THE YOUNGS PIZZA MANAGEMENT SYSTEM (YPMS)

Youngs Pizza is a growing family-owned pizzeria that is doing great, but they are still writing every order down on paper. It is a messy system that is starting to slow them down and cause problems in holding their daily operations back. These include:

- *Operational Inefficiency and Errors:* Handwritten orders are often very hard to read as they are not clear enough which can lead to wrong, unhappy customers and wasted food.
- *No Centralized Data*: Customer information, order history and menu details are spread across different slips and notebooks making it hard to quickly access preferences of customers, track and analyze sales trends, or manage inventory properly.
- *Poor Order and Delivery Tracking*: There's no proper system to track the status of an order from when it was placed down to delivery. This causes delays, miscommunication and makes it hard to hold staff accountable.
- *Financial Reporting Challenges*: Reconciling daily sales and tracking different payment types is time-consuming and often prone to errors.

The purpose of the Youngs Pizza Management System (YPMS) is to solve these problems by providing a centralized, automated and reliable relational database. This system will serve as the main source of information for all business operations, streamlining workflows, improving customer service and enabling data-driven decision-making.

Technical Requirements

The successful implementation of the YPMS depends on the following technical requirements:

i. *Relational Database Management System (RDBMS):* The system will be built using Oracle Database, developed and maintained using Oracle SQL Developer as the primary interface. This enterprise-grade database platform provides strong data security, reliability and powerful SQL querying capabilities.

ii. Normalized Database Schema: The database will be designed to at least the Third Normal Form (3NF) to eliminate redundancy and prevent update anomalies. The main entities that will be implemented are:

1. Customer
2. Employee
3. MenuItem
4. Order
5. OrderItem (Junction Table)
6. Payment

iii. Data Integrity and Constraints: The schema will enforce:

- *Primary Keys*: Unique identifiers for each record. For example, CustomerID, OrderID.
- *Foreign Key Constraints*: To make sure related tables are correctly linked. For instance, an Order must have a valid CustomerID.
- Data Type Validation: Using suitable data types such as INT, VARCHAR, DECIMAL, and DATETIME for all attributes.
- Business Rule Constraints: Using CHECK and NOT NULL constraints to ensure valid data. For example, Price must be greater than 0; Order Status can only be specific values or words like 'Pending', 'Completed'.

iv. *Complex Query Support*: The database will be designed to efficiently handle a variety of complex business queries including:

- Generating detailed customer order histories.
- Calculating total sales revenue for a specific period.
- Identifying the most popular items on the menu.

Listing all orders handled by a specific employee.

DATABASE DESIGN AND ENTITY RELATIONSHIPS

Entity-Relationship (ER) Modeling: A well-defined ER Diagram was created to visually represent the entities, their attributes and the relationships including cardinality between them before building and implementing the database.

Key Relationships:

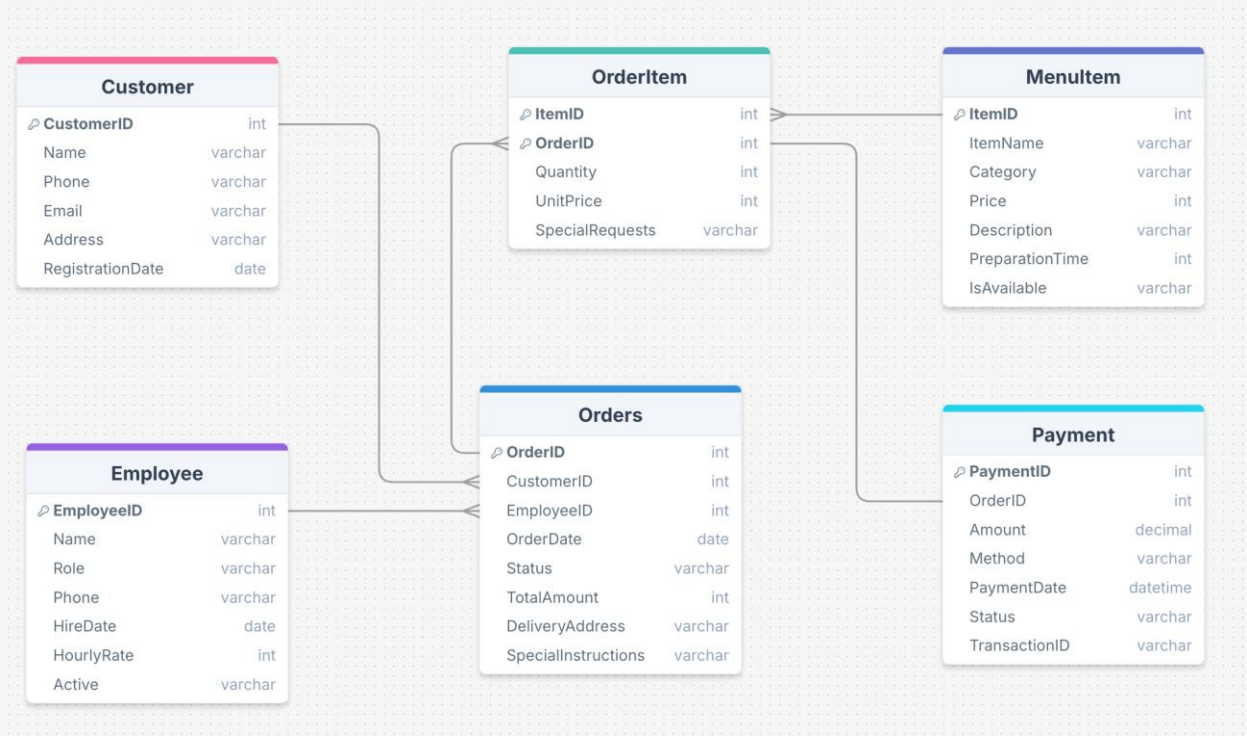
Customer to Orders: One customer places many orders. CustomerID is the foreign key in Order

Employee to Orders: One employee handles many orders. EmployeeID is the foreign key in Order.

Orders to OrderItem: One order has many order items. OrderID is the foreign key in OrderItem.

MenuItem to OrderItem: One menu item can appear in many order items. ItemID is the foreign key in OrderItem.

Order to Payment: One order can have many payments. OrderID is the foreign key in Payment



SEQUENCE CREATION FOR PRIMARY KEYS:

For each entity table sequences were set up. The goal is to automate primary key generation. Each sequence starts at a unique range. This keeps IDs organized by table. Customer IDs start at 1001, employees start at 2001, menu items at 3001, orders at 4001, and payments at 5001. Using sequences avoids duplicates and removes manual ID entry**.**

```
-- Create Sequences for Primary Keys
CREATE SEQUENCE seq_ypms_customer_id START WITH 1001 INCREMENT BY 1;
CREATE SEQUENCE seq_ypms_employee_id START WITH 2001 INCREMENT BY 1;
CREATE SEQUENCE seq_ypms_menuitem_id START WITH 3001 INCREMENT BY 1;
CREATE SEQUENCE seq_ypms_order_id START WITH 4001 INCREMENT BY 1;
CREATE SEQUENCE seq_ypms_payment_id START WITH 5001 INCREMENT BY 1;
```

CREATION OF TABLES AND INSERTION OF DATA SAMPLES

Customer Table: We used standard fields such as name, phone, email, and address. RegistrationDate uses SYSDATE so new records get stamped with the current date. For the data sample inserts. Each row uses the customer sequence to create the primary key. These inserts add four sample customers with realistic contact details. The sequence generated customer IDs automatically.

The sample data in the output shows that the table structure works and the inserts run without errors.

Output:

	CUSTOMERID	NAME	PHONE	EMAIL	ADDRESS	REGISTRATIONDATE
1	1001	Maria Garcia	(555) 100 1234	maria.g@email.com	815 Ohio Ave, Youngstown	22-NOV-25
2	1002	James Wilson	(555) 100 9876	james.wilson@email.com	250 Lincoln Avenue, Younsgtown	22-NOV-25
3	1003	Lisa Chen	(555) 100 0987	lisa.chen@email.com	3230 Belmont Avenue, Youngstown	22-NOV-25
4	1004	Robert Brown	(555) 100 5432	robert.b@email.com	321 Pine Groove, Girard	22-NOV-25

Employee Table: The table stores key information about each worker and the primary key is EmployeeID. Name, role, phone, and hire date help track staff records. HourlyRate stores wage

information with a constraint to prevent negative values. Active shows if the employee still works in the business.

For the data sample inserts, the lines add four sample employees to the table. Each worker has a different role which gives us a diverse dataset for testing queries. The hire date uses SYSDATE, so the database sets the date automatically.

Output:

	EMPLOYEEID	NAME	ROLE	PHONE	HIREDATE	HOURLYRATE	ACTIVE
1	2001	Tom Anderson	Manager	(555) 200 4680	22-NOV-25	25	Y
2	2002	Sarah Martinez	Cashier	(555) 200 2345	22-NOV-25	18.5	Y
3	2003	David Kim	Chef	(555) 200 9753	22-NOV-25	22	Y
4	2004	Emily Johnson	Delivery Staff	(555) 200 7913	22-NOV-25	16	Y

MenuItem Table: This table holds all the items the pizzeria sells. Key attributes are defined like the item name, category, price, a short description, and the preparation time. A check constraint is included to make sure items are assigned only to valid categories like Pizza, Drink, Side, or Dessert. The output contains a few sample menu items and confirms that the table was created successfully and the values were inserted as expected.

Output:

	ITEMID	ITEMNAME	CATEGORY	PRICE	DESCRIPTION	PREPARATIONTIME	ISAVAILABLE
1	3001	Classic Margherita	Pizza	14.99	Fresh mozzarella, tomato sauce, basil	15	Y
2	3002	Pepperoni Feast	Pizza	16.99	Double pepperoni, extra cheese	18	Y
3	3003	Vegetarian Supreme	Pizza	17.99	Mushrooms, peppers, onions, olives, tomatoes	20	Y
4	3004	Hawaiian Paradise	Pizza	16.49	Ham, pineapple, mozzarella	16	Y
5	3005	Coca-Cola	Drink	2.99	500ml bottle	0	Y
6	3006	Garlic Breadsticks	Side	5.99	6 pieces with marinara sauce	8	Y
7	3007	Chocolate Lava Cake	Dessert	6.99	Warm cake with molten chocolate center	10	Y

Order Table: Every order placed by a customer is stored here. Foreign keys were added, linking orders to customers and employees. The order status, total amount, delivery address and special instructions were added as well. The output demonstrates different orders in different statuses,

verifying the orders were inserted correctly, showing each order with its timestamp and other details.

Output:

	ORDERID	CUSTOMERID	EMPLOYEEID	ORDERDATE	STATUS	TOTALAMOUNT	DELIVERYADDRESS	SPECIALINSTRUCTIONS
1	4001	1001	2002	22-NOV-25 03.28.18.539394000 PM	Delivered	38.97	815 Ohio Ave, Youngstown	(null)
2	4002	1002	2002	22-NOV-25 03.28.18.578050000 PM	Out for Delivery	24.98	250 Lincoln Avenue, Younsgtown	(null)
3	4003	1003	2002	22-NOV-25 03.28.18.587282000 PM	Preparing	17.99	(null)	Extra cheese, cut into 8 slices

OrderItem (Junction Table): This table breaks each order into its individual items So instead of just knowing an order total, we can see exactly which items were purchased, the quantity, the unit price and any special requests. The table includes foreign keys linking each item back to both the order and the menu item. The output shows real-world examples like adding “extra basil” or “no olives” and confirms the line items were inserted successfully.

	ORDERID	ITEMID	QUANTITY	UNITPRICE	SPECIALREQUESTS
1	4001	3001	1	14.99	Extra basil
2	4001	3005	2	2.99	(null)
3	4001	3006	1	5.99	(null)
4	4002	3002	1	16.99	(null)
5	4002	3005	2	2.99	(null)
6	4003	3003	1	17.99	No olives

Payment Table: This table handles how orders are paid for, whether by cash, card, debit card or online. Fields for the payment amount, method, status and a transaction ID for tracking were included. The output includes completed payments and one pending payment and shows that the table is working correctly.

Output:

	PAYMENTID	ORDERID	AMOUNT	METHOD	PAYMENTDATE	STATUS	TRANSACTIONID
1	5001	4001	38.97	Credit Card	22-NOV-25 03.52.24.105703000 PM	Completed	TXN001234
2	5002	4002	24.98	Cash	22-NOV-25 03.52.24.172098000 PM	Completed	(null)
3	5003	4003	17.99	Online	22-NOV-25 03.52.24.188351000 PM	Pending	(null)

SCENARIO BASED INSIGHTS:

Scenario 1: Finding the top 5 most popular menu items by quantity sold

```
--- "What are our top 5 most popular menu items by quantity sold?"

SELECT
    m.ItemName,
    m.Category,
    m.Price,
    SUM(oi.Quantity) AS Total_Quantity_Sold,
    COUNT(oi.OrderID) AS Number_of_Orders,
    SUM(oi.Quantity * oi.UnitPrice) AS Total_Revenue
FROM MenuItem m
JOIN OrderItem oi ON m.ItemID = oi.ItemID
JOIN Orders o ON oi.OrderID = o.OrderID
WHERE o.Status != 'Cancelled'
GROUP BY m.ItemName, m.Category, m.Price
ORDER BY Total_Quantity_Sold DESC;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 5 in 0.115 seconds

	ITEMNAME	CATEGORY	PRICE	TOTAL_QUANTITY_SOLD	NUMBER_OF_ORDERS	TOTAL_REVENUE
1	Coca-Cola	Drink	2.99	4	2	11.96
2	Vegetarian Supreme	Pizza	17.99	1	1	17.99
3	Pepperoni Feast	Pizza	16.99	1	1	16.99
4	Classic Margherita	Pizza	14.99	1	1	14.99
5	Garlic Breadsticks	Side	5.99	1	1	5.99

Scenario 2: Finding orders paid in Cash

```
---- Finding all orders paid in cash.
SELECT p.PaymentID, p.OrderID, p.Amount, p.Method, p.PaymentDate, c.Name AS CustomerName
FROM Payment p
JOIN Orders o ON p.OrderID = o.OrderID
JOIN Customer c ON o.CustomerID = c.CustomerID
WHERE p.Method = 'Cash';
```

Script Output | Query Result

SQL | All Rows Fetched: 1 in 0.156 seconds

	PAYMENTID	ORDERID	AMOUNT	METHOD	PAYMENTDATE	CUSTOMERNAME
1	5002	4002	24.98	Cash	25-NOV-25 03.09.44.430794000 PM	James Wilson

Scenario 3: Finding the average order value

```sql
---- Find the average order value.
SELECT ROUND(AVG(TotalAmount),2) AS AvgOrderValue
FROM Orders;
```

Script Output | Query Result | Query Result 1

SQL | All Rows Fetched: 1 in 0.072 seconds

	AVGORDERVALUE
1	27.31