Frozen Custard System Management

Using customized DBMS

Team 3 – Session 9:50 AM

The Original Frozen Custard– DBMS Course Project

Hummarah Shahzad

Avishek Dasgupta

Samarth Bansal

Siddhant Treasure

Mannat Singh

Kai

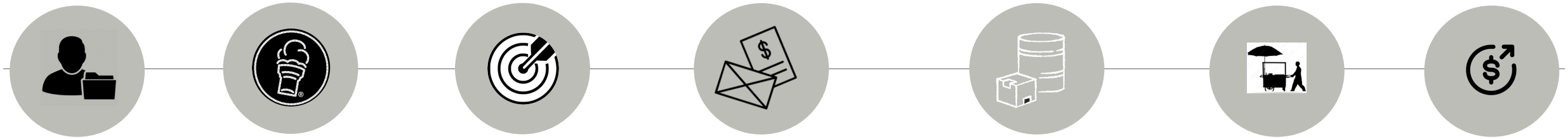
Project Map

Original Frozen Custard– DBMS Course Project

1. **Business Problem –** defining business objectives for the project
2. **Conceptual Design –** developing ER diagram for the problem
3. **Logical Design –** developing relational schema from ERD
4. **Normalization –** ensuring logical design is in 4NF before SQL modeling
5. **Implementing Database –** creating database in SQL
6. **Querying Database –** developing queries to meet our business objectives
7. **Business Insights –** Business findings and way forward

Business Problem

Our project aims to model a centralized data base for a local business using multiple sources of data to help management generate relevant insights for data driven business decisions



Employees–
Employees are responsible for selling products

Products–
Products are offered by the business/employees to the customers

Sales –
Products generate sales for the business

Marketing–
Each channel has a marketing expense

Raw material –
Each product is made using different raw materials

Vendors –
Raw materials are provided by various vendors

Expense–
Manufacturing selling and marketing expense along with running expenses are recorded

Objectives

Identify ***star selling products*** to ensure right marketing strategy in place

Employees Annual bonus with respect to ***yearly performance matrix***

Multiple sorting parameters to improve ***inventory management***

Designing DBMS

Conceptual Design

Developing entity relationship diagram for the problem

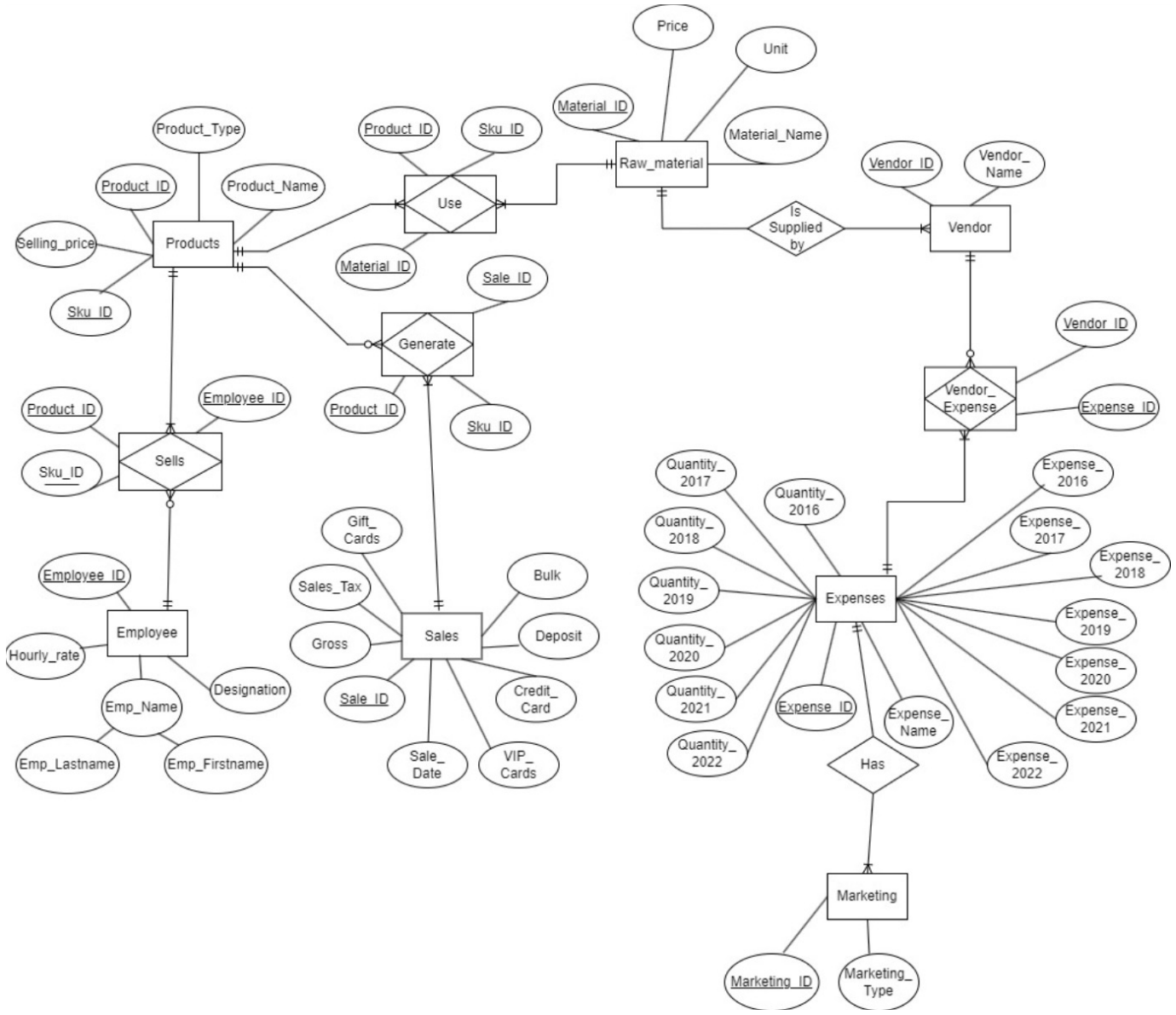
Relationship between Employees> Product > Sales> Raw material & Vendor & Expenses is a many-to-many relationship, we create three associative entity tables:

Employee-Product (Sells) –
Employee annual bonus

Product-Sales (Generate)–
Identify highest selling products

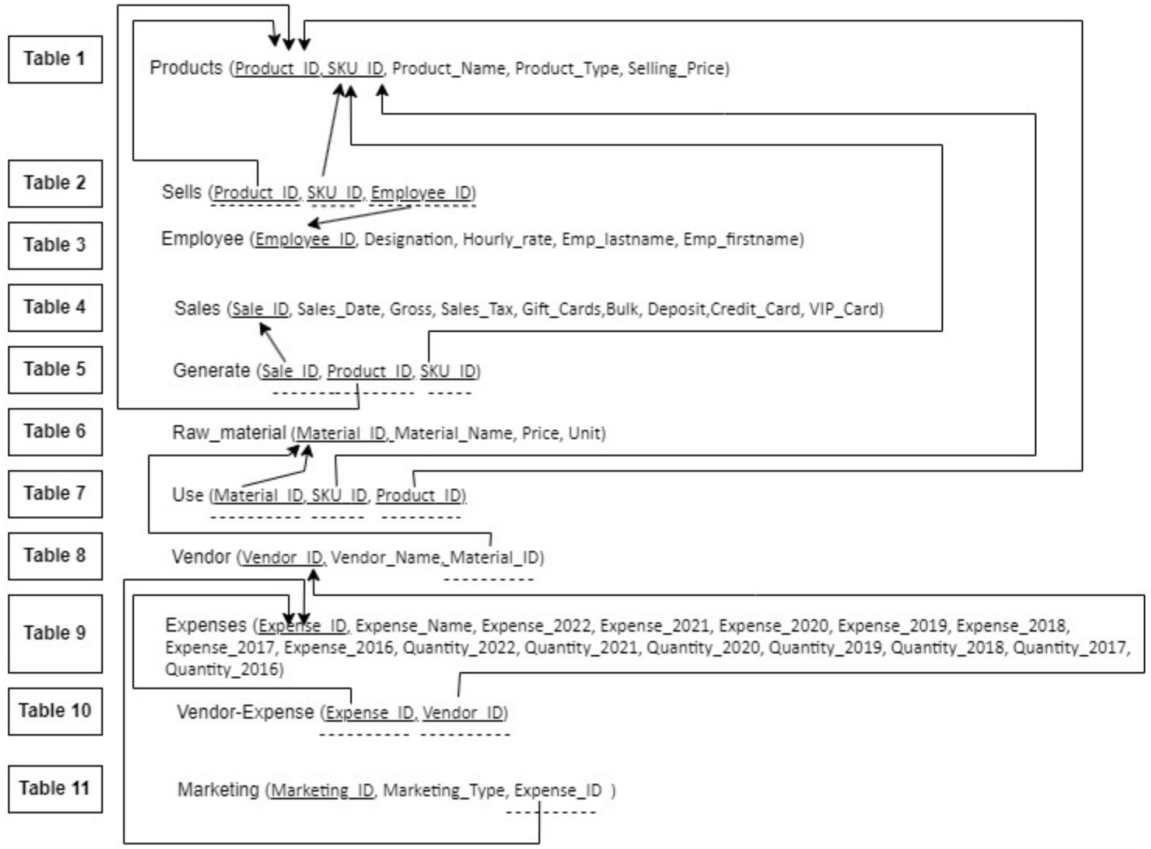
Product-Raw Material (Use)–
To optimize inventory level to meet product demand

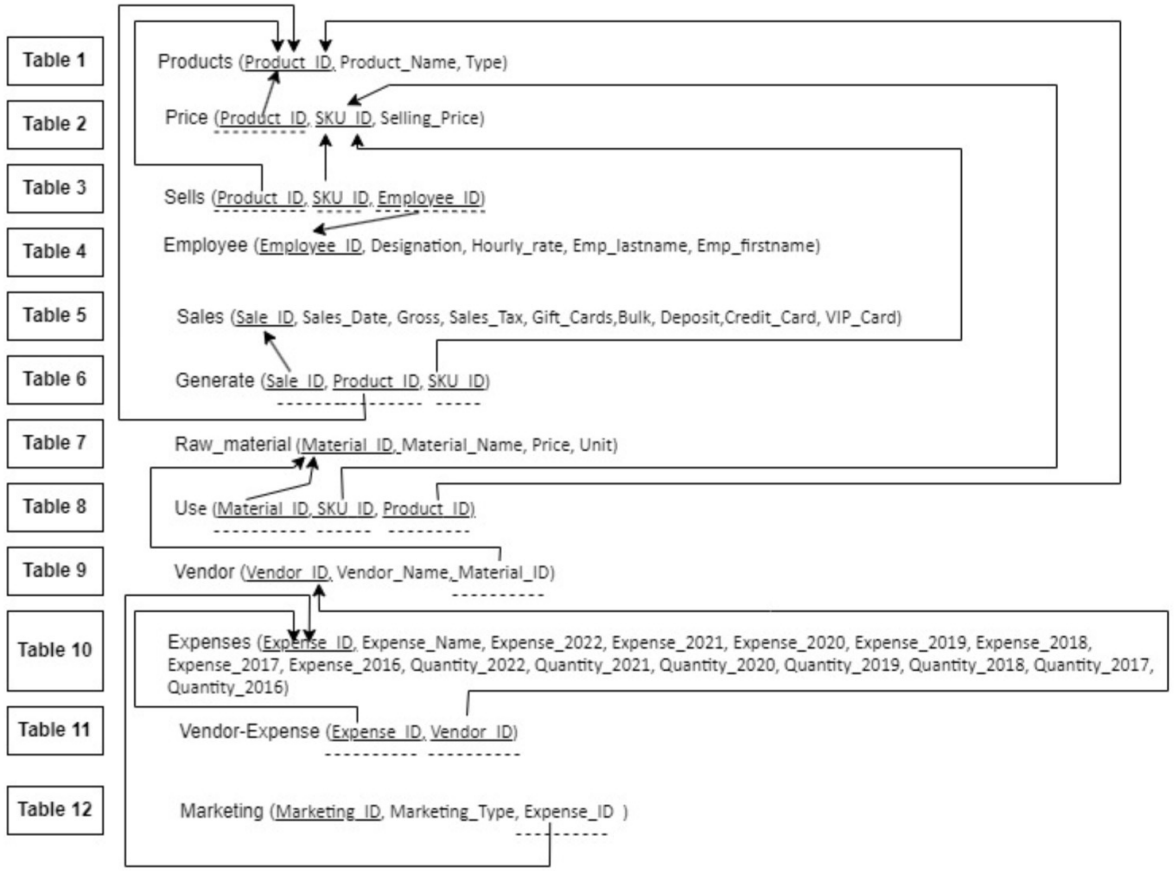
Entity Relationship Diagram



Logical Design

Developing relational schema from ER diagram





Relational schema allows us to set the framework for implementation of our DBMS

Normalization

Entities needs to be in 3NF before implementation

Validation for 1NF

Multi-valued or composite attributes?

- No multi-valued or composite attributes.

Validation for 2NF

Partial dependencies in the data?.

- It was found that Table 1 in the Relational Schema has partial dependencies. Therefore, Table 1 was split into 2 tables to remove the partial dependencies.

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)

Validation for 3 NF:

Transitive Dependencies?

- After modifying the Schema for 2 NF, the data available in all the tables were validated for 3NF. It was observed that the data is already in 3 NF, since no transitive dependencies were observed.

Implementing DBMS in SQL

Implementation

Creating database in SQL

Using **CREATE** to generate entity tables

```
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
```

```
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
```

```
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
```

Using **CREATE** to generate associative entity tables

```
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
```

Implementation

Creating database in SQL

Using **CREATE** to generate entity tables

```
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
```

```
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
```

Using **CREATE** to generate associative entity tables

```
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
```

Implementation

Creating database in SQL

Using **CREATE** to generate entity tables

```
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
```

```
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
```

Using **CREATE** to generate associative entity tables

```
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
```

Implementation

Creating database in SQL

Using **CREATE** to generate entity tables

```
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
```

```
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
```

Using **CREATE** to generate associative entity tables

```
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
```

Implementation

Relational instance in database

Using **SELECT** to view corresponding tables

```
1 •  SELECT * FROM project.employees;
2 •  SELECT * FROM project.expenses;
3 •  SELECT * FROM project.generate;
4 •  SELECT * FROM project.marketing;
5 •  SELECT * FROM project.price;
6 •  SELECT * FROM project.product;
7 •  SELECT * FROM project.raw_material;
8 •  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	231.61	11	56.08	3

expenses 1

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
12	Sponsorships	3

Querying the DBMS

Business inferences from our database

Query Objective: show the total sales for year from 2017 to 2020

Query

```
1   #Total sales every year
2
3 • select SUBSTRING(sale_date, 7, 4) as year , sum(credit_card ) + sum(deposit) as Total_Sales
4   from project.sales
5   group by SUBSTRING(sale_date, 7, 4) order by 1 desc;
6
```

Output

year	Total_Sales
2020	492276.1400000002
2019	497374.0700000007
2018	512358.6599999998
2017	551388.94

Business Implications

The above output can act as a reference data to evaluate year on year total sales for the business and identify the year in which sales was the highest, this information can be further used to find out the contributors of high sales in the respective year

Querying the DBMS

Business inferences from our database

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

Query

```
7   #Total Expenses every year
8
9   select '2018' as year, sum(expense_2018) as Total_expense from project.expenses
10  union
11  select '2017' as year, sum(expense_2017) as Total_expense from project.expenses
12  union
13  select '2019' as year, sum(expense_2019) as Total_expense from project.expenses
14  union
15  select '2020' as year, sum(expense_2020) as Total_expense from project.expenses
16  order by 1 desc;
17
```

Output

year	Total_expense
2020	608916.7699999998
2019	495136.78
2018	503128.6
2017	567319.48

Business Implications

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

Querying the DBMS

Business inferences from our database

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

Query

```
18    #Profit or Loss for all years
19
20    with sales as (select SUBSTRING(sale_date, 7, 4) as year , sum(credit_card ) + sum(deposit) as Total_Sales
21    from project.sales
22    group by SUBSTRING(sale_date, 7, 4) order by 1 desc),
23
24    expense as(
25    select '2018' as year, sum(expense_2018) as Total_expense from project.expenses
26    union
27    select '2017' as year, sum(expense_2017) as Total_expense from project.expenses
28    union
29    select '2019' as year, sum(expense_2019) as Total_expense from project.expenses
30    union
31    select '2020' as year, sum(expense_2020) as Total_expense from project.expenses
32    order by 1 desc)
33
34    select a.year,a.total_sales, b.total_expense, a.total_sales - b.total_expense as Profit_Loss
35    from sales a join expense b on a.year = b.year;
```

Output

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

Business Implications

In addition to the previous output, it is important to also 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

Querying the DBMS

Business inferences from our database

Query Objective: To get the top highest selling months.

Query

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

Output

Top_months	Total_Sales
June	380116.02
July	365433.92000000004
May	335407.00999999995

Business Implications

Identifying three highest selling months could help in prioritizing marketing activities and manage inventory efficiently to meet the demand accordingly

Querying the DBMS

Business inferences from our database

Query Objective: give the lowest month and its sales.

Query

```
50 • select case when a.sale_month = 03 then 'March'
51              when a.sale_month = 04 then 'April'
52              when a.sale_month = 05 then 'May'
53              when a.sale_month = 06 then 'June'
54              when a.sale_month = 07 then 'July'
55              when a.sale_month = 08 then 'August'
56              when a.sale_month = 09 then 'September'
57              when a.sale_month = 10 then 'October'end as Top_months,  a.Total_Sales from
58 (select SUBSTRING(sale_date, 4, 2) as sale_month , sum(credit_card ) + sum(deposit) as Total_Sales
59 from project.sales
60 group by SUBSTRING(sale_date, 4, 2))a order by 2 limit 1;
```

Output

	Top_months	Total_Sales
▶	March	114727.64000000001

Business Implications

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

Querying the DBMS

Business inferences from our database

Query Objective: to find the employee of the month based on the number of product sold each month

Query

```
63    # Employee of the month
64
65    select a.employee_id , b.EMPLOYEE_FIRSTNAME , b.EMPLOYEE_LASTNAME
66    from project.sells a join  project.employees b on a.employee_id = b.employee_id
67    order by 1 desc limit 1;
```

Output

	employee_id	EMPLOYEE_FIRSTNAME	EMPLOYEE_LASTNAME
▶	15	Jordan	Samman

Business Implications

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

Querying the DBMS

Business inferences from our database

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

SQL Function

```
#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;
```

Output

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

Business Implications

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

Querying the DBMS

Business inferences from our database

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

SQL Query

```
75    #Top products in all product_type
76
77 •  select x.product_id , x.product_name , x.product_type from (
78    select count(a.product_id) as count, a.product_id , b.product_name, b.product_type,
79     rank() over (partition by b.product_type order by a.product_id desc) as product_rank
80    from project.sells a join project.product b on a.product_id = b.product_id
81    group by a.product_id) x where x.product_rank = 1;
```

Output

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

Business Implications

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

Querying the DBMS

Business inferences from our database

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

SQL Query

```
99    #Raw material for the top products for different product_type
100
101 • select distinct  x.product_name , x.product_type , z.material_name from (
102   select count(a.product_id) as count, a.product_id , b.product_name, b.product_type,
103    rank() over (partition by b.product_type order by a.product_id desc) as product_rank
104   from project.sells a join project.product b on a.product_id = b.product_id
105   group by a.product_id) x left outer join project.uses y on x.product_id = y.product_id
106   left outer join project.raw_material z on y.material_id = z.material_id
107   where x.product_rank = 1;
```

Output

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

Business Implications

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

Querying the DBMS

Business inferences from our database

Query Objective: To find the highest expenses occured in each year

Query

```
83  # Most expenses occured each year
84
85  select '2022' as year, EXPENSE_NAME from project.expenses where expense_2022 in (select max(expense_2022) as max_expense from project.expen
86  union
87  select '2021' as year,EXPENSE_NAME from project.expenses where expense_2021 in (select max(expense_2021) as max_expense from project.expens
88  union
89  select '2020' as year,EXPENSE_NAME from project.expenses where expense_2020 in (select max(expense_2020) as max_expense from project.expens
90  union
91  select '2019' as year,EXPENSE_NAME from project.expenses where expense_2019 in (select max(expense_2019) as max_expense from project.expens
92  union
93  select '2018' as year,EXPENSE_NAME from project.expenses where expense_2018 in (select max(expense_2018) as max_expense from project.expens
94  union
95  select '2017' as year,EXPENSE_NAME from project.expenses where expense_2017 in (select max(expense_2017) as max_expense from project.expens
96  union
97  select '2016' as year,EXPENSE_NAME from project.expenses where expense_2016 in (select max(expense_2016) as max_expense from project.expens
```

Output

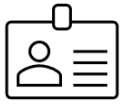
year	EXPENSE_NAME
2022	Salaries&Wages
2021	PayrollTaxes
2020	Security-CAMERAS&GUARD
2019	PayrollTaxes
2018	PayrollTaxes
2017	Supplies-Kitchen
2016	Supplies-Kitchen

Business Implications

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

Business Recommendations

All business findings



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

THANK YOU!