Part B / Design Normalized Database

Course Name: CS5200 Database Management Systems

Sai Karthikeyan, Sura

Spring 2025

1. Introduction

This document presents an approach to normalized relational database schema for managing restaurant visits. The schema follows Third Normal Form (3NF) to eliminate redundancy and ensure data integrity. The steps involved include listing the initial functional dependencies for a restaurant visit relation from the raw data, performing decomposition to ensure normalization to 3NF, and designing an ERD to illustrate the final structure.

2. Functional Dependencies From Raw Database

$$\text{VisitID} \rightarrow \left\{ \begin{array}{l} \text{Restaurant, ServerEmpID, ServerName, StartDateHired, EndDateHired, HourlyRate,} \\ \text{ServerBirthDate, ServerTIN, VisitDate, VisitTime, MealType, PartySize, Genders,} \\ \text{WaitTime, CustomerName, CustomerPhone, CustomerEmail, LoyaltyMember,} \\ \text{FoodBill, TipAmount, DiscountApplied, PaymentMethod, OrderedAlcohol, AlcoholBill} \end{array} \right\}$$

$$\text{ServerEmpID} \rightarrow \{\ \text{ServerName, StartDateHired, EndDateHired, HourlyRate, ServerBirthDate, ServerTIN}\ \}$$

$$\text{CustomerName} \rightarrow \{\text{CustomerPhone, CustomerEmail, LoyaltyMember}\}$$

3. Normalization

3.1. Approach To Normalization

Decomposing into 3NF

The normalization process was essential to ensure that the database adheres to the principles of 3NF to eliminate redundancy, enhance data integrity, and streamline data management.

The original relation had multiple attributes that didn't adhere to the principle of atomicity, leading to redundancy and potential anomalies during insert, update, and delete operations. By decomposing the relation into smaller, more focused entities, I ensured that each table now represents a single concept and is free from redundancy.

Partial dependencies were eliminated by ensuring each table has a single primary key. Transitive dependencies were removed by creating separate tables for related entities. For example, Restaurant in the original table was replaced with RestaurantID in the Visit table, ensuring that RestaurantName is stored only once in the Restaurant table.

Functional Dependencies From Normalized Database

$$\text{PaymentID} \rightarrow \{\text{PaymentMethod}\}$$

$$\text{CustomerID} \rightarrow \{\text{CustomerName}, \text{CustomerPhone}, \text{CustomerEmail}, \text{LoyaltyMember}\}$$

$$\text{RestaurantID} \rightarrow \{\text{RestaurantName}\}$$

$$\text{MealTypeID} \rightarrow \{\text{MealType}\}$$

$$\text{ServerEmpID} \rightarrow \left\{ \text{ServerName}, \text{StartDateHired}, \text{EndDateHired}, \text{HourlyRate}, \text{ServerBirthDate}, \text{ServerTIN} \right\}$$

$$\text{VisitID} \rightarrow \left\{ \begin{array}{l} \text{RestaurantID}, \text{ServerEmpID}, \text{VisitDate}, \text{VisitTime}, \text{MealTypeID}, \text{PartySize}, \text{Genders}, \\ \text{WaitTime}, \text{CustomerID} \end{array} \right\}$$

$$\text{BillID} \rightarrow \left\{ \begin{array}{l} \text{VisitID}, \text{FoodBill}, \text{TipAmount}, \text{DiscountApplied}, \text{PaymentID}, \text{OrderedAlcohol}, \\ \text{AlcoholBill} \end{array} \right\}$$

Designing the Entity Relationship Diagram

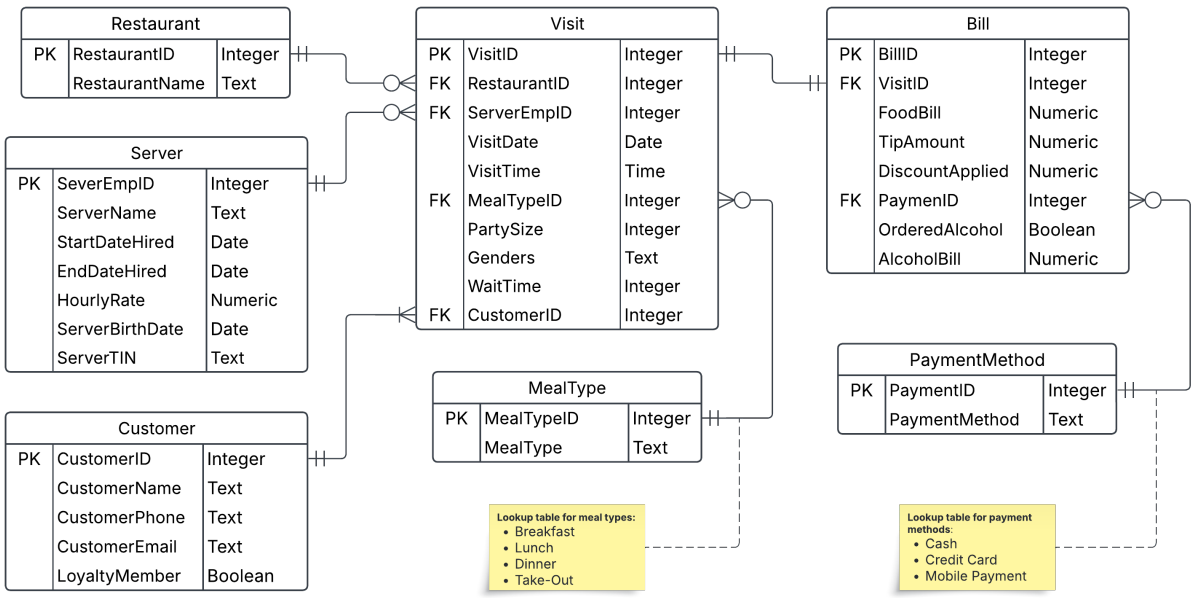

Figure 1: Entity Relationship Diagram

3.2 Proof To Normalization

PaymentMethod

1NF: All attributes (PaymentID, PaymentMethod) are atomic with no repeating groups.
2NF: PaymentID uniquely identifies each payment method, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; PaymentMethod depends directly on PaymentID.

Customer

1NF: All attributes (CustomerID, CustomerName, CustomerPhone, CustomerEmail, LoyaltyMember) are atomic with no repeating groups.
2NF: CustomerID uniquely identifies each customer, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; all attributes depend directly on CustomerID.

Restaurant

1NF: All attributes (RestaurantID, RestaurantName) are atomic with no repeating groups.
2NF: RestaurantID uniquely identifies each restaurant, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; RestaurantName depends directly on RestaurantID.

MealType

1NF: All attributes (MealTypeID, MealType) are atomic with no repeating groups.
2NF: MealTypeID uniquely identifies each meal type, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; MealType depends directly on MealTypeID.

Server

1NF: All attributes (ServerEmpID, ServerName, StartDateHired, EndDateHired, HourlyRate, ServerBirthDate, ServerTIN) are atomic with no repeating groups.
2NF: ServerEmpID uniquely identifies each server, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; all attributes depend directly on ServerWmpID.

Visit

1NF: All attributes (VisitID, RestaurantID, ServerEmpID, VisitDate, VisitTime, MealTypeID, PartySize, Genders, WaitTime, CustomerID) are atomic with no repeating groups.
2NF: VisitID uniquely identifies each visit, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; all attributes depend directly on VisitID.

Bill

1NF: All attributes (BillID, VisitID, FoodBill, TipAmount, DiscountApplied, PaymentID, OrderedAlcohol, AlcoholBill) are atomic with no repeating groups.
2NF: BillID uniquely identifies each bill, and all non-key attributes fully depend on it.
3NF: No transitive dependencies; all attributes depend directly on BillID.

4. Database Population and Data Management

Default Values for Inconsistent Data For Server, I introduced a default row with ServerEmpID equal to '0' to handle cases where server data is missing or inconsistent

For Customer, I introduced a default row with CustomerID equal to '0' to handle visits where customer details are not provided.

Data Population I inserted data into the tables in a logical order, First I populated independent tables (PaymentMethod, Restaurant, MealType, Server, Customer). Then, I populated the Visit table, mapping foreign keys (CustomerID, RestaurantID, MealTypeID, ServerEmpID) using the data from the independent tables. Finally, I populated the Bill table, linking it to Visit and PaymentMethod. I handled the insertion in batches to manage the large dataset efficiently.