Coffee Ordering System

Student: [YE Chenwei, Jesse] ID: 21199517 Course: ISOM5260

October 17, 2025

Contents

1 Project Initiation Document **2**

1.1 Business Values of the System 2

1.2 Scope of the System 2

1.2.1 Core Functional Requirements 2

1.2.2 System Boundaries 3

2 Design Specification **3**

2.1 Conceptual Data Model & Business Rules 3

Business Rules (from ERD) 3

Functional Dependencies by Table and Normal Forms 4

2.2 Data Dictionary (Excerpt) 5

3 Configuration Specification (SQL Queries) **5**

3.1 Data Retrieval 5

3.2 Reporting and Analytics 6

4 Generating Charts with SQL (From Aggregation to Visualization) **8**

4.1 Concept Overview 8

4.2 Common Business Questions → SQL Aggregation Examples 8

4.3 Export and Visualization Steps 9

4.4 Example: CSV to Line Chart (Minimal Python) 9

4.5 SQL Compatibility and Notes 10

5 Project Conclusion **10**

Appendix: Selected DDL Snippets **10**

1 Project Initiation Document

1.1 Business Values of the System

Traditional coffee shop operations rely on manual processes, leading to inefficiency and lack of insight. The Coffee Ordering System delivers value to owners (efficiency, insights, data-driven pricing, scalability), customers (convenience, personalization, transparency), and staff (streamlined operations).

1.2 Scope of the System

1.2.1 Core Functional Requirements

- **Customer Management**:
  - Walk-in customer registration and order processing
  - Member account creation with secure authentication (password hashing)
  - Customer profile management (personal information, preferences)
  - Member-specific features (birthday tracking, registration dates)
- **Product Catalog Management**:
  - Hierarchical product categorization (Coffee, Tea, Dessert, Light Meal)
  - Product lifecycle management (active/inactive status control)
  - Price management and product information updates
  - Image support for visual product presentation
- **Order Processing System**:
  - Shopping cart functionality for order building
  - Order creation with automatic total calculation
  - Order status tracking (pending, completed, cancelled, refunded)
  - Complete order history for customers and administrators
  - Multiple payment method support (Cash, Credit Card, Alipay, WeChat Pay)
- **Administrative Analytics & Reporting**:
  - Daily sales reports with revenue and order count metrics
  - Product performance analysis (best sellers, revenue leaders)
  - Customer behavior insights (spending patterns, loyalty metrics)
  - Payment method distribution analysis
  - Customer segmentation and retention tracking
- **Data Management & Security**:
  - Referential integrity enforcement through foreign key constraints
  - Data validation and business rule enforcement
  - Secure password storage using SHA-256 hashing
  - Database normalization to third normal form (3NF)

1.2.2 System Boundaries

In Scope Core ordering workflow, customer relationship management, basic analytics, and data integrity.

Out of Scope

- Inventory management and stock level tracking
- Staff scheduling and employee management
- Financial accounting and tax calculations
- Mobile application development
- Third-party payment gateway integration
- Advanced marketing automation and campaigns
- Multi-location support (single store focus)
- Advanced loyalty program features

Future Enhancement Opportunities: The system architecture supports potential expansion into inventory management, multi-location operations, advanced analytics dashboards, and API development for mobile applications.

2 Design Specification

2.1 Conceptual Data Model & Business Rules

Core entities: **Customer**, **Member_Customers**, **Member_Preferences**, **Category**, **Product**, **Orders**, **Order_Items**.

Business Rules (from ERD)

- Only `MEMBER` customers can have a row in `MEMBER_CUSTOMERS`; if a member row exists, `CUSTOMER.CUSTOMER_TYPE` cannot be changed to `GUEST` (trigger enforced).
- `ORDERS.PAYMENT_METHOD` ∈ {`CASH, CREDIT CARD, ALIPAY, WECHAT`}; values are validated with `UPPER()` CHECKs.
- `PRODUCT.IS_ACTIVE` ∈ {`Y,N`}; each product must belong to a valid category.
- Quantities and prices are non-negative; in `ORDER_ITEMS`, `LINE_AMOUNT` is a virtual column defined as `QUANTITY * UNIT_PRICE`.
- `ORDERS.TOTAL_AMOUNT` is maintained by triggers as the sum of its order items (controlled denormalization for performance).
- Natural key: `CATEGORY.CATEGORY_NAME` is `UNIQUE`; `CUSTOMER.EMAIL` is indexed for lookup.

Referential integrity (Foreign Keys)

We enforce the following FKs (child.column → parent.column) and cardinalities:

- **ORDERS.customer_id → CUSTOMER.customer_id** (1 : N)
- **MEMBER_CUSTOMERS.customer_id → CUSTOMER.customer_id** (1 : 0..1)

- **MEMBER_PREFERENCES.customer_id → CUSTOMER.customer_id** (1 : N)
- **PRODUCT.category_id → CATEGORY.category_id** (1 : N)
- **ORDER_ITEMS.order_id → ORDERS.order_id** (1 : N)
- **ORDER_ITEMS.product_id → PRODUCT.product_id** (1 : N)

Textual ER Overview

```
Customer (1) ----< Orders (M)
   | |
   | +----< Order_Items (M) >----(1) Product
   | |
   +----< Member_Customers (0/1) +----(M) Product >----(1) Category
   |
   +----< Member_Preferences (M)
```

Functional Dependencies by Table and Normal Forms

Note: Primary keys, candidate/natural keys and `UNIQUE` constraints are listed in the Data Dictionary. This section focuses on determinants, dependencies, and normal forms.

- **`CUSTOMER`**: The primary key functionally determines {`name, phone, email, address, customer_type`}. This relation is in 3NF (and would be in BCNF if there are no non-key determinants).
- **`MEMBER_CUSTOMERS`**: The primary key functionally determines {`password_hash, date_of_birth, registration_date`}. The membership rule is enforced by a trigger. The relation is in 3NF / BCNF.
- **`MEMBER_PREFERENCES`**: The primary key functionally determines {`customer_id, preference_type, preference_value, created_date`}. There is an optional candidate key defined by `UNIQUE(customer_id, preference_type, preference_value)`. It is in 3NF (and would be in BCNF if the UNIQUE constraint is treated as a candidate key).
- **`CATEGORY`**: The primary key functionally determines {`category_name, description`}. A natural key exists via `UNIQUE(category_name)`. The relation is in BCNF.
- **`PRODUCT`**: The primary key functionally determines {`name, price, is_active, category_id`}. The attribute `category_id` is a foreign key and does not create a transitive dependency inside the `PRODUCT` relation. It is in 3NF / BCNF.
- **`ORDERS`**: The primary key functionally determines {`customer_id, order_date, status, payment_method, total_amount`}. The `total_amount` is a trigger-maintained cache of the sums of the order items. The relation is in 3NF (with its base attributes satisfying BCNF).
- **`ORDER_ITEMS`**: The primary key functionally determines {`order_id, product_id, quantity, unit_price`}. The `line_amount` is a virtual column, calculated as `quantity` × `unit_price`. The relation is in 3NF / BCNF.

2.2 Data Dictionary (Excerpt)

Table	Key Attributes	Notes
CUSTOMER	CUSTOMER_ID (PK)	name, phone, email, address; customer_type {MEMBER/GUEST}; email indexed
MEMBER_CUSTOMERS	CUSTOMER_ID (PK, FK→CUSTOMER)	password_hash, date_of_birth, registration_date; only for MEMBER; parent delete cascades
MEMBER_PREFERENCES	PREFERENCE_ID (PK), CUSTOMER_ID (FK→CUSTOMER)	preference_type, preference_value, created_date; optional dedup: `UNIQUE(customer_id, preference_type, preference_value)`
CATEGORY	CATEGORY_ID (PK)	category_name **UNIQUE**, description
PRODUCT	PRODUCT_ID (PK)	name, price, is_active {Y/N}, category_id (FK→CATEGORY)
ORDERS	ORDER_ID (PK)	customer_id (FK→CUSTOMER), order_date, status {PLACED/COMPLETED/CANCELLED/REFUNDED}, payment_method {CASH/CREDIT CARD/ALIPAY/WECHAT}, total_amount (trigger-maintained)
ORDER_ITEMS	ORDER_ITEM_ID (PK)	order_id (FK→ORDERS), product_id (FK→PRODUCT), quantity > 0, unit_price ≥ 0, line_amount = quantity × unit_price (virtual); optional UNIQUE(order_id, product_id) depending on business rules

3 Configuration Specification (SQL Queries)

This section lists representative SQL used by admin/user reports. Purposes are annotated for each query.

3.1 Data Retrieval

Active Products (Menu) – list active items for ordering

```
SELECT p.PRODUCT_ID, p.NAME, p.PRICE, p.IS_ACTIVE, c.CATEGORY_NAME
FROM CYEAE_PRODUCT p
LEFT JOIN CYEAE_CATEGORY c ON p.CATEGORY_ID = c.CATEGORY_ID
WHERE p.IS_ACTIVE = ’Y’
ORDER BY c.CATEGORY_NAME, p.NAME;
```

Categories – populate category dropdown

```
SELECT CATEGORY_ID, CATEGORY_NAME, DESCRIPTION
FROM CYEAE_CATEGORY
ORDER BY CATEGORY_NAME;
```

Order History (by customer) – account page

```
SELECT o.ORDER_ID, c.NAME, o.ORDER_DATE, o.STATUS, o.PAYMENT_METHOD, o.TOTAL_AMOUNT
FROM CYEAE_ORDERS o
JOIN CYEAE_CUSTOMER c ON o.CUSTOMER_ID = c.CUSTOMER_ID
WHERE o.CUSTOMER_ID = :id
ORDER BY o.ORDER_DATE DESC;
```

3.2 Reporting and Analytics

Daily Sales (date range) – ops dashboard

```
SELECT DATE(o.ORDER_DATE) AS order_date,
      COUNT(o.ORDER_ID) AS order_count,
      SUM(o.TOTAL_AMOUNT) AS total_sales,
      AVG(o.TOTAL_AMOUNT) AS avg_order_value
FROM CYEAE_ORDERS o
WHERE DATE(o.ORDER_DATE) BETWEEN :from AND :to
GROUP BY DATE(o.ORDER_DATE)
ORDER BY order_date DESC;
```

Product Sales – revenue leaderboard

```
SELECT p.NAME AS product_name, c.CATEGORY_NAME,
      SUM(oi.QUANTITY) AS total_quantity,
      SUM(oi.LINE_AMOUNT) AS total_revenue
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN CYEAE_CATEGORY c ON p.CATEGORY_ID = c.CATEGORY_ID
GROUP BY p.PRODUCT_ID, p.NAME, c.CATEGORY_NAME
ORDER BY total_revenue DESC;
```

Customer Summary – high spenders

```
SELECT c.CUSTOMER_ID, c.NAME,
      COUNT(o.ORDER_ID) AS order_count,
      SUM(o.TOTAL_AMOUNT) AS total_spent,
      AVG(o.TOTAL_AMOUNT) AS avg_order_value,
      MAX(o.ORDER_DATE) AS last_order_date
FROM CYEAE_CUSTOMER c
LEFT JOIN CYEAE_ORDERS o ON c.CUSTOMER_ID = o.CUSTOMER_ID
GROUP BY c.CUSTOMER_ID, c.NAME
ORDER BY total_spent DESC;
```

Last Order per Customer – recency

```
SELECT o.CUSTOMER_ID, MAX(o.ORDER_DATE) AS last_order_date
FROM CYEAE_ORDERS o
GROUP BY o.CUSTOMER_ID;
```

Top-N Products in Period – merchandising

```
SELECT p.NAME, SUM(oi.QUANTITY) AS qty
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN CYEAE_ORDERS o ON oi.ORDER_ID = o.ORDER_ID
WHERE DATE(o.ORDER_DATE) BETWEEN :from AND :to
GROUP BY p.NAME
ORDER BY qty DESC
LIMIT 10;
```

Orders by Payment Method – channel mix

```
SELECT PAYMENT_METHOD, COUNT(*) AS cnt, SUM(TOTAL_AMOUNT) AS revenue
FROM CYEAE_ORDERS
GROUP BY PAYMENT_METHOD
ORDER BY revenue DESC;
```

Average Basket Size – merchandising effectiveness

```sql
SELECT AVG(items_per_order) AS avg_items
FROM (
  SELECT COUNT(*) AS items_per_order
  FROM CYEAE_ORDER_ITEMS
  GROUP BY ORDER_ID
);
```

Inactive Products (never sold) – catalog hygiene

```sql
SELECT p.PRODUCT_ID, p.NAME
FROM CYEAE_PRODUCT p
LEFT JOIN CYEAE_ORDER_ITEMS oi ON oi.PRODUCT_ID = p.PRODUCT_ID
WHERE oi.ORDER_ID IS NULL;
```

Member Favorite Products – personalization seed

```sql
SELECT p.PRODUCT_ID, p.NAME,
       SUM(oi.QUANTITY) AS total_quantity,
       COUNT(oi.ORDER_ID) AS order_count
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN CYEAE_ORDERS o ON oi.ORDER_ID = o.ORDER_ID
WHERE o.CUSTOMER_ID = :customer_id
GROUP BY p.PRODUCT_ID, p.NAME
ORDER BY total_quantity DESC, order_count DESC
LIMIT 5;
```

Customers with No Orders in 30 Days – reactivation

```sql
SELECT c.CUSTOMER_ID, c.NAME
FROM CYEAE_CUSTOMER c
LEFT JOIN (
  SELECT CUSTOMER_ID, MAX(ORDER_DATE) AS last_dt
  FROM CYEAE_ORDERS
  GROUP BY CUSTOMER_ID
) x ON x.CUSTOMER_ID = c.CUSTOMER_ID
WHERE x.last_dt IS NULL OR DATE(x.last_dt) < DATE('now','-30 day');
```

Daily Active Customers – engagement

```sql
SELECT DATE(ORDER_DATE) AS d, COUNT(DISTINCT CUSTOMER_ID) AS active_customers
FROM CYEAE_ORDERS
GROUP BY DATE(ORDER_DATE)
ORDER BY d DESC;
```

Category Mix Share – category performance

```sql
SELECT c.CATEGORY_NAME,
       SUM(oi.LINE_AMOUNT) AS revenue,
       ROUND(100.0 * SUM(oi.LINE_AMOUNT) / (SELECT SUM(TOTAL_AMOUNT) FROM CYEAE_ORDERS), 2) AS
           pct
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN CYEAE_CATEGORY c ON p.CATEGORY_ID = c.CATEGORY_ID
GROUP BY c.CATEGORY_NAME
ORDER BY revenue DESC;
```

4 Generating Charts with SQL (From Aggregation to Visualization)

4.1 Concept Overview

Charts are built from **aggregating** granular data (grouping by `date/category/product/customer`, etc.) into structured results (typically `x, y` or multi-series), then exporting the result (CSV/JSON) to a visualization tool (Python/Excel/BI).

4.2 Common Business Questions → SQL Aggregation Examples

Run the following SQL directly on the operational database to produce chart-ready datasets. Export the results as CSV and use them for line/bar/pie visualizations.

Daily Sales Trend (Line Chart)

```
SELECT DATE(o.ORDER_DATE) AS order_date,
      SUM(o.TOTAL_AMOUNT) AS total_sales,
      COUNT(o.ORDER_ID) AS order_count
FROM CYEAE_ORDERS o
GROUP BY DATE(o.ORDER_DATE)
ORDER BY order_date;
```

Top Products by Revenue (Bar Chart)

```
SELECT p.NAME AS product_name,
      SUM(oi.LINE_AMOUNT) AS total_revenue,
      SUM(oi.QUANTITY) AS total_quantity
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
GROUP BY p.PRODUCT_ID, p.NAME
ORDER BY total_revenue DESC
LIMIT 10; -- Top 10
```

Payment Method Mix (Pie/Donut)

```
SELECT o.PAYMENT_METHOD,
      COUNT(*) AS order_count,
      SUM(o.TOTAL_AMOUNT) AS revenue
FROM CYEAE_ORDERS o
GROUP BY o.PAYMENT_METHOD
ORDER BY revenue DESC;
```

Customer Spend Distribution (Long Tail/Segmented Bars)

```
SELECT c.CUSTOMER_ID,
      c.NAME,
      SUM(o.TOTAL_AMOUNT) AS total_spent,
      COUNT(o.ORDER_ID) AS order_count
FROM CYEAE_CUSTOMER c
LEFT JOIN CYEAE_ORDERS o ON c.CUSTOMER_ID = o.CUSTOMER_ID
GROUP BY c.CUSTOMER_ID, c.NAME
ORDER BY total_spent DESC;
```

Category Revenue Share (Stacked Bars/Pie)

```
SELECT c.CATEGORY_NAME,
      SUM(oi.LINE_AMOUNT) AS revenue
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN CYEAE_CATEGORY c ON p.CATEGORY_ID = c.CATEGORY_ID
GROUP BY c.CATEGORY_NAME
ORDER BY revenue DESC;
```

Top-N Products in a Time Window (Ranked Bars)

```
SELECT p.NAME,
      SUM(oi.QUANTITY) AS qty
FROM CYEAE_ORDER_ITEMS oi
JOIN CYEAE_PRODUCT p ON oi.PRODUCT_ID = p.PRODUCT_ID
JOIN CYEAE_ORDERS o ON oi.ORDER_ID = o.ORDER_ID
WHERE DATE(o.ORDER_DATE) BETWEEN :from AND :to
GROUP BY p.NAME
ORDER BY qty DESC
LIMIT 10;
```

4.3 Export and Visualization Steps

1. Run the SQL above in your database client or built-in query tool (add date/other filters as needed).
2. **Export as CSV** with column headers.
3. Choose a visualization tool:
   - **Python/Matplotlib/Seaborn**: read CSV and plot line/bar/pie charts.
   - **Excel/Numbers**: insert charts and select appropriate chart types.
   - **BI tools** (e.g., Power BI/Tableau): connect the CSV or live database to build dashboards.
4. Save images (PNG/SVG/PDF) for reports or web display.

4.4 Example: CSV to Line Chart (Minimal Python)

The snippet below demonstrates plotting the ”Daily Sales” CSV as a line chart:

```
-- First, export two columns (order_date,total_sales) to sales_trends.csv via SQL
```

```
# sales_trends_plot.py
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('sales_trends.csv', parse_dates=['order_date'])
df = df.sort_values('order_date')

plt.figure(figsize=(8,4))
plt.plot(df['order_date'], df['total_sales'], marker='o')
plt.title('Daily Sales')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.tight_layout()
plt.savefig('sales_trends.png', dpi=150)
```

4.5 SQL Compatibility and Notes

- Date functions differ across databases (e.g., DATE(), TRUNC(), CAST()); adjust for your target engine when migrating.
- Precision/rounding: consider applying ROUND() to monetary aggregates as appropriate.
- Filters: most charts benefit from adding WHERE clauses (time window, category, customer segments).
- Performance: on large tables, add indexes (e.g., ORDER_DATE, CUSTOMER_ID, PRODUCT_ID) to keep aggregations fast.

5 Project Conclusion

The solution achieves 3NF design, enforces business rules at the database layer, and provides actionable SQL for management reporting. Future work: inventory, loyalty program, multi-location, and richer dashboards/APIs.

Appendix: Selected DDL Snippets

CHECK Constraints with UPPER()

```
ALTER TABLE orders ADD CONSTRAINT chk_orders_status_uc
  CHECK (UPPER(status) IN (’PLACED’,’COMPLETED’,’CANCELLED’,’REFUNDED’));

ALTER TABLE orders ADD CONSTRAINT chk_orders_pay_method_uc
  CHECK (UPPER(payment_method) IN (’CASH’,’CREDIT CARD’,’ALIPAY’,’WECHAT’));
```

Member Rule Triggers

```
CREATE OR REPLACE TRIGGER trg_mc_insupd
BEFORE INSERT OR UPDATE ON member_customers
FOR EACH ROW
DECLARE
  v_type customer.customer_type%TYPE;
BEGIN
  SELECT customer_type INTO v_type
  FROM customer WHERE customer_id = :NEW.customer_id;
  IF v_type <> ’MEMBER’ THEN
    RAISE_APPLICATION_ERROR(-20001,
      ’CustomerType must be MEMBER to have member_customers row.’);
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(-20003, ’Customer does not exist.’);
END;
/
```