIT` text,
  `BULK` text,
  `Sales_Tax` double DEFAULT NULL,
  `Gift_Cards` text,
  PRIMARY KEY (`Sale_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



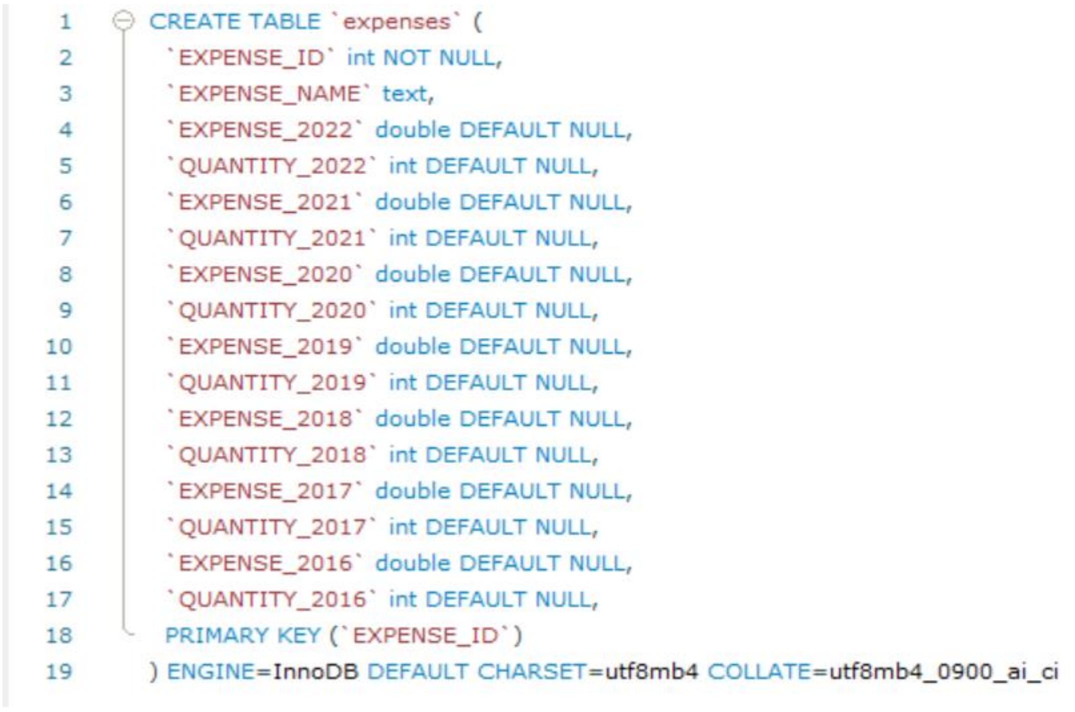
```sql
CREATE TABLE `product` (
  `Product_ID` int NOT NULL,
  `Product_name` text,
  `Product_type` text,
  PRIMARY KEY (`Product_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



```sql
CREATE TABLE `raw_material` (
  `MATERIAL_ID` int NOT NULL,
  `MATERIAL_NAME` text,
  `PRICE` double DEFAULT NULL,
  `UNIT` text,
  PRIMARY KEY (`MATERIAL_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

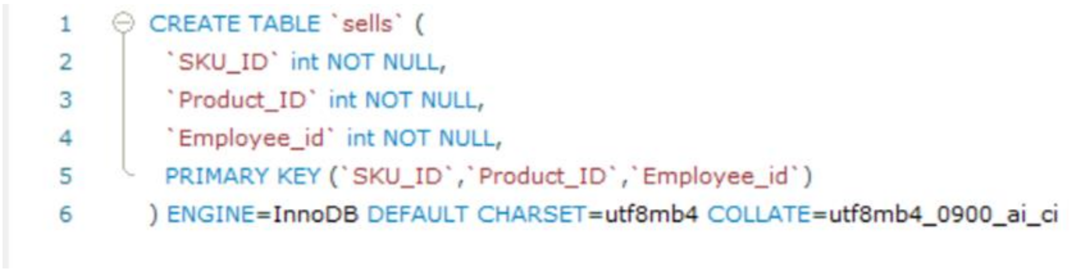


```sql
CREATE TABLE `vendor` (
  `VENDOR_ID` int NOT NULL,
  `VENDOR_NAME` text,
  `MATERIAL_ID` int NOT NULL,
  PRIMARY KEY (`VENDOR_ID`,`MATERIAL_ID`),
  KEY `MATERIAL_ID_idx` (`MATERIAL_ID`),
  CONSTRAINT `MATERIAL_ID` FOREIGN KEY (`MATERIAL_ID`) REFERENCES `raw_material` (`MATERIAL_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

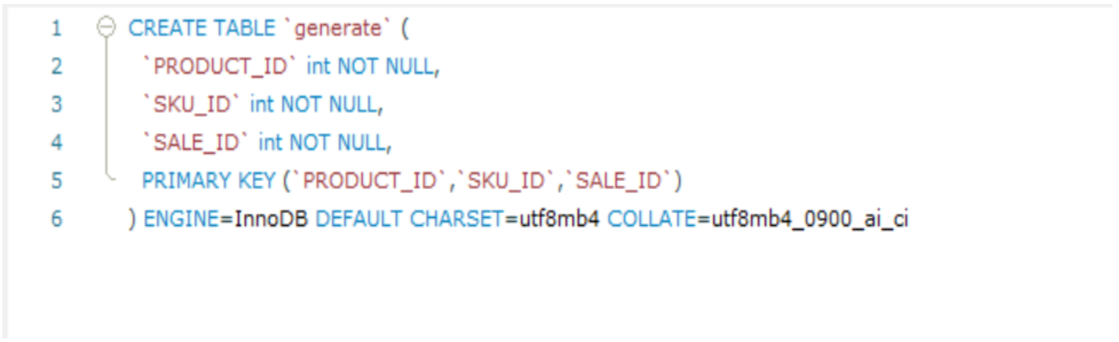


```sql
CREATE TABLE `expenses` (
  `EXPENSE_ID` int NOT NULL,
  `EXPENSE_NAME` text,
  `EXPENSE_2022` double DEFAULT NULL,
  `QUANTITY_2022` int DEFAULT NULL,
  `EXPENSE_2021` double DEFAULT NULL,
  `QUANTITY_2021` int DEFAULT NULL,
  `EXPENSE_2020` double DEFAULT NULL,
  `QUANTITY_2020` int DEFAULT NULL,
  `EXPENSE_2019` double DEFAULT NULL,
  `QUANTITY_2019` int DEFAULT NULL,
  `EXPENSE_2018` double DEFAULT NULL,
  `QUANTITY_2018` int DEFAULT NULL,
  `EXPENSE_2017` double DEFAULT NULL,
  `QUANTITY_2017` int DEFAULT NULL,
  `EXPENSE_2016` double DEFAULT NULL,
  `QUANTITY_2016` int DEFAULT NULL,
  PRIMARY KEY (`EXPENSE_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

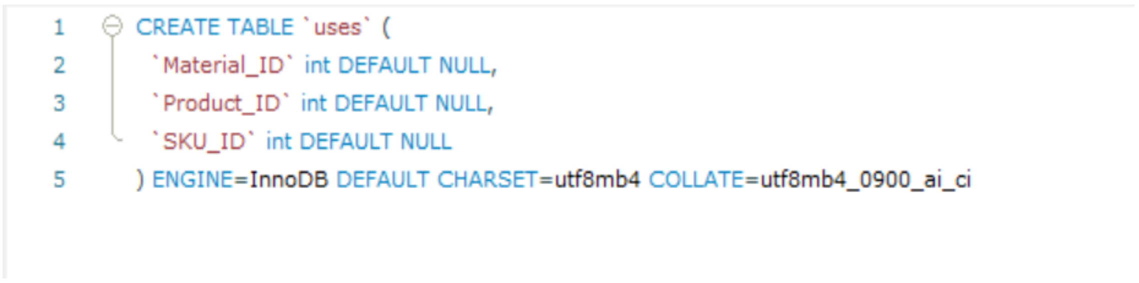
7.2. Using CREATE to generate associative entity tables.



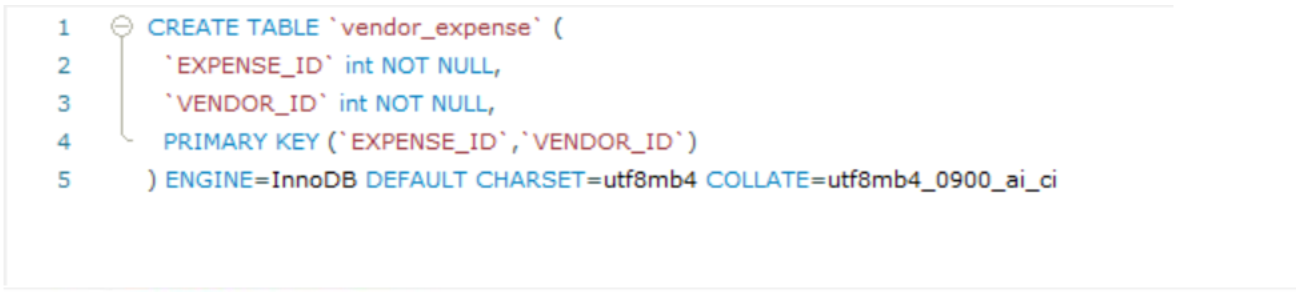
```sql
CREATE TABLE `sells` (
  `SKU_ID` int NOT NULL,
  `Product_ID` int NOT NULL,
  `Employee_id` int NOT NULL,
  PRIMARY KEY (`SKU_ID`,`Product_ID`,`Employee_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



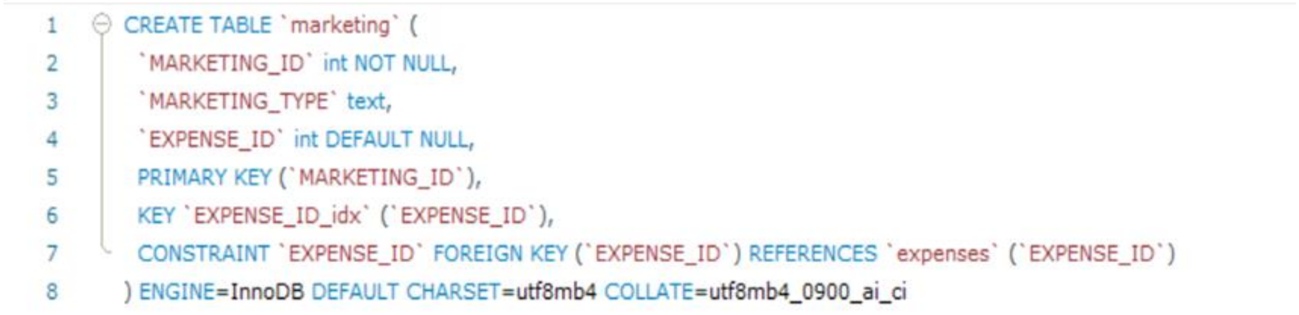
```sql
CREATE TABLE `generate` (
  `PRODUCT_ID` int NOT NULL,
  `SKU_ID` int NOT NULL,
  `SALE_ID` int NOT NULL,
  PRIMARY KEY (`PRODUCT_ID`,`SKU_ID`,`SALE_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



```sql
CREATE TABLE `uses` (
  `Material_ID` int DEFAULT NULL,
  `Product_ID` int DEFAULT NULL,
  `SKU_ID` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

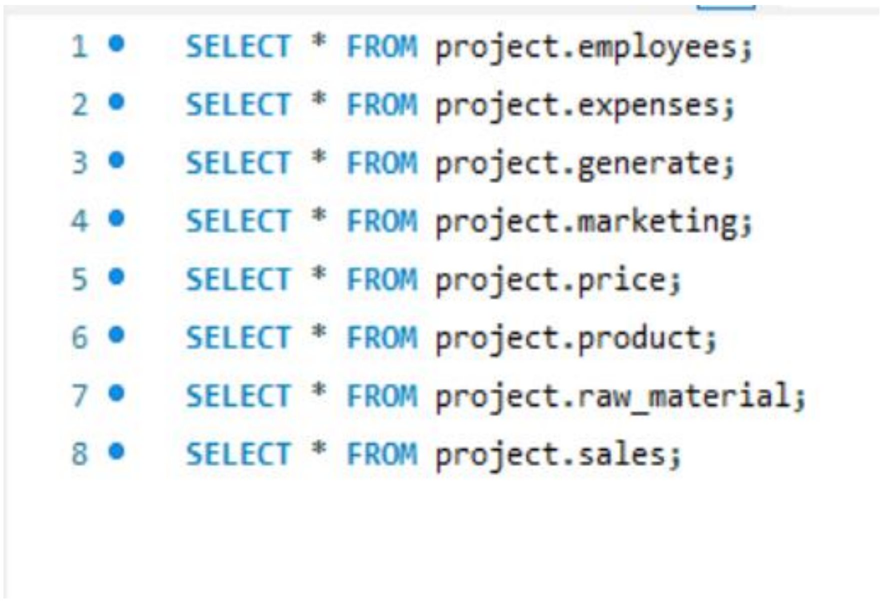


```
CREATE TABLE `vendor_expense` (
  `EXPENSE_ID` int NOT NULL,
  `VENDOR_ID` int NOT NULL,
  PRIMARY KEY (`EXPENSE_ID`,`VENDOR_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

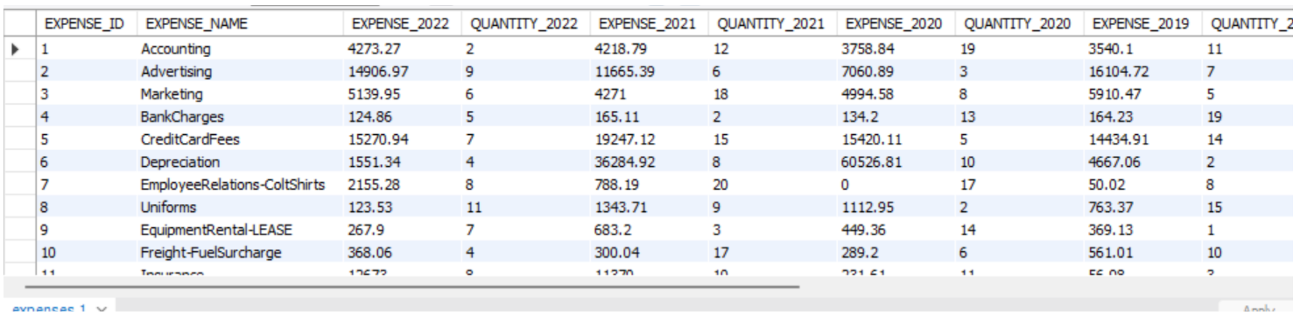


```
CREATE TABLE `marketing` (
  `MARKETING_ID` int NOT NULL,
  `MARKETING_TYPE` text,
  `EXPENSE_ID` int DEFAULT NULL,
  PRIMARY KEY (`MARKETING_ID`),
  KEY `EXPENSE_ID_idx` (`EXPENSE_ID`),
  CONSTRAINT `EXPENSE_ID` FOREIGN KEY (`EXPENSE_ID`) REFERENCES `expenses` (`EXPENSE_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

7.3. Using SELECT to view corresponding tables.



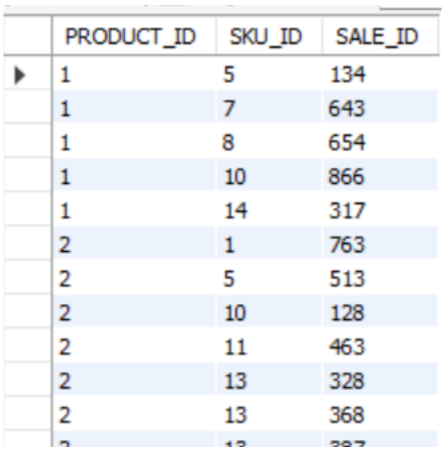
```
SELECT * FROM project.employees;
SELECT * FROM project.expenses;
SELECT * FROM project.generate;
SELECT * FROM project.marketing;
SELECT * FROM project.price;
SELECT * FROM project.product;
SELECT * FROM project.raw_material;
SELECT * FROM project.sales;
```



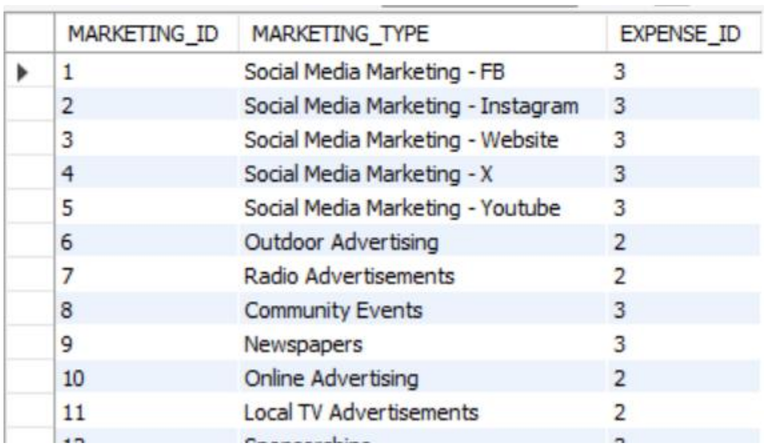
EXPENSE_ID	EXPENSE_NAME	EXPENSE_2022	QUANTITY_2022	EXPENSE_2021	QUANTITY_2021	EXPENSE_2020	QUANTITY_2020	EXPENSE_2019	QUANTITY_2
1	Accounting	4273.27	2	4218.79	12	3758.84	19	3540.1	11
2	Advertising	14906.97	9	11665.39	6	7060.89	3	16104.72	7
3	Marketing	5139.95	6	4271	18	4994.58	8	5910.47	5
4	BankCharges	124.86	5	165.11	2	134.2	13	164.23	19
5	CreditCardFees	15270.94	7	19247.12	15	15420.11	5	14434.91	14
6	Depreciation	1551.34	4	36284.92	8	60526.81	10	4667.06	2
7	EmployeeRelations-ColtShirts	2155.28	8	788.19	20	0	17	50.02	8
8	Uniforms	123.53	11	1343.71	9	1112.95	2	763.37	15
9	EquipmentRental-LEASE	267.9	7	683.2	3	449.36	14	369.13	1
10	Freight-FuelSurcharge	368.06	4	300.04	17	289.2	6	561.01	10
11	Insurance	12673	8	11370	10	221.61	11	56.08	3

expenses 1

Apply



PRODUCT_ID	SKU_ID	SALE_ID
1	5	134
1	7	643
1	8	654
1	10	866
1	14	317
2	1	763
2	5	513
2	10	128
2	11	463
2	13	328
2	13	368

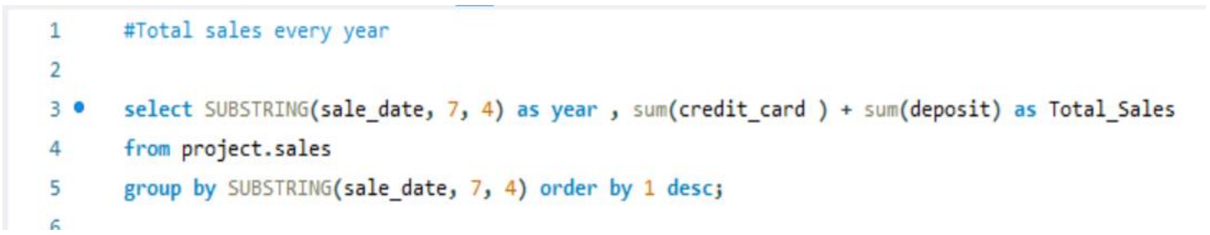


MARKETING_ID	MARKETING_TYPE	EXPENSE_ID
1	Social Media Marketing - FB	3
2	Social Media Marketing - Instagram	3
3	Social Media Marketing - Website	3
4	Social Media Marketing - X	3
5	Social Media Marketing - Youtube	3
6	Outdoor Advertising	2
7	Radio Advertisements	2
8	Community Events	3
9	Newspapers	3
10	Online Advertising	2
11	Local TV Advertisements	2

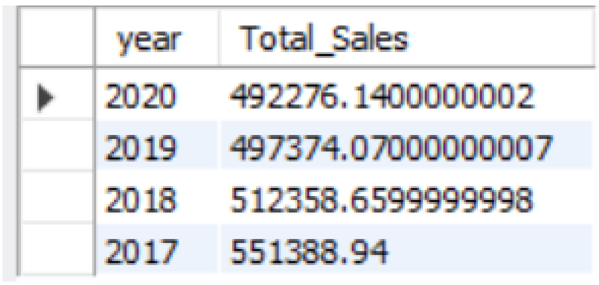
8. Queries:

8.1. Query Objective: show the total sales for years from 2017 to 2020.

The output can act as reference data to evaluate year-on-year total sales for the business and identify the year in which sales were the highest.



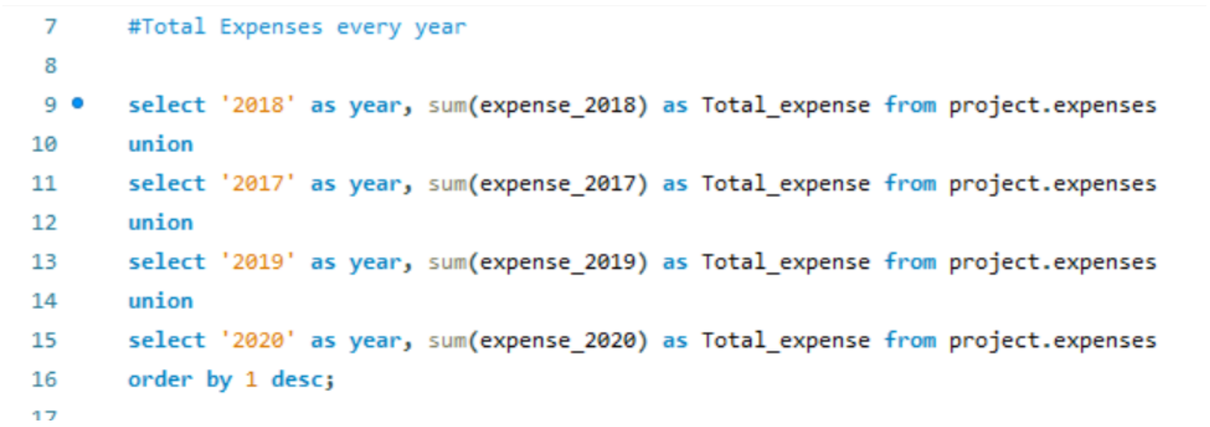
```
#Total sales every year

select SUBSTRING(sale_date, 7, 4) as year , sum(credit_card ) + sum(deposit) as Total_Sales
from project.sales
group by SUBSTRING(sale_date, 7, 4) order by 1 desc;
```



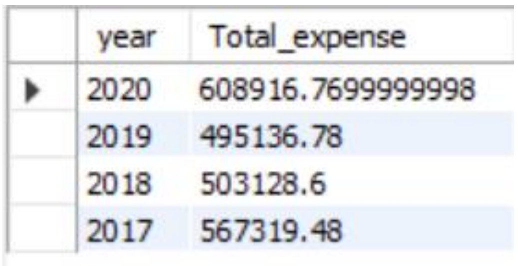
year	Total_Sales
2020	492276.1400000002
2019	497374.07000000007
2018	512358.6599999998
2017	551388.94

8.2. Query Objective: show the total expense for all years from 2017 to 2020.

This query helps us extract the total expense on an annual basis, we can further use the total output to select a year where we want to dig deeper into the expense table and understand where we can cut down on them.



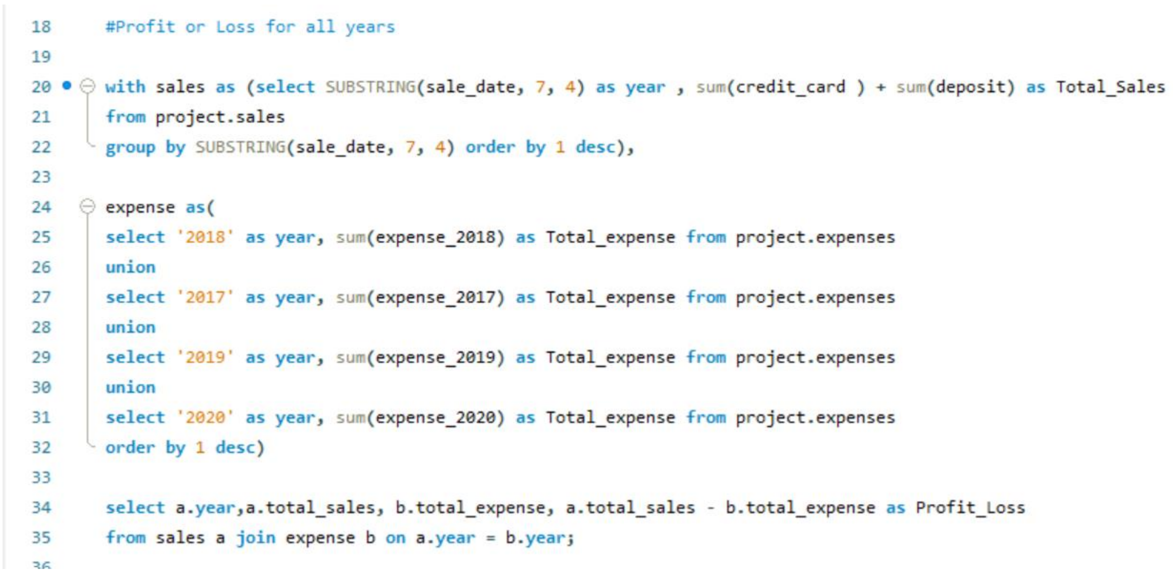
```
#Total Expenses every year

select '2018' as year, sum(expense_2018) as Total_expense from project.expenses
union
select '2017' as year, sum(expense_2017) as Total_expense from project.expenses
union
select '2019' as year, sum(expense_2019) as Total_expense from project.expenses
union
select '2020' as year, sum(expense_2020) as Total_expense from project.expenses
order by 1 desc;
```



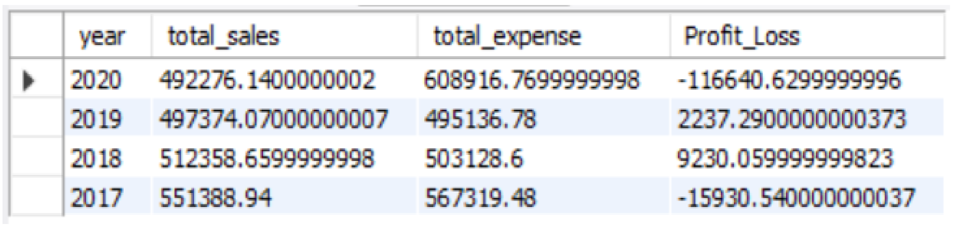
year	Total_expense
2020	608916.7699999998
2019	495136.78
2018	503128.6
2017	567319.48

8.3. Query Objective: show the profit or loss of each year from 2017 to 2020.

In addition to the previous output, it is important also to know the bottom line of the financials. The output not only gives data for profit/loss but also gives the corresponding revenue and expense for deeper reference.



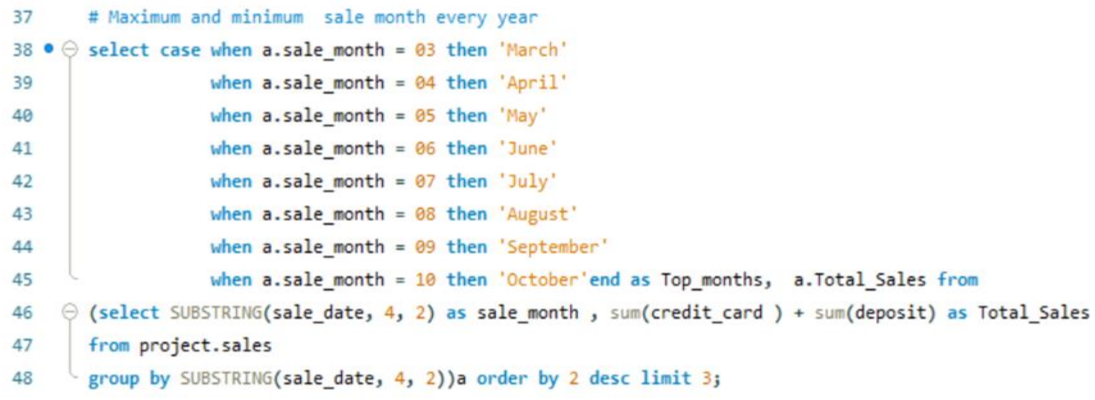
```sql
#Profit or Loss for all years

with sales as (select SUBSTRING(sale_date, 7, 4) as year , sum(credit_card ) + sum(deposit) as Total_Sales
from project.sales
group by SUBSTRING(sale_date, 7, 4) order by 1 desc),

expense as(
select '2018' as year, sum(expense_2018) as Total_expense from project.expenses
union
select '2017' as year, sum(expense_2017) as Total_expense from project.expenses
union
select '2019' as year, sum(expense_2019) as Total_expense from project.expenses
union
select '2020' as year, sum(expense_2020) as Total_expense from project.expenses
order by 1 desc)

select a.year,a.total_sales, b.total_expense, a.total_sales - b.total_expense as Profit_Loss
from sales a join expense b on a.year = b.year;
```



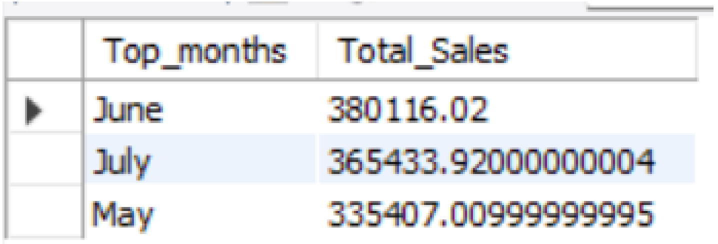
year	total_sales	total_expense	Profit_Loss
2020	492276.1400000002	608916.7699999998	-116640.6299999996
2019	497374.07000000007	495136.78	2237.2900000000373
2018	512358.6599999998	503128.6	9230.059999999823
2017	551388.94	567319.48	-15930.540000000037

8.4. Query Objective: To get the top 3-month sales.

Identifying the top 3 months' sales could help in prioritizing marketing activities and manage inventory efficiently.



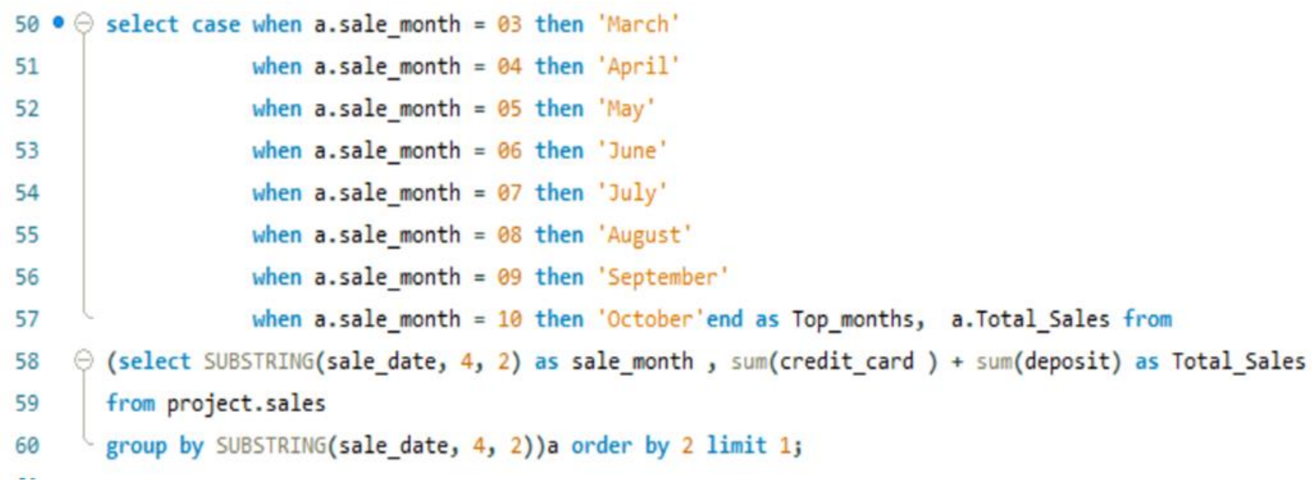
```sql
# Maximum and minimum  sale month every year
select case when a.sale_month = 03 then 'March'
            when a.sale_month = 04 then 'April'
            when a.sale_month = 05 then 'May'
            when a.sale_month = 06 then 'June'
            when a.sale_month = 07 then 'July'
            when a.sale_month = 08 then 'August'
            when a.sale_month = 09 then 'September'
            when a.sale_month = 10 then 'October'end as Top_months,  a.Total_Sales from
(select SUBSTRING(sale_date, 4, 2) as sale_month , sum(credit_card ) + sum(deposit) as Total_Sales
from project.sales
group by SUBSTRING(sale_date, 4, 2))a order by 2 desc limit 3;
```



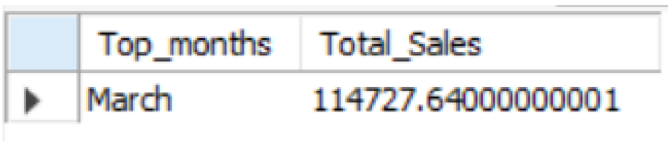
Top_months	Total_Sales
June	380116.02
July	365433.92000000004
May	335407.00999999995

8.5. Query Objective: give the lowest month and its sales.

By identifying the lowest performing month using this query, the business can further dig deep into the sales data and identify patterns to address low sales.



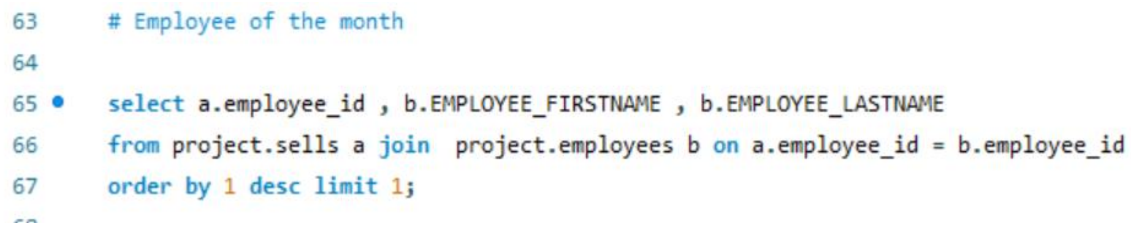
```sql
select case when a.sale_month = 03 then 'March'
            when a.sale_month = 04 then 'April'
            when a.sale_month = 05 then 'May'
            when a.sale_month = 06 then 'June'
            when a.sale_month = 07 then 'July'
            when a.sale_month = 08 then 'August'
            when a.sale_month = 09 then 'September'
            when a.sale_month = 10 then 'October'end as Top_months,  a.Total_Sales from
(select SUBSTRING(sale_date, 4, 2) as sale_month , sum(credit_card ) + sum(deposit) as Total_Sales
from project.sales
group by SUBSTRING(sale_date, 4, 2))a order by 2 limit 1;
```



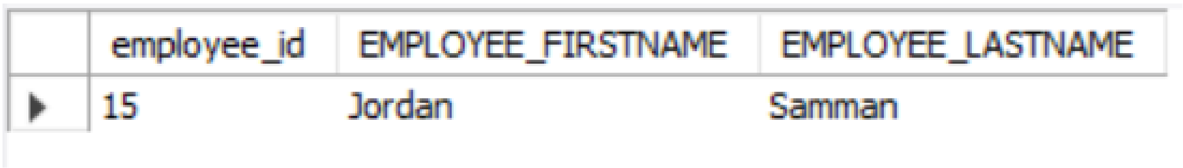
	Top_months	Total_Sales
▶	March	114727.64000000001

8.6. Query Objective: to find the employee of the month based on the number of products sold each month.

Identifying 'Employee of the month' is a key component in employee management. Any incentive tied to performance could further encourage employees to strive to achieve this goal.



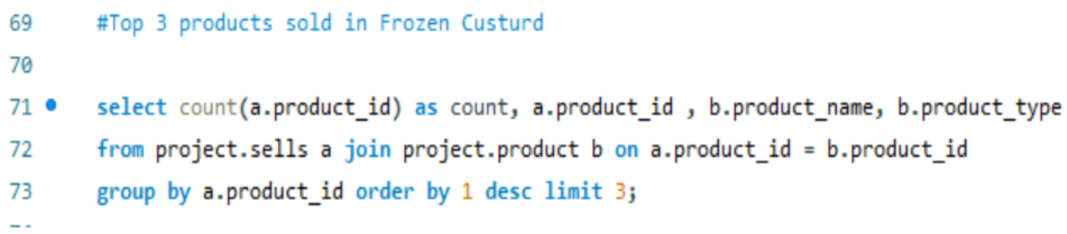
```sql
# Employee of the month

select a.employee_id , b.EMPLOYEE_FIRSTNAME , b.EMPLOYEE_LASTNAME
from project.sells a join  project.employees b on a.employee_id = b.employee_id
order by 1 desc limit 1;
```



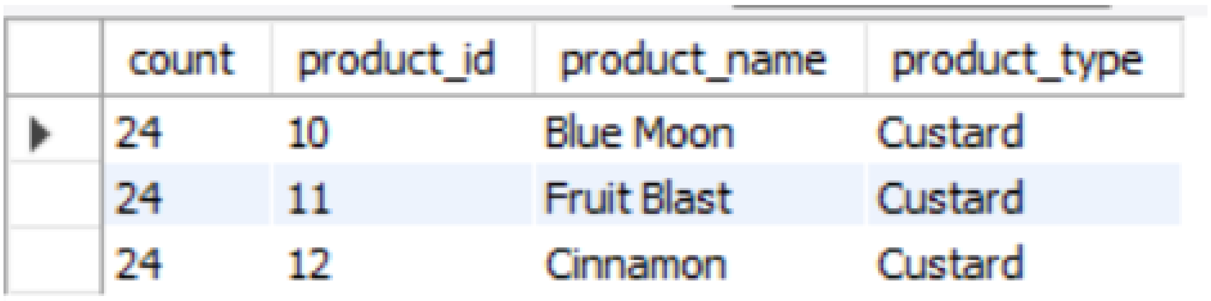
	employee_id	EMPLOYEE_FIRSTNAME	EMPLOYEE_LASTNAME
▶	15	Jordan	Samman

8.7. Query Objective: To find the top 3 most sold products at Frozen Custard

Identifying top 3 sold products would help in managing inventory for flavors, prioritize product promotions and add bargaining leverage when negotiating with vendors.



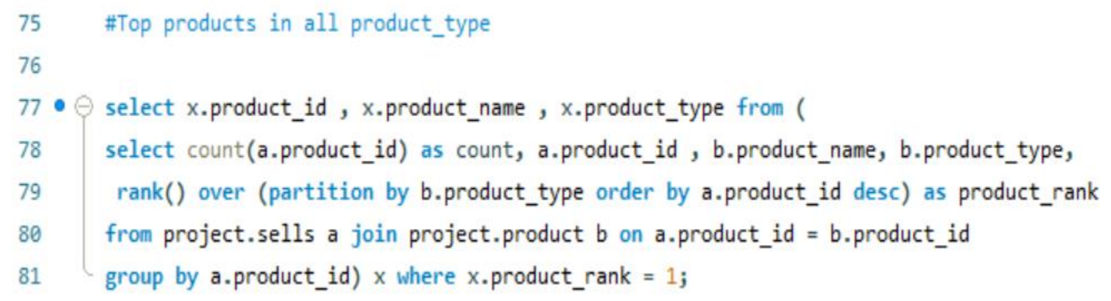
```sql
#Top 3 products sold in Frozen Custurd

select count(a.product_id) as count, a.product_id , b.product_name, b.product_type
from project.sells a join project.product b on a.product_id = b.product_id
group by a.product_id order by 1 desc limit 3;
```



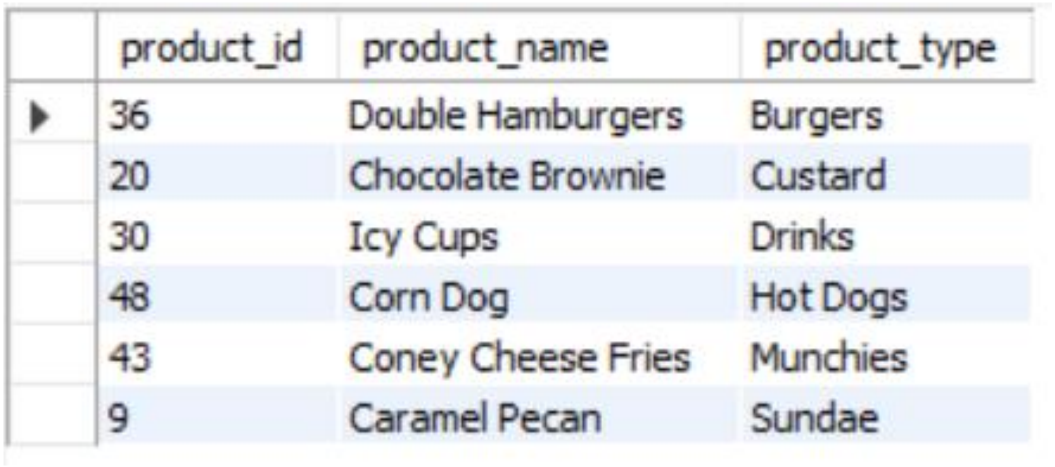
count	product_id	product_name	product_type
24	10	Blue Moon	Custard
24	11	Fruit Blast	Custard
24	12	Cinnamon	Custard

8.8. Query Objective: To find the top products sold in all categories.

Identifying top products outside of the frozen custard menu would help in identifying products that complement well to the frozen custard menu and give inspiration for introducing new products as well.



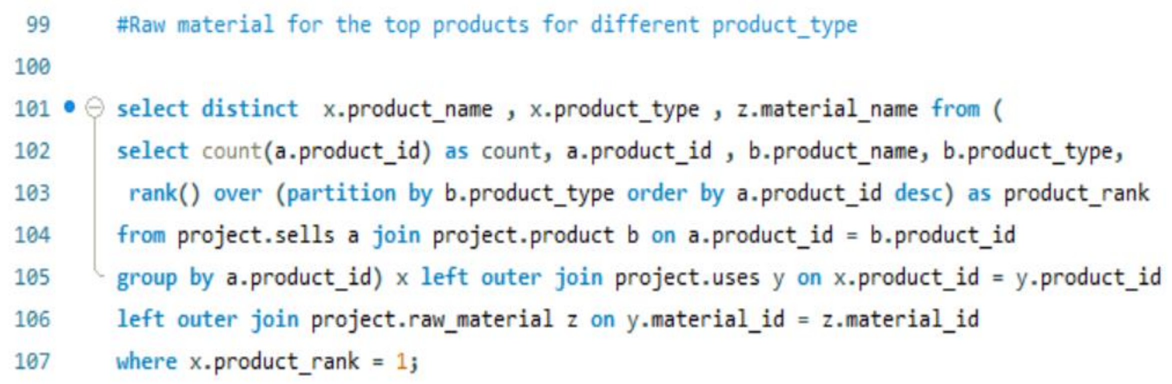
```
#Top products in all product_type

select x.product_id , x.product_name , x.product_type from (
select count(a.product_id) as count, a.product_id , b.product_name, b.product_type,
 rank() over (partition by b.product_type order by a.product_id desc) as product_rank
from project.sells a join project.product b on a.product_id = b.product_id
group by a.product_id) x where x.product_rank = 1;
```



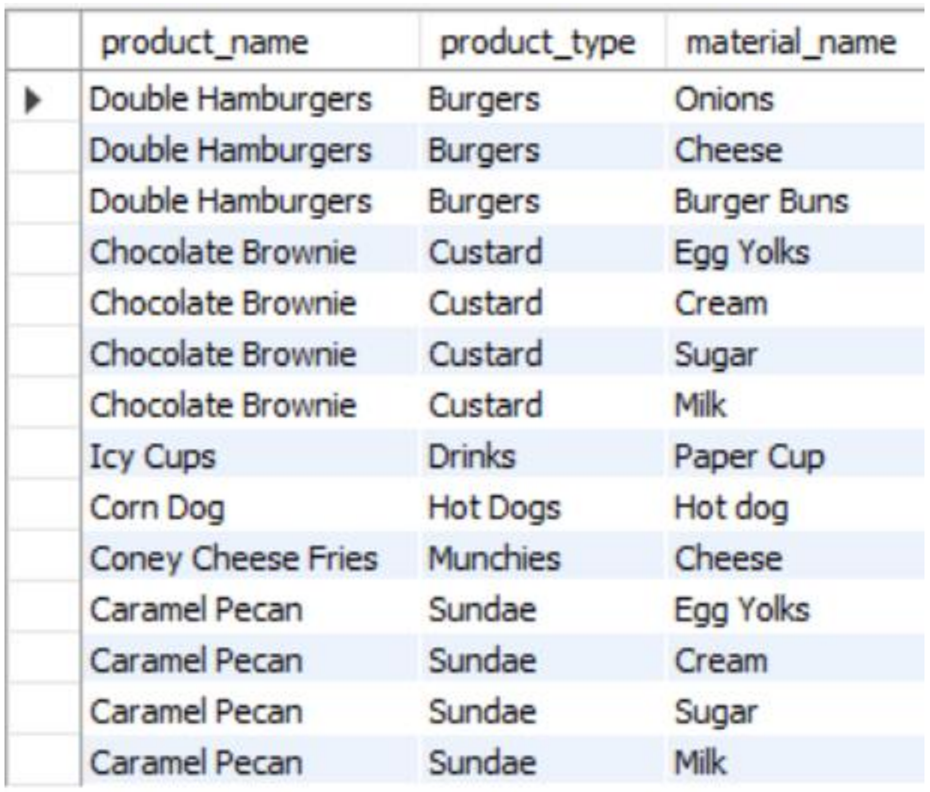
product_id	product_name	product_type
36	Double Hamburgers	Burgers
20	Chocolate Brownie	Custard
30	Icy Cups	Drinks
48	Corn Dog	Hot Dogs
43	Coney Cheese Fries	Munchies
9	Caramel Pecan	Sundae

8.9. Query Objective: To find the raw materials used for most sold products in all categories.

This query is crucial for inventory management of other products as these products are not the primary selling items for the business and mismanagement of inventory could lead to unwanted expenses.



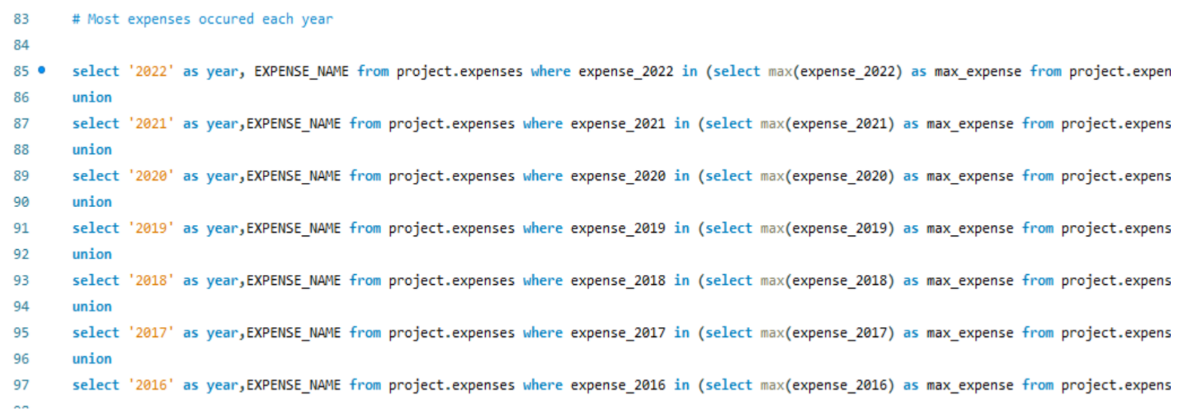
```
#Raw material for the top products for different product_type

select distinct  x.product_name , x.product_type , z.material_name from (
select count(a.product_id) as count, a.product_id , b.product_name, b.product_type,
 rank() over (partition by b.product_type order by a.product_id desc) as product_rank
from project.sells a join project.product b on a.product_id = b.product_id
group by a.product_id) x left outer join project.uses y on x.product_id = y.product_id
left outer join project.raw_material z on y.material_id = z.material_id
where x.product_rank = 1;
```



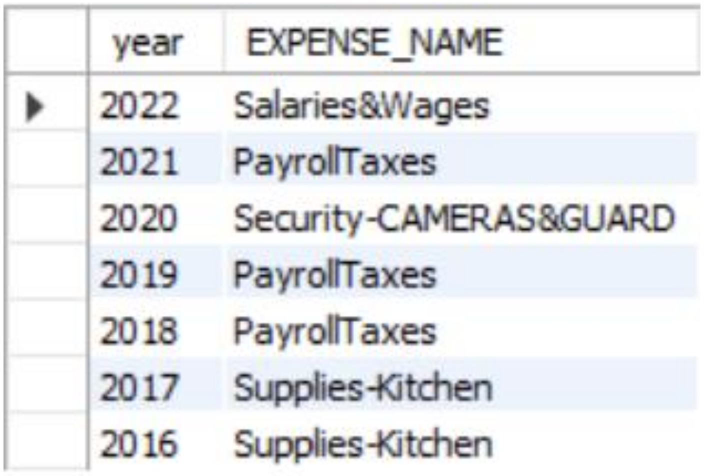
product_name	product_type	material_name
Double Hamburgers	Burgers	Onions
Double Hamburgers	Burgers	Cheese
Double Hamburgers	Burgers	Burger Buns
Chocolate Brownie	Custard	Egg Yolks
Chocolate Brownie	Custard	Cream
Chocolate Brownie	Custard	Sugar
Chocolate Brownie	Custard	Milk
Icy Cups	Drinks	Paper Cup
Corn Dog	Hot Dogs	Hot dog
Coney Cheese Fries	Munchies	Cheese
Caramel Pecan	Sundae	Egg Yolks
Caramel Pecan	Sundae	Cream
Caramel Pecan	Sundae	Sugar
Caramel Pecan	Sundae	Milk

8.10. Query Objective: To find the highest expenses occurred in each year.

As a seasonal business it is important to have a track of expenses. The output from this query would help in identifying the area that costs the business the most.



```
# Most expenses occured each year

select '2022' as year, EXPENSE_NAME from project.expenses where expense_2022 in (select max(expense_2022) as max_expense from project.expen
union
select '2021' as year,EXPENSE_NAME from project.expenses where expense_2021 in (select max(expense_2021) as max_expense from project.expens
union
select '2020' as year,EXPENSE_NAME from project.expenses where expense_2020 in (select max(expense_2020) as max_expense from project.expens
union
select '2019' as year,EXPENSE_NAME from project.expenses where expense_2019 in (select max(expense_2019) as max_expense from project.expens
union
select '2018' as year,EXPENSE_NAME from project.expenses where expense_2018 in (select max(expense_2018) as max_expense from project.expens
union
select '2017' as year,EXPENSE_NAME from project.expenses where expense_2017 in (select max(expense_2017) as max_expense from project.expens
union
select '2016' as year,EXPENSE_NAME from project.expenses where expense_2016 in (select max(expense_2016) as max_expense from project.expens
```



year	EXPENSE_NAME
2022	Salaries&Wages
2021	PayrollTaxes
2020	Security-CAMERAS&GUARD
2019	PayrollTaxes
2018	PayrollTaxes
2017	Supplies-Kitchen
2016	Supplies-Kitchen

9. Business Recommendations

- Use the employee data in the database to create efficient work schedules and monitor performance. Implement training and development programs based on employee data to enhance skills and customer service.
- Monitor and analyze expense data to identify areas for cost savings and efficiency improvements. Regularly review operational expenses and vendor contracts to optimize spending.
- Utilize sales data to tailor marketing campaigns and promotions. Implement email marketing, social media engagement, and loyalty programs to attract and retain customers.
- At later stages, implement a CRM system within the database to integrate and track customer interactions on online and offline channels. This will enable personalized marketing campaigns, loyalty programs, and targeted promotions.
- Provide ongoing training and support to staff members responsible for using the database. Encourage feedback from employees and customers regarding the database system's usability and functionality. Continuously improve the system based on user input.