Sri Sivasubramaniya Nadar College of Engineering,
Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)

Computer Science and Engineering

UCS2404 - DATABASE MANAGEMENT SYSTEMS

LIBRARY MANAGEMENT SYSTEM

Mugilkrishna D U – 3122225001073
Nikilesh Jayaguptha – 3122225001081
Niranjan B – 3122225001082

PHASE – 1: IDENTIFICATION OF CONSTRAINTS AND FUNCTIONAL DEPENDENCIES (FD) AMONG SET OF ATTRIBUTES

PROBLEM STATEMENT:

The Library Management System (LMS) project implemented in NetBeans focuses on automating library operations through efficient book and patron management, borrowing, returning, and reservation functionalities. SQLPlus, a command-line interface for Oracle Database, is utilized within the project for database management tasks such as creating schemas, tables, and executing SQL queries to ensure data integrity and normalization. The LMS allows librarians to add, update, and remove books with details like title, author, genre, and availability, while patrons can search and browse the catalog, check out books, and manage their accounts. SQLPlus facilitates backend operations crucial for maintaining a well-organized database structure, supporting seamless integration with the frontend developed in NetBeans for a user-friendly interface.

ASSUMPTIONS:

This database design models the flow of a library management system. Here's a detailed breakdown of each table and its role in the management system:

1. **Book**

This table stores information about the books available in the library.

Attributes:

- **BookID (ISBN):** The International Standard Book Number, a unique identifier for the book.
- **Title:** The title of the book.
- **Author:** The author(s) of the book.
- **Publisher:** The publishing company of the book.
- **Genre:** The genre or category of the book (e.g., Fiction, Non-fiction, Science).
- **Pub_yr:** The year the book was published.
- **Language:** The language in which the book is written.

- **Status:** The current status of the book (e.g., Available, Checked Out, Reserved).
- **Type:** The type of book (e.g., Hardcover, Paperback, eBook).

2. Member Info

This table stores information about the members of the library.

Attributes:

- **Member ID:** A unique identifier for each member.
- **Name:** The name of the member.
- **Address:** The address of the member.
- **Phone no:** The contact phone number of the member.
- **E mail:** The email address of the member.
- **Mem type:** The type of membership (e.g., Regular, Premium).
- **Join Date:** The date when the member joined the library.
- **Exp Date:** The expiration date of the membership.
- **Fines:** The total amount of fines the member has accumulated.
- **Rolename:** The role or title of the member if applicable (e.g., Student, Teacher).
- **Salary:** The salary of the member if applicable.
- **Role id:** The identifier for the role if applicable.

3. Worker Info

This table stores information about the workers in the library.

Attributes:

- **Worker ID:** A unique identifier for each worker.
- **Name:** The name of the worker.
- **Role:** The role or position of the worker (e.g., Librarian, Assistant).
- **Phno:** The contact phone number of the worker.
- **Email ID:** The email address of the worker.
- **Hire date:** The date when the worker was hired.

4. Payment

This table stores information about the payment transactions made by members.

Attributes:

- **TransID:** A unique identifier for each transaction.
- **MemID:** The ID of the member making the payment.
- **Fine amt:** The amount of fine being paid.
- **Method:** The method of payment (e.g., Credit Card, Cash).
- **Status:** The status of the transaction (e.g., Successful, Failed).
- **Transaction reference:** A reference number for the transaction, not present if the transaction fails.
- **Date:** The date when the transaction was made.

5. Order

This table stores information about book orders made by members.

Attributes:

- **OrderID:** A unique identifier for each order.
- **MemID:** The ID of the member who made the order.
- **Checkout date:** The date when the order was checked out.
- **Due date:** The date when the order is due to be returned.
- **Return date:** The date when the order was returned.
- **Fine amt:** The amount of fine applicable if the order is returned late.

6. Inventory

This table stores information about the inventory of books in different branches of the library.

Attributes:

- **BookID:** The ID of the book.
- **BranchID:** The ID of the branch.
- **Total copies:** The total number of copies of the book available in the branch.

- **Available copies:** The number of copies currently available for lending.
- **Lending type:** The type of lending available (e.g., Short-term, Long-term).

7. Branch:

A table containing information about the branches.

Attributes:

- **BranchID:** The ID of the branch.
- **Name:** The name of the branch.
- **Location:** The location of the branch.
- **Contact_no:** The contact number of the branch.

8. Delivery

This table stores information about the delivery of ordered books.

Attributes:

- **Delid:** A unique identifier for each delivery.
- **Orderid:** The ID of the order being delivered.
- **Del_address:** The address to which the order is being delivered.
- **Del_status:** The status of the delivery (e.g., Dispatched, Delivered).
- **Dispatch_date:** The date when the order was dispatched.
- **Deli_personid:** The ID of the delivery person.
- **Deli_person name:** The name of the delivery person.
- **Deli_person phone_no:** The contact phone number of the delivery person.

9. Reservation

This table stores information about reservations made by members for books.

Attributes:

- **Resid:** A unique identifier for each reservation.
- **Bookid:** The ID of the book being reserved.
- **Memberid:** The ID of the member making the reservation.
- **Res_date:** The date when the reservation was made.

- **Status:** The status of the reservation (e.g., Pending, Completed).

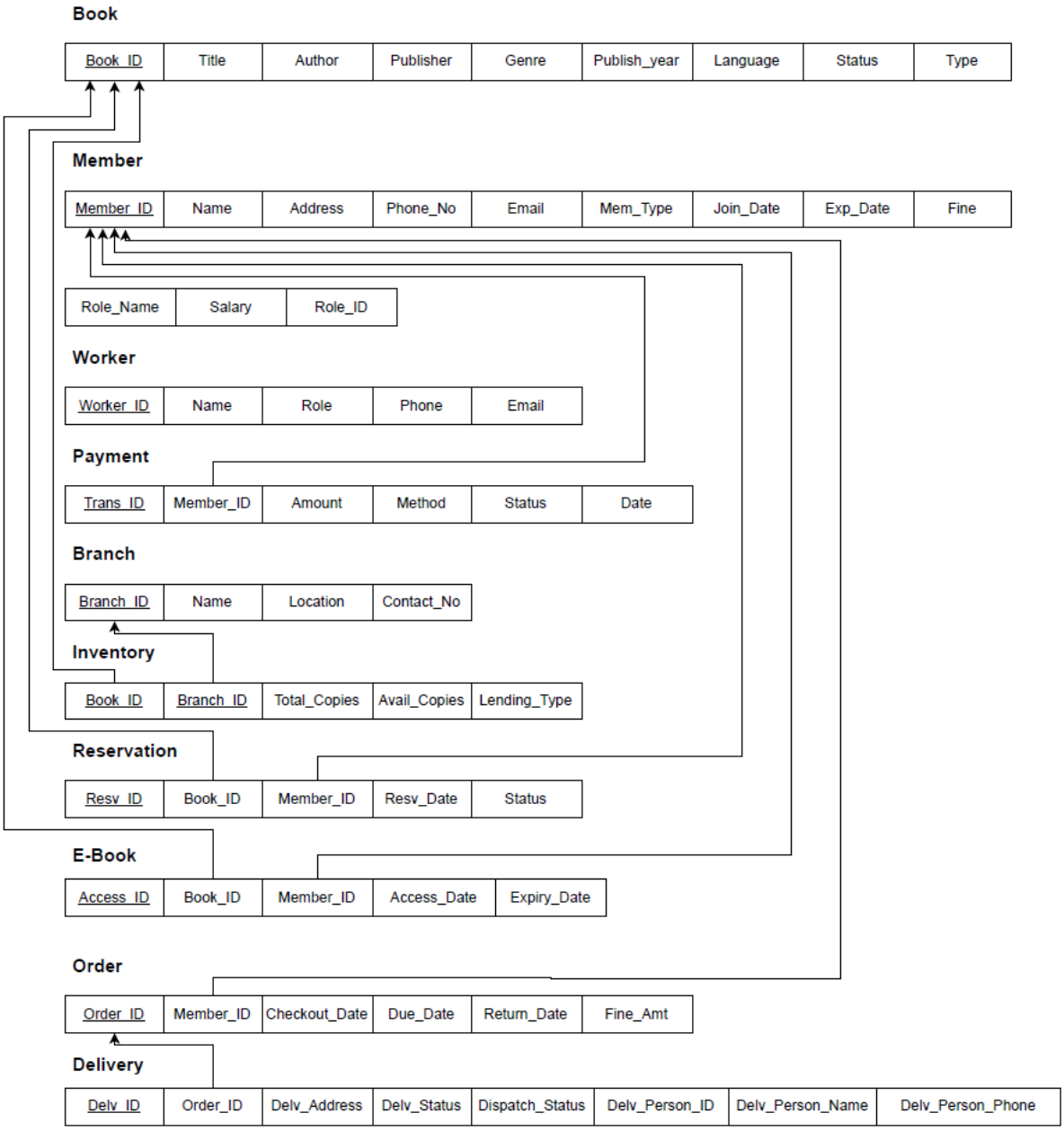
10. E-Book Access

This table stores information about the access of e-books by members.

Attributes:

- **Accessid:** A unique identifier for each e-book access.
- **Bookid:** The ID of the e-book.
- **Memid:** The ID of the member accessing the e-book.
- **Access_date:** The date when the e-book was accessed.
- **Expiry:** The expiration date of the e-book access.

SCHEMA DIAGRAM:



ER DIAGRAM


Email_id
Hire_date
Workerinfo
Phone_no
Role
Workerid
Name
IsTaken_by
Transaction_reference
Trans_id
Membership_id
Payment
Fine_amount
Status
Date
Method
Makes
Fines
Expiry_date
Name
Member_info
Address
Member_id
joindate
Email
Reserved
Phoneno
Delivery_person_Phonenumber
delivery_id
delivery_address
Order_id
Fine_amount
Membership_id
Order
Places
delivery_person_name
Has
Checkout_date
delivery_date
Delivery
Return_date
Due_date
delivery_status
delivery_personid
dispatch_state
Has
Access_id
access_date
has_access_to
Ebook_access
expiry
is_accessed
Bookid
Genre
Title
Author
Reservation
reservation_id
status
Reservation_date
is_reserved
Publisher_yr
Book
Language
Status
Type
Publisher
Is_a_part_of
Total_Copies
Inventory
Lending_type
Available_copies
Holds
Branch_id
Branches
Name
Contact_no
Location

ENTITIES AND ATTRIBUTES:

I. Book

A: BookID (ISBN)

B: Title

C: Author

D: Publisher

E: Genre

F: Pub_yr

G: Language

H: Status

I: Type

Identified FDs:

Set of FDs: {
    A → BCDEFGHI,
    BC → DE
}

Decomposition:

Decomposed set of FDs: {
    A → B,
    A → C,
    A → D,
    A → E,
    A → F,
    A → G,

A → H,
A → I,
BC → D,
BC → E
}

Prime Attributes: {A}

Finding irreducible set of FDs:

Step 1: Decomposition

A -> B
A -> C
A -> D
A -> E
A -> F
A -> G
A -> H
A -> I
BC -> D
BC -> E

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side, no attributes can be removed.

Step 3: Check for Redundant FDs

1. **Check A -> B**:
    - Compute closure of {A} without A -> B: {A}+ = {A, C, D, E, F, G, H, I}
        - B cannot be derived, so A -> B is not redundant.
2. **Check A -> C**:
    - Compute closure of {A} without A -> C: {A}+ = {A, B, D, E, F, G, H, I}
        - C cannot be derived, so A -> C is not redundant.
3. **Check A -> D**:
    - Compute closure of {A} without A -> D: {A}+ = {A, B, C, E, F, G, H, I}

        - D cannot be derived, so A -> D is not redundant.
4. **Check A -> E**:
    - Compute closure of {A} without A -> E: {A}+ = {A, B, C, D, F, G, H, I}
        - E cannot be derived, so A -> E is not redundant.
5. **Check A -> F**:
    - Compute closure of {A} without A -> F: {A}+ = {A, B, C, D, E, G, H, I}
        - F cannot be derived, so A -> F is not redundant.
6. **Check A -> G**:
    - Compute closure of {A} without A -> G: {A}+ = {A, B, C, D, E, F, H, I}
        - G cannot be derived, so A -> G is not redundant.
7. **Check A -> H**:
    - Compute closure of {A} without A -> H: {A}+ = {A, B, C, D, E, F, G, I}
        - H cannot be derived, so A -> H is not redundant.
8. **Check A -> I**:
    - Compute closure of {A} without A -> I: {A}+ = {A, B, C, D, E, F, G, H}
        - I cannot be derived, so A -> I is not redundant.
9. **Check BC -> D**:
    - Compute closure of {BC} without BC -> D: {BC}+ = {B, C, D, E}
        - D can be derived, so BC -> D is redundant.
10. **Check BC -> E**:
    - Compute closure of {BC} without BC -> E: {BC}+ = {B, C, D, E}
        - E can be derived, so BC -> E is redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the irreducible set of functional dependencies is:

A -> BCDEFGHI
BC -> DE

II. Member Info

A: Member ID

B: Name

C: Address

D: Phone no

E: E mail

F: Mem type

G: Join Date

H: Exp Date

I: Fines

J: Rolename

K: Salary

L: Role id

Identified FDs:

Set of FDs: {
A → BCDEFGHIJKL,
G → H,
L → JK,
E → BCDFGHIJKL
}

Finding irreducible set of FDs:

Step 1: Decomposition

A -> B

A -> C

A -> D

A -> E

A -> F

A -> G

A -> H

A -> I

A -> J

A -> K

A -> L

G -> H

L -> J

L -> K

E -> B

E -> C

E -> D

E -> F

E -> G

E -> H

E -> I

E -> J

E -> K

E -> L

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side, no attributes can be removed.

Step 3: Check for Redundant FDs

1. **Check A -> B**:
    - Compute closure of {A} without A -> B: {A}+ = {A, C, D, E, F, G, H, I, J, K, L}
        - B cannot be derived, so A -> B is not redundant.
2. **Check A -> C**:
    - Compute closure of {A} without A -> C: {A}+ = {A, B, D, E, F, G, H, I, J, K, L}
        - C cannot be derived, so A -> C is not redundant.
3. **Check A -> D**:
    - Compute closure of {A} without A -> D: {A}+ = {A, B, C, E, F, G, H, I, J, K, L}
        - D cannot be derived, so A -> D is not redundant.
4. **Check A -> E**:
    - Compute closure of {A} without A -> E: {A}+ = {A, B, C, D, F, G, H, I, J, K, L}
        - E cannot be derived, so A -> E is not redundant.
5. **Check G -> H**:
    - Compute closure of {G} without G -> H: {G}+ = {G}
        - H cannot be derived, so G -> H is not redundant.
6. **Check L -> J**:
    - Compute closure of {L} without L -> J: {L}+ = {L, K}
        - J cannot be derived, so L -> J is not redundant.
7. **Check L -> K**:
    - Compute closure of {L} without L -> K: {L}+ = {L, J}
        - K cannot be derived, so L -> K is not redundant.
8. **Check E -> B**:
    - Compute closure of {E} without E -> B: {E}+ = {E, C, D, F, G, H, I, J, K, L}

        - B cannot be derived, so E -> B is not redundant.
9. **Check E -> C**:
    - Compute closure of {E} without E -> C: {E}+ = {E, B, D, F, G, H, I, J, K, L}
        - C cannot be derived, so E -> C is not redundant.
10. **Check E -> D**:
    - Compute closure of {E} without E -> D: {E}+ = {E, B, C, F, G, H, I, J, K, L}
        - D cannot be derived, so E -> D is not redundant.
11. **Check E -> F**:
    - Compute closure of {E} without E -> F: {E}+ = {E, B, C, D, G, H, I, J, K, L}
        - F cannot be derived, so E -> F is not redundant.
12. **Check E -> G**:
    - Compute closure of {E} without E -> G: {E}+ = {E, B, C, D, F, H, I, J, K, L}
        - G cannot be derived, so E -> G is not redundant.
13. **Check E -> H**:
    - Compute closure of {E} without E -> H: {E}+ = {E, B, C, D, F, G, I, J, K, L}
        - H cannot be derived, so E -> H is not redundant.
14. **Check E -> I**:
    - Compute closure of {E} without E -> I: {E}+ = {E, B, C, D, F, G, H, J, K, L}
        - I cannot be derived, so E -> I is not redundant.
15. **Check E -> J**:
    - Compute closure of {E} without E -> J: {E}+ = {E, B, C, D, F, G, H, I, K, L}
        - J cannot be derived, so E -> J is not redundant.
16. **Check E -> K**:
    - Compute closure of {E} without E -> K: {E}+ = {E, B, C, D, F, G, H, I, J, L}
        - K cannot be derived, so E -> K is not redundant.
17. **Check E -> L**:

- Compute closure of {E} without E -> L: {E}+ = {E, B, C, D, F, G, H, I, J, K}
  - L cannot be derived, so E -> L is not redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the irreducible set of functional dependencies is:

{A -> E, G -> H,L -> JK,E -> BCDFGHIL}

III. Worker Info

A: Worker ID

B: Name

C: Role

D: Phno

E: Email ID

F: Hire date

Identified FDs:

Set of FDs: {
A → BCDE,
E → BCD
}

Finding irreducible set of FDs:

Step 1: Decomposition

A -> B
A -> C
A -> D
A -> E

E -> B
E -> C
E -> D

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side, no attributes can be removed.

Step 3: Check for Redundant FDs

1. **Check A -> B**:
    - Compute closure of {A} without A -> B: {A}+ = {A, C, D, E}
        - B cannot be derived, so A -> B is not redundant.
2. **Check A -> C**:
    - Compute closure of {A} without A -> C: {A}+ = {A, B, D, E}
        - C cannot be derived, so A -> C is not redundant.
3. **Check A -> D**:
    - Compute closure of {A} without A -> D: {A}+ = {A, B, C, E}
        - D cannot be derived, so A -> D is not redundant.
4. **Check A -> E**:
    - Compute closure of {A} without A -> E: {A}+ = {A, B, C, D}
        - E cannot be derived, so A -> E is not redundant.
5. **Check E -> B**:
    - Compute closure of {E} without E -> B: {E}+ = {E, C, D}
        - B cannot be derived, so E -> B is not redundant.
6. **Check E -> C**:
    - Compute closure of {E} without E -> C: {E}+ = {E, B, D}
        - C cannot be derived, so E -> C is not redundant.
7. **Check E -> D**:
    - Compute closure of {E} without E -> D: {E}+ = {E, B, C}
        - D cannot be derived, so E -> D is not redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the irreducible set of functional dependencies is:

{A -> E, E -> BCD}

IV. Payment

A: TransID

B: MemID

C: Fine amt

D: Method

E: Status

F: Date

Identified FDs:
{
A → BCDEF
}

Since A → BCDEF has all attributes on the right-hand side and no attributes can be removed without changing the closure of A, this FD is already minimal.

Therefore, the minimal set of functional dependencies is simply:

Minimal Set of FDs:
{
A → BCDEF
}

V. Order

A: OrderID

B: MemID

C: Checkout date

D: Due date

E: Return date

F: Fine amt

G: Bookid

Identified FDs:

{
AG → BCDEF,
A → BCDEF,
C → D
}

Step 1: Decomposition

AG → B
AG → C
AG → D
AG → E
AG → F
A → B
A → C
A → D
A → E
A → F
C → D

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side and each right-hand side has only one attribute due to the decomposition, no attributes can be removed in this step.

Step 3: Check for Redundant FDs

Now, let's check each FD to see if any can be removed without affecting the closure of attributes.

1. **Check AG → B**:
    - Compute closure of {AG} without AG → B: {AG}+ = {A, G, C, D, E, F}
        - B cannot be derived, so AG → B is not redundant.
2. **Check AG → C**:
    - Compute closure of {AG} without AG → C: {AG}+ = {A, G, B, D, E, F}
        - C cannot be derived, so AG → C is not redundant.
3. **Check AG → D**:
    - Compute closure of {AG} without AG → D: {AG}+ = {A, G, B, C, E, F}
        - D cannot be derived, so AG → D is not redundant.
4. **Check AG → E**:
    - Compute closure of {AG} without AG → E: {AG}+ = {A, G, B, C, D, F}
        - E cannot be derived, so AG → E is not redundant.
5. **Check AG → F**:
    - Compute closure of {AG} without AG → F: {AG}+ = {A, G, B, C, D, E}
        - F cannot be derived, so AG → F is not redundant.
6. **Check A → B**:
    - Compute closure of {A} without A → B: {A}+ = {A, C, D, E, F}
        - B can be derived from A → B, so A → B is redundant.
7. **Check A → C**:
    - Compute closure of {A} without A → C: {A}+ = {A, B, D, E, F}
        - C cannot be derived, so A → C is not redundant.
8. **Check A → D**:
    - Compute closure of {A} without A → D: {A}+ = {A, B, C, E, F}
        - D cannot be derived, so A → D is not redundant.
9. **Check A → E**:
    - Compute closure of {A} without A → E: {A}+ = {A, B, C, D, F}
        - E cannot be derived, so A → E is not redundant.
10. **Check A → F**:
    - Compute closure of {A} without A → F: {A}+ = {A, B, C, D, E}
        - F cannot be derived, so A → F is not redundant.
11. **Check C → D**:
    - Compute closure of {C} without C → D: {C}+ = {C}
        - D cannot be derived, so C → D is not redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the minimal set of functional dependencies is:

Minimal Set of FDs:

{

AG → BCDEF,

A → BCDEF,

C → D

}

VI. Inventory

A: BookID

B: BranchID

C: Total copies

D: Available copies

E: Lending type

Identified FDs:

{

AB → CDE,

A → E

}

Finding minimal set of FDS:

Step 1: Decomposition

AB → C

AB → D

AB → E

A → E

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side and each right-hand side has only one attribute due to the decomposition, no attributes can be removed in this step.

Step 3: Check for Redundant FDs

Now, let's check each FD to see if any can be removed without affecting the closure of attributes.

1. Check AB → C:
    - Compute closure of {AB} without AB → C: {AB}+ = {A, B, D, E}
        - C cannot be derived, so AB → C is not redundant.
2. Check AB → D:
    - Compute closure of {AB} without AB → D: {AB}+ = {A, B, C, E}
        - D cannot be derived, so AB → D is not redundant.
3. Check AB → E:
    - Compute closure of {AB} without AB → E: {AB}+ = {A, B, C, D}
        - E cannot be derived, so AB → E is not redundant.
4. Check A → E:
    - Compute closure of {A} without A → E: {A}+ = {A}
        - E can be derived from A → E, so A → E is redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the minimal set of functional dependencies is:

AB → C
AB → D
AB → E

Minimal Set of FDs:

{

AB → CDE,
A → E

}

VII. Branch

A: BranchID

B: Name

C: Location

D: Contact_no

Identified FDs:
{
A → BCD
}

Since A→BCD has all attributes on the right-hand side and no attributes can be removed without changing the closure of A, this FD is already minimal.

Therefore, the minimal set of functional dependencies is simply:

Minimal Set of FDs:
{
A → BCD
}

VIII. Delivery

A: Delid

B: Orderid

C: Del_address

D: Del_status

E: Dispatch_date

F: Deli_personid

G: Deli_person name

H: Deli_person phone_no

Identified FDs:

{

A → BCDEFGH

F → GH

}

Finding minimal set of FDs:

Step 1: Decomposition

A → B

A → C

A → D

A → E

A → F

A → G

A → H

F → G

F → H

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side and each right-hand side has only one attribute due to the decomposition, no attributes can be removed in this step.

Step 3: Check for Redundant FDs

Now, let's check each FD to see if any can be removed without affecting the closure of attributes.

1. Check A → B:
    - Compute closure of {A} without A → B: {A}+ = {A, C, D, E, F, G, H}
        - B can be derived from A → B, so A → B is not redundant.

2. Check A → C:
    - Compute closure of {A} without A → C: {A}+ = {A, B, D, E, F, G, H}
        - C can be derived from A → C, so A → C is not redundant.
3. Check A → D:
    - Compute closure of {A} without A → D: {A}+ = {A, B, C, E, F, G, H}
        - D can be derived from A → D, so A → D is not redundant.
4. Check A → E:
    - Compute closure of {A} without A → E: {A}+ = {A, B, C, D, F, G, H}
        - E can be derived from A → E, so A → E is not redundant.
5. Check A → F:
    - Compute closure of {A} without A → F: {A}+ = {A, B, C, D, E, G, H}
        - F can be derived from A → F, so A → F is not redundant.
6. Check A → G:
    - Compute closure of {A} without A → G: {A}+ = {A, B, C, D, E, F, H}
        - G can be derived from A → G, so A → G is redundant.
7. Check A → H:
    - Compute closure of {A} without A → H: {A}+ = {A, B, C, D, E, F, G}
        - H can be derived from A → H, so A → H is redundant.
8. Check F → G:
    - Compute closure of {F} without F → G: {F}+ = {F, H}
        - G can be derived from F → G, so F → G is not redundant.
9. Check F → H:
    - Compute closure of {F} without F → H: {F}+ = {F, G}
        - H can be derived from F → H, so F → H is not redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the minimal set of functional dependencies is:

A → B
A → C
A → D
A → E
A → F
F → G
F → H

Minimal Set of FDs:

{

A → BCDEF

F →GH

}

IX. Reservation

A: Resid

B: Bookid

C: Memberid

D: Res_date

E: Status

Identified FDs:

{

A → BCDE

}

Finding minimal set of FDs

Since A→BCDE has all attributes on the right-hand side and no attributes can be removed without changing the closure of A, this FD is already minimal.

Therefore, the minimal set of functional dependencies is simply:

Minimal Set of FDs:

{

A → BCDE

}

X. E-book Access

A: Accessid

B: Bookid

C: Memid

D: Access_date

E: Expiry

Identified FDs:

{

A → BCDE

D → E

}

Step 1: Decomposition

A → B

A → C

A → D

A → E

D → E

Step 2: Remove Extraneous Attributes

Since all FDs have only one attribute on the left-hand side and each right-hand side has only one attribute due to the decomposition, no attributes can be removed in this step.

Step 3: Check for Redundant FDs

Now, let's check each FD to see if any can be removed without affecting the closure of attributes.

1. Check A → B:
    - Compute closure of {A} without A → B: {A}+ = {A, C, D, E}
        - B cannot be derived, so A → B is not redundant.
2. Check A → C:
    - Compute closure of {A} without A → C: {A}+ = {A, B, D, E}
        - C cannot be derived, so A → C is not redundant.

3. Check A → D:
    - Compute closure of {A} without A → D: {A}+ = {A, B, C, E}
        - D cannot be derived, so A → D is not redundant.
4. Check A → E:
    - Compute closure of {A} without A → E: {A}+ = {A, B, C, D,E}
        - E cannot be derived, so A → E is redundant.
5. Check D → E:
    - Compute closure of {D} without D → E: {D}+ = {D}
        - E can be derived from D → E, so D → E is not redundant.

Resulting Minimal Set of FDs:

After removing the redundant FDs, the minimal set of functional dependencies is:

A → B
A → C
A → D
D → E

Minimal Set of FDs:

{
A → BCD
D → E
}

PHASE – 2: NORMALIZING THE RELATIONS INTO 3NF OR BCNF

1. **Book**

Step-by-Step Normalization:

Analysis:

1. **First Normal Form (1NF):**
    - The given table is assumed to be in 1NF as no multi-valued attributes are provided.
2. **Second Normal Form (2NF):**
    - The relation is already in 1NF.
    - Candidate Key: A (since A→BCDEFGHIA)
    - Non-prime attributes: B, C, D, E, F, G, H, I
    - **Check for partial dependencies:**
        - A→BCDEFGHI : Full functional dependency (no partial dependency on part of a composite key as A is not composite).
        - BC→DEBC : This is a partial dependency if we consider BC as a subset of A (if A were composite, but A is a single attribute).
3. Since A→BCDEFGHIA: holds and covers all attributes, there are no partial dependencies, making the relation in 2NF directly.
4. **Third Normal Form (3NF):**
    - The relation is in 2NF.
    - To be in 3NF, we must ensure there are no transitive dependencies.
5. **Check dependencies:**
    - A→BCDEFGHIA: A is a candidate key, so all attributes are directly dependent on A.
    - BC→DEBC: Here, BC is not a candidate key for the entire relation, indicating a transitive dependency when considering A as a superkey (i.e., A determines BC, which determines DE).
6. To resolve this transitive dependency, we must decompose the relation.

Decomposition to Resolve Transitive Dependency:

1. Decompose based on the transitive dependency BC→DEBC:
    - Table 1: A→BCFGHI with attributes {A, B, C, F, G, H, I}

    - Table 2: BC→DE with attributes {B, C, D, E}

Checking Normal Forms After Decomposition:

Table 1: {A, B, C, F, G, H, I}

- FD: A→BCFGHIA \rightarrow BCFGHIA→BCFGHI
- Candidate Key: A
- In 2NF (as there are no partial dependencies).
- In 3NF (as there are no transitive dependencies).
- Also in BCNF (since A is a candidate key and the only determinant).

Table 2: {B, C, D, E}

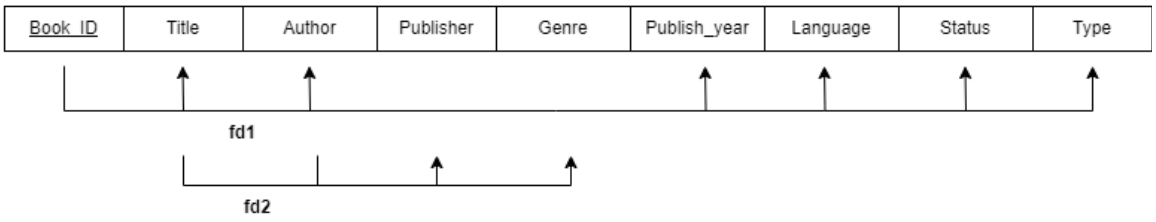
- FD: BC→DE
- Candidate Key: BC
- In 2NF (as all non-prime attributes are fully functionally dependent on the entire primary key BC).
- In 3NF (as there are no transitive dependencies within this table).
- Also in BCNF (since BC is a candidate key and the only determinant).

Final Normalized Tables:

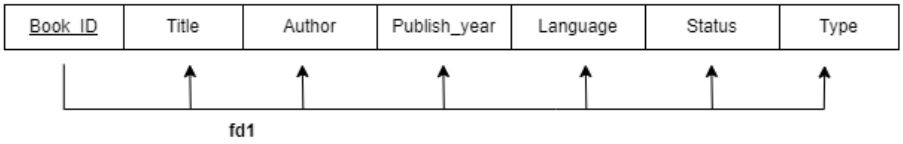
After normalization, the tables are:

1. Table 1:
    - Attributes: {A, B, C, F, G, H, I}
    - Functional Dependency: A→BCFGHIA
2. Table 2:
    - Attributes: {B, C, D, E}
    - Functional Dependency: BC→DEBC

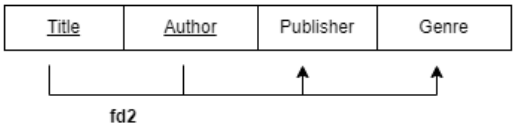
Book



Book Details



Book Category



2. **Member**

Analysis and Normalization:

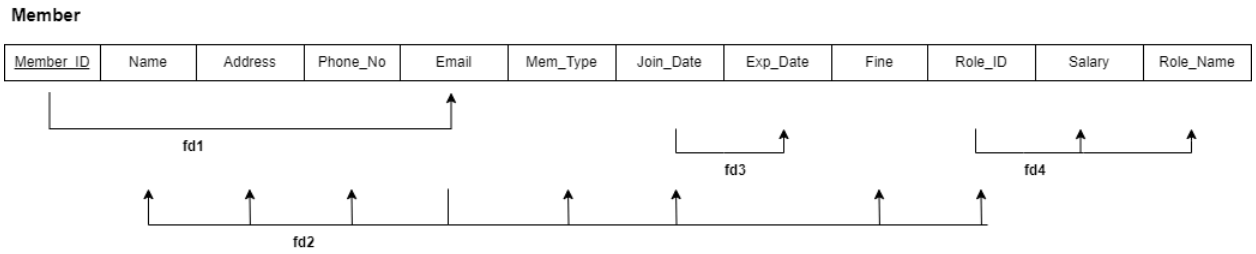
1. **First Normal Form (1NF):**
    - To be in 1NF, each attribute must hold atomic (indivisible) values, and there should be no repeating groups or arrays.
    - The given FDs do not affect 1NF directly since they describe relationships rather than the form of data storage.
2. **Second Normal Form (2NF):**
    - A relation is in 2NF if it is in 1NF and every non-prime attribute is fully functionally dependent on the whole of every candidate key.
    - Candidate keys need to be identified:
        - From E -> BCDFGHIL, we deduce that E is a candidate key because it determines all other attributes.
3. **Normalization to 2NF:**
    - Decompose the relation based on partial dependencies:
        - Table 1: E -> BCDFGHIL with attributes {E, B, C, D, F, G, H, I, L}
        - Table 2: A -> E with attributes {A, E}

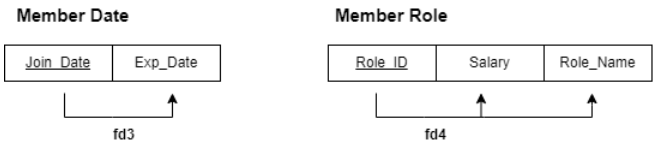
        - Table 3: G -> H with attributes {G, H}
        - Table 4: L -> JK with attributes {L, J, K}
4. Each table now has attributes that are fully dependent on the respective candidate keys (E, A, G, L).
5. **Third Normal Form (3NF):**
    - A relation is in 3NF if it is in 2NF and no transitive dependencies exist.
    - Check for transitive dependencies:
        - E -> BCDFGHIL is a transitive dependency if we consider the closure of E.
        - E -> B, E -> C, etc., which are dependencies on parts of E.
        - Normalize further to remove transitive dependencies:
            - Table 1: E -> BCDFGHIL with attributes {E, B, C, D, F, G, H, I, L}
            - Table 2: A -> E with attributes {A, E}
            - Table 3: G -> H with attributes {G, H}
            - Table 4: L -> JK with attributes {L, J, K}
6. Each table is now in 3NF since there are no non-key attributes that are transitively dependent on the primary key.
7. **Boyce-Codd Normal Form (BCNF):**
    - A relation is in BCNF if for every functional dependency X -> Y, X must be a superkey.
    - Check for BCNF:
        - Table 1: E -> BCDFGHIL, where E is the candidate key.
        - Table 2: A -> E, where A is not a superkey by itself (only E is).
        - Table 3: G -> H, where G is not a superkey.
        - Table 4: L -> JK, where L is not a superkey.
8. **Normalize to BCNF:**
    - Ensure every dependency has a superkey on the left-hand side:
        - Combine tables where possible to ensure each table has a superkey as the determinant.

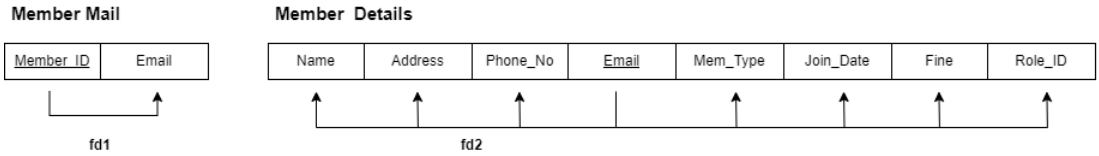
Final Normalized Tables:

Based on the given FDs and normalization steps, the final tables in BCNF would typically be:

1. **Table 1:**
    - Attributes: {E, B, C, D, F, G, H, I, L}
    - Functional Dependency: E -> BCDFGHIL (E is the candidate key)
2. **Table 2:**
    - Attributes: {A, E}
    - Functional Dependency: A -> E (E is the candidate key)
3. **Table 3:**
    - Attributes: {G, H}
    - Functional Dependency: G -> H (G is the candidate key)
4. **Table 4:**
    - Attributes: {L, J, K}
    - Functional Dependency: L -> JK (L is the candidate key)







3. **Worker**

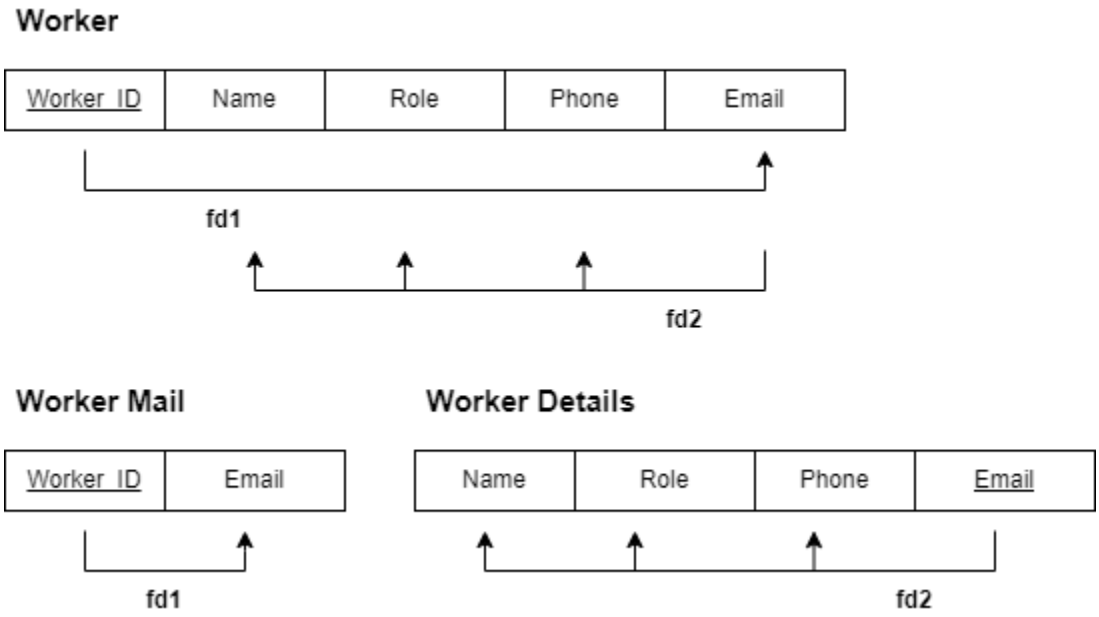
Analysis and Normalisation:

Let's analyze the normalization condition for each normal form:

1. **1NF (First Normal Form):**
    - A table is in 1NF if it doesn't have repeating groups or arrays as columns.
    - Since we only have attributes on the right-hand side of the functional dependencies, there's no violation of 1NF here.
2. **2NF (Second Normal Form):**
    - A table is in 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key.
    - To check for 2NF:
        - The candidate keys are A and E
        - A→E E is fully dependent on A, satisfying 2NF.
        - E→BCD,B,C, and D are fully dependent on EEE, satisfying 2NF.
3. **3NF (Third Normal Form):**
    - A table is in 3NF if it is in 2NF and no transitive dependencies exist where non-key attributes depend on other non-key attributes.
    - Analyzing for 3NF:
        - E→BCDE introduces a transitive dependency where B,C,B, C,B,C, and DDD are dependent on EEE, which is not a candidate key.
        - This violates 3NF because B,C, and D are not directly dependent on the candidate keys (A or E) but on a non-key attribute (EEE).
4. **Normalization and BCNF (Boyce-Codd Normal Form):**
    - To normalize to 3NF, we decompose the table into two tables:
        - Table 1: {A→E}
        - Table 2: {E→BCD}
    - Both resulting tables are in 3NF because they satisfy the conditions for 2NF and 3NF independently.
    - To check BCNF:
        - BCNF states that every functional dependency X→YX , X must be a superkey.
        - In the original set, E→BCDE violates BCNF because E is not a candidate key (it's not minimal in terms of determining all attributes on the right side).
        - After decomposition, each table satisfies BCNF because each functional dependency's left-hand side is a candidate key.

Final table:

- Table 1: {A→E}
- Table 2: {E→BCD}



4. **Payment**

It satisfies all normal forms:

1. First Normal Form (1NF): A relation is in 1NF if all attributes contain atomic (indivisible) values, meaning no attribute can have a set or list of values.

The functional dependency {a} -> {bcdef} indicates that for every value of a, there is a unique set of values {bcdef}. This implies that the attribute bcdef is functionally dependent on a, and therefore for each a, bcdef is a single value, satisfying the requirement of 1NF.

2. Second Normal Form (2NF): A relation is in 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on the whole primary key.

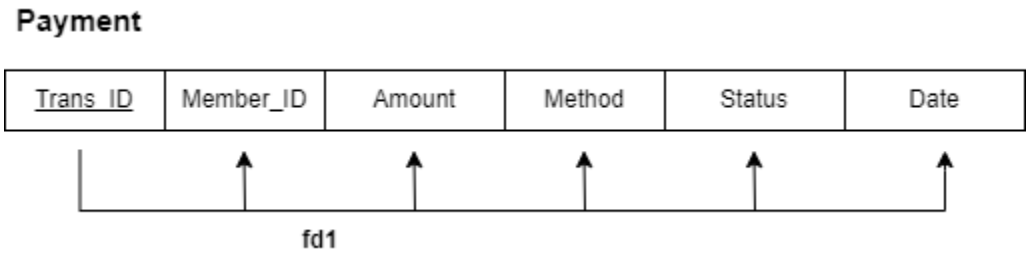
In our case, {a} is the primary key since {a} -> {bcdef} holds. bcdef is fully functionally dependent on a (since {bcdef} is determined by {a}), satisfying 2NF.

3. Third Normal Form (3NF): A relation is in 3NF if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.

Since there is only one non-key attribute set {bcdef}, and it is directly determined by the primary key {a}, there are no transitive dependencies involving non-key attributes. Therefore, the relation satisfies 3NF.

4. Boyce-Codd Normal Form (BCNF): A relation is in BCNF if for every functional dependency X -> Y, where X is a superkey, X -> Y is a trivial dependency (where Y is a subset of X).

In our case, {a} is the only candidate key (and also the primary key), and {a} -> {bcdef} is not a trivial dependency because {bcdef} is not a subset of {a}. However, since {a} is the superkey and {a} -> {bcdef} holds, the relation satisfies BCNF.



5. **Order**

Step-by-Step Normalization:

Analysis:

Given the relation for the table Order:

Attributes: {A, B, C, D, E, F, G}

Functional Dependencies (FDs): { AG → BCDEF, A → BCDEF, C → D }

First Normal Form (1NF):

To ensure 1NF, we need to ensure all attributes are atomic.

The relation is assumed to be in 1NF as all attributes are atomic.

Second Normal Form (2NF):

Check for partial dependencies:

AG → BCDEF: Full functional dependency (no partial dependency).

A → BCDEF: Full functional dependency (no partial dependency).

C → D: Full functional dependency (no partial dependency).

To achieve 2NF:

Decompose the relation to remove partial dependencies:

Table 1: AG with attributes {A, G} Table 2: CD with attributes {C, D} Table 3: ABCEF with attributes {A, B, C, E, F}

Third Normal Form (3NF):

Check for transitive dependencies:

For Table 1 (AG): No transitive dependencies as AG is a candidate key.

For Table 2 (CD): No transitive dependencies as C is a candidate key.

For Table 3 (ABCEF): No transitive dependencies as A is a candidate key.

The relation is in 3NF after decomposition.

Boyce-Codd Normal Form (BCNF):

Check for candidate keys and ensure all FDs are satisfied:

Candidate Keys: AG (since AG → BCDEF) A (since A → BCDEF) C (since C → D)

All FDs are already covered by the candidate keys: AG → BCDEF A → BCDEF C → D

The relation satisfies BCNF.

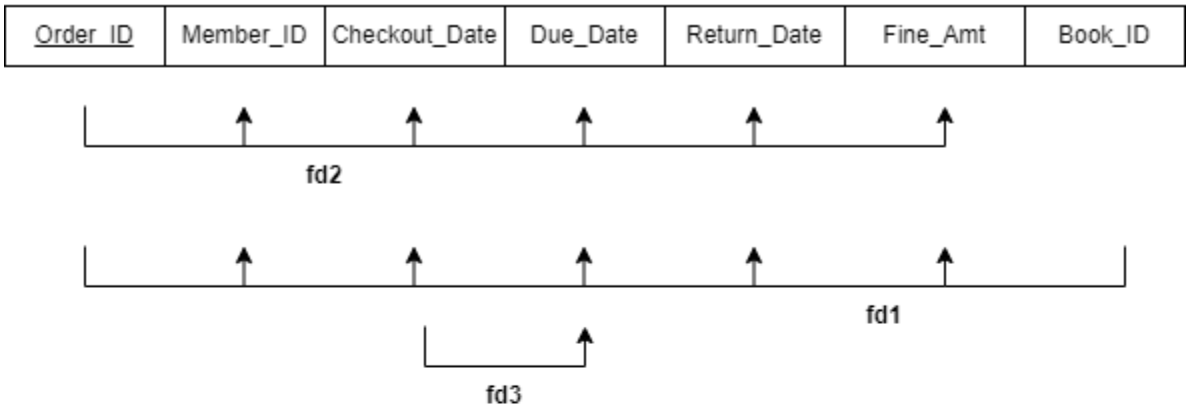
Final Normalized Tables:

After normalization, the tables are: Table 1: Attributes: {A, G} Functional Dependency: AG → BCDEF

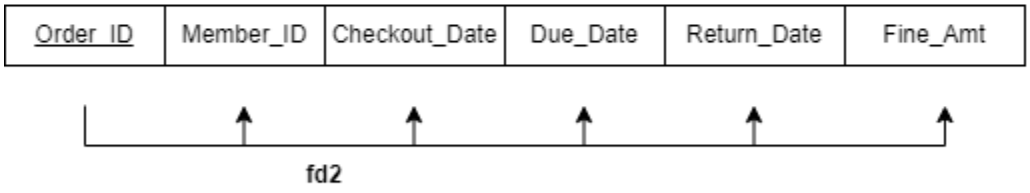
Table 2: Attributes: {C, D} Functional Dependency: C → D

Table 3: Attributes: {A, B, C, E, F} Functional Dependency: A → BCDEF

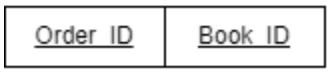
Order



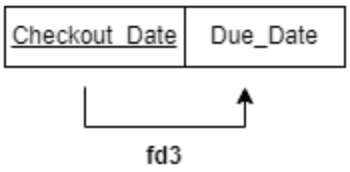
Order Details



Order Book



Order Date



6. **Inventory**

Step-by-Step Normalization:

Analysis:

Given the relation for the table Inventory:

Attributes: {A, B, C, D, E}

Functional Dependencies (FDs): { AB → CDE, A → E }

First Normal Form (1NF):

To ensure 1NF, we need to ensure all attributes are atomic.

The relation is assumed to be in 1NF as all attributes are atomic.

Second Normal Form (2NF):

Check for partial dependencies:

AB → CDE: Full functional dependency (no partial dependency).

A → E: Partial dependency because A determines E, and A is not a superkey.

To achieve 2NF:

Decompose the relation to remove partial dependencies:

Table 1: AB → CD with attributes {A, B, C, D} Table 2: A → E with attributes {A, E}

Third Normal Form (3NF):

Check for transitive dependencies:

For Table 1 (AB → CD): No transitive dependencies as AB is a candidate key.

For Table 2 (A → E): No transitive dependencies as A is a candidate key.

The relation is in 3NF after decomposition.

Boyce-Codd Normal Form (BCNF):

Check for candidate keys and ensure all FDs are satisfied:

Candidate Key: AB (since AB → CD)

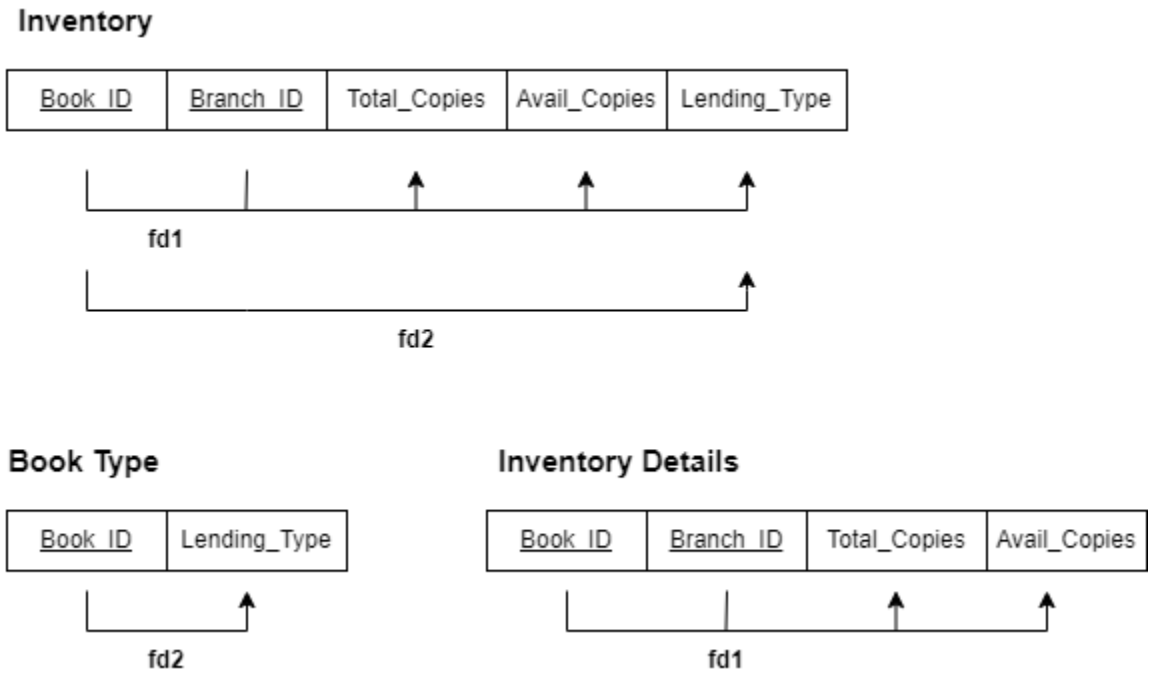
All FDs are already covered by the candidate key: AB → CD A → E

The relation satisfies BCNF.

Final Normalized Tables:

After normalization, the tables are: Table 1: Attributes: {A, B, C, D} Functional Dependency: AB → CD

Table 2: Attributes: {A, E} Functional Dependency: A → E



7. **Branch**

Step-by-Step Normalization:

Analysis:

Given the relation for the table Branch:

Attributes: {A, B, C, D}

Functional Dependencies (FDs): { A → BCD }

First Normal Form (1NF):

To ensure 1NF, we need to ensure all attributes are atomic.

The relation is not in 1NF because D is multivalued.

Decompose the relation into 1NF:

Table 1: AD with attributes {A, D} Table 2: ABC with attributes {A, B, C}

Second Normal Form (2NF):

Check for partial dependencies:

For Table 1 (AD): A → D: Full functional dependency (no partial dependency).

For Table 2 (ABC): No partial dependencies as all non-prime attributes are fully functionally dependent on the candidate key (A).

The relation is in 2NF after decomposition.

Third Normal Form (3NF):

Check for transitive dependencies:

For Table 1 (AD): No transitive dependencies as A is a candidate key.

For Table 2 (ABC): No transitive dependencies as A is a candidate key.

The relation is in 3NF after decomposition.

Boyce-Codd Normal Form (BCNF):

Check for candidate keys and ensure all FDs are satisfied:

Candidate Key: A (since A → BCD)

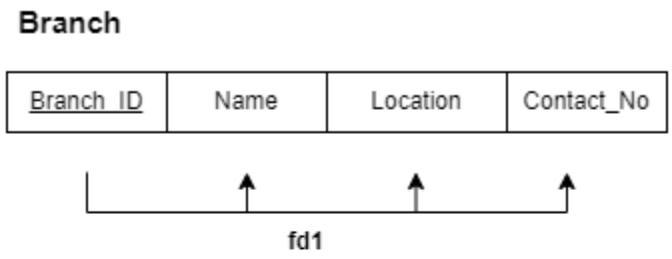
All FDs are already covered by the candidate key: A → BCD

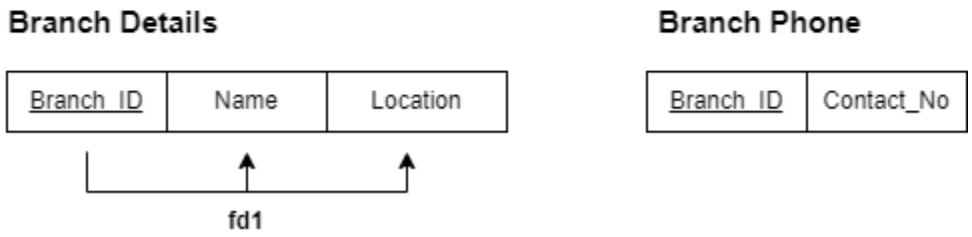
The relation satisfies BCNF.

Final Normalized Tables:

After normalization, the tables are: Table 1: Attributes: {A, D} Functional Dependency: A → D

Table 2: Attributes: {A, B, C} Functional Dependency: No additional FDs (already fully functionally dependent on A)





8. **Delivery**

Step-by-Step Normalization:

Analysis:

Given the relation for the table Delivery:

Attributes: {A, B, C, D, E, F, G, H}

Functional Dependencies (FDs): { A → BCDEFGH F → GH }

First Normal Form (1NF): The relation is assumed to be in 1NF as all attributes are atomic.

Second Normal Form (2NF): The relation is already in 1NF.

Candidate Key: A (since A → BCDEFGH)

Non-prime attributes: B, C, D, E, F, G, H

Check for partial dependencies:

A → BCDEFGH: Full functional dependency (no partial dependency on part of a composite key as A is not composite). F → GH: This is a partial dependency since F determines only part of the non-prime attributes GH.

To achieve 2NF: Decompose the relation to remove partial dependencies:

Table 1: A → BCDE with attributes {A, B, C, D, E}

Table 2: F → GH with attributes {F, G, H}

Third Normal Form (3NF): The relation is in 2NF.

Check for transitive dependencies: A → BCDE: A is a candidate key, so all attributes are directly dependent on A. F → GH: No transitive dependency as F is not a superkey.

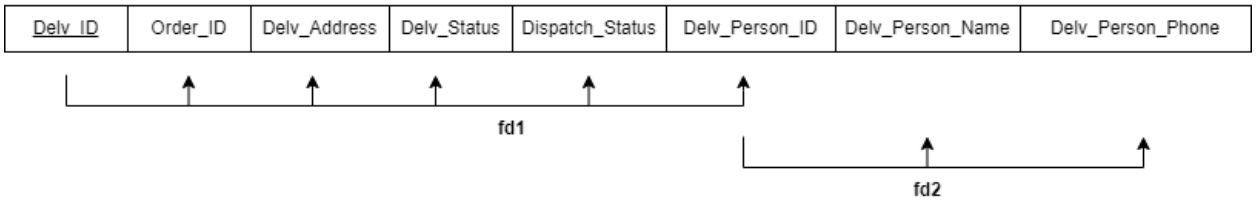
To achieve 3NF: No further decomposition is needed since there are no transitive dependencies.

Final Normalized Tables: After normalization, the tables are:

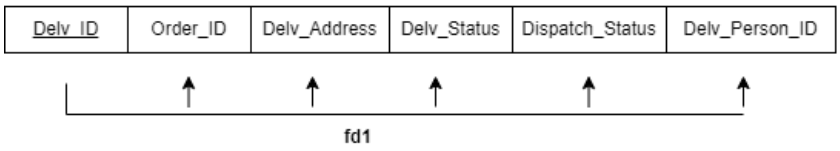
Table 1: Attributes: {A, B, C, D, E, F} Functional Dependency: A → BCDEF

Table 2: Attributes: {F, G, H} Functional Dependency: F → GH

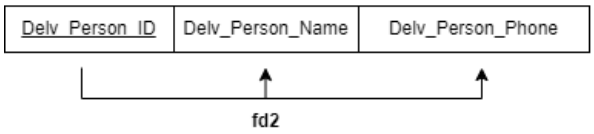
Delivery



Delivery Details



Delivery Person Details



9. **Reservation**

Step-by-Step Normalization:

Analysis: Given the relation for the table Reservation:

Attributes: {A, B, C, D, E}

Functional Dependencies (FDs): { A → BCDE }

First Normal Form (1NF): The relation is assumed to be in 1NF as all attributes are atomic.

Second Normal Form (2NF): The relation is already in 1NF.

Candidate Key: A (since A → BCDE)

Non-prime attributes: B, C, D, E

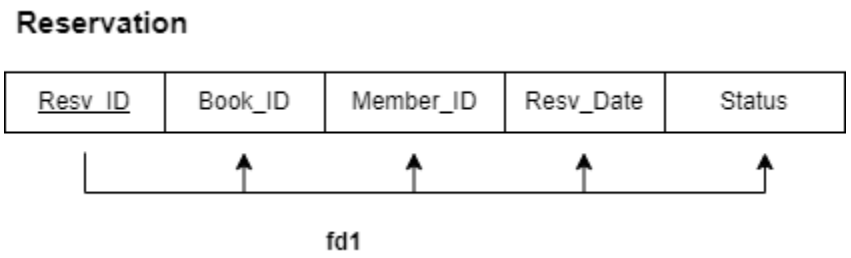
Check for partial dependencies:

A → BCDE: Full functional dependency (no partial dependency on part of a composite key as A is not composite).

To achieve 2NF: No decomposition is needed since there are no partial dependencies.

Third Normal Form (3NF): Check for transitive dependencies: A → BCDE: A is a candidate key, so all attributes are directly dependent on A.

Final Normalized Tables: After normalization, the table is: Table 1: Attributes: {A, B, C, D, E} Functional Dependency: A → BCDE



10. **E - Book**

Step-by-Step Normalization:

Analysis: Given the relation for the table E-book access:

Attributes: {A, B, C, D, E}

Functional Dependencies (FDs): { A → BCDE D → E }

First Normal Form (1NF): The relation is assumed to be in 1NF as all attributes are atomic.

Second Normal Form (2NF): The relation is already in 1NF.

Candidate Key: A (since A → BCDE)

Non-prime attributes: B, C, D, E

Check for partial dependencies:

A → BCDE: Full functional dependency (no partial dependency on part of a composite key as A is not composite).

D → E: Full functional dependency (no partial dependency as D determines E completely).

To achieve 2NF: No decomposition is needed since there are no partial dependencies.

Third Normal Form (3NF):

Check for transitive dependencies: A → BCDE: A is a candidate key, so all attributes are directly dependent on A.

D → E: No transitive dependency as D is not a superkey.

Final Normalized Tables:

After normalization, the table is:

Table 1: Attributes: {A, B, C, D, E} Functional Dependencies: { A → BCDE, D → E }