LIBRARY MANAGEMENT SYSTEM

UCS2304 – Database Management Systems

Documentation

Submitted By

Samyuktha D (3122 22 5001 309)

Rithick.R.Rahul (3122 22 5001 107)

Preethi Prative (3122 22 5001 098)

Department of Computer Science and Engineering

Sri Sivasubramaniya Nadar College of Engineering

Kalavakkam – 603110

TABLES AND THEIR ATTRIBUTES

1. **Branch:**

    1.1 Branch_ID
    1.2 Branch_Name
    1.3 City
    1.4 State
    1.5 Pincode

2. **Author:**

    2.1 Author_ID
    2.2 Author_Name
    2.3 Gender

3. **Publisher:**

    3.1 Publisher_ID
    3.2 Publisher_Name
    3.3 City
    3.4 State
    3.5 Pincode

4. **Genre:**

    4.1 Genre_ID
    4.2 Genre

5. **Catalog:**

    5.1 Branch_ID
    5.2 Title
    5.3 ISBN
    5.4 Author_ID
    5.5 Publisher_ID
    5.6 Genre_ID
    5.7 Format
    5.8 Price
    5.9 Publication_Year
    5.10 Copies_Count

6. **Librarian:**

6.1 Librarian_ID
6.2 Branch_ID
6.3 Name
6.4 Gender

7. **Member:**

7.1 Member_ID
7.2 Name
7.3 Gender
7.4 City
7.5 State
7.6 Pincode

8. **Transactions:**

8.1 Transaction_ID
8.2 Member_ID
8.3 Librarian_ID
8.4 Title
8.5 ISBN
8.6 Author_ID
8.7 Publisher_ID
8.8 Genre
8.9 Format
8.10 Price
8.11 Issue_Date
8.12 Due_Date
8.13 Return_Date
8.14 Penalty
8.15 Publication_Year

FUNCTIONAL DEPENDENCIES AND NORMALIZATION

Branch:

Attributes:

- Branch_ID
- Branch_Name
- City
- State
- Pincode

Functional Dependencies:

- Branch_ID -> Branch_Name , City , State , Pincode
- Pincode -> City , State

Step 1:

Decomposition:

- Branch_ID -> Branch_Name
- Branch_ID -> City
- Branch_ID -> State
- Branch_ID -> Pincode
- Pincode -> City
- Pincode -> State

Step 2:

Checking for extraneous attributes:

No attributes to check as each FD is already atomic.

Step 3:

Checking for Redundancies:

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Branch_ID -> Branch_Name is redundant**

Remove Branch_ID -> Branch_Name and check if the remaining FDs can determine Branch Name:

Remaining FDs: {Branch_ID->City , Branch_ID->State , Branch_ID->Pincode , Pincode->City , Pincode->State}

Closure of {Branch_ID}:

{Branch_ID} -> {Branch_ID , City , State , Pincode}

Since Branch_Name is not included in the closure of {Branch_ID} without Branch_ID->Branch_Name, this FD is not redundant.

b) **Checking if Branch_ID -> City is redundant**

Remove Branch_ID -> City and check if the remaining FDs can determine City:

Remaining FDs: {Branch_ID->Branch_Name , Branch_ID->State , Branch_ID->Pincode , Pincode->City , Pincode->State}

Closure of {Branch_ID}:

{Branch_ID} -> {Branch_ID , Branch_Name , City , State , Pincode}

Since Pincode is in closure set of Branch_ID, and City is in closure set of Pincode, City is in closure set of Branch_ID.

Since City is included in the closure of {Branch_ID} ,the FD Branch_Name -> City is redundant

c) **Checking if Branch_ID -> State is redundant**

Remove Branch_ID -> State and check if the remaining FDs can determine State:

Remaining FDs: {Branch_ID->Branch_Name , Branch_ID->City , Branch_ID->Pincode , Pincode->City , Pincode->State}

Closure of {Branch_ID}:

{Branch_ID} -> {Branch_ID , Branch_Name , City , State , Pincode}

Since Pincode is in closure set of Branch_ID, and State is in closure set of Pincode, City is in closure set of Branch_ID.

Since State is included in the closure of {Branch_ID}, the FD Branch_ID -> State is redundant

d) **Checking if Branch_ID -> Pincode is redundant**

Remove Branch_ID -> Pincode and check if the remaining FDs can determine Pincode:

Remaining FDs: {Branch_ID->Branch_Name , Branch_ID->City , Branch_ID->State , Pincode->City , Pincode->State}

Closure of {Branch_ID}:

{Branch_ID} -> {Branch_ID , Branch_Name , City , State}

Since Pincode is not included in the closure of {Branch_ID} without Branch_ID->Pincode, this FD is not redundant

e) Checking if Pincode -> City is redundant

Remove Pincode -> City and check if the remaining FDs can determine City:

Remaining FDs: {Branch_ID->Branch_Name , Branch_ID->City , Branch_ID->State , Branch_ID->Pincode , Pincode->State}

Closure of {Pincode}:

{Pincode} -> {Pincode , State}

Since City is not included in the closure of {Pincode} without Pincode->City, this FD is not redundant

f) Checking if Pincode -> State is redundant

Remove Pincode -> State and check if the remaining FDs can determine State:

Remaining FDs: {Branch_ID->Branch_Name , Branch_ID->City , Branch_ID->State , Branch_ID->Pincode , Pincode->City}

Closure of {Branch_ID}:

{Pincode} -> {Pincode , City}

Since State is not included in the closure of {Pincode} without Pincode -> State, this FD is not redundant

Final Analysis:

After checking all functional dependencies, there are two redundant FDs, i.e., Branch_ID->City , Branch_ID->State.

Primary Key Determination:

The primary key for the “Branch” table is Branch_ID because it uniquely determines all other attributes in the table.

Minimal Set of FDs:

- Branch_ID -> Branch_Name
- Branch_ID -> Pincode
- Pincode -> City
- Pincode -> State

Step 4:

Normalization:

Minimal FDs

- Branch_ID -> Branch_Name
- Branch_ID -> Pincode
- Pincode -> City
- Pincode -> State

Candidate key: Branch_ID

Prime Attributes: Branch_ID**,** Pincode**,** City**,** State

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

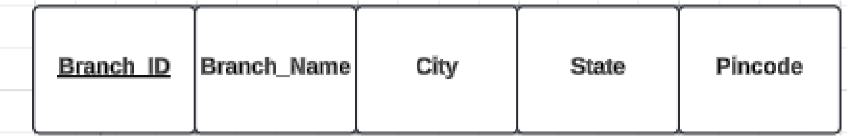
Analyzing for Partial Dependencies:

- Branch_ID -> Branch_Name
- Branch_ID -> Pincode

  Both Branch_Name and Pincode are fully dependent on Branch_ID, which is a part of the candidate key. Hence**, no partial dependency exists.**

Conclusion for 2NF:

The **table is already in 2NF** because there are **no partial dependencies**.



Branch_ID	Branch_Name	City	State	Pincode

Step 3: Third Normal Form (3NF)

3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

- Pincode -> City
- Pincode -> State

  This indicate a transitive dependency where Pincode determines both City and State.

  Pincode, City and State are NPA.

Decomposition to Achieve 3NF:

To remove the transitive dependency, we decompose the table into two tables: **Branch** and **Pincode**.

Branch Table:

Branch_ID (Primary Key)

Branch_Name

Pincode (Foreign Key)

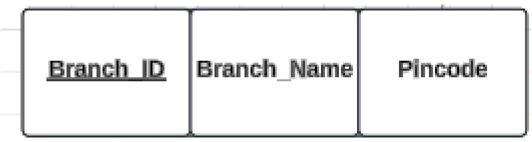
Pincode Table:

Pincode (Primary Key)

City

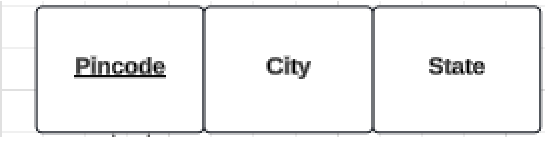
State

TABLE FOR BRANCH:



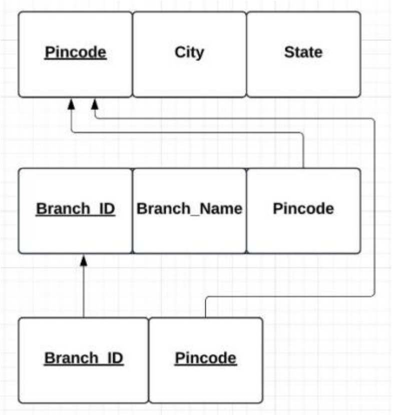
Branch_ID	Branch_Name	Pincode

TABLE FOR LOCATION:



Pincode	City	State

TABLE FOR 3NF:



Checking 3NF Compliance:

- **Branch Table:**

  Branch_ID is the primary key.

  Branch_Name is directly dependent on Branch_ID.

  Pincode is directly dependent on Branch_ID.

- **Pincode Table:**

  Pincode is the primary key.

  City and State are directly dependent on Pincode.

FINAL ANALYSIS:

By decomposing the Branch table into Branch and Pincode tables, we achieve 3NF

Branch Table:

Branch_ID (Primary Key)

Branch_Name

Pincode (Foreign Key)

Pincode Table:

Pincode (Primary Key)

City

State

This decomposition ensures that:

1NF: Attributes are atomic.

2NF: No partial dependencies.

3NF: No transitive dependencies.

Author

Attributes:

- **Author_ID**
- **Author_Name**
- **Gender**

Functional Dependencies:

- Author_ID -> Author_Name , Gender

Step 1:

Decomposition:

- Author_ID -> Author_Name
- Author_ID -> Gender

Step 2:

Checking for extraneous attributes

No attributes to check as each FD is already atomic

Step 3:

Checking for Redundancies:

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Author_ID -> Author_Name is redundant**

   Remove Author_ID -> Author_Name and check if the remaining FDs can determine Author Name:

   Remaining FDs: {Author_ID -> Gender}

   Closure of {Author_ID}:

   {Author_ID} -> {Author_ID , Gender}

   Since Author_Name is not included in the closure of {Author_ID} without Author_ID->Author_Name, this FD is not redundant.

b) **Checking if Author_ID -> Gender is redundant**

   Remove Author_ID -> Author_Name and check if the remaining FDs can determine Gender:

   Remaining FDs: {Author_ID -> Author_Name}

   Closure of {Author_ID}:

   {Author_ID} -> {Author_ID , Author_Name}

   Since Gender is not included in the closure of {Author_ID} without Author_ID->Gender, this FD is not redundant.

Final Analysis:

After checking all functional dependencies, there are no redundant FDs.

Primary Key Determination:

The primary key for the “Author” table is Author_ID because it uniquely determines all other attributes in the table.

Minimal Set of FDs:

- Author_ID -> Author_Name
- Author_ID -> Gender

Step 4:

Normalization:

Minimal FDs

- Author_ID -> Author_Name
- Author_ID -> Gender

Candidate key: Author_ID

Prime Attributes: Author_ID

Non-Prime Attributes: Author_Name, Gender

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

Analyzing for Partial Dependencies:

- Author_ID -> Author_Name
- Author_ID -> Gender

Both Author_Name and Gender are fully dependent on Author_ID, which is a part of the candidate key. Hence**, no partial dependency exists.**

Conclusion for 2NF:

The table is already in 2NF because there are no partial dependencies.

Step 3: Third Normal Form (3NF)

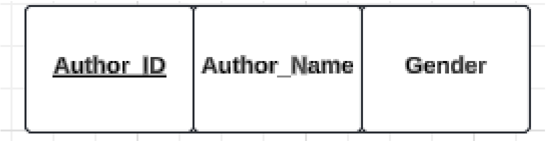
3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

- Author_ID -> Author_Name
- Author_ID -> Gender

There are no transitive dependencies in the current schema because Author_Name and Gender depend directly on the candidate key (Author_ID), which is the primary key.

TABLE FOR AUTHOR:



Author_ID	Author_Name	Gender

FINAL ANALYSIS:

The Author table, based on the given functional dependencies, is already in 1NF, 2NF, 3NF. There are no violations of these normal forms because:

- The table is properly structured with atomic values.
- All non-prime attributes depend on the candidate key (Author_ID), and there are no transitive dependencies.
- Thus, the Author table is well-normalized according to the principles of relational database normalization.

Publisher

Attributes:

- Publisher_ID
- Publisher_Name
- City
- State
- Pincode

Functional Dependencies:

- Publisher_ID -> Publisher_Name , City , State , Pincode
- Pincode -> City , State

Step 1:

Decomposition:

- Publisher_ID -> Publisher_Name
- Publisher_ID -> City
- Publisher_ID -> State
- Publisher_ID -> Pincode
- Pincode -> City
- Pincode -> State

Step 2:

Checking for extraneous attributes

No attributes to check as each FD is already atomic.

Step 3:

Checking for redundancies

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Publisher_ID -> Publisher_Name is redundant**

Remove Publisher_ID -> Publisher_Name and check if the remaining FDs can determine Publisher Name:

Remaining FDs: { Publisher_ID -> City , Publisher_ID -> State , Publisher_ID -> Pincode , Pincode -> City , Pincode -> State}

Closure of {Publisher_ID}:

{Publisher_ID} -> {Publisher_ID , City , State , Pincode}

Since Publisher_Name is not included in the closure of {Publisher_ID} without Publisher_ID->Publisher_Name, this FD is not redundant.

b) **Checking if Publisher_ID -> City is redundant**

Remove Publisher_ID -> City and check if the remaining FDs can determine City:

Remaining FDs: {Publisher_ID -> Publisher_Name , Publisher_ID -> State , Publisher_ID -> Pincode , Pincode -> City , Pincode -> State}

Closure of {Publisher_ID}:

{Publisher_ID} -> {Publisher_ID , Publisher_Name , City , State , Pincode}

Since Pincode is in closure set of Publisher_ID, and City is in closure set of Pincode, City is in closure set of Publisher_ID.

Since City is included in the closure of {Publisher_ID}, the FD Publisher_ID -> City is redundant.

c) **Checking if Publisher_ID -> State is redundant**

Remove Publisher_ID -> State and check if the remaining FDs can determine State:

Remaining FDs: {Publisher_ID->Publisher_Name , Publisher_ID->City , Publisher_ID->Pincode , Pincode->City , Pincode->State}

Closure of {Publisher_ID}:

{Publisher_ID} -> {Publisher_ID , Publisher_Name , City , State , Pincode}

Since Pincode is in closure set of Publisher_ID, and State is in closure set of Pincode, State is in closure set of Publisher_ID.

Since State is included in the closure of {Publisher_ID}, the FD Publisher_ID -> State is redundant

d) Checking if Publisher_ID -> Pincode is redundant

Remove Publisher_ID -> Pincode and check if the remaining FDs can determine Pincode:

Remaining FDs: {Publisher_ID->Publisher_Name , Publisher_ID->City , Publisher_ID->State , Pincode->City , Pincode->State}

Closure of {Publisher_ID}:

{Publisher_ID} -> {Publisher_ID , Publisher_Name , City , State}

Since Pincode is not included in the closure of {Publisher_ID} without Publisher_ID->Pincode, this FD is not redundant

e) Checking if Pincode -> City is redundant

Remove Pincode -> City and check if the remaining FDs can determine City:

Remaining FDs: {Publisher_ID->Publisher_Name , Publisher_ID->City , Publisher_ID->State , Publisher_ID->Pincode , Pincode->State}

Closure of {Pincode}:

{Pincode} -> {Pincode , State}

Since City is not included in the closure of {Pincode} without Pincode->City, this FD is not redundant

f) Checking if Pincode -> State is redundant

Remove Pincode -> State and check if the remaining FDs can determine State:

Remaining FDs: {Publisher_ID->Publisher_Name , Publisher_ID->City , Publisher_ID->State , Publisher_ID->Pincode , Pincode->City}

Closure of {Publisher_ID}:

{Pincode} -> {Pincode , City}

Since State is not included in the closure of {Pincode} without Pincode -> State, this FD is not redundant

Final Analysis:

After checking all functional dependencies, there are two redundant FDs, i.e., Publisher_ID->City , Publisher_ID->State.

Primary Key Determination:

The primary key for the "Publisher" table is Publisher_ID because it uniquely determines all other attributes in the table.

Minimal Set of FDs:

- Publisher_ID -> Publisher_Name
- Publisher_ID -> Pincode
- Pincode -> City
- Pincode -> State

Step 4:

Normalization:

Minimal FDs

- Publisher_ID -> Publisher_Name , City , State , Pincode
- Pincode -> City , State

Candidate key: Publisher_ID

Prime Attributes: Publisher_ID

Non-Prime Attributes: Publisher_Name , City , State , Pincode, City , State

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

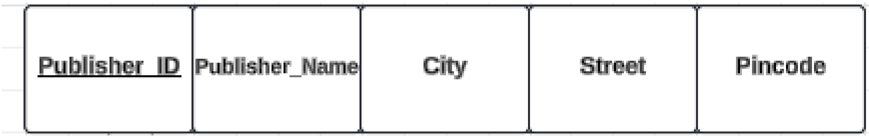
Analyzing for Partial Dependencies:

- Publisher_ID -> Publisher_Name , City , State , Pincode
- Pincode -> City , State

Publisher_Name , City , State , Pincode are fully dependent on Author_ID, which is a part of the candidate key. Hence**, no partial dependency exists.**

Conclusion for 2NF:

The table is already in 2NF because there are no partial dependencies.



Publisher_ID	Publisher_Name	City	Street	Pincode

Step 3: Third Normal Form (3NF)

3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

From the functional dependencies, **Pincode -> City, State** creates **a transitive dependency**. This means City and State are transitively dependent on Publisher_ID through Pincode.

To resolve this, we decompose the table to eliminate the transitive dependency:

Decomposition:

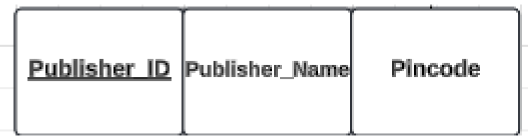
Publisher Table:

Publisher_ID (PK)
Publisher_Name
Pincode (FK)

Location Table:
Pincode (PK)
City
State

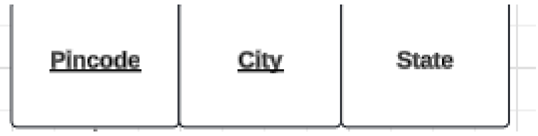
Now, the Publisher table has no transitive dependencies and is in 3NF.

TABLE FOR PUBLISHER:



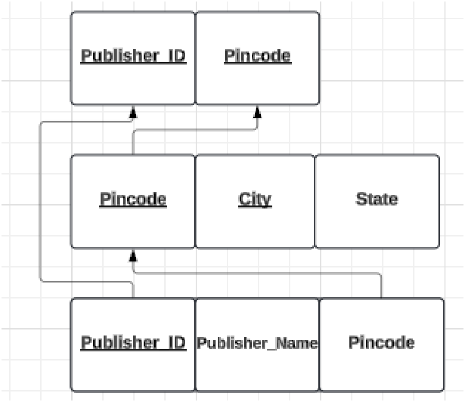
Publisher_ID	Publisher_Name	Pincode

TABLE FOR LOCATION:



Pincode	City	State

TABLE FOR 3NF:



FINAL ANALYSIS:

By decomposing the Publisher table into Publisher and Pincode tables, we achieve 3NF

This decomposition ensures that:

1NF: Attributes are atomic.

2NF: No partial dependencies.

3NF: No transitive dependencies.

Genre:

Attributes:

- Genre_ID
- Genre

Functional Dependencies:

- Genre_ID -> Genre

Step 1:

Decomposition:

Both left and right hand side of the FD has only one attribute. The FD is fully decomposed.

Step 2:

Checking for extraneous attributes:

No attributes to check as the FD is already atomic.

Step 3:

Checking for redundancies

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Genre_ID -> Genre is redundant**

Remove Genre_ID -> Genre and check if the remaining FDs can determine Publisher Name:

Remaining FDs: { }

Closure of {Genre_ID}:

{Genre_ID} -> {Genre_ID}

Since Genre is not included in the closure of {Genre_ID} without Genre_ID->Genre, this FD is not redundant.

Final Analysis:

After checking all functional dependencies, there are no redundant FDs.

Primary Key Determination:

The primary key for the "Genre" table is Genre_ID because it uniquely determines all other attributes in the table.

Minimal Set of FDs:

- Genre_ID -> Genre

Step 4:

Normalization:

Minimal FDs

- Genre_ID -> Genre

Candidate key: Genre_ID

Prime Attributes: Genre_ID

Non-Prime Attributes: Genre

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

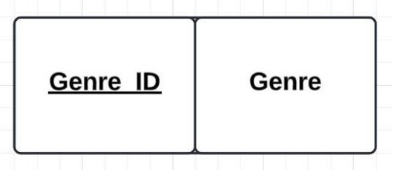
Analyzing for Partial Dependencies:

- Genre_ID -> Genre

  Genre is fully dependent on Genre_ID, which is a part of the candidate key. Hence**, no partial dependency exists.**

Conclusion for 2NF:

The **table is already in 2NF** because there are **no partial dependencies**.



Genre_ID	Genre

Step 3: Third Normal Form (3NF)

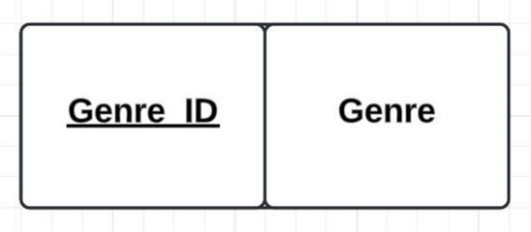
3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

- Genre_ID -> Genre

There are no transitive dependencies in the current schema because Genre depend directly on the candidate key (Genre_ID), which is the primary key.

TABLE FOR GENRE:



Genre_ID	Genre

FINAL ANALYSIS:

The Genre table, based on the given functional dependencies, is already in 1NF, 2NF, 3NF. There are no violations of these normal forms because:

- The table is properly structured with atomic values.

- All non-prime attributes depend on the candidate key (Genre_ID), and there are no transitive dependencies.
- Thus, the Genrer table is well-normalized according to the principles of relational database normalization.

Catalog:

Attributes:

- Branch_ID
- ISBN
- Title
- Author_ID
- Publisher_ID
- Genre_ID
- Format
- Price
- Publication_Year
- Copies_Count

Functional Dependencies:

- Branch_ID, ISBN -> Title, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count
- ISBN -> Title, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year

Step 1:

Decomposition:

- Branch_ID , ISBN -> Title
- Branch_ID , ISBN -> Author_ID
- Branch_ID , ISBN -> Publisher_ID
- Branch_ID , ISBN -> Genre_ID
- Branch_ID , ISBN -> Format
- Branch_ID , ISBN -> Price
- Branch_ID , ISBN -> Publication_Year
- Branch_ID , ISBN -> Copies_Count
- ISBN -> Title

- ISBN -> Author_ID
- ISBN -> Publisher_ID
- ISBN -> Genre_ID
- ISBN -> Format
- ISBN -> Price
- ISBN -> Publication_Year

Step 2:

Checking for extraneous attributes:

Closure of Branch_ID = {Branch_ID}

So Branch_ID alone cannot determine all other attributes.

Closure of ISBN = {Title , Author_ID , Publisher_ID , Genre_ID , Format , Price , Publication_Year}

So ISBN alone can determine Title , Author_ID , Publisher_ID , Genre_ID , Format , Price , Publication_Year.

Hence, Branch_ID is an extraneous attribute.

Step 3:

Checking for Redundancies:

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Branch_ID, ISBN -> Title is redundant:**

Remove Branch_ID, ISBN -> Title and check if the remaining FDs can determine Title:

Remaining FDs: {Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Title is in closure set of ISBN, Title is in closure set of {Branch_ID , ISBN}.

Since Title is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Title is redundant.

b) Checking if Branch_ID, ISBN -> Author_ID is redundant:

Remove Branch_ID, ISBN -> Author_ID and check if the remaining FDs can determine Author_ID:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Author_ID is in closure set of ISBN, Author_ID is in closure set of {Branch_ID , ISBN}.

Since Author_ID is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Author_ID is redundant.

c) Checking if Branch_ID, ISBN -> Publisher_ID is redundant:

Remove Branch_ID, ISBN -> Publisher_ID and check if the remaining FDs can determine Publisher_ID:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Publisher_ID is in closure set of ISBN, Publisher_ID is in closure set of {Branch_ID , ISBN}.

Since Publisher_ID is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Publisher_ID is redundant.

d) Checking if Branch_ID, ISBN -> Genre_ID is redundant:

Remove Branch_ID, ISBN -> Genre_ID and check if the remaining FDs can determine Genre_ID:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Genre is in closure set of ISBN, Genre is in closure set of {Branch_ID , ISBN}.

Since Genre_ID is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Genre_ID is redundant.

e) Checking if Branch_ID, ISBN -> Format is redundant:

Remove Branch_ID, ISBN -> Format and check if the remaining FDs can determine Format:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Format is in closure set of ISBN, Format is in closure set of {Branch_ID , ISBN}.

Since Format is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Format is redundant.

f) **Checking if Branch_ID, ISBN -> Price is redundant:**

Remove Branch_ID, ISBN -> Price and check if the remaining FDs can determine Price:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Price is in closure set of ISBN, Price is in closure set of {Branch_ID , ISBN}.

Since Price is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Price is redundant.

g) **Checking if Branch_ID, ISBN -> Publication_Year is redundant:**

Remove Branch_ID, ISBN -> Publication_Year and check if the remaining FDs can determine Publication_Year:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year, Copies_Count}

Since ISBN is in closure set of {Branch_ID , ISBN} , and Publication_Year is in closure set of ISBN, Publication_Year is in closure set of {Branch_ID , ISBN}.

Since Publication_Year is included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Publication_Year is redundant.

h) **Checking if Branch_ID, ISBN -> Copies_Count is redundant:**

Remove Branch_ID, ISBN -> Copies_Count and check if the remaining FDs can determine Copies_Count:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {Branch_ID, ISBN}:

{Branch_ID, ISBN} -> {Branch_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year}

Since Copies_Count is not included in the closure of {Branch_ID , ISBN}, the FD {Branch_ID , ISBN} -> Copies_Count is not redundant.

i) Checking if ISBN -> Title is redundant:

Remove ISBN -> Title and check if the remaining FDs can determine Title:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year}

Since Title is not included in the closure of {ISBN}, the FD {ISBN} -> Title is not redundant.

j) Checking if ISBN -> Author_ID is redundant:

Remove ISBN -> Author_ID and check if the remaining FDs can determine Author_ID:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title , Publisher_ID, Genre_ID, Format, Price, Publication_Year}

Since Author_ID is not included in the closure of {ISBN}, the FD {ISBN} -> Author_ID is not redundant.

k) Checking if ISBN -> Publisher_ID is redundant:

Remove ISBN -> Publisher_ID and check if the remaining FDs can determine Publisher_ID:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title , Genre_ID, Format, Price, Publication_Year}

Since Publisher_ID is not included in the closure of {ISBN}, the FD {ISBN} -> Publisher_ID is not redundant.

l) **Checking if ISBN -> Genre_ID is redundant:**

Remove ISBN -> Genre_ID and check if the remaining FDs can determine Genre_ID:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title , Author_ID, Publisher_ID, Format, Price, Publication_Year}

Since Genre_ID is not included in the closure of {ISBN}, the FD {ISBN} -> Genre_ID is not redundant.

m) **Checking if ISBN -> Format is redundant:**

Remove ISBN -> Format and check if the remaining FDs can determine Format:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Price, ISBN -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre_ID, Price, Publication_Year}

Since Format is not included in the closure of {ISBN}, the FD {ISBN} -> Format is not redundant.

n) Checking if ISBN -> Price is redundant:

Remove ISBN -> Price and check if the remaining FDs can determine Price:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title , Author_ID, Publisher_ID, Genre_ID, Format, Publication_Year}

Since Price is not included in the closure of {ISBN}, the FD {ISBN} -> Price is not redundant.

o) Checking if ISBN -> Publication_Year is redundant:

Remove ISBN -> Publication_Year and check if the remaining FDs can determine Publication_Year:

Remaining FDs: {Branch_ID , ISBN -> Title , Branch_ID, ISBN -> Author_ID, Branch_ID, ISBN -> Publisher_ID, Branch_ID, ISBN -> Genre_ID, Branch_ID, ISBN -> Format, Branch_ID, ISBN -> Price, Branch_ID, ISBN -> Publication_Year, Branch_ID, ISBN -> Copies_Count, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre_ID, ISBN -> Format, ISBN -> Price}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title , Author_ID, Publisher_ID, Genre_ID, Format, Price}

Since Publication_Year is not included in the closure of {ISBN}, the FD {ISBN} -> Publication_Year is not redundant.

Final Analysis:

After checking all functional dependencies, there are 7 redundant FDs. They are {Branch_ID,ISBN} -> Title, {Branch_ID,ISBN} -> Author_ID, {Branch_ID,ISBN} -> Publisher_ID , {Branch_ID,ISBN} -> Genre_ID , {Branch_ID,ISBN} -> Format, {Branch_ID,ISBN} -> Price, {Branch_ID,ISBN} -> Publication_Year.

Primary Key Determination:

The primary key for the "Catalog" table is a combined key of (Branch_ID,ISBN} because it uniquely determines all other attributes in the table.

Minimal Set of FDs:

- Branch_ID,ISBN -> Copies_Count
- ISBN -> Title
- ISBN -> Author_ID
- ISBN -> Publisher_ID
- ISBN -> Genre_ID
- ISBN -> Format
- ISBN -> Price
- ISBN -> Publication_Year

Step 4:

Normalization:

Minimal FDs

- Branch_ID, ISBN -> Copies_Count
- ISBN -> Title
- ISBN -> Author_ID
- ISBN -> Publisher_ID
- ISBN -> Genre_ID
- ISBN -> Format

- ISBN -> Price
- ISBN -> Publication_Year

Candidate key: Branch_ID, ISBN

Prime Attributes: Branch_ID, ISBN

Non-Prime Attributes: Title, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

Analyzing for Partial Dependencies:

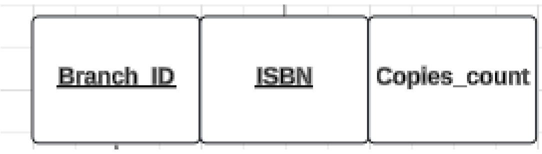
> From the FDs, all non-prime attributes (Title, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year) depend only on ISBN, **which is a subset of the primary key (Branch_ID, ISBN).**
>
> This indicates a partial dependency, meaning the table is not in 2NF.
>
> To resolve this, we decompose the table to ensure that all non-prime attributes are fully functionally dependent on the entire primary key**.**

Decomposition:

1. **Catalog Table:**
    - Branch_ID (PK)
    - ISBN (PK)
    - Copies_Count
2. **Book Table:**

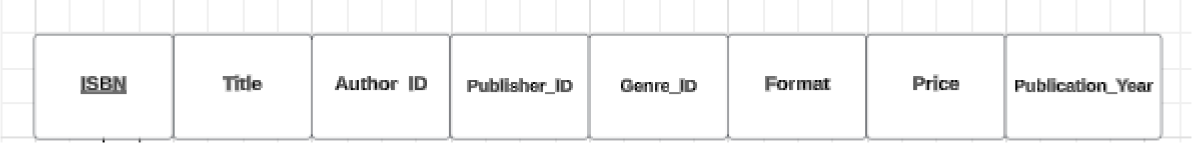
    - ISBN (PK)
    - Title
    - Author_ID
    - Publisher_ID
    - Genre_ID
    - Format
    - Price
    - Publication_Year

TABLE FOR CATALOG:



Branch_ID	ISBN	Copies_count

TABLE FOR BOOKS:



ISBN	Title	Author_ID	Publisher_ID	Genre_ID	Format	Price	Publication_Year

Step 3: Third Normal Form (3NF)

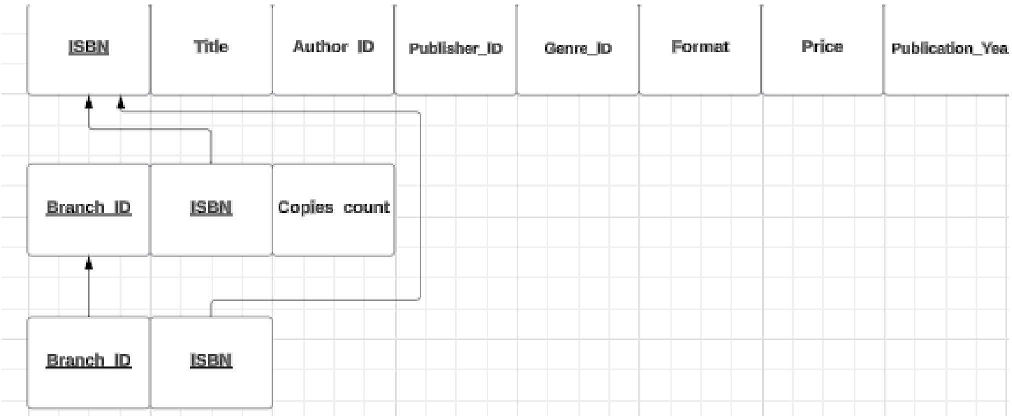
3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

- There are no transitive dependencies because there are only two attributes: (Branch_ID, ISBN) which is the primary key, and Copies_Count which is fully dependent on the primary key.

- Each non-prime attribute (Title, Author_ID, Publisher_ID, Genre_ID, Format, Price, Publication_Year) is directly dependent on the primary key, ISBN, and there are no dependencies among the non-prime attributes themselves.

TABLE FOR 3NF:



ISBN	Title	Author ID	Publisher_ID	Genre_ID	Format	Price	Publication_Yea

Branch ID	ISBN	Copies count

Branch ID	ISBN

FINAL ANALYSIS:

By decomposing the Catalog table into Catalog table and Book tables, we achieve 2NF

This decomposition ensures that:

1NF: Attributes are atomic.

2NF: No partial dependencies.

3NF: No transitive dependencies.

Librarian:

Attributes:

- Librarian_ID
- Branch_ID
- Name
- Gender

Functional Dependencies:

- Librarian_ID -> Branch_ID , Name , Gender

Step 1:

Decomposition:

- Librarian_ID -> Branch_ID
- Librarian_ID -> Name
- Librarian_ID -> Gender

Step 2:

Checking for extraneous attributes

No attributes to check as each FD is already atomic.

Step 3:

Checking for redundancies

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) Checking if Librarian_ID -> Branch_ID is redundant:

Remove Librarian_ID -> Branch_ID and check if the remaining FDs can determine Branch_ID:

Remaining FDs: {Librarian_ID -> Name , Librarian_ID -> Gender}

Closure of {Librarian_ID}:

{Librarian_ID} -> {Librarian_ID , Name , Gender}

Since Branch_ID is not included in the closure of {Librarian_ID}, the FD {Librarian_ID} -> Branch_ID is not redundant.

b) **Checking if Librarian_ID -> Name is redundant:**

Remove Librarian_ID -> Name and check if the remaining FDs can determine Name:

Remaining FDs: { Librarian_ID -> Branch_ID , Librarian_ID -> Gender}

Closure of {Librarian_ID}:

{Librarian_ID} -> {Librarian_ID , Branch_ID , Gender }

Since Name is not included in the closure of {Librarian_ID}, the FD {Librarian_ID} -> Name is not redundant.

c) **Checking if Librarian_ID -> Gender is redundant:**

Remove Librarian_ID -> Gender and check if the remaining FDs can determine Gender:

Remaining FDs: {Librarian_ID -> Branch_ID , Librarian_ID -> Name}

Closure of {Librarian_ID}:

{Librarian_ID} -> {Librarian_ID , Branch_ID , Name , Genre }

Since Gender is not included in the closure of {Librarian_ID}, the FD {Librarian_ID} -> Gender is not redundant.

Final Analysis:

After checking all functional dependencies, there are no redundant FDs.

Primary Key Determination:

The primary key for the "Librarian" table is Librarian_ID because it uniquely determines all other attributes in the table.

Minimal set of FDs:

- Librarian_ID -> Branch_ID
- Librarian_ID -> Name
- Librarian_ID -> Gender

Step 4:

Normalization:

Minimal FDs

- Librarian_ID -> Branch_ID
- Librarian_ID -> Name
- Librarian_ID -> Gender

Candidate Key

- Librarian_ID

Prime Attributes

- Librarian_ID

Non-Prime Attributes

- Branch_ID
- Name
- Gender

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

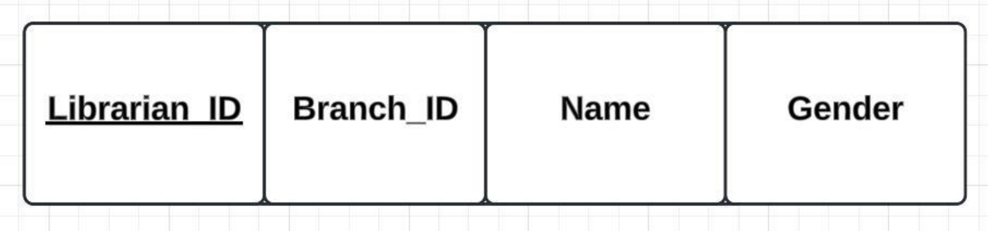
All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

Analyzing for Partial Dependencies

- All non-prime attributes (Branch_ID, Name, Gender) depend entirely on **Librarian_ID**, which is the primary key.
- Since **Librarian_ID** is the only candidate key and there are no partial dependencies, the table is already in 2NF.



Librarian_ID	Branch_ID	Name	Gender

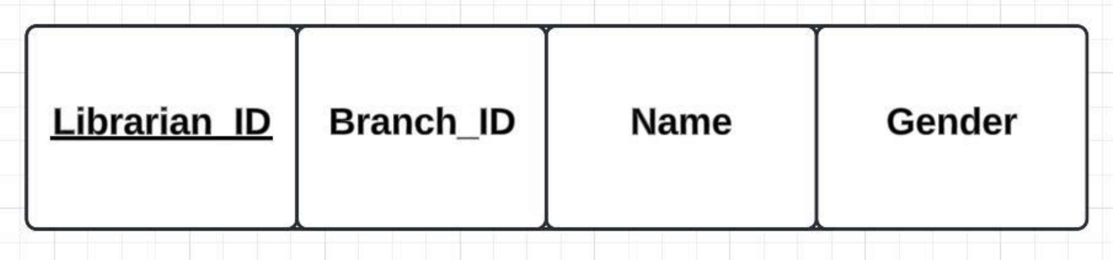
Step 3: Third Normal Form (3NF)

3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

Each non-prime attribute (Branch_ID, Name, Gender) is directly dependent on the primary key Librarian_ID, and there are no dependencies among the non-prime attributes themselves.

TABLE FOR 3NF:



Librarian_ID	Branch_ID	Name	Gender

FINAL ANALYSIS:

Since the table is in 1NF and there are no partial or transitive dependencies, the table is already in 2NF and 3NF.

Primary Keys and Foreign Keys

Librarian Table:

Primary Key (PK): Librarian_ID

This decomposition ensures that:

1NF: Attributes are atomic.

2NF: No partial dependencies.

3NF: No transitive dependencies.

Member:

Attributes:

- Member_ID
- Name
- Gender
- City
- State
- Pincode

Functional Dependencies:

- Member_ID -> Name , Gender , City , State , Pincode
- Pincode -> City , State

Step 1:

Decomposition:

- Member_ID -> Name
- Member_ID -> Gender
- Member_ID -> City
- Member_ID -> State
- Member_ID -> Pincode
- Pincode -> City
- Pincode -> State

Step 2:

Checking for extraneous attributes:

No attributes to check as each FD is already atomic.

Step 3:

Checking for redundancies:

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Member_ID -> Name is redundant:**

Remove Member_ID -> Name and check if the remaining FDs can determine Name:

Remaining FDs: {Member_ID -> Gender, Member_ID -> City, Member_ID -> State, Member_ID -> Pincode, Pincode -> City, Pincode -> State}

Closure of {Member_ID}:

{Member_ID} -> {Member_ID , Gender , City , State , Pincode}

Since Name is not included in the closure of {Member_ID}, the FD {Member_ID} -> Name is not redundant.

b) **Checking if Member_ID -> Gender is redundant:**

Remove Member_ID -> Gender and check if the remaining FDs can determine Gender:

Remaining FDs: { Member_ID -> Name, Member_ID -> City, Member_ID -> State, Member_ID -> Pincode, Pincode -> City, Pincode -> State}

Closure of {Member_ID}:

{Member_ID} -> {Member_ID , Name , City , State , Pincode}

Since Gender is not included in the closure of {Member_ID}, the FD {Member_ID} -> Gender is not redundant.

c) Checking if Member_ID -> City is redundant:

Remove Member_ID -> City and check if the remaining FDs can determine City:

Remaining FDs: {Member_ID -> Name, Member_ID -> Gender, Member_ID -> State, Member_ID -> Pincode, Pincode -> City, Pincode -> State}

Closure of {Member_ID}:

{Member_ID} -> {Member_ID , Name , Gender , City , State , Pincode}

Since Pincode is in closure set of Member_ID, and City is in closure set of Pincode, City is in closure set of Member_ID.

Since City is included in the closure of {Member_ID}, the FD {Member_ID} -> City is redundant.

d) Checking if Member_ID -> State is redundant:

Remove Member_ID -> State and check if the remaining FDs can determine State:

Remaining FDs: {Member_ID -> Name, Member_ID -> Gender, Member_ID -> City, Member_ID -> Pincode, Pincode -> City, Pincode -> State}

Closure of {Member_ID}:

{Member_ID} -> {Member_ID , Name , Gender , City , Pincode}

Since Pincode is in closure set of Member_ID, and Street is in closure set of Pincode, Street is in closure set of Member_ID.

Since State is not included in the closure of {Member_ID}, the FD {Member_ID} -> State is redundant.

e) Checking if Member_ID -> Pincode is redundant:

Remove Member_ID -> Pincode and check if the remaining FDs can determine Pincode:

Remaining FDs: {Member_ID -> Name, Member_ID -> Gender, Member_ID -> City, Member_ID -> State, Pincode -> City, Pincode -> State}

Closure of {Member_ID}:

{Member_ID} -> {Member_ID , Name , Gender , City , State}

Since Pincode is not included in the closure of {Member_ID}, the FD {Member_ID} -> Name is not redundant.

f) Checking if Pincode -> City is redundant:

Remove Pincode -> City and check if the remaining FDs can determine City:

Remaining FDs: {Member_ID -> Name, Member_ID -> Gender, Member_ID -> City, Member_ID -> State, Pincode -> State}

Closure of {Pincode}:

{Pincode} -> {Pincode, State }

Since City is not included in the closure of {Pincode}, the FD {Pincode} -> City is not redundant.

g) Checking if Pincode -> Street is redundant:

Remove Pincode -> Street and check if the remaining FDs can determine Street:

Remaining FDs: {Member_ID -> Name, Member_ID -> Gender, Member_ID -> City, Member_ID -> State, Pincode -> City}

Closure of {Pincode}:

{Pincode} -> {Pincode , Street}

Since Street is not included in the closure of {Pincode}, the FD {Pincode} -> Street is not redundant.

Final Analysis:

After checking all functional dependencies, there are two redundant FDs, which are Member_ID -> City and Member_ID -> Street.

Primary Key Determination:

The primary key for the “Member” table is Member_ID because it uniquely determines all other attributes in the table.

Minimal set of FDs:

- Member_ID -> Name
- Member_ID -> Gender
- Member_ID -> Pincode
- Pincode -> City
- Pincode -> State

Step 4:

Normalization:

Minimal set of FDs

- Member_ID -> Name
- Member_ID -> Gender
- Member_ID -> Pincode
- Pincode -> City
- Pincode -> State

Candidate Key

- Member_ID

Prime Attributes

- Librarian_ID

Non-Prime Attributes

- Name
- Gender
- Pincode
- City
- State

Step 1: First Normal Form (1NF)

To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

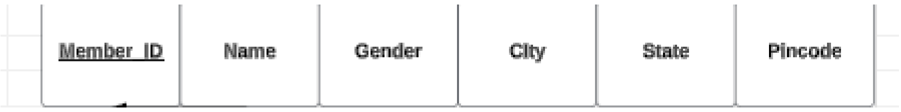
All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

2NF requires that the table is in 1NF and there are **no partial dependencies**, meaning no non-prime attribute is dependent on only a part of any candidate key.

Analyzing for Partial Dependencies

- All non-prime attributes (Name, Gender, Pincode) depend entirely on Member_ID, which is the primary key.
- There are no partial dependencies because the primary key is a single attribute (Member_ID), so the table satisfies 2NF



Member_ID	Name	Gender	City	State	Pincode

Step 3: Third Normal Form (3NF)

3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

Pincode -> City and Pincode -> State are transitive dependencies because City and State are dependent on Pincode, which is not a candidate key but a non-prime attribute.

Decomposition to Achieve 3NF

To achieve 3NF, we decompose the table to remove the transitive dependencies.

Decomposed Tables:

Member Table:

Member_ID (PK)

Name

Gender

Pincode (FK)

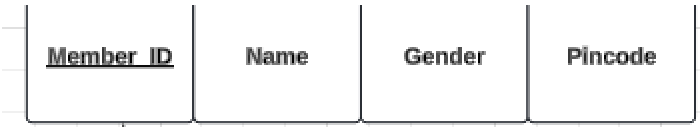
Pincode Table:

Pincode (PK)

City

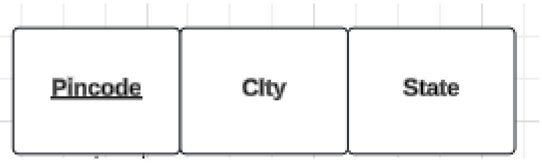
State

TABLE FOR MEMBER:



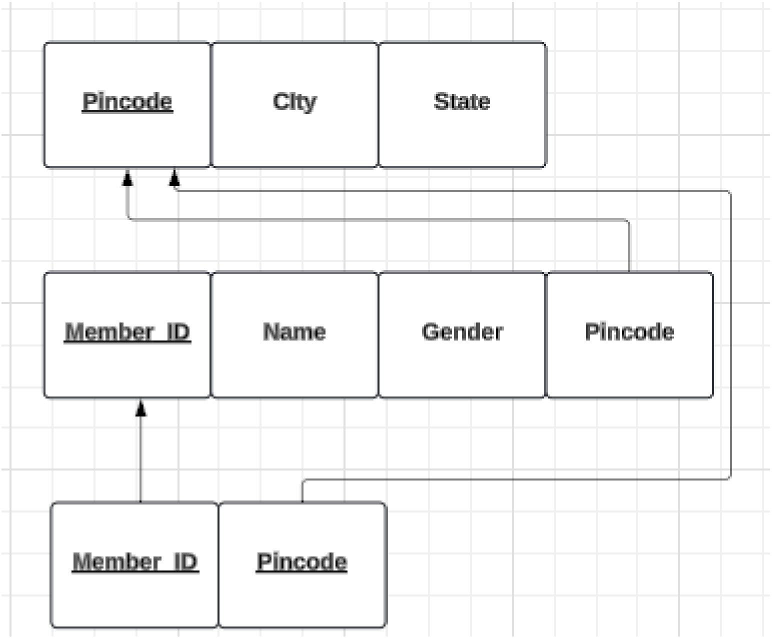
Member_ID	Name	Gender	Pincode

TABLE FOR LOCATION:



Pincode	City	State

TABLE FOR 3NF:



Pincode	City	State

Member_ID	Name	Gender	Pincode

Member_ID	Pincode

FINAL ANALYSIS:

By decomposing the original table into **Member** and **Pincode** tables, we achieve 3NF:

- **1NF:** Attributes are atomic.
- **2NF:** No partial dependencies because each non-prime attribute is fully dependent on the primary key.

- **3NF:** No transitive dependencies because each non-prime attribute is directly dependent on the primary key.

Primary Keys and Foreign Keys

Member Table:

- Primary Key (PK): Member_ID
- Foreign Key (FK): Pincode (references Pincode table)

Pincode Table:

- Primary Key (PK): Pincode

This decomposition ensures that:

1NF: Attributes are atomic.

2NF: No partial dependencies.

3NF: No transitive dependencies.

Transactions:

Attributes:

- Transaction_ID
- Member_ID
- Librarian_ID
- Title
- ISBN
- Author_ID
- Publisher_ID
- Genre_ID
- Format
- Price
- Issue_Date

- Due_Date
- Return_Date
- Penalty
- Publication_Year

Functional Dependencies:

- Transaction_ID -> Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty
- ISBN -> Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year
- Title -> Author_ID , Publisher_ID , Genre , Format , Price , Publication_Year

Step 1:

Decomposition:

- Transaction_ID -> Member_ID
- Transaction_ID -> Librarian_ID
- Transaction_ID -> Title
- Transaction_ID -> ISBN
- Transaction_ID -> Author_ID
- Transaction_ID -> Publisher_ID
- Transaction_ID -> Genre_ID
- Transaction_ID -> Format
- Transaction_ID -> Price
- Transaction_ID -> Issue_Date
- Transaction_ID -> Due_Date
- Transaction_ID -> Return_Date
- Transaction_ID -> Penalty
- ISBN -> Title
- ISBN -> Author_ID
- ISBN -> Publisher_ID
- ISBN -> Genre
- ISBN -> Format
- ISBN -> Price
- ISBN -> Publication_Year
- Title -> Author_ID
- Title -> Publisher_ID
- Title -> Genre
- Title -> Format
- Title -> Price
- Title -> Publication_Year

Step 2:

Checking for extraneous attributes:

No attributes to check as each FD is already atomic.

Step 3:

Checking for redundancies:

We need to check if any of the decomposed functional dependencies can be derived from the others, thus indicating redundancy.

a) **Checking if Transaction_ID -> Member_ID is redundant**:

Remove Transaction_ID -> Member_ID and check if the remaining FDs can determine Member_ID:

Remaining FDs: {Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID , Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since Member_ID is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Member_ID is not redundant.

b) **Checking if Transaction_ID -> Librarian_ID is redundant:**

Remove Transaction_ID -> Librarian_ID and check if the remaining FDs can determine Librarian_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID , Member_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since Librarian_ID is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Librarian_ID is not redundant.

c) Checking if Transaction_ID -> Title is redundant:

Remove Transaction_ID -> Title and check if the remaining FDs can determine Title:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> { Transaction_ID , Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is in closure set of Transaction_ID, and Title is in closure set of ISBN, Title is in closure set of Transaction_ID.

Since Title is included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Title is redundant.

d) Checking if Transaction_ID -> ISBN is redundant:

Remove Transaction_ID -> ISBN and check if the remaining FDs can determine ISBN:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID ,Member_ID, Librarian_ID, Title, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> ISBN is not redundant.

e) Checking if Transaction_ID -> Author_ID is redundant:

Remove Transaction_ID -> Author_ID and check if the remaining FDs can determine Author_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID , Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is in closure set of Transaction_ID, and Author_ID is in closure set of ISBN, Author_ID is in closure set of Transaction_ID.

Since Author_ID is included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Author_ID is redundant.

f) Checking if Transaction_ID -> Publisher_ID is redundant:

Remove Transaction_ID -> Publisher_ID and check if the remaining FDs can determine Publisher_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID , Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is in closure set of Transaction_ID, and Publisher_ID is in closure set of ISBN, Publisher_ID is in closure set of Transaction_ID.

Since Publisher_ID is included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Publisher_ID is redundant.

g) Checking if Transaction_ID -> Genre_ID is redundant:

Remove Transaction_ID -> Genre_ID and check if the remaining FDs can determine Genre_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN ->

Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is in closure set of Transaction_ID, and Genre_ID is in closure set of ISBN, Genre_ID is in closure set of Transaction_ID.

Since Genre_ID is included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Genre_ID is redundant.

h) Checking if Transaction_ID -> Format is redundant:

Remove Transaction_ID -> Format and check if the remaining FDs can determine Format:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is in closure set of Transaction_ID, and Format is in closure set of ISBN, Format is in closure set of Transaction_ID.

Since Format is included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Format is redundant.

i) Checking if Transaction_ID -> Price is redundant:

Remove Transaction_ID -> Price and check if the remaining FDs can determine Price:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> {Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since ISBN is in closure set of Transaction_ID, and Price is in closure set of ISBN, Price is in closure set of Transaction_ID.

Since Price is included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Price is redundant.

j) Checking if Transaction_ID -> Issue_Date is redundant:

Remove Transaction_ID -> Issue_Date and check if the remaining FDs can determine Issue_Date:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> { Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Due_Date, Return_Date, Penalty}

Since Issue_Date is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Issue_Date is not redundant.

k) Checking if Transaction_ID -> Due_Date is redundant:

Remove Transaction_ID -> Due_Date and check if the remaining FDs can determine Due_Date:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> { Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Return_Date, Penalty}

Since Due_Date is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Due_Date is not redundant.

l) Checking if Transaction_ID -> Return_Date is redundant:

Remove Transaction_ID -> Return_Date and check if the remaining FDs can determine Return_Date:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> { Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since Return_Date is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Return_Date is not redundant.

m) Checking if Transaction_ID -> Penalty is redundant:

Remove Transaction_ID -> Penalty and check if the remaining FDs can determine Penalty:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Transaction_ID}:

{Transaction_ID} -> { Transaction_ID, Member_ID, Librarian_ID, Title, ISBN, Author_ID, Publisher_ID, Genre_ID, Format, Price, Issue_Date, Due_Date, Return_Date, Penalty}

Since Penalty is not included in the closure of {Transaction_ID}, the FD {Transaction_ID} -> Penalty is not redundant.

n) Checking if ISBN -> Title is redundant:

Remove ISBN -> Title and check if the remaining FDs can determine Title:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is not included in the closure of {ISBN}, the FD {ISBN} -> Title is not redundant.

o) Checking if ISBN -> Author_ID is redundant:

Remove ISBN -> Author_ID and check if the remaining FDs can determine Author_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is in closure set of ISBN, and Author_ID is in closure set of Title, Author_ID is in closure set of ISBN.

Since Author_ID is included in the closure of {ISBN}, the FD {ISBN} -> Author_ID is redundant.

p) Checking if ISBN -> Publisher_ID is redundant:

Remove ISBN -> Publisher_ID and check if the remaining FDs can determine Publisher_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is in closure set of ISBN, and Publisher_ID is in closure set of Title, Publisher_ID is in closure set of ISBN.

Since Publisher_ID is included in the closure of {ISBN}, the FD {ISBN} -> Publisher_ID is redundant.

q) Checking if ISBN -> Genre is redundant:

Remove ISBN -> Genre and check if the remaining FDs can determine Genre:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is in closure set of ISBN, and Genre is in closure set of Title, Genre is in closure set of ISBN.

Since Genre is included in the closure of {ISBN}, the FD {ISBN} -> Genre is redundant.

r) Checking if ISBN -> Format is redundant:

Remove ISBN -> Format and check if the remaining FDs can determine Format:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is in closure set of ISBN, and Format is in closure set of Title, Format is in closure set of ISBN.

Since Format is included in the closure of {ISBN}, the FD {ISBN} -> Format is redundant.

s) Checking if ISBN -> Price is redundant:

Remove ISBN -> Price and check if the remaining FDs can determine Price:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is in closure set of ISBN, and Price is in closure set of Title, Price is in closure set of ISBN.

Since Price is included in the closure of {ISBN}, the FD {ISBN} -> Price is redundant.

t) Checking if ISBN -> Publication_Year is redundant:

Remove ISBN -> Publication_Year and check if the remaining FDs can determine Publication_Year:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID ->

Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {ISBN}:

{ISBN} -> {ISBN, Title, Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}

Since Title is in closure set of ISBN, and Publication_Year is in closure set of Title, Publication is in closure set of ISBN.

Since Publication_Year is included in the closure of {ISBN}, the FD {ISBN} -> Publication_Year is redundant.

u) Checking if Title -> Author_ID is redundant:

Remove Title -> Author_ID and check if the remaining FDs can determine Author_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Title}:

{Title} -> {Title, Publisher_ID , Genre , Format , Price , Publication_Year}

Since Author_ID is not included in the closure of {Title}, the FD {Title} -> Author_ID is not redundant.

v) Checking if Title -> Publisher_ID is redundant:

Remove Title -> Publisher_ID and check if the remaining FDs can determine Publisher_ID:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Genre, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Title}:

{Title} -> {Title, Author_ID , Genre , Format , Price , Publication_Year}

Since Publisher_ID is not included in the closure of {Title}, the FD {Title} -> Publisher_ID is not redundant.

w) Checking if Title -> Genre is redundant:

Remove Title -> Genre and check if the remaining FDs can determine Genre:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Format, Title -> Price, Title -> Publication_Year}

Closure of {Title}:

{Title} -> {Title, Author_ID , Publisher_ID , Format , Price , Publication_Year}

Since Genre is not included in the closure of {Title}, the FD {Title} -> Genre is not redundant.

x) Checking if Title -> Format is redundant:

Remove Title -> Format and check if the remaining FDs can determine Format:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID ->

Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Price, Title -> Publication_Year}

Closure of {Title}:

{Title} -> {Title, Author_ID , Publisher_ID , Genre , Price , Publication_Year}

Since Format is not included in the closure of {Title}, the FD {Title} -> Format is not redundant.

y) Checking if Title -> Price is redundant:

Remove Title -> Price and check if the remaining FDs can determine Price:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Publication_Year}

Closure of {Title}:

{Title} -> {Title, Author_ID , Publisher_ID , Genre , Format , Publication_Year}

Since Price is not included in the closure of {Title}, the FD {Title} -> Price is not redundant.

z) Checking if Title -> Publication_Year is redundant:

Remove Title -> Publication_Year and check if the remaining FDs can determine Publication_Year:

Remaining FDs: { Transaction_ID -> Member_ID, Transaction_ID -> Librarian_ID, Transaction_ID -> Title, Transaction_ID -> ISBN, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, Transaction_ID -> Issue_Date, Transaction_ID -> Due_Date, Transaction_ID -> Return_Date, Transaction_ID -> Penalty, ISBN -> Title, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, ISBN -> Publication_Year, Title -> Author_ID, Title -> Publisher_ID, Title -> Genre, Title -> Format, Title -> Price}

Closure of {Title}:

{Title} -> {Title, Author_ID , Publisher_ID , Genre , Format , Price}

Since Publication_Year is not included in the closure of {Title}, the FD {Title} -> Publication_Year is not redundant.

Final Analysis:

After checking all functional dependencies, there are twelve redundant FDs. They are Transaction_ID -> Title, Transaction_ID -> Author_ID, Transaction_ID -> Publisher_ID, Transaction_ID -> Genre_ID, Transaction_ID -> Format, Transaction_ID -> Price, ISBN -> Author_ID, ISBN -> Publisher_ID, ISBN -> Genre, ISBN -> Format, ISBN -> Price, and ISBN -> Publication_Year.

Primary Key Determination:

The primary key for the “Transaction” table is Transaction_ID because it uniquely determines all other attributes in the table.

Minimal set of FDs:

- Transaction_ID -> Member_ID
- Transaction_ID -> Librarian_ID
- Transaction_ID -> ISBN
- Transaction_ID -> Issue_Date
- Transaction_ID -> Due_Date
- Transaction_ID -> Return_Date
- Transaction_ID -> Penalty
- ISBN -> Title
- Title -> Author_ID
- Title -> Publisher_ID
- Title -> Genre
- Title -> Format
- Title -> Price
- Title -> Publication_Year

Step 4:

Normalization:

Transactions

- Transaction_ID -> Member_ID
- Transaction_ID -> Librarian_ID
- Transaction_ID -> ISBN
- Transaction_ID -> Issue_Date
- Transaction_ID -> Due_Date
- Transaction_ID -> Return_Date
- Transaction_ID -> Penalty
- ISBN -> Title
- Title -> Author_ID
- Title -> Publisher_ID
- Title -> Genre
- Title -> Format
- Title -> Price
- Title -> Publication_Year

Candidate Key

- Transaction_ID

Prime Attributes

- Transaction_ID

Non-Prime Attributes

- **Member_ID**
- **Librarian_ID**
- **ISBN**
- **Issue_Date**
- **Due_Date**
- **Return_Date**
- **Penalty**
- **Title**

- **Author_ID**
- **Publisher_ID**
- **Genre**
- **Format**
- **Price**
- **Publication_Year**

Step 1: First Normal Form (1NF)

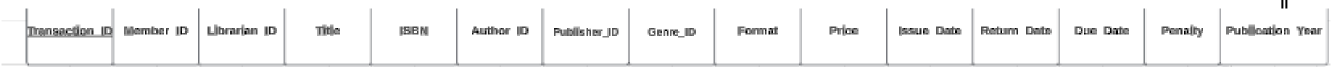
To ensure 1NF, we need to ensure that each attribute contains atomic values and there are no repeating groups or arrays of values.

All attributes in the **FDs are atomic (single-valued)**, and there are no repeating groups. Therefore, the table already **satisfies 1NF.**

Step 2: Second Normal Form (2NF)

- 2NF requires that the table is in 1NF and there are no partial dependencies, meaning no non-prime attribute is dependent on only a part of any candidate key.

- Since the primary key is Transaction_ID and all non-prime attributes are fully dependent on Transaction_ID, there are no partial dependencies. Therefore, the table is in 2NF.



Transaction_ID	Member ID	Librarian ID	Title	ISBN	Author ID	Publisher_ID	Genre_ID	Format	Price	Issue Date	Return Date	Due Date	Penalty	Publication Year

Step 3: Third Normal Form (3NF)

3NF requires that the table is in 2NF and there are **no transitive dependencies**, meaning no non-prime attribute is transitively dependent on any other non-prime attribute.

Analyzing for Transitive Dependencies:

- The dependencies **ISBN -> Title** and **Title -> {Author_ID, Publisher_ID, Genre, Format, Price, Publication_Year}** indicate transitive dependencies through **Title**.
- Since **Title** is a non-prime attribute, the dependencies involving **Title** are transitive.

Decomposition to Achieve 3NF

We decompose the table to remove the transitive dependencies by creating additional tables.

Decomposed Tables:

Transactions Table:

Transaction_ID (PK)

Member_ID

Librarian_ID

ISBN (FK)

Issue_Date

Due_Date

Return_Date

Penalty

Book Table:

ISBN (PK)

Title

Book_Details Table:

Title (PK)

Author_ID

Publisher_ID

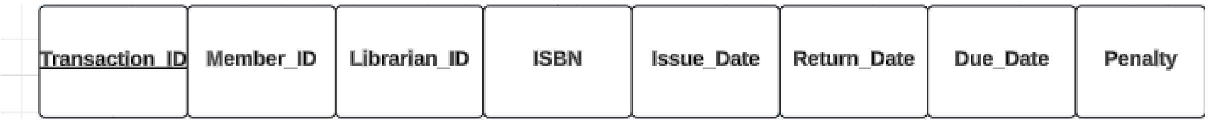
Genre

Format

Price

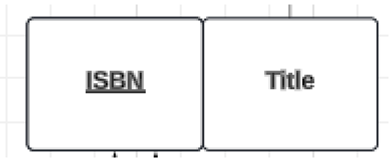
Publication_Year

TABLE FOR TRANSACTIONS:



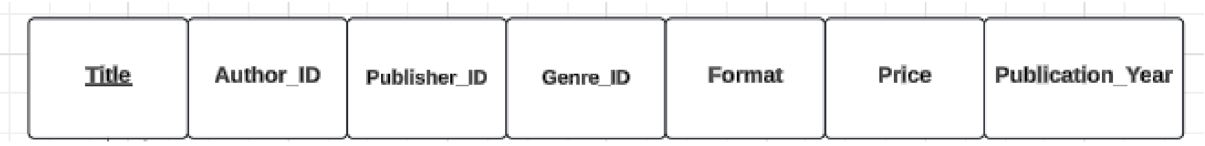
Transaction_ID	Member_ID	Librarian_ID	ISBN	Issue_Date	Return_Date	Due_Date	Penalty

TABLE FOR BOOKS:



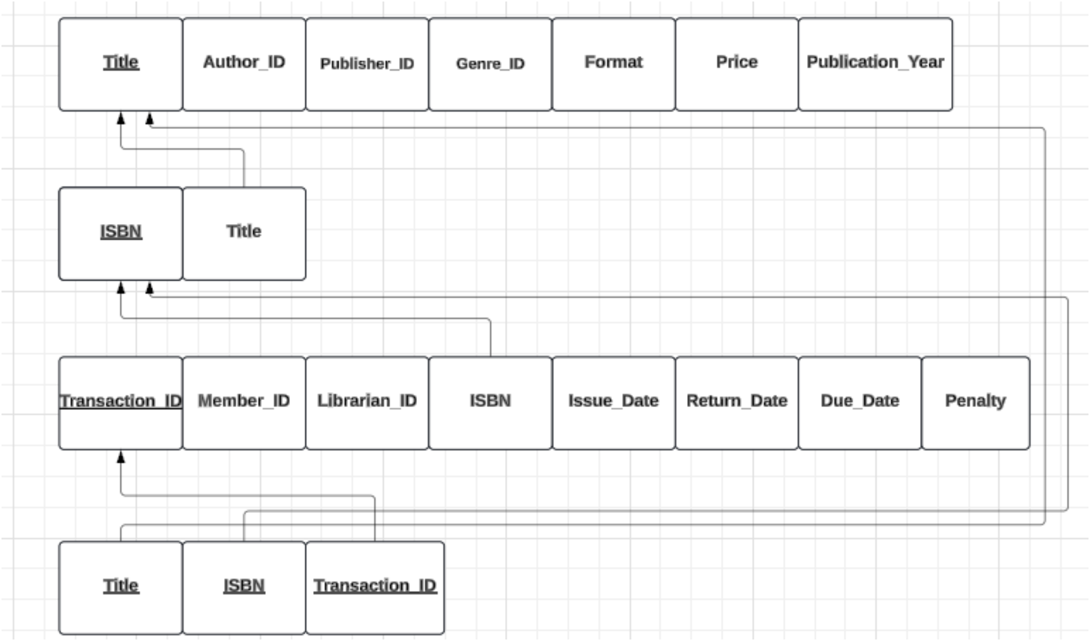
ISBN	Title

TABLE FOR BOOK_DETAILS:



Title	Author_ID	Publisher_ID	Genre_ID	Format	Price	Publication_Year

TABLE FOR 3NF:



Primary Keys and Foreign Keys

Transactions Table:

- Primary Key (PK): Transaction_ID
- Foreign Key (FK): ISBN

Book Table:

- Primary Key (PK): ISBN

Book_Details Table:

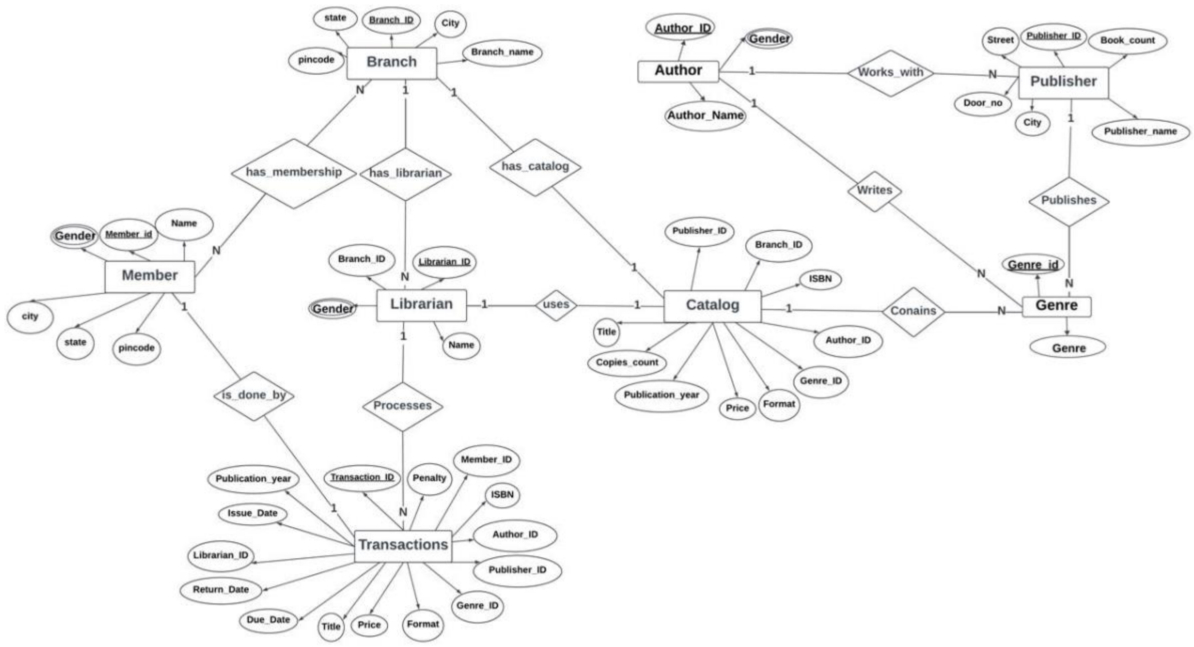
- Primary Key (PK): Title

FINAL ANALYSIS:

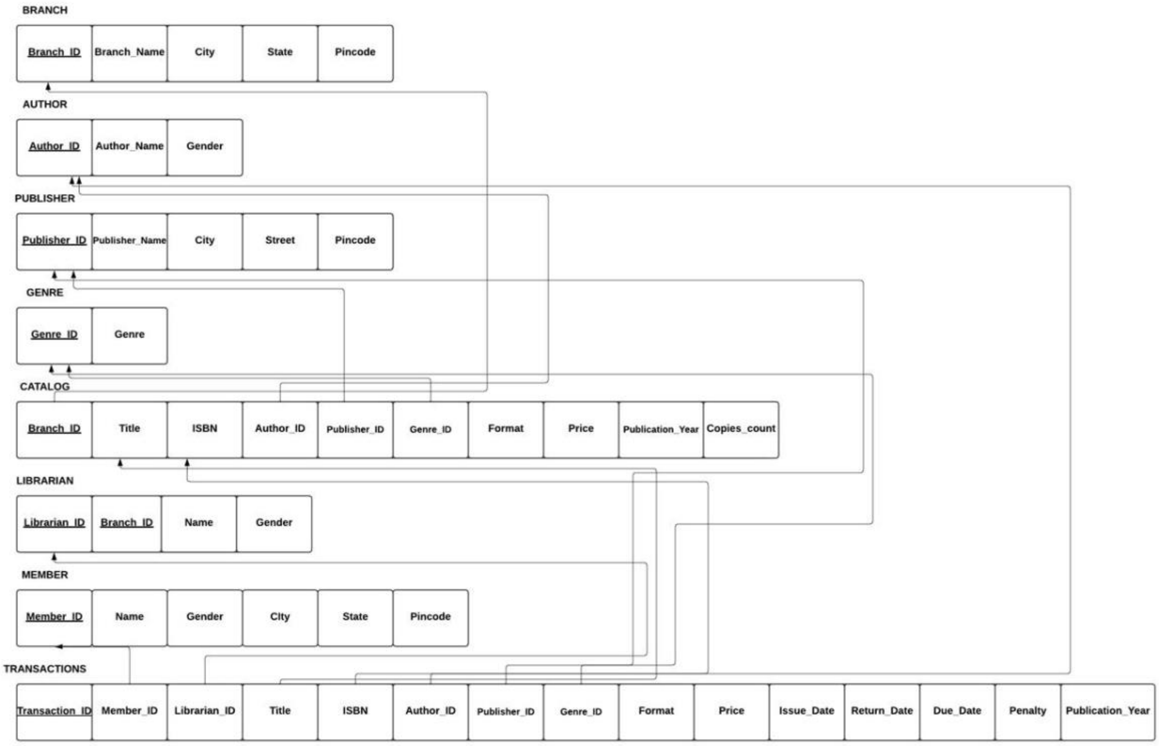
By decomposing the original table into Transactions, Book, and Title_Details tables, we achieve 3NF:

- 1NF: Attributes are atomic.
- 2NF: No partial dependencies.
- 3NF: No transitive dependencies.

ENTITY RELATIONSHIP DIAGRAM (ER DIAGRAM)



SCHEMA DIAGRAM: BEFORE NORMALIZATION



BRANCH

Branch_ID	Branch_Name	City	State	Pincode

AUTHOR

Author_ID	Author_Name	Gender

PUBLISHER

Publisher_ID	Publisher_Name	City	Street	Pincode

GENRE

Genre_ID	Genre

CATALOG

Branch_ID	Title	ISBN	Author_ID	Publisher_ID	Genre_ID	Format	Price	Publication_Year	Copies_count

LIBRARIAN

Librarian_ID	Branch_ID	Name	Gender

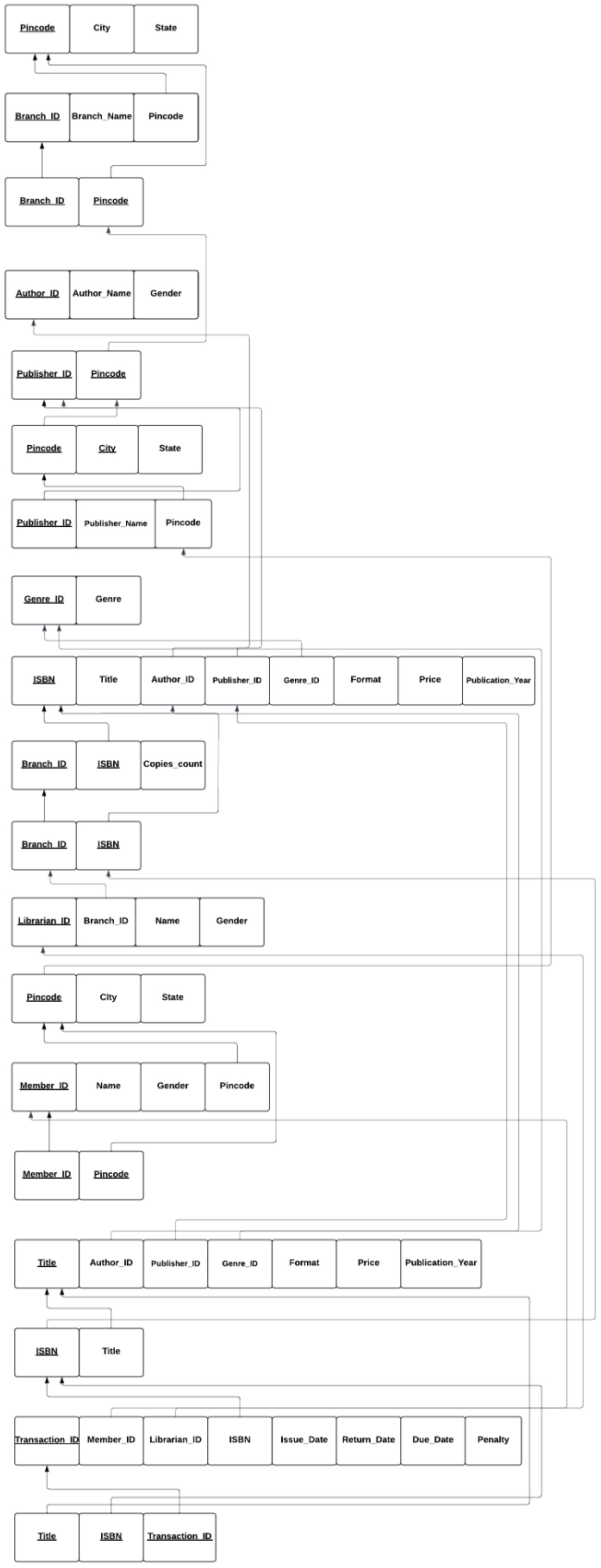
MEMBER

Member_ID	Name	Gender	City	State	Pincode

TRANSACTIONS

Transaction_ID	Member_ID	Librarian_ID	Title	ISBN	Author_ID	Publisher_ID	Genre_ID	Format	Price	Issue_Date	Return_Date	Due_Date	Penalty	Publication_Year

SCHEMA DIAGRAM: AFTER NORMALIZATION


Pincode | City | State
Branch_ID | Branch_Name | Pincode
Branch_ID | Pincode
Author_ID | Author_Name | Gender
Publisher_ID | Pincode
Pincode | City | State
Publisher_ID | Publisher_Name | Pincode
Genre_ID | Genre
ISBN | Title | Author_ID | Publisher_ID | Genre_ID | Format | Price | Publication_Year
Branch_ID | ISBN | Copies_count
Branch_ID | ISBN
Librarian_ID | Branch_ID | Name | Gender
Pincode | City | State
Member_ID | Name | Gender | Pincode
Member_ID | Pincode
Title | Author_ID | Publisher_ID | Genre_ID | Format | Price | Publication_Year
ISBN | Title
Transaction_ID | Member_ID | Librarian_ID | ISBN | Issue_Date | Return_Date | Due_Date | Penalty
Title | ISBN | Transaction_ID

NETBEANS CODE:

CODE FOR AUTHOR:

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package com.mycompany.lms;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
/**
 *
 * @author RRR
 */
public class author extends javax.swing.JFrame {

    /**
     * Creates new form author
     */
    Connection con;
    Statement st;
    PreparedStatement ps;
    ResultSet rs;
```

```
public author() {
    initComponents();
    try{
        Class.forName("oracle.jdbc.OracleDriver");
        JOptionPane.showMessageDialog(null,"Driver Loaded");

        try{
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
            JOptionPane.showMessageDialog(null,"Connected to Database");
        }
        catch(SQLException ex){
            Logger.getLogger(author.class.getName()).log(Level.SEVERE,null,ex);
        }
    }
    catch(ClassNotFoundException ex){
        Logger.getLogger(author.class.getName()).log(Level.SEVERE,null,ex);
    }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
```

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    authorid = new javax.swing.JTextField();
    jLabel3 = new javax.swing.JLabel();
    name = new javax.swing.JTextField();
    jLabel4 = new javax.swing.JLabel();
    gender = new javax.swing.JTextField();
    insert = new javax.swing.JButton();
    delete = new javax.swing.JButton();
    search = new javax.swing.JButton();
    update = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jLabel1.setText("AUTHOR DETAILS");

    jLabel2.setText("authorid");

    jLabel3.setText("name");

    jLabel4.setText("gender");
```

```
insert.setText("insert");
insert.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        insertActionPerformed(evt);
    }
});

delete.setText("delete");
delete.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        deleteActionPerformed(evt);
    }
});

search.setText("search");
search.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        searchActionPerformed(evt);
    }
});

update.setText("update");
update.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        updateActionPerformed(evt);
```

```
            }

        });


        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

        getContentPane().setLayout(layout);

        layout.setHorizontalGroup(

            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(layout.createSequentialGroup()

                .addGap(63, 63, 63)

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                    .addComponent(jLabel2)

                    .addComponent(jLabel3)

                    .addComponent(jLabel4))

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)

                    .addComponent(authorid)

                    .addComponent(name)

                    .addComponent(gender, javax.swing.GroupLayout.DEFAULT_SIZE, 130,
Short.MAX_VALUE))

                .addGap(22, 22, 22))

            .addGroup(layout.createSequentialGroup()

                .addGap(44, 44, 44)

                .addComponent(insert)

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                    .addGroup(layout.createSequentialGroup()
            .addGap(38, 38, 38)

            .addComponent(jLabel1))

          .addGroup(layout.createSequentialGroup()

            .addGap(18, 18, 18)

            .addComponent(delete)

            .addGap(18, 18, 18)

            .addComponent(search)

            .addGap(18, 18, 18)

            .addComponent(update)))

        .addContainerGap(14, Short.MAX_VALUE))

    );

    layout.setVerticalGroup(

      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

      .addGroup(layout.createSequentialGroup()

        .addGap(18, 18, 18)

        .addComponent(jLabel1)

        .addGap(32, 32, 32)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

          .addComponent(jLabel2)

          .addComponent(authorid, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(34, 34, 34)

        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

          .addComponent(jLabel3)

          .addComponent(name, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                .addGap(31, 31, 31)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jLabel4)
                    .addComponent(gender, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(36, 36, 36)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(insert)
                    .addComponent(delete)
                    .addComponent(search)
                    .addComponent(update))
                .addContainerGap(44, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    private void insertActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String sql="insert into author(authorid,name,gender) values(?,?,?)";
        try{
            ps=con.prepareStatement(sql);

            ps.setString(1,authorid.getText());
            ps.setString(2,name.getText());
            ps.setString(3,gender.getText());
            ps.executeUpdate();
            JOptionPane.showMessageDialog(null,"Inserted Successfully");
        }
        catch(SQLException ex){
            Logger.getLogger(author.class.getName()).log(Level.SEVERE,null,ex);
        }
    }

    private void deleteActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String sql = "delete from author where authorid=?";
        try {
            ps = con.prepareStatement(sql);
            ps.setString(1, authorid.getText());
            int rowsAffected = ps.executeUpdate();
            if (rowsAffected > 0) {
                JOptionPane.showMessageDialog(null, "Successfully deleted!");
            } else {
                JOptionPane.showMessageDialog(null, "No record found with the provided ID.");
            }
        } catch (SQLException ex) {
            Logger.getLogger(author.class.getName()).log(Level.SEVERE, null, ex);
        }
```

```
}

private void searchActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String sql="select * from author where authorid='"+authorid.getText()+"'";
    try
    {
        st=con.createStatement();
        rs=st.executeQuery(sql);
        if(rs.next()){
            name.setText(rs.getString(2));
        }
        JOptionPane.showMessageDialog(null, "Searched");
    }
    catch(SQLException ex){
            Logger.getLogger(author.class.getName()).log(Level.SEVERE,null,ex);
     }
}

private void updateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String sql = "update author set name=?,gender=? where authorid=?";
    try{
        ps=con.prepareStatement(sql);
        ps.setString(1,name.getText());
```

```
            ps.setString(2,gender.getText());

            ps.setString(3,authorid.getText());

            ps.executeUpdate();



            JOptionPane.showMessageDialog(null,"successfully Updated");

        }

        catch(SQLException ex){

            Logger.getLogger(author.class.getName()).log(Level.SEVERE,null,ex);

        }

    }



    /**

     * @param args the command line arguments

     */

    public static void main(String args[]) {

        /* Set the Nimbus look and feel */

        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

         */

        try {

            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {

                if ("Nimbus".equals(info.getName())) {

                    javax.swing.UIManager.setLookAndFeel(info.getClassName());

                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(author.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(author.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(author.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(author.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex);
        }
        //</editor-fold>


        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new author().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JTextField authorid;
    private javax.swing.JButton delete;
    private javax.swing.JTextField gender;
    private javax.swing.JButton insert;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JTextField name;
    private javax.swing.JButton search;
    private javax.swing.JButton update;
    // End of variables declaration
}
```

CODE FOR GENRE:

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package com.mycompany.lms;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
/**
 *
 * @author RRR
 */
public class genre extends javax.swing.JFrame {

    /**
     * Creates new form genre
 */
Connection con;
Statement st;
PreparedStatement ps;
ResultSet rs;
public genre() {
  initComponents();
  try{
    Class.forName("oracle.jdbc.OracleDriver");
    JOptionPane.showMessageDialog(null,"Driver Loaded");

    try{
      con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
      JOptionPane.showMessageDialog(null,"Connected to Database");
    }
    catch(SQLException ex){
      Logger.getLogger(genre.class.getName()).log(Level.SEVERE,null,ex);
    }
  }
  catch(ClassNotFoundException ex){
    Logger.getLogger(genre.class.getName()).log(Level.SEVERE,null,ex);
  }
}

/**
```

```
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        genreid = new javax.swing.JTextField();
        jLabel2 = new javax.swing.JLabel();
        genre = new javax.swing.JTextField();
        insert = new javax.swing.JButton();
        delete = new javax.swing.JButton();
        search = new javax.swing.JButton();
        jLabel3 = new javax.swing.JLabel();
        update = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jLabel1.setText("genreid");

        jLabel2.setText("genre");

        insert.setText("insert");
```

```
insert.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        insertActionPerformed(evt);
    }
});

delete.setText("delete");
delete.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        deleteActionPerformed(evt);
    }
});

search.setText("search");
search.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        searchActionPerformed(evt);
    }
});

jLabel3.setText("GENRE DETAILS");

update.setText("update");
update.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        updateActionPerformed(evt);
      }
    });

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
        .addGap(16, 16, 16)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jLabel1)
            .addComponent(jLabel2))
          .addGroup(layout.createSequentialGroup()
            .addComponent(insert)
            .addGap(10, 10, 10)))
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
          .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
              .addComponent(genre, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 116, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(genreid, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.PREFERRED_SIZE, 116,
javax.swing.GroupLayout.PREFERRED_SIZE))
```

```
                        .addGap(70, 70, 70))
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(delete)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(search)
                        .addGap(18, 18, 18)
                        .addComponent(update)
                        .addGap(0, 59, Short.MAX_VALUE))))
            .addGroup(layout.createSequentialGroup()
                .addGap(145, 145, 145)
                .addComponent(jLabel3)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(19, 19, 19)
                .addComponent(jLabel3)
                .addGap(18, 18, 18)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(genreid, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(52, 52, 52)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(genre, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 68,
    Short.MAX_VALUE)

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

                    .addComponent(insert)

                    .addComponent(delete)

                    .addComponent(search)

                    .addComponent(update))

                .addGap(60, 60, 60))

        );


        pack();

      }// </editor-fold>


      private void insertActionPerformed(java.awt.event.ActionEvent evt) {

        // TODO add your handling code here:

        String sql="insert into genre(genreid,genre) values(?,?)";

        try{

            ps=con.prepareStatement(sql);


            ps.setString(1,genreid.getText());

            ps.setString(2,genre.getText());
```

```
        ps.executeUpdate();

        JOptionPane.showMessageDialog(null,"Inserted Successfully");

    }

    catch(SQLException ex){

        Logger.getLogger(genre.class.getName()).log(Level.SEVERE,null,ex);

    }

}


private void deleteActionPerformed(java.awt.event.ActionEvent evt) {

    // TODO add your handling code here:

    String sql = "delete from genre where genreid=?";

    try {

        ps = con.prepareStatement(sql);

        ps.setString(1, genreid.getText());

        int rowsAffected = ps.executeUpdate();

        if (rowsAffected > 0) {

            JOptionPane.showMessageDialog(null, "Successfully deleted!");

        } else {

            JOptionPane.showMessageDialog(null, "No record found with the provided ID.");

        }

    } catch (SQLException ex) {

        Logger.getLogger(genre.class.getName()).log(Level.SEVERE, null, ex);

    }

}
```

```
private void searchActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String sql="select * from genre where genreid='"+genreid.getText()+"'";
    try
    {
        st=con.createStatement();
        rs=st.executeQuery(sql);
        if(rs.next()){
            genre.setText(rs.getString(2));
        }
        JOptionPane.showMessageDialog(null, "Searched");
    }
    catch(SQLException ex){
            Logger.getLogger(genre.class.getName()).log(Level.SEVERE,null,ex);
     }
}

private void updateActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String sql = "update genre set genre=? where genreid=?";
    try{
        ps = con.prepareStatement(sql);
        ps.setString(1,genreid.getText());
        ps.setString(2,genre.getText());
        ps.executeUpdate();
```

```
            JOptionPane.showMessageDialog(null,"successfully Updated");
        }
        catch(SQLException ex){
            Logger.getLogger(genre.class.getName()).log(Level.SEVERE,null,ex);
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
```

```
java.util.logging.Logger.getLogger(genre.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);

        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(genre.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);

        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(genre.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(genre.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);

        }

        //</editor-fold>



        /* Create and display the form */

        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {

                new genre().setVisible(true);

            }

        });

    }


    // Variables declaration - do not modify

    private javax.swing.JButton delete;

    private javax.swing.JTextField genre;
    private javax.swing.JTextField genreid;

    private javax.swing.JButton insert;

    private javax.swing.JLabel jLabel1;

    private javax.swing.JLabel jLabel2;

    private javax.swing.JLabel jLabel3;

    private javax.swing.JButton search;

    private javax.swing.JButton update;

    // End of variables declaration

}
```

CODE FOR LIBRARIAN:

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/GUIForms/JFrame.java to edit this template
 */
package com.mycompany.lms;
import java.sql.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;


/**
```

```
 *
 * @author RRR
 */
public class librarian extends javax.swing.JFrame {
    Connection con;
    Statement st;
    PreparedStatement ps;
    ResultSet rs;

    /**
     * Creates new form librarian
     */
    public librarian() {
        initComponents();
        try{
            Class.forName("oracle.jdbc.OracleDriver");
            JOptionPane.showMessageDialog(null,"Driver Loaded");

            try{
                con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
                JOptionPane.showMessageDialog(null,"Connected to Database");
            }
            catch(SQLException ex){
                Logger.getLogger(librarian.class.getName()).log(Level.SEVERE,null,ex);
            }
        }
        catch(ClassNotFoundException ex){
            Logger.getLogger(librarian.class.getName()).log(Level.SEVERE,null,ex);
        }
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        libid = new javax.swing.JTextField();
        insert = new javax.swing.JButton();
        jLabel2 = new javax.swing.JLabel();
        name = new javax.swing.JTextField();
        jLabel3 = new javax.swing.JLabel();
        gender = new javax.swing.JTextField();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        branchid = new javax.swing.JTextField();
```

```
delete = new javax.swing.JButton();
search = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jLabel1.setText("libid");

insert.setText("insert");
insert.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        insertActionPerformed(evt);
    }
});

jLabel2.setText("name");

jLabel3.setText("gender");

jLabel4.setText("Librarian Details");

jLabel5.setText("branchid");

branchid.setHorizontalAlignment(javax.swing.JTextField.LEFT);

delete.setText("delete");
```

```
        delete.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                deleteActionPerformed(evt);
            }
        });

        search.setText("search");
        search.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                searchActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
                .addContainerGap(25, Short.MAX_VALUE)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(insert, javax.swing.GroupLayout.Alignment.TRAILING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jLabel2)
                        .addComponent(jLabel3)
                        .addComponent(jLabel5)
```

```
              .addComponent(jLabel1)))
          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
              .addGap(127, 127, 127)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(libid, javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(branchid, javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(name, javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addComponent(gender, javax.swing.GroupLayout.PREFERRED_SIZE, 123,
javax.swing.GroupLayout.PREFERRED_SIZE)))
            .addGroup(layout.createSequentialGroup()
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
              .addComponent(delete)
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
              .addComponent(search)))
          .addGap(53, 53, 53))
        .addGroup(layout.createSequentialGroup()
          .addGap(139, 139, 139)
          .addComponent(jLabel4)
          .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
          .addGap(24, 24, 24)

          .addComponent(jLabel4)

          .addGap(27, 27, 27)

          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

            .addComponent(jLabel1)

            .addComponent(libid, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 21,
Short.MAX_VALUE)

          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

            .addComponent(branchid, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

            .addComponent(jLabel5))

          .addGap(18, 18, 18)

          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

            .addComponent(jLabel2)

            .addComponent(name, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

          .addGap(18, 18, 18)

          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

            .addComponent(jLabel3)

            .addComponent(gender, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

          .addGap(31, 31, 31)

          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

            .addComponent(insert)

            .addComponent(delete)

            .addComponent(search))
          .addGap(34, 34, 34))
  );

  pack();
}// </editor-fold>

private void insertActionPerformed(java.awt.event.ActionEvent evt) {

  // TODO add your handling code here:
  String sql="insert into librarian(libid,branchid,name,gender) values(?,?,?,?)";
  try{
      ps=con.prepareStatement(sql);

      ps.setString(1,libid.getText());
      ps.setString(2,branchid.getText());
      ps.setString(3,name.getText());
      ps.setString(4,gender.getText());



      ps.executeUpdate();
      JOptionPane.showMessageDialog(null,"Inserted Successfully");
  }
  catch(SQLException ex){
      Logger.getLogger(librarian.class.getName()).log(Level.SEVERE,null,ex);
  }
```

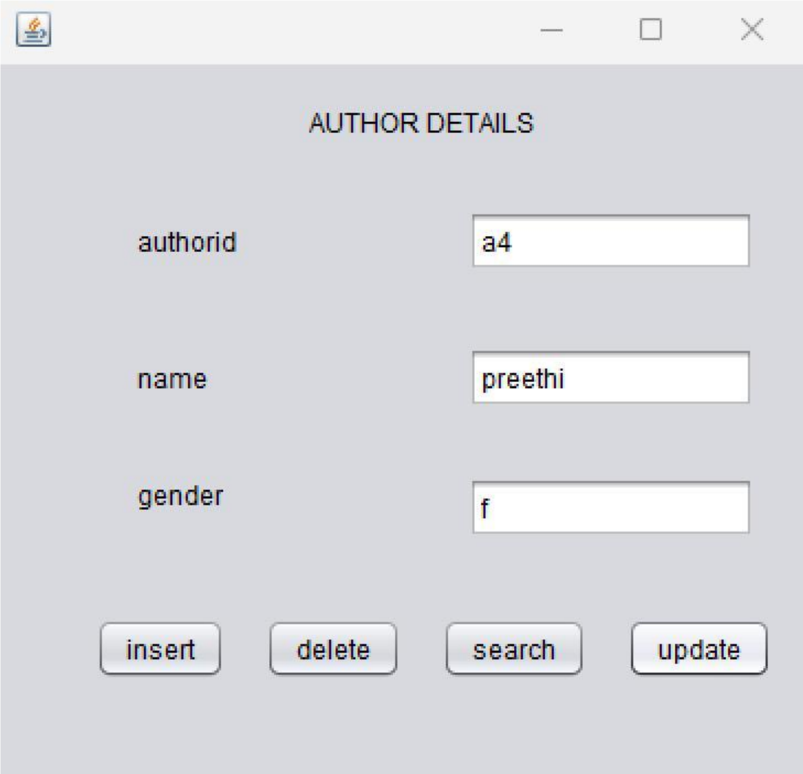
```
}


private void deleteActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String sql = "delete from librarian where libid=?";
    try {
        ps = con.prepareStatement(sql);
        ps.setString(1, libid.getText());
        int rowsAffected = ps.executeUpdate();
        if (rowsAffected > 0) {
            JOptionPane.showMessageDialog(null, "Successfully deleted!");
        } else {
            JOptionPane.showMessageDialog(null, "No record found with the provided ID.");
        }
    } catch (SQLException ex) {
        Logger.getLogger(librarian.class.getName()).log(Level.SEVERE, null, ex);
    }

}


private void searchActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String sql="select * from librarian where libid='"+libid.getText()+"'";
    try
    {
            st=con.createStatement();

            rs=st.executeQuery(sql);

            if(rs.next()){

                name.setText(rs.getString(3));

            }

            JOptionPane.showMessageDialog(null, "Searched");

        }

        catch(SQLException ex){

                Logger.getLogger(librarian.class.getName()).log(Level.SEVERE,null,ex);

         }


    }


    /**

     * @param args the command line arguments

     */

    public static void main(String args[]) {

        /* Set the Nimbus look and feel */

        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

         */

        try {

            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {

                if ("Nimbus".equals(info.getName())) {
```

```
                javax.swing.UIManager.setLookAndFeel(info.getClassName());
                break;
            }
        }
    } catch (ClassNotFoundException | InstantiationException | IllegalAccessException | 
javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(librarian.class.getName()).log(java.util.logging.Level.SEVERE, 
null, ex);
    }
    //</editor-fold>

    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(() -> {
        new librarian().setVisible(true);
    });
}

// Variables declaration - do not modify
private javax.swing.JTextField branchid;
private javax.swing.JButton delete;
private javax.swing.JTextField gender;
private javax.swing.JButton insert;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
```

```
    private javax.swing.JLabel jLabel3;

    private javax.swing.JLabel jLabel4;

    private javax.swing.JLabel jLabel5;

    private javax.swing.JTextField libid;

    private javax.swing.JTextField name;

    private javax.swing.JButton search;

    // End of variables declaration

}
```

NETBEANS OTPUT:

INSERATION:

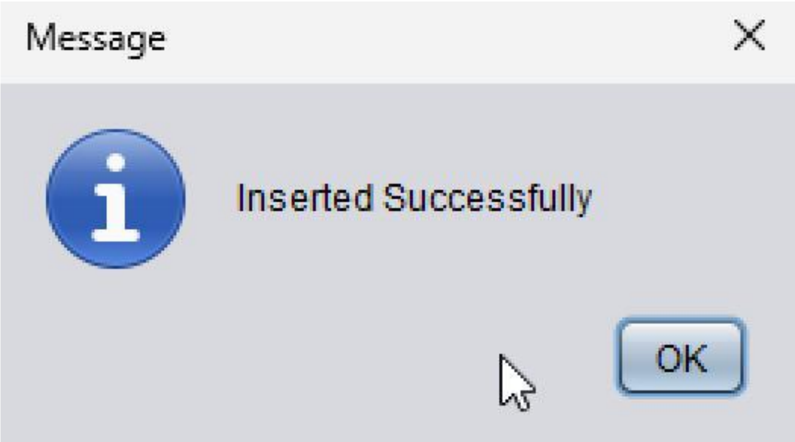


AUTHOR DETAILS

authorid: a4

name: preethi

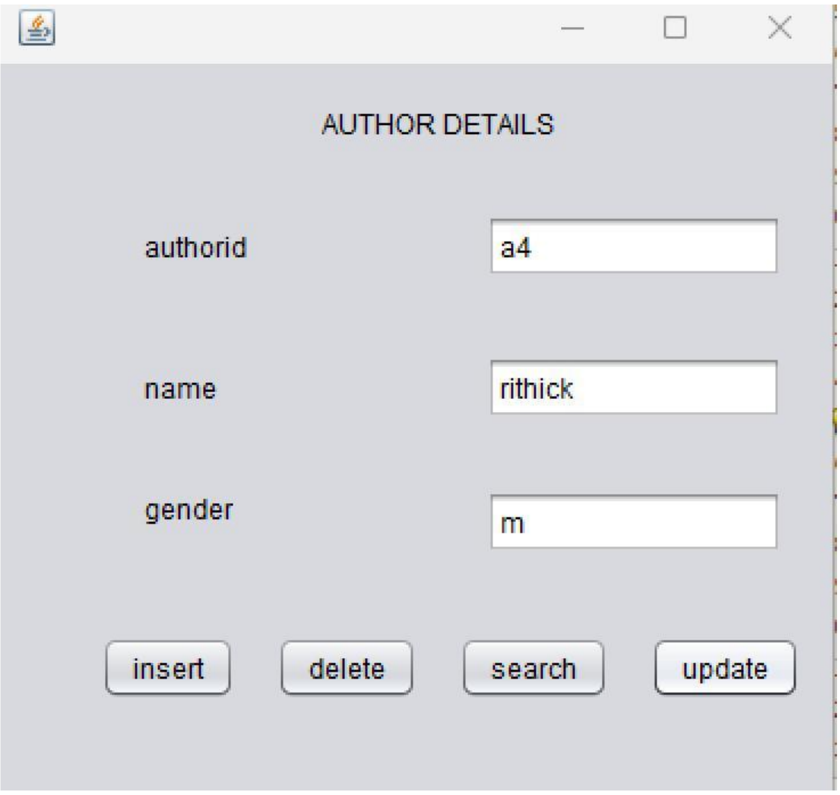
gender: f



Message

Inserted Successfully

UPDATE:

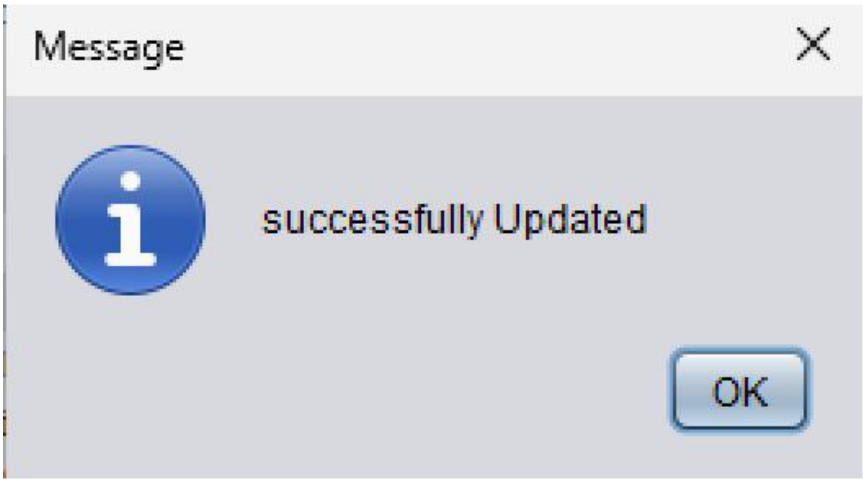


AUTHOR DETAILS

authorid: a4

name: rithick

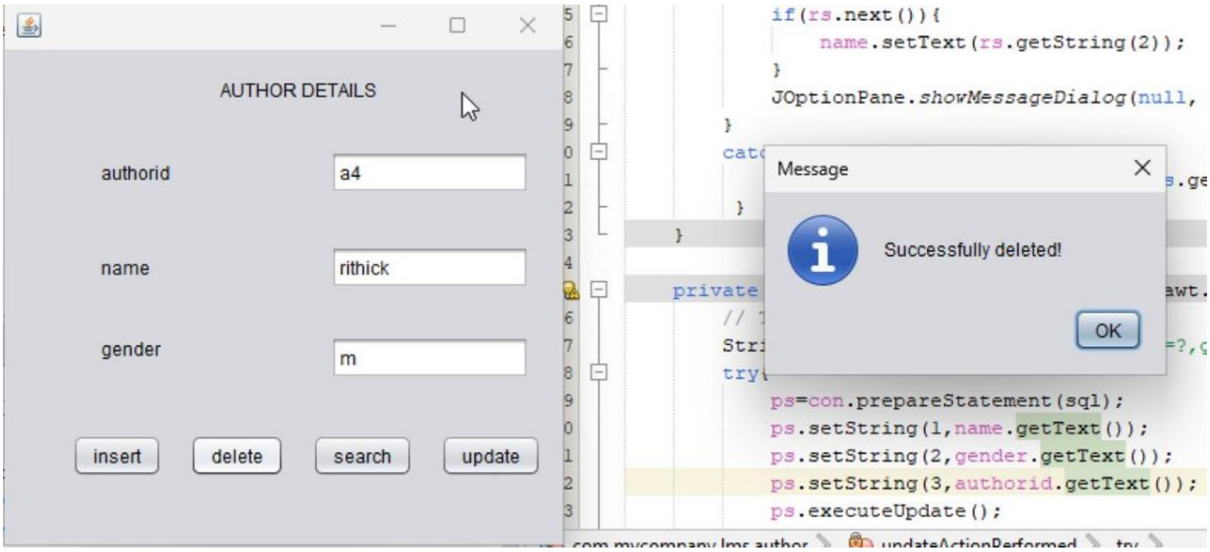
gender: m



Message

successfully Updated

DELETE:



AUTHOR DETAILS

authorid: a4

name: rithick

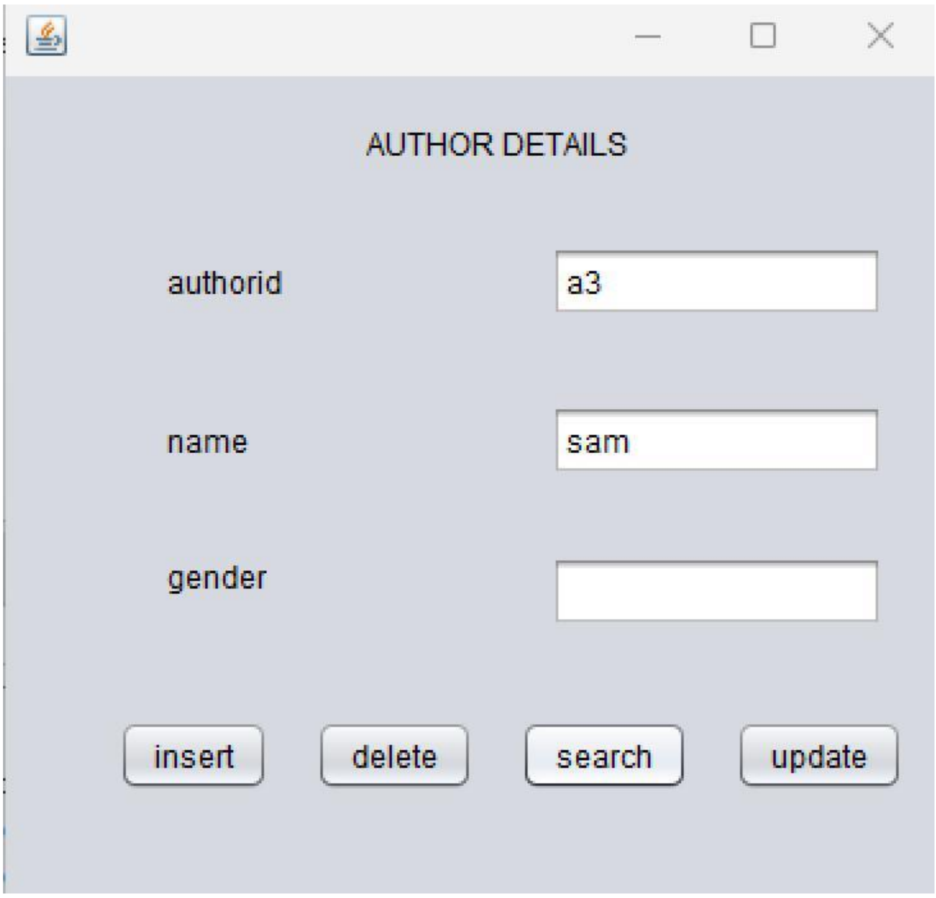
gender: m

```
            if(rs.next()){
                name.setText(rs.getString(2));
            }
            JOptionPane.showMessageDialog(null,
        }
        catc                                        s.ge
         }
    }

    private                                         awt.
        // T
        Stri                                        =?,g
        try{
            ps=con.prepareStatement(sql);
            ps.setString(1,name.getText());
            ps.setString(2,gender.getText());
            ps.setString(3,authorid.getText());
            ps.executeUpdate();
```

Message

Successfully deleted!

SEARCH:

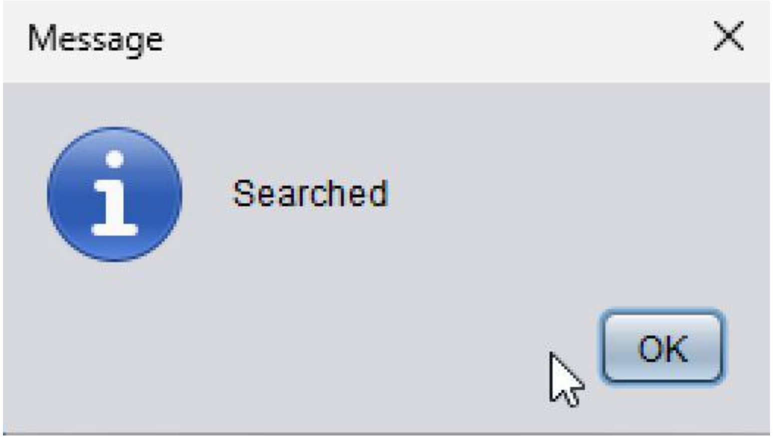


AUTHOR DETAILS

authorid: a3

name: sam

gender:



Searched