SSN COLLEGE OF ENGINEERING, KALAVAKKAM-603110

Department of Computer Science and Engineering

UCS2404 – Database Management Systems
II Year CSE - (IV Semester)

Academic Year 2022-23
Batch: 2021- 2025

CULINARY COMPASS

Culinary Compass: Restaurant Management System

Database Design, Identification of Constraints and Functional Dependencies (FD)
Normalization of relations into 3NF or BCNF

Faculty In-charge: Dr. N Sujaudeen

Project Students:
Mohith A (3122 21 5001 053)
Keerthan V S (3122 21 5001 043)
Jahnavi Murali (3122 21 5001 038)

Description and Database Requirements:

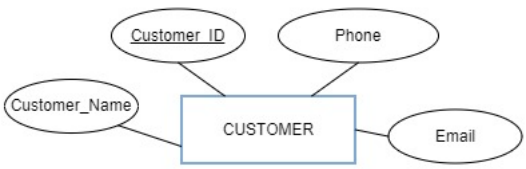
A database schema design is created based on the following requirements of the restaurant database:

1. The restaurant has multiple customers who are identified by a unique customer ID.
    a. Each customer has a name, email address and phone number.
    b. A customer can place any number of orders and make any number of reservations.
2. The restaurant keeps track of the various reservations made by the customers.
    a. Each reservation has a unique reservation ID, date and time of reservation and the number of diners.
    b. A reservation can be made for multiple tables at the restaurant.
3. The restaurant employs several people in different roles, namely, the manager, chefs and waiters.
    a. Each employee has a unique employee ID, name, address, phone number, email address, position, and salary.
    b. A waiter can wait/ service multiple orders in the restaurant.
4. The restaurant has multiple tables/ seating areas to host the customers.
    a. Each table is identified by a unique table number and can host a limited number of diners, feature, and status (Occupied/Unoccupied).
    b. A table is serviced by only one waiter.
5. The restaurant keeps track of the various ingredients required to cook several dishes and the respective quantities available.
6. The restaurant has a menu of food items.
    a. Each food item has its own unique ID, name, type/ cuisine it belongs to and its price per unit.
    b. Each food item requires/ needs multiple food (basic) ingredients and the quantity needed for each.
7. The restaurant keeps track of the orders placed.
    a. Each order is identified by a unique ID.
    b. An order can contain multiple food items of varying quantities.
    c. An order also consists of the net price for each food item and the total bill amount.

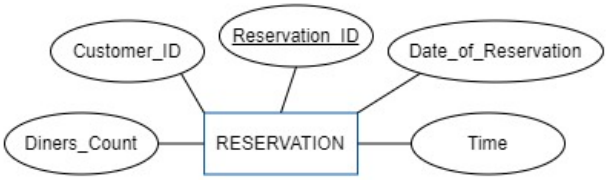
Identifying Entity Types:

The following entity types have been identified from the requirements mentioned in the section above.

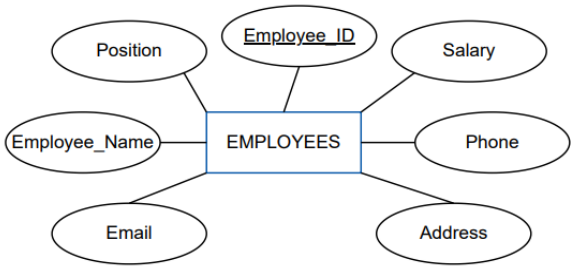
1. Customer (Customer_ID, Customer_Name, Email, Phone_number)



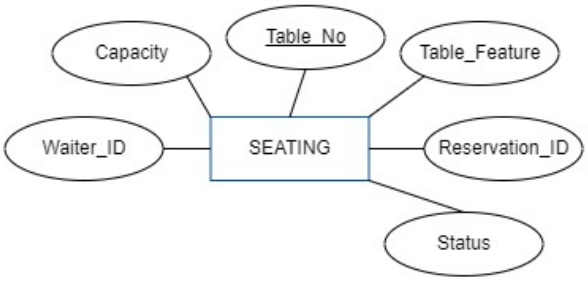
2. Reservation (Reservation_ID, Customer_ID, Date_of_reservation, Time, Diners_Count)



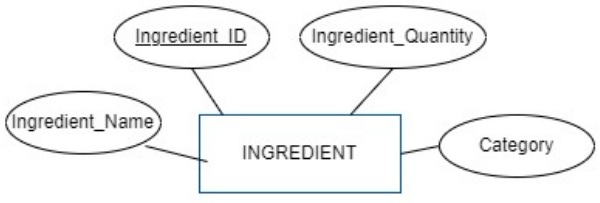
3. Employee (Employee_ID, Employee_Name, Address, Email, Phone, Salary, Position)



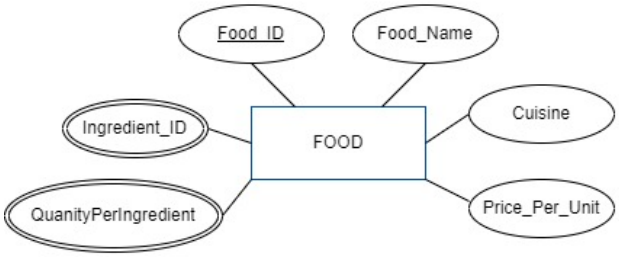
4. Seating (Table_No, Capacity, Table_Feature, Waiter_ID, Reservation_ID, Status)



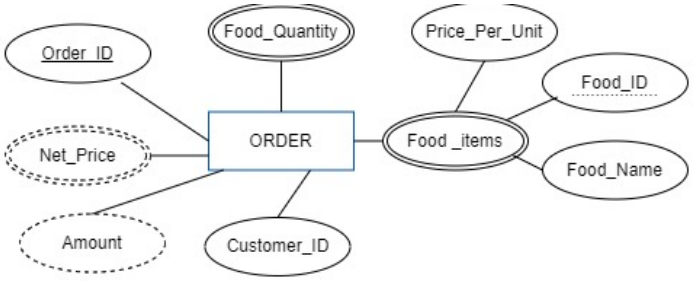
5. Ingredient (Ingredient_ID, Ingredient_Name, Category, Ingredient_Quantity)



6. Food (Food_ID, Food_Name, Cuisine, Price_Per_Unit, {Ingredient_ID}, {Quantity_Per_Ingredient})



7. Order (Order_ID, {Food_items (Food_ID, Food_Name, Price_Per_Unit)}, {Quantity}, Net_Price, Amount)



Identifying Relationships:

The following relationships have been identified from the database requirements given.

1. *Makes:* Relationship between Customer and Reservation. A customer can make any number of reservations which is always made by a single customer.
   Cardinality Ratio – 1: N

2. *Made for:* Relationship between Reservation and Seating. A reservation can be made for any number of tables which is reserved by one only at a time.
   Cardinality Ratio – 1: N

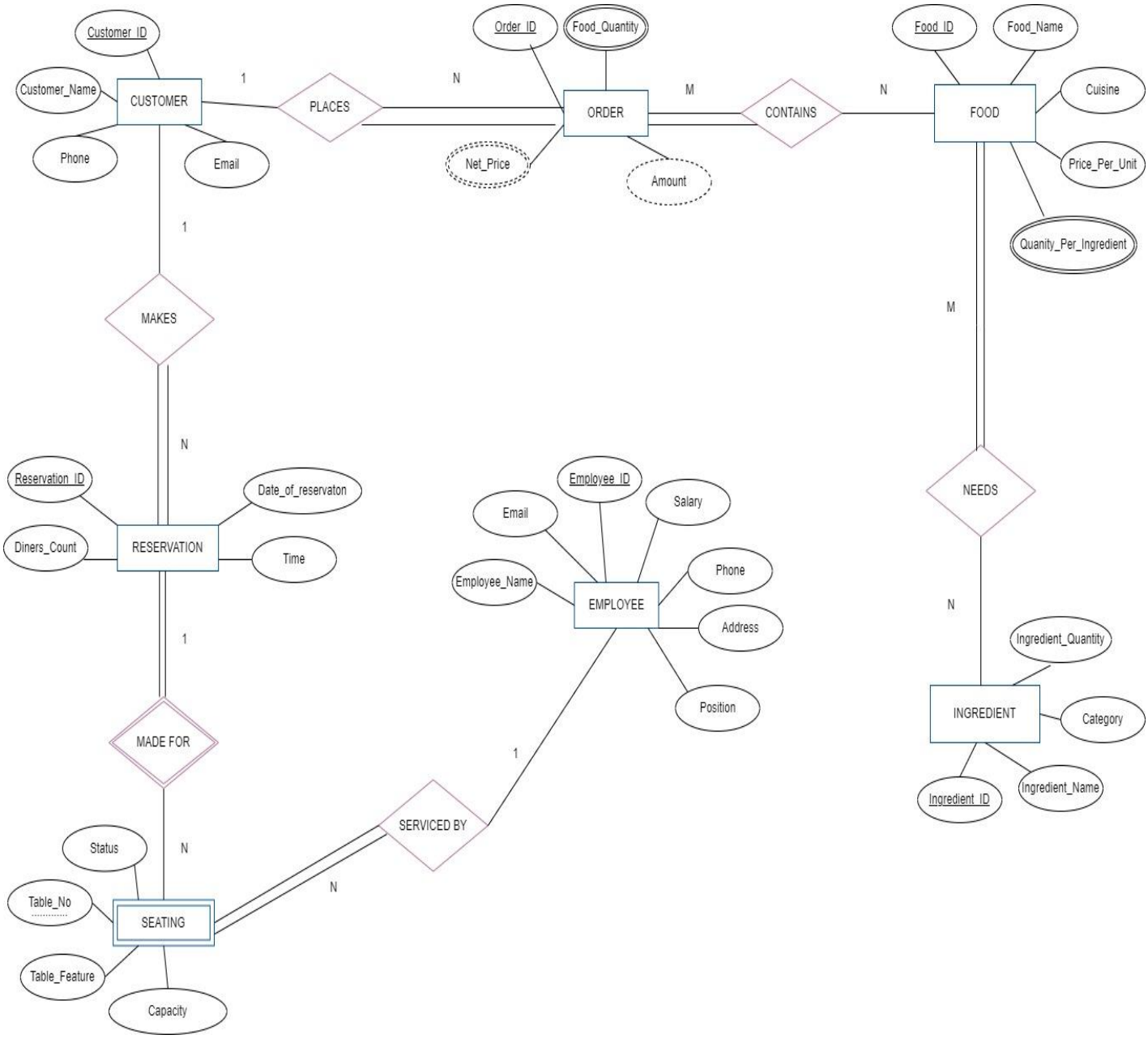
3. *Serviced by:* Relationship between Seating and Employee. A waiter can wait multiple tables but a table is always serviced by one only.
   Cardinality Ratio – N: 1

4. *Places:* Relationship between Customer and Order. A customer can place any number of orders and an order is always placed by 1 customer.
   Cardinality Ratio – 1: N

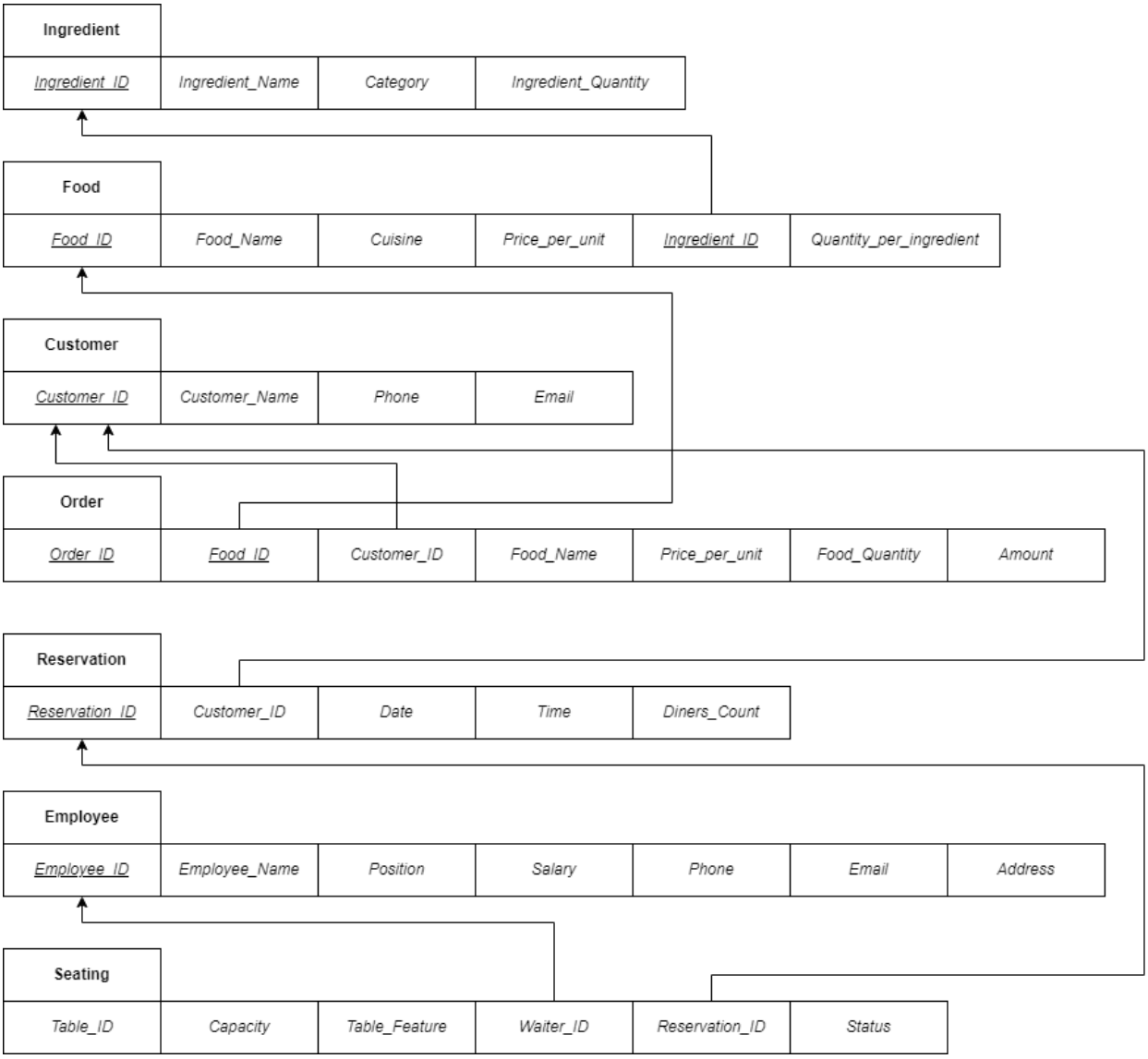
5. *Contains:* Relationship between Order and Menu entity types. Every order contains several food items.
   Cardinality Ratio - M: N

6. *Needs:* Relationship between Menu and Ingredients entity types. Every food item needs several ingredients.
   Cardinality Ratio - M: N

Entity-Relationship Diagram:



Schema Diagram/ Relational Model:



Tables and Attributes:

Tables	**Attributes**
Ingredient	• Ingredient_ID • Ingredient_Name • Category • Ingredient_Quantity
Food	• Food_ID • Food_Name

	• Cuisine • Price_per_unit • Ingredient_ID • Quantity_per_ingredient
Customer	• Customer_ID • Customer_Name • Email • Phone
Order	• Order_ID • Food_ID • Customer_ID • Food_Name • Price_per_unit • Food_Quantity • Amount
Reservation	• Reservation_ID • Customer_ID • Date • Time • Diners_Count
Employee	• Employee_ID • Employee_Name • Position • Salary • Phone • Email • Address
Seating	• Table_ID • Capacity • Table_Feature • Waiter_ID • Reservation_ID • Status

Constraints:

Relation: Ingredient

Key Constraints:

PRIMARY KEY: Ingredient_ID

Domain Constraints:

CHECK: Ingredient_Quantity >= 0

Relation: Food

Key Constraints:

PRIMARY KEY: Food_ID

FOREIGN KEY: Ingredient_ID referencing Ingredient.Ingredient_ID

Domain Constraints:

- CHECK: Quantity_per_ingredient > 0
- CHECK: Price_per_unit > 0
- NOT NULL: Food_Name

Relation: Customer

Key Constraints:

PRIMARY KEY: Customer_ID

Domain Constraints:

- CHECK: Email LIKE '%@% (the check constraint LIKE '%@%' ensures that the email address column (Email) contains the '@' symbol)

Relation: Order

Key Constraints:

PRIMARY KEY: Order_ID, Food_ID

FOREIGN KEY: Food_ID referencing Food.Food_ID and Customer_ID referencing Customer. Customer_ID

Domain Constraints:

- CHECK: Price_per_unit > 0 (should be greater than zero)
- CHECK: Food_Quantity > 0 (should be greater than zero)

Relation: Reservation

Key Constraints:

PRIMARY KEY: Reservation_ID

FOREIGN KEY: and Customer_ID referencing Customer. Customer_ID

Domain Constraints:

- CHECK: Diners_Count > 0 (should be greater than zero)
- NOT NULL: Customer_ID

Relation: Employee

Key Constraints:

PRIMARY KEY: Employee_ID

Domain Constraints:

- CHECK: Salary >= 5000 (minimum salary of waiter should be 5000)
- CHECK: Email LIKE '%@% (the check constraint LIKE '%@%' ensures that the email address column (Email) contains the '@' symbol.)

Relation: Seating

Key Constraints:

FOREIGN KEY: Reservation_ID referencing Reservation. Reservation_ID and Waiter_ID referencing Employee.Employee_ID

Domain Constraints:

CHECK: Capacity > 0 (should be greater than zero)

Identifying Functional Dependencies:

Relation	**Functional Dependencies (FDs)**
Ingredient	Ingredient_ID → Ingredient_Name, Category, Ingredient_Quantity
Food	Food_ID → Food_Name, Cuisine, Price_per_unit Food_ID, Ingredient_ID → Quantity_per_ingredient
Customer	Customer_ID → Customer_Name, Phone, Email
Order	Order_ID → Customer_ID, Amount Food_ID → Food_Name, Price_per_unit Order_ID, Food_ID → Food_Quantity
Reservation	Reservation_ID → Customer_ID, Date_of_Reservation, Time, Diners_Count
Employee	Employee_ID → Employee_Name, Position, Salary, Phone, Email, Address
Seating	Table_ID → Capacity, Table_Feature, Waiter_ID

Identifying Candidate Keys:

Relation: Ingredient

Ingredient_ID → Ingredient_Name, Category, Ingredient_Quantity
a → b c d

$a^+ \longrightarrow$ { a, b, c, d }

Ingredient_ID is a super key while its proper subset is not.

Therefore, Candidate key = Ingredient_ID

Relation: Food

Food_ID → Food_Name, Cuisine, Price_per_unit
a → b c d

Food_ID, Ingredient_ID → Quantity_per_ingredient
a e → f

$abcdef^+ \longrightarrow$	{ a, b, c, d, e, f }	| Super key
$aef^+ \longrightarrow$	{ a, b, c, d, e, f}	| Super key
$ae^+ \longrightarrow$	{ a, b, c, d, e, f }	| Super key
$a^+ \longrightarrow$	{ b, c, d }	| Not a super key
$e^+ \longrightarrow$	{ e }	| Not a super key

Food_ID, Ingredient_ID i.e., ae, is a super key while its proper subset is not.

Therefore, Candidate key = Food_ID, Ingredient_ID

Relation: Customer

Customer_ID → Customer_Name, Phone, Email
a → b c d

$a^+ \longrightarrow$ { a, b, c, d }

Customer_ID is a super key while its proper subset is not.

Therefore, Candidate key = Customer_ID

Relation: Order

Order_ID → Customer_ID, Amount
a → c g

Food_ID → Food_Name, Price_per_unit
b → d e

Order_ID, Food_ID → Food_Quantity
a b → f

$abcdefg^+ \longrightarrow$	{ a, b, c, d, e, f , g }	| Super key
$abf^+ \longrightarrow$	{ a, b, e, g, c, d, f }	| Super key

$ab^+ \longrightarrow \{ a, b, g, c, d, e, f \}$ | Super key
$a^+ \longrightarrow \{ b, g \}$ | Not a super key
$b^+ \longrightarrow \{ b, c, d \}$ | Not a super key

Order_ID, Food_ID i.e., ab is a super key while its proper subset is not.

Therefore, Candidae key = Order_ID, Food_ID

Relation: Reservation

Reservation_ID → Customer_ID, Date_of_reservation, Time, Diners_Count
a → b c d e

$a^+ \longrightarrow \{ a, b, c, d, e \}$

Reservation_ID is a super key while its proper subset is not.

Therefore, Candidate key = Reservation_ID

Relation: Employee

Employee_ID → Employee_Name, Position, Salary, Phone, Email, Address
a → b c d e f g

$a^+ \longrightarrow \{ a, b, c, d, e, f, g \}$

Employee_ID is a super key while its proper subset is not.

Therefore, Candidate key = Employee_ID

Relation: Seating

Table_No → Capacity, Table_Feature, Waiter_ID
a → b c d

$abcdef^+ \longrightarrow \{ a, b, c, d, e, f \}$ | Super key
$aef^+ \longrightarrow \{ a, b, c, d, e, f \}$ | Super key

Table_ID, Reservation_ID, Status is a super key while its proper subset is not.

Therefore, Candidate key = Table_ID, Reservation_ID, Status

Tables	**Candidate Key(s)**
Ingredient	Ingredient_ID
Food	Food_ID, Ingredient_ID

Customer	Customer_ID
Order	Order_ID, Food_ID
Reservation	Reservation_ID
Employee	Employee_ID
Seating	Table_ID, Reservation_ID, Status

Normalisation:

Relation: Ingredient

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Ingredient_ID → Ingredient_Name, Category, Ingredient_Quantity

There is no partial dependency. Therefore, the relation is in 2NF.

Checking 3NF:

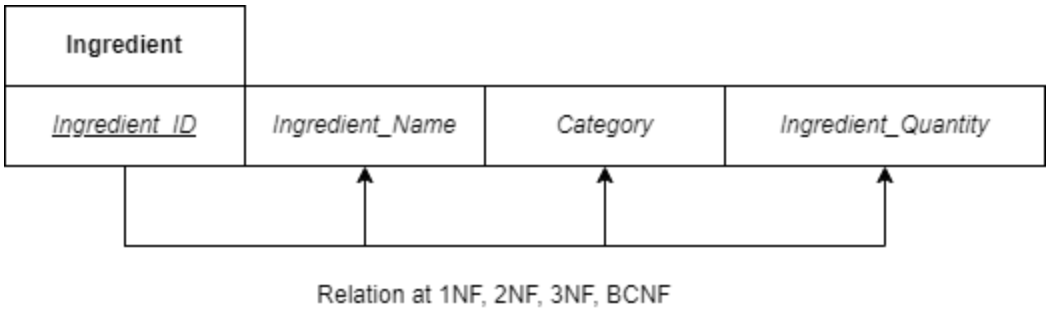
FD: Ingredient_ID → Ingredient_Name, Category, Ingredient_Quantity

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

Checking BCNF:

FD: Ingredient_ID → Ingredient_Name, Category, Ingredient_Quantity

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.



Relation: Food

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Food_ID → Food_Name, Cuisine, Price_per_unit
Food_ID, Ingredient_ID → Quantity_per_ingredient

Candidate Key: {Food_ID, Ingredient_ID}

The subset of Candidate Key (Food_ID) determines non-prime attributes. Hence, FD: (Food_ID → Food_Name, Cuisine, Price_per_unit) is a partial dependency. Therefore, the relation is not in 2NF.

Normalize to 2NF:

By splitting the relation,

(1) Food: (Food_ID, Food_Name, Cuisine, Price_per_unit)
FD: Food_ID → Food_Name, Cuisine, Price_per_unit
Candidate Key: {Food_ID}

(2) Food_Ingredient: (Food_ID, Ingredient_ID, Quantity_per_ingredient)
FD: Food_ID, Ingredient_ID → Quantity_per_ingredient
Candidate Key: {Food_ID, Ingredient_ID}

The new relations are in 2NF.

Checking 3NF:

For the split relations,

(1) Food: (Food_ID, Food_Name, Cuisine, Price_per_unit)
FD: Food_ID → Food_Name, Cuisine, Price_per_unit

Candidate Key: {Food_ID}

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

(2) Food_Ingredient: (Food_ID, Ingredient_ID, Quantity_per_ingredient)
FD: Food_ID, Ingredient_ID → Quantity_per_ingredient
Candidate Key: {Food_ID, Ingredient_ID}

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

Checking BCNF:

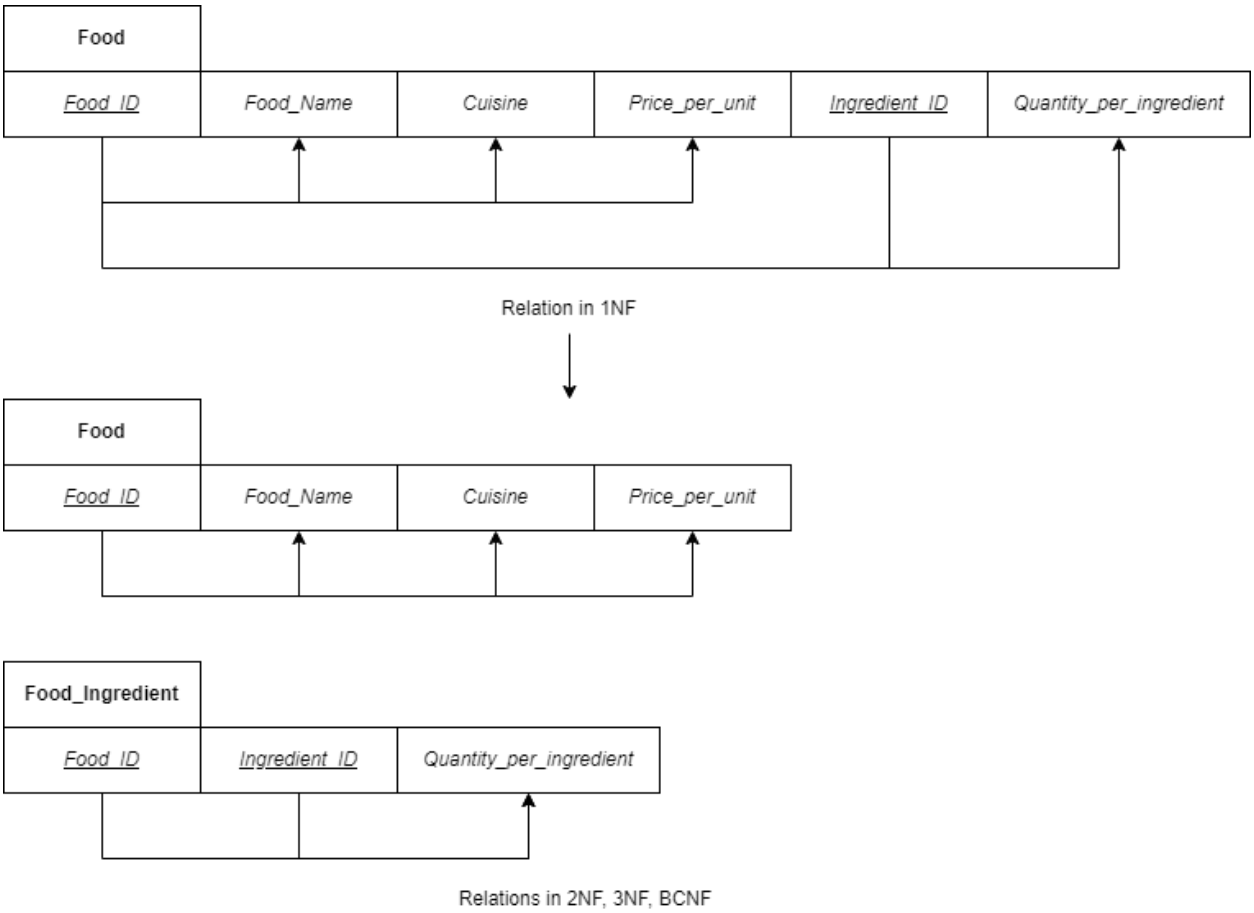
For the split relations,

(1) Food: (Food_ID, Food_Name, Cuisine, Price_per_unit)
FD: Food_ID → Food_Name, Cuisine, Price_per_unit
Candidate Key: {Food_ID}

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.

(2) Food_Ingredient: (Food_ID, Ingredient_ID, Quantity_per_ingredient)
FD: Food_ID, Ingredient_ID → Quantity_per_ingredient
Candidate Key: {Food_ID, Ingredient_ID}

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.



Relation: Customer

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Customer_ID → Customer_Name, Phone, Email

There is no partial dependency. Therefore, the relation is in 2NF.

Checking 3NF:

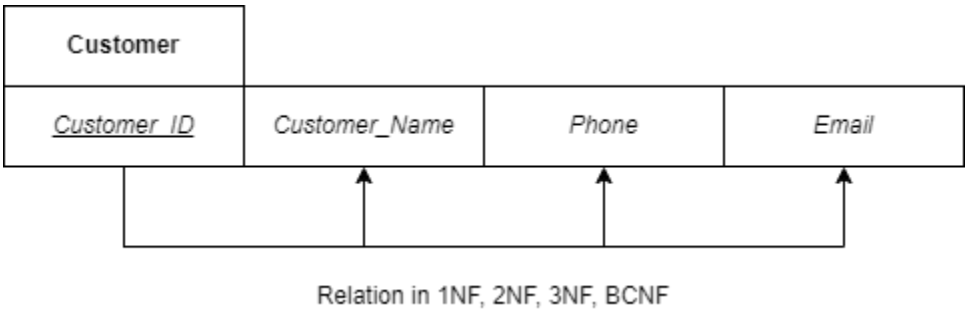
FD: Customer_ID → Customer_Name, Phone, Email

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

Checking BCNF:

FD: Customer_ID → Customer_Name, Phone, Email

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.



Relation: Order

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Order_ID → Customer_ID, Amount
Food_ID → Food_Name, Price_per_unit
Order_ID, Food_ID → Food_Quantity

Candidate Key: {Food_ID, Order_ID}

The subset of Candidate Key (Order_ID) / (Food_ID) determines non-prime attributes. Hence, FDs: (Order_ID → Customer_ID, Amount) / (Food_ID → Food_Name, Price_per_unit) are partial dependencies. Therefore, the relation is not in 2NF.

Normalize to 2NF:

By splitting the relation,

(1) Bill: (Order_ID, Customer_ID, Amount)
FD: Order_ID → Customer_ID, Amount
Candidate Key: {Order_ID}

(2) Food: (Food_ID, Food_Name, Price_per_unit)
FD: Food_ID → Food_Name, Price_per_unit
Candidate Key: {Food_ID}

This relation is redundant. Therefore, we neglect it.

(3) Order: (Order_ID, Food_ID, Food_Quantity)
FD: Order_ID, Food_ID → Food_Quantity
Candidate Key: {Order_ID, Food_ID}

The new relations Bill and Order are in 2NF.

Checking 3NF:

For the split relations,

(1) Bill: (Order_ID, Customer_ID, Amount)
FD: Order_ID → Customer_ID, Amount
Candidate Key: {Order_ID}

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

(2) Order: (Order_ID, Food_ID, Food_Quantity)
FD: Order_ID, Food_ID → Food_Quantity
Candidate Key: {Order_ID, Food_ID}

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

Checking BCNF:

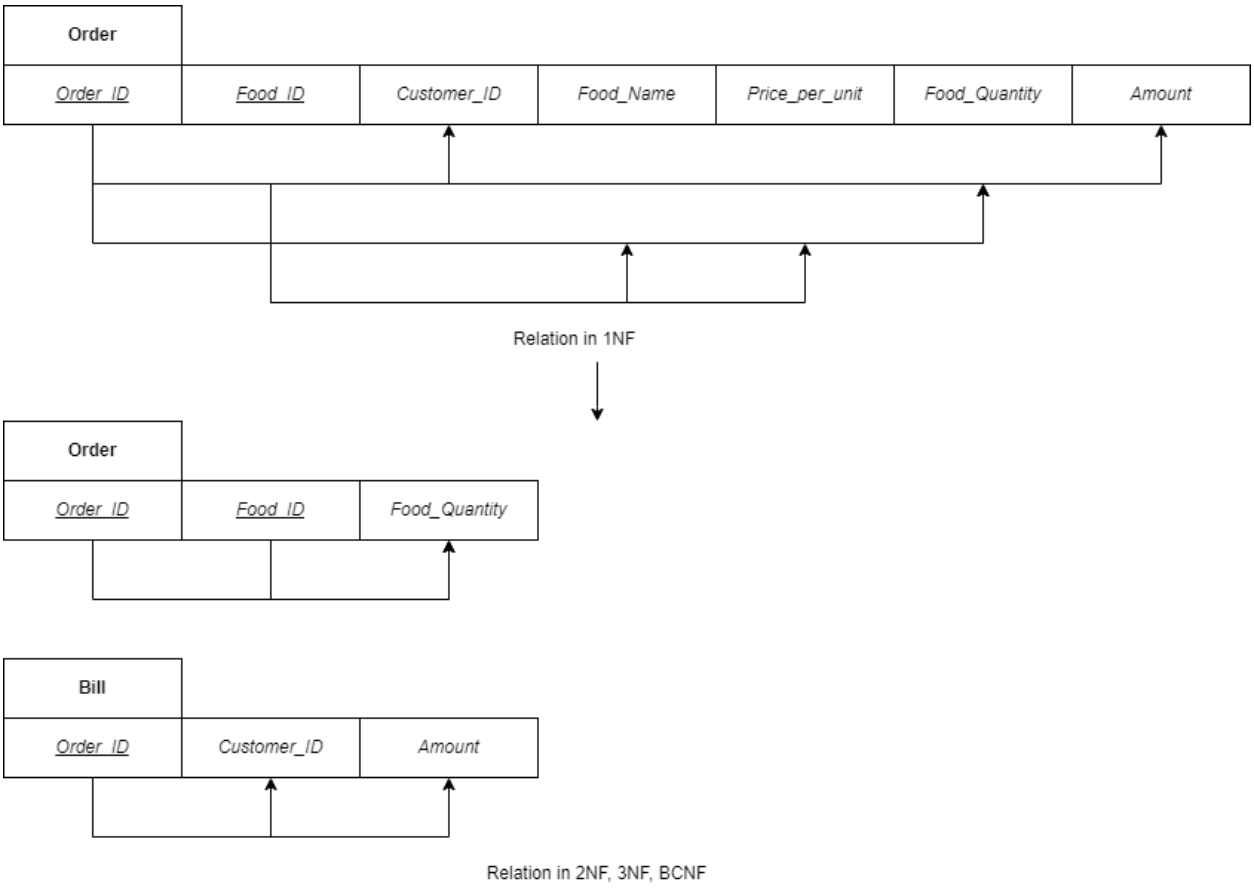
For the split relations,

(1) Bill: (Order_ID, Customer_ID, Amount)
FD: Order_ID → Customer_ID, Amount
Candidate Key: {Order_ID}

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.

(2) Order: (Order_ID, Food_ID, Food_Quantity)
FD: Order_ID, Food_ID → Food_Quantity
Candidate Key: {Order_ID, Food_ID}

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.



Relation: Reservation

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Reservation_ID → Customer_ID, Date_of_Reservation, Time, Diners_Count

There is no partial dependency. Therefore, the relation is in 2NF.

Checking 3NF:

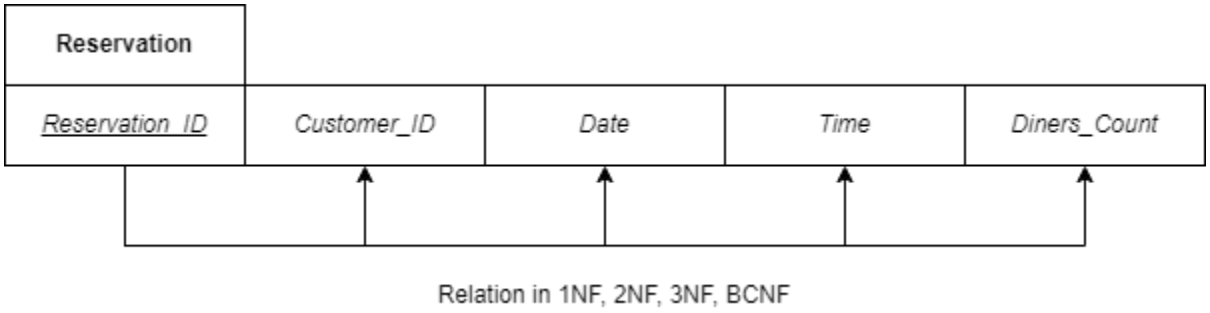
FD: Reservation_ID → Customer_ID, Date_of_Reservation, Time, Diners_Count

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

Checking BCNF:

FD: Reservation_ID → Customer_ID, Date_of_Reservation, Time, Diners_Count

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.



Relation in 1NF, 2NF, 3NF, BCNF

Relation: Employee

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Employee_ID → Employee_Name, Position, Salary, Phone, Email, Address

There is no partial dependency. Therefore, the relation is in 2NF.

Checking 3NF:

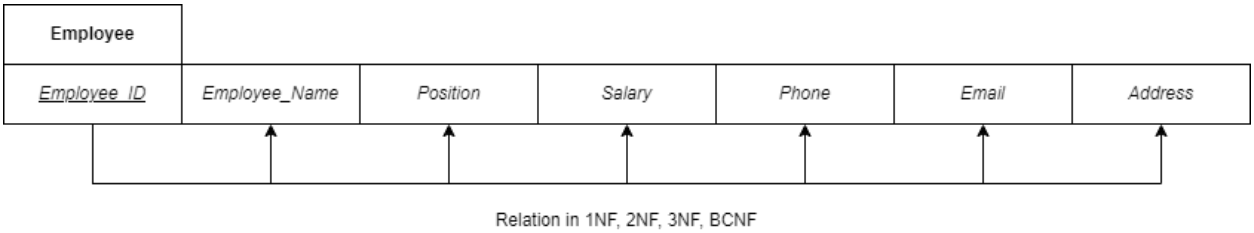
FD: Employee_ID → Employee_Name, Position, Salary, Phone, Email, Address

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

Checking BCNF:

FD: Employee_ID → Employee_Name, Position, Salary, Phone, Email, Address

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.



Relation: Seating

Checking 1NF:

The relation consists of no multi-valued or composite attribute and has only atomic values, so it is in 1NF.

Checking 2NF:

FD: Table_ID → Capacity, Table_Feature, Waiter_ID

Candidate Key: {Table_ID, Reservation_ID, Status}

The subset of Candidate Key (Table_ID) determines non-prime attributes. Hence, FD: (Table_ID → Capacity, Table_Feature, Waiter_ID) is a partial dependency. Therefore, the relation is not in 2NF.

Normalize to 2NF:

By splitting the relation,

(1) Seating: (Table_ID, Capacity, Table_Feature, Waiter_ID)
FD: Table_ID → Capacity, Table_Feature, Waiter_ID
Candidate Key: {Table_ID}

(2) Table_Status: (Table_ID, Reservation_ID, Status)
FD: None
Candidate Key: {Table_ID, Reservation_ID, Status}

The new relations are in 2NF.

Checking 3NF:

For the split relations,

(1) Seating: (Table_ID, Capacity, Table_Feature, Waiter_ID)
FD: Table_ID → Capacity, Table_Feature, Waiter_ID
Candidate Key: {Table_ID}

In the given FD, LHS is a superkey and hence there is no transitive dependency. Therefore, the relation is in 3NF.

(2) Table_Status: (Table_ID, Reservation_ID, Status)
FD: None
Candidate Key: {Table_ID, Reservation_ID, Status}

There is no FD. Therefore, the relation is in 3NF.

Checking BCNF:

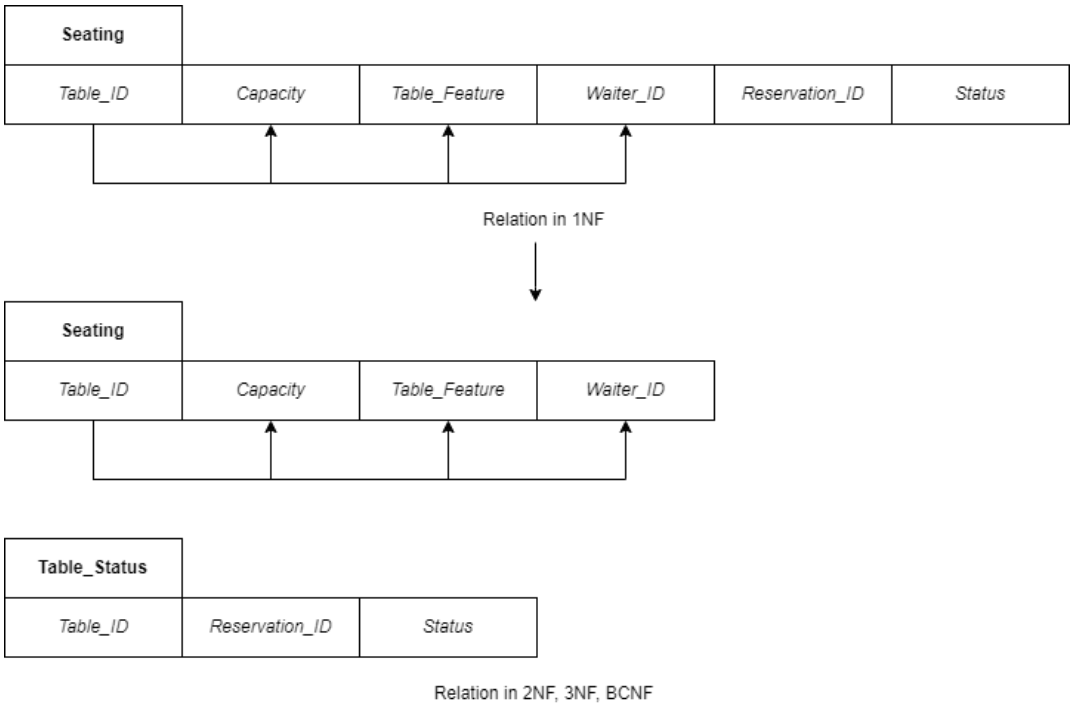
For the split relations,

(1) Seating: (Table_ID, Capacity, Table_Feature, Waiter_ID)
FD: Table_ID → Capacity, Table_Feature, Waiter_ID
Candidate Key: {Table_ID}

In the given FD, LHS is a superkey. Therefore, the relation is in BCNF.

(2) Table_Status: (Table_ID, Reservation_ID, Status)
FD: None
Candidate Key: {Table_ID, Reservation_ID, Status}

There is no FD. Therefore, the relation is in BCNF.



Schema Diagram: