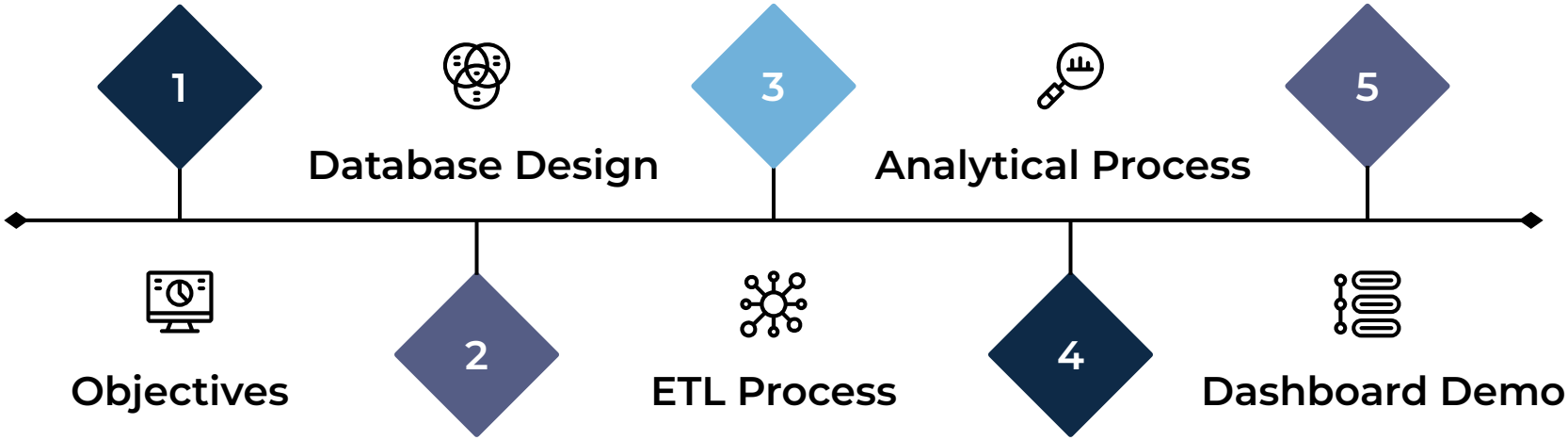
ABC Foodmart

Group 7
Yifan Lei, Zijian Li, Miaoer Mo, Isabella Shen,
Yifei Wan, Rui Yan

Content


1
Objectives
2
3
Database Design
ETL Process
4
5
Analytical Process
Dashboard Demo

Background Info & Objectives

Background Info

Client :

- ABC Foodmart
- Neighborhood grocery chain with 2 locations in Queens NY and planning to establish 3 more locations in Brooklyn

Business Challenge:

- Outdated data system with spreadsheet and paper binders
- Inefficiency, erros, slow decision-making
- Inadequate fo expansion

Define Objectives

- Design and implement a integrated data management system
- Improve efficiency and accuracy in decision-making process
- Support the opening of three new locations

Proposed Solution

Relational Database Design:

- Normalized relational database
- ER-Diagram

Data Migration:

- Data cleanse
- ETL process

Analysis & Support:

- Business Analysis
- Ongoing technical Support

Original Datasets

Customers

1

Tracking customer interactions, purchase history, and loyalty program participation

Employees

2

Manage staffing, scheduling, and operational efficiency

Product Stock

3

Inventory levels, helping to manage stock-outs or overstock situations

Products

4

Inventory tracking, sales analysis, and vendor management

Stores

5

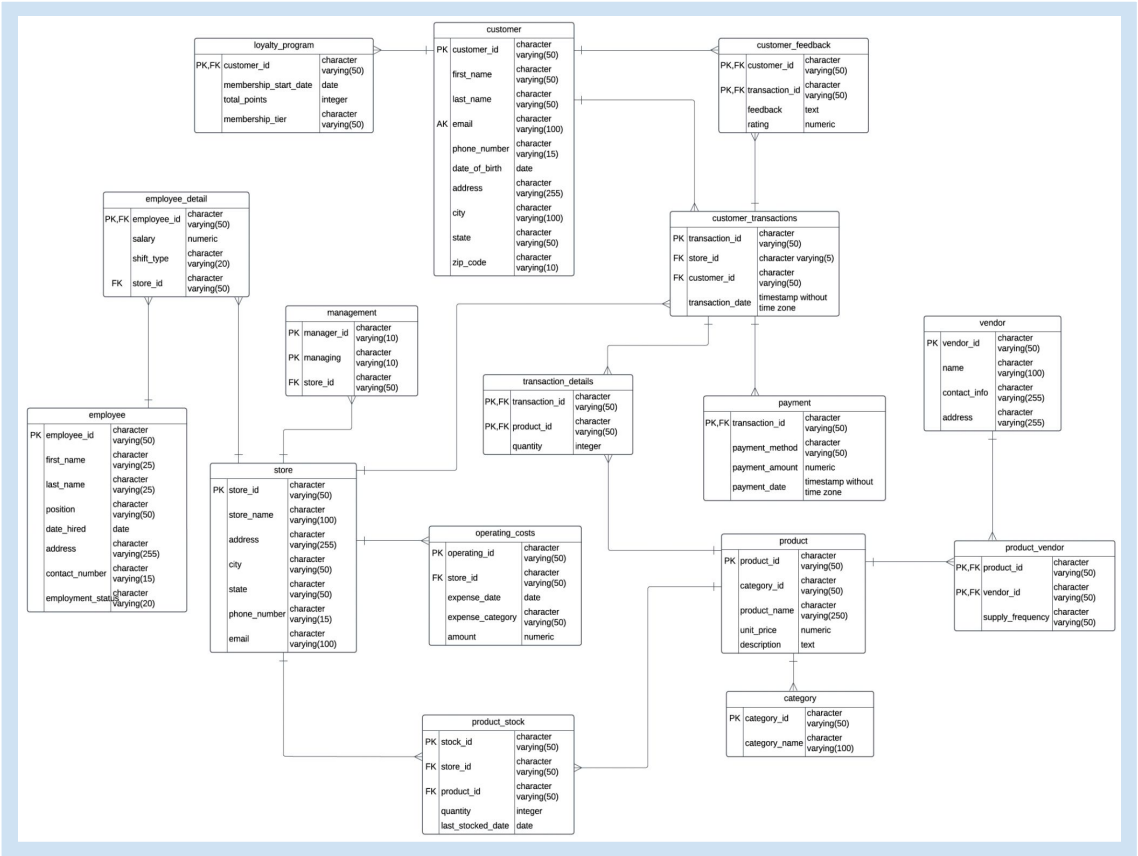
Managing store locations and linking transactions or stock data to specific stores

Transactions

6

Analyzing sales trends, transaction profits and customer purchase patterns

Database Design



Core Tables: employee, store, customer, product, and category

Dependent Tables: employee_detail, product_stock, and customer_transactions

Supporting Tables: loyalty_program, operating_costs, payment, and transaction_details

Key Relationships and Cardinalities

1. Employee and Store Management
2. Customer Management
3. Product and Inventory Management
4. Financial and Transaction Management
5. Operational Costs

ETL Process

6 Original Datasets → 16 normalized (3NF) Tables

Operating_costs table

```
In [7]: # Create opr_cost_df with the selected columns
        opr_cost_df = df1[['store_id', 'expense_date', 'expense_category', 'amount']]

        # Add Operating_id column with values starting from O1
        opr_cost_df.insert(0, 'operating_id', ['O' + str(i) for i in range(1, len(opr_cost_df) + 1)])

        # Display the result
        print(opr_cost_df.head())
        opr_cost_df.to_sql(name='operating_costs', con=engine, if_exists='append', index=False)
```

	operating_id	store_id	expense_date	expense_category	amount
0	O1	2	2022/4/29	Rent	3729.39
1	O2	2	2023/5/3	Maintenance	2210.33
2	O3	3	2023/8/19	Rent	3208.38
3	O4	1	2024/7/25	Maintenance	1135.39
4	O5	4	2022/1/21	Utilities	904.52

Add `id` column to product/category/vendor table

```
#Insert into product table
# Add `product_id` column based on unique `product_name`
product_mapping = {name: f'P{i+1}' for i, name in enumerate(df2['product_name'].unique())}
df2['product_id'] = df2['product_name'].map(product_mapping)

#  Add `category_id` column based on unique `Category`
category_mapping = {name: f'CE{i+1}' for i, name in enumerate(df2['Category'].unique())}
df2['category_id'] = df2['Category'].map(category_mapping)

# Add `vendor_id` column based on unique `vendor_name`
vendor_mapping = {name: f'V{i+1}' for i, name in enumerate(df2['vendor_name'].unique())}
df2['vendor_id'] = df2['vendor_name'].map(vendor_mapping)
```

Management table

```
        # Drop rows with missing store_id
        manager_table = manager_table.dropna(subset=['store_id'])

        # Display the first few rows of the manager_table
        print(manager_table.head())
        manager_table.to_sql(name='management', con=engine, if_exists='append', index=False)
```

	manager_id	managing	store_id
0	E1	E6	1
1	E1	E11	1
2	E1	E16	1
3	E1	E21	1
4	E2	E7	2

```
Out[27]: 20
```

Stores

- **store:** assigned a unique store_id by mapping store_name to a number, and captured static store information
- **operating_cost:** isolates dynamic financial data, linking expenses to their corresponding stores via store_id

Products

- **product:** stores product details like product_name, unit_price with a unique product_id
- **category:** assigned a category_id, linking products to their categories
- **vendor:** stores supplier details such as vendor_name, contact_info with a unique vendor_id
- **product_vendor:** Represents the many-to-many relationship between products and vendors

Employees

- **employee:** Each employee was assigned a unique identifier, and duplicates were removed
- **employee_detail:** Include Job details such as salary. Store_name was mapped to store_id from store table
- **management:** normalized hierarchical relationships by splitting the management. Manager_id and managing employees were distinctly identified

ETL Process

6 Original Datasets → **16 normalized (3NF) Tables**

Product_stock table

```
# Map product_name to product_id
df4['product_id'] = df4['product_name'].map(product_mapping)

df4 = df4[['stock_id', 'store_id', 'product_id', 'quantity', 'last_stocked_date']]

print(df4.head())
```

```
  stock_id  store_id product_id  quantity last_stocked_date
0       S1         1         P1       247         5/22/2023
1       S2         2         P1       920          7/8/2023
2       S3         3         P1       696         8/30/2023
3       S4         4         P1       772        11/12/2024
4       S5         5         P1       980         7/16/2023
```

Customer table

```
# Insert the data into the customer table in the database
customer_df.to_sql(name='customer', con=engine, if_exists='append', index=False)
```

```
  customer_id first_name last_name    phone_number  \
0          C1      Uriah   Bridges  (401) 552-4059
1          C2      Paula     Small  (688) 210-1295
2          C3     Edward      Buck  (434) 593-6233
3          C4    Michael   Riordan  (517) 270-8979
4          C5    Jasmine     Onque  (214) 940-9676

                        email date_of_birth          address         city  \
0    uriah.bridges@example.com    1977/10/1  8413 Madison Ave     New York
1      paula.small@example.com     1969/7/5     5064 Broadway  Jersey City
2      edward.buck@example.com    2000/3/26      8055 Main St     New York
3  michael.riordan@example.com     1981/2/8      2880 Main St     New York
4    jasmine.onque@example.com    1961/4/10     9447 Broadway     New York
```

Transaction_Details table

```
# Ensure no duplicates in transaction_id and product_id pairs (use for validation, optional)
duplicate_check = transaction_details_df.duplicated(subset=['transaction_id', 'product_id'], keep=False)
if duplicate_check.any():
    print("Warning: Duplicates detected in transaction_id and product_id pairs!")

print(transaction_details_df.head(20))
```

```
    transaction_id product_id  quantity
0        123000012       P112         2
1        123000012       P125         3
2        123000012        P74         2
3        123000012       P885         2
4        123000012       P911         2
5        123000012        P98         2
6        123000064      P1276         2
7        123000064       P205         2
8        123000064       P291         3
9        123000064        P42         3
```

Product Stock

→ **product_stock:** assigned unique stock_id and mapping store_name to store_id and product_name to product_id, creating a link between inventory, stores, and products

Customers

→ **customer:** assigned sequential customer_ids (e.g., C1, C2) and aligning column names to create a clean repository for static customer information.

→ **loyalty_program:** extracted loyalty-specific attributes and mapping customer_ids based on email as a unique key, ensuring referential integrity.

Transactions

→ **customer_transaction:** mapping phone_number to customer_id, removing duplicate transaction entries

→ **transaction_details:** mapping product_name to its id, ensure unique combinations of transaction and product

→ **payment:** calculated payment as the sum of unit_price multiplied by quantity for each transaction, and aggregating additional fields grouped by transaction_id

→ **customer_feedback:** merged transactions with customer data to link customer_id to feedback

Targeted Audiences

eg: Which customers spend the most?

Data Analyst

Use **PgAdmin** to directly query and retrieve results in tabular data

customer_id character varying (50)	customer_name text	total_spent numeric
C776	Shari Ngodup	5235.78
C196	Harrison Hudson	4886.87
C429	Marlee Stevens	4834.63
C1001	Mylee Snow	4792.15
C679	Sincere Rivera	4759.56
C1401	Ellie Ortega	4753.11
C1169	Lamont Cook	4658.09
C1741	Valentino Myers	4605.54
C1562	Terrell Guerra	4484.28
C1209	Lilliana Walker	4101.56

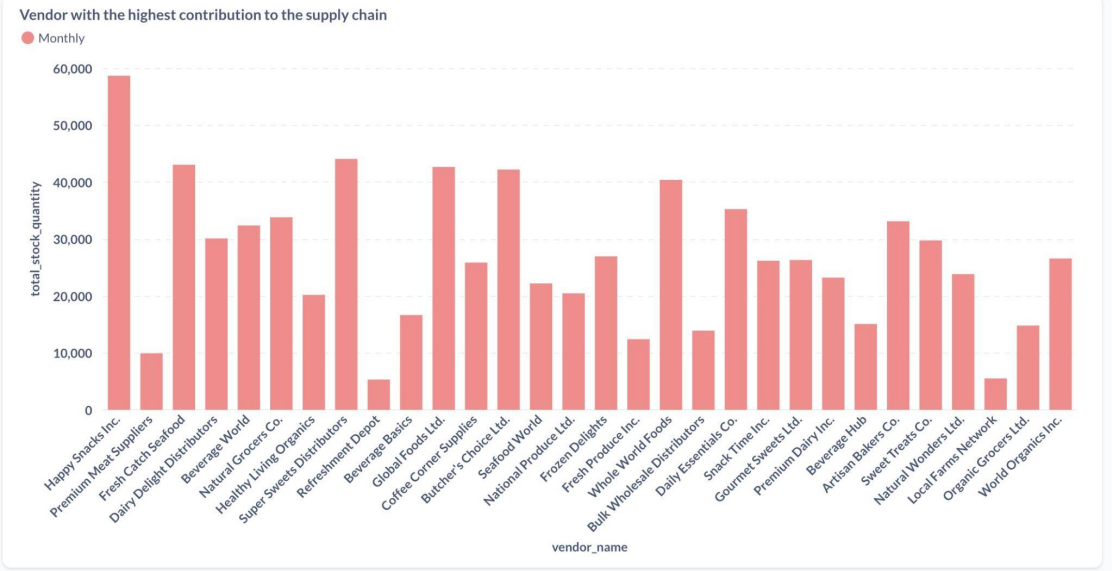
C-level Officers

Use **Metabase** Dashboards to help understand quickly what the key insights are

Market Insights

Vendors Contribute the most

```sql
SELECT
    v.vendor_id,
    v.name AS vendor_name,
    pv.supply_frequency,
    COUNT(DISTINCT pv.product_id) AS total_products_supplied,
    SUM(ps.quantity) AS total_stock_quantity
FROM product_vendor pv
JOIN product_stock ps ON pv.product_id = ps.product_id
JOIN vendor v ON pv.vendor_id = v.vendor_id
GROUP BY v.vendor_id, v.name, pv.supply_frequency
ORDER BY total_stock_quantity DESC;
```



Market Insights

Most Loyal Customers

```sql
SELECT
    c.customer_id,
    c.first_name || ' ' || c.last_name AS full_name,
    lp.membership_start_date,
    lp.total_points,
    lp.membership_tier,
    c.city
FROM
    loyalty_program lp
JOIN
    customer c ON lp.customer_id = c.customer_id
ORDER BY
    lp.total_points DESC,
    lp.membership_start_date ASC;
```

Revenue per Category

```sql
SELECT
    c.category_name,
    SUM(td.quantity * p.unit_price) AS total_revenue
FROM
    transaction_details td
JOIN
    product p ON td.product_id = p.product_id
JOIN
    category c ON p.category_id = c.category_id
GROUP BY
    c.category_name
ORDER BY
    total_revenue DESC;
```

Market Insights

Average Rating for Each Store

```sql
SELECT
    s.store_id,
    s.store_name,
    ROUND(AVG(cf.rating),2) AS average_rating
FROM
    customer_feedback cf
JOIN
    customer_transactions ct ON cf.transaction_id = ct.transaction_id
JOIN
    store s ON ct.store_id = s.store_id
GROUP BY
    s.store_id, s.store_name
ORDER BY
    average_rating DESC;
```

Monthly Sales Trends

```sql
SELECT
    TO_CHAR(ct.transaction_date, 'YYYY-MM') AS month,
    SUM(td.quantity * p.unit_price) AS total_revenue
FROM
    transaction_details td
JOIN
    customer_transactions ct ON td.transaction_id = ct.transaction_id
JOIN
    product p ON td.product_id = p.product_id
GROUP BY
    TO_CHAR(ct.transaction_date, 'YYYY-MM')
ORDER BY
    month;
```

Thank you !