Name: Richard Raymond J. Canda **Date:** February 28, 2025

C.Y.S.: BSCpE – 3A

Laboratory Activity 6:

Laboratory Title: Normalization - Second Normal Form (2NF)
Chapter No. and Topic: Chapter 3 - Database Design and Modeling
Discussions:
This activity will cover the process of converting a table to the Second Normal Form (2NF).

Activity Description:
Given a 1NF table, remove partial dependencies to achieve 2NF.

Objectives:

- Remove partial dependencies and achieve 2NF.

Materials:

- SQL client

Procedure:

1. Create a 1NF table:

```
sql

Copy code

CREATE TABLE Books_1NF (

   BookID INT,

   Title VARCHAR(100),

   Author VARCHAR(100),

   Genre VARCHAR(50),

   Publisher VARCHAR(100),

   PublisherAddress VARCHAR(100)

);
```

1. Insert sample data:

```
sql

Copy code

INSERT INTO Books_1NF (BookID, Title, Author, Genre, Publisher,
PublisherAddress)
```

```
VALUES

(1, 'Book A', 'Author1', 'Fiction', 'Publisher1', 'Address1'),

(2, 'Book B', 'Author2', 'Non-Fiction', 'Publisher1',
'Address1');
```

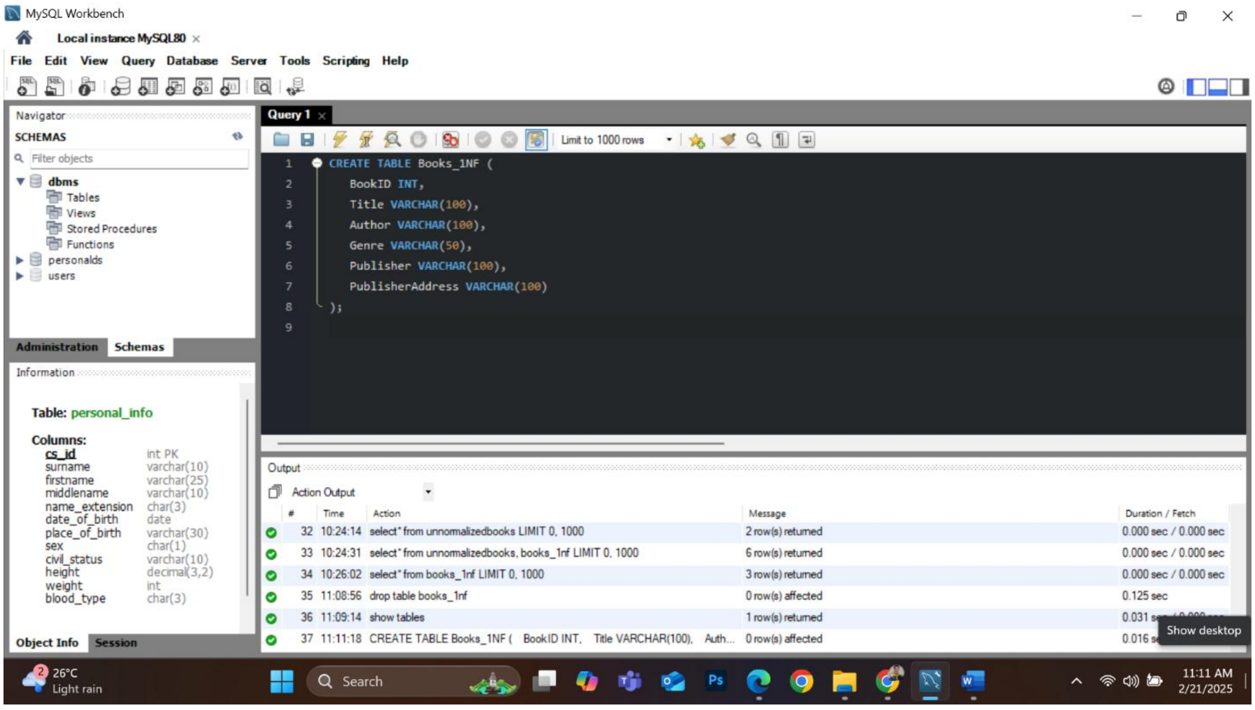
1. Create two separate tables to remove partial dependency:

```
sql

Copy code

CREATE TABLE Books_2NF (

   BookID INT PRIMARY KEY,

   Title VARCHAR(100),

   Author VARCHAR(100),

   Genre VARCHAR(50)

);


CREATE TABLE Publishers (

   PublisherID INT PRIMARY KEY,

   PublisherName VARCHAR(100),

   PublisherAddress VARCHAR(100)

);
```

4. Move Publisher data into the `Publishers` table and adjust `Books_2NF` to include `PublisherID` as a foreign key.

Result:
The table is now in 2NF with no partial dependencies.

1. Create a 1NF table:

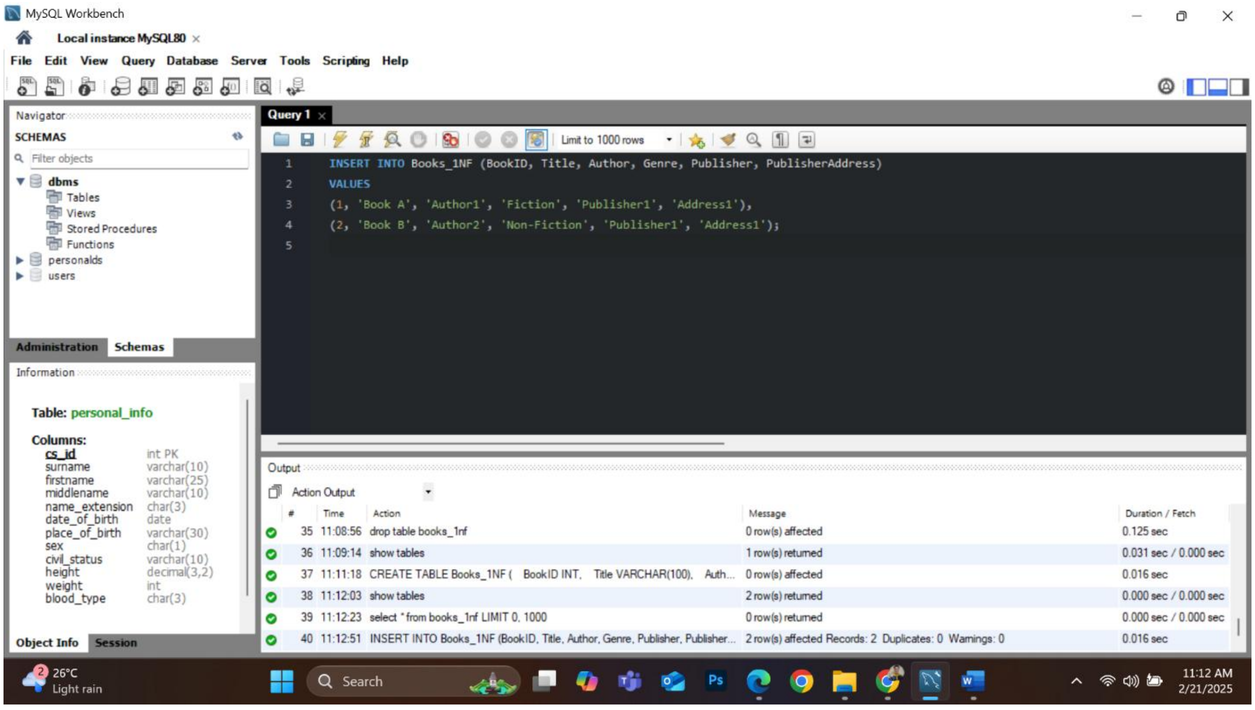


```sql
CREATE TABLE Books_1NF (
    BookID INT,
    Title VARCHAR(100),
    Author VARCHAR(100),
    Genre VARCHAR(50),
    Publisher VARCHAR(100),
    PublisherAddress VARCHAR(100)
);
```

```
Table: personal_info

Columns:
  cs_id            int PK
  surname          varchar(10)
  firstname        varchar(25)
  middlename       varchar(10)
  name_extension   char(3)
  date_of_birth    date
  place_of_birth   varchar(30)
  sex              char(1)
  civil_status     varchar(10)
  height           decimal(3,2)
  weight           int
  blood_type       char(3)
```

#	Time	Action	Message	Duration / Fetch
32	10:24:14	select * from unnormalizedbooks LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
33	10:24:31	select * from unnormalizedbooks, books_1nf LIMIT 0, 1000	6 row(s) returned	0.000 sec / 0.000 sec
34	10:26:02	select * from books_1nf LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
35	11:08:56	drop table books_1nf	0 row(s) affected	0.125 sec
36	11:09:14	show tables	1 row(s) returned	0.031 se
37	11:11:18	CREATE TABLE Books_1NF (BookID INT, Title VARCHAR(100), Auth...	0 row(s) affected	0.016 s

2. Insert sample data:

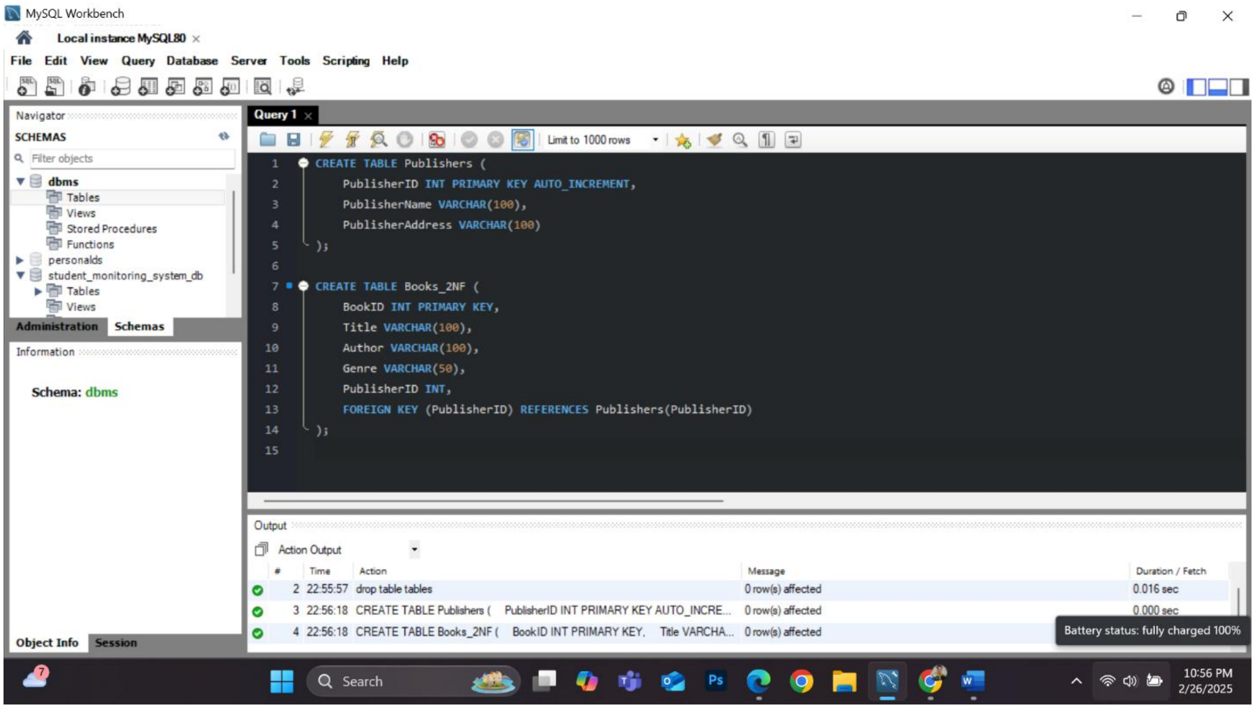


```sql
INSERT INTO Books_1NF (BookID, Title, Author, Genre, Publisher, PublisherAddress)
VALUES
(1, 'Book A', 'Author1', 'Fiction', 'Publisher1', 'Address1'),
(2, 'Book B', 'Author2', 'Non-Fiction', 'Publisher1', 'Address1');
```

```
Table: personal_info

Columns:
  cs_id            int PK
  surname          varchar(10)
  firstname        varchar(25)
  middlename       varchar(10)
  name_extension   char(3)
  date_of_birth    date
  place_of_birth   varchar(30)
  sex              char(1)
  civil_status     varchar(10)
  height           decimal(3,2)
  weight           int
  blood_type       char(3)
```

#	Time	Action	Message	Duration / Fetch
35	11:08:56	drop table books_1nf	0 row(s) affected	0.125 sec
36	11:09:14	show tables	1 row(s) returned	0.031 sec / 0.000 sec
37	11:11:18	CREATE TABLE Books_1NF (BookID INT, Title VARCHAR(100), Auth...	0 row(s) affected	0.016 sec
38	11:12:03	show tables	2 row(s) returned	0.000 sec / 0.000 sec
39	11:12:23	select * from books_1nf LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
40	11:12:51	INSERT INTO Books_1NF (BookID, Title, Author, Genre, Publisher, Publisher...	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.016 sec

3. Create two separate tables to remove partial dependency:

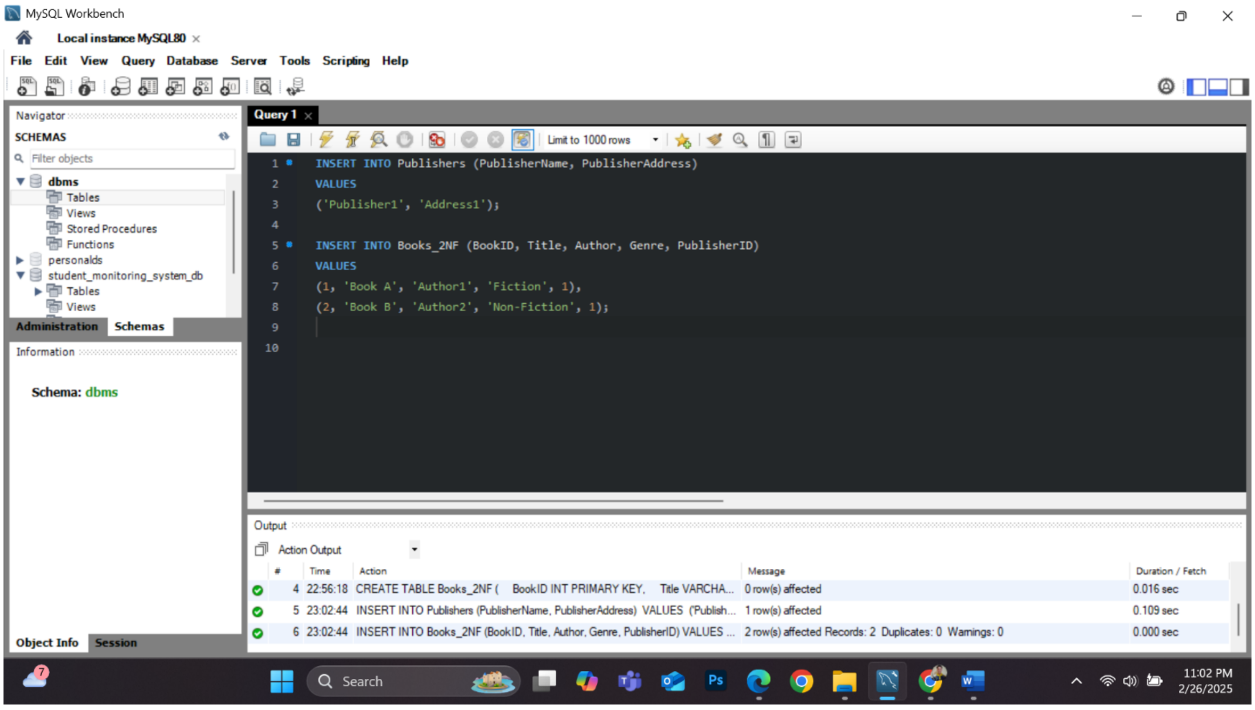


```sql
CREATE TABLE Publishers (
    PublisherID INT PRIMARY KEY AUTO_INCREMENT,
    PublisherName VARCHAR(100),
    PublisherAddress VARCHAR(100)
);

CREATE TABLE Books_2NF (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(100),
    Genre VARCHAR(50),
    PublisherID INT,
    FOREIGN KEY (PublisherID) REFERENCES Publishers(PublisherID)
);
```

```
#  Time      Action                                                                     Message              Duration / Fetch
2  22:55:57  drop table tables                                                          0 row(s) affected    0.016 sec
3  22:56:18  CREATE TABLE Publishers (    PublisherID INT PRIMARY KEY AUTO_INCRE...    0 row(s) affected    0.000 sec
4  22:56:18  CREATE TABLE Books_2NF (    BookID INT PRIMARY KEY,    Title VARCHA...    0 row(s) affected
```

In this step, which separates the publishers Name and Address. This will help to reduce repeating atomic value for the publisherName and publisherAddress.

4. Move Publisher data into the `Publishers` table and adjust `Books_2NF` to include `PublisherID` as a foreign key.

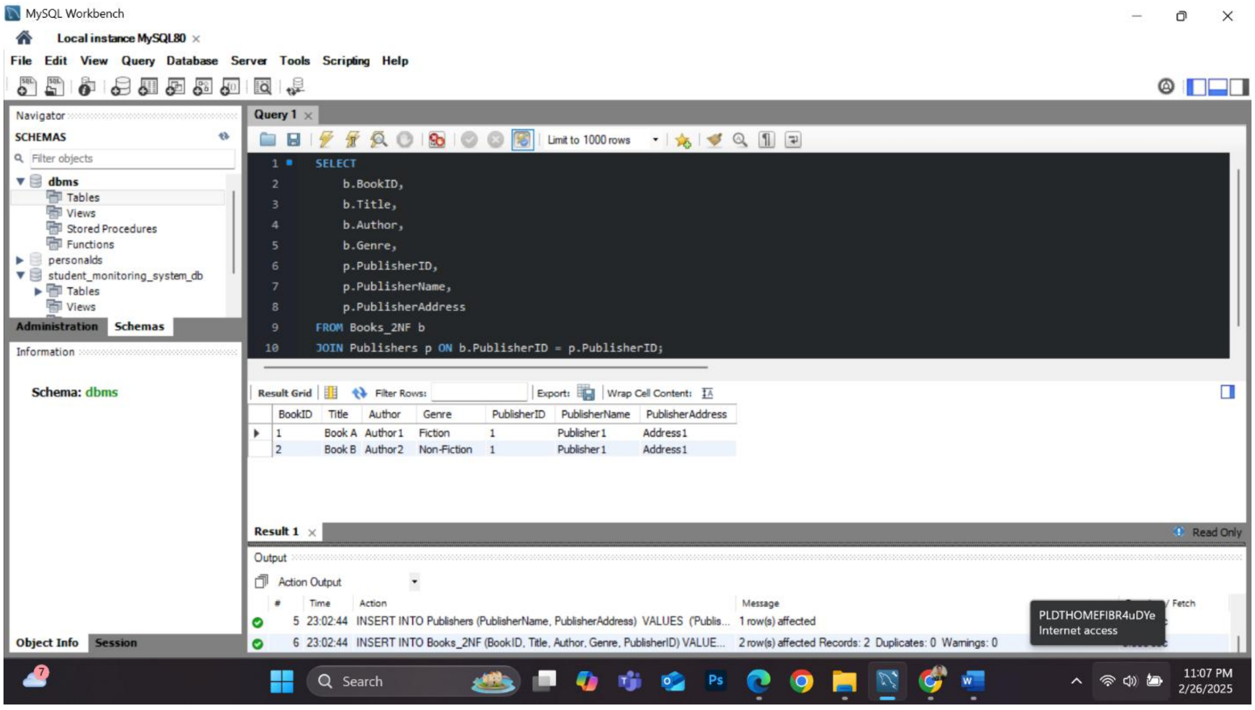


```sql
INSERT INTO Publishers (PublisherName, PublisherAddress)
VALUES
('Publisher1', 'Address1');

INSERT INTO Books_2NF (BookID, Title, Author, Genre, PublisherID)
VALUES
(1, 'Book A', 'Author1', 'Fiction', 1),
(2, 'Book B', 'Author2', 'Non-Fiction', 1);
```

```
#  Time      Action                                                                           Message                                                 Duration / Fetch
4  22:56:18  CREATE TABLE Books_2NF (    BookID INT PRIMARY KEY,    Title VARCHA...          0 row(s) affected                                       0.016 sec
5  23:02:44  INSERT INTO Publishers (PublisherName, PublisherAddress)  VALUES  ('Publish...  1 row(s) affected                                       0.109 sec
6  23:02:44  INSERT INTO Books_2NF (BookID, Title, Author, Genre, PublisherID) VALUES ...     2 row(s) affected Records: 2  Duplicates: 0  Warnings: 0  0.000 sec
```

I moved the publishers data into the publishers table. The publishers data can be retrieve by pointing to the publishersID.

Here is what it looks like:



```sql
SELECT
    b.BookID,
    b.Title,
    b.Author,
    b.Genre,
    p.PublisherID,
    p.PublisherName,
    p.PublisherAddress
FROM Books_2NF b
JOIN Publishers p ON b.PublisherID = p.PublisherID;
```

BookID	Title	Author	Genre	PublisherID	PublisherName	PublisherAddress
1	Book A	Author1	Fiction	1	Publisher1	Address1
2	Book B	Author2	Non-Fiction	1	Publisher1	Address1

```
#  Time      Action                                                                    Message
5  23:02:44  INSERT INTO Publishers (PublisherName, PublisherAddress)  VALUES  ('Publis...  1 row(s) affected
6  23:02:44  INSERT INTO Books_2NF (BookID, Title, Author, Genre, PublisherID) VALUE...    2 row(s) affected Records: 2  Duplicates: 0  Warnings: 0
```

Additional Questions/Discussions:

- What is a partial dependency, and how does 2NF eliminate it?

Answer:

A **partial dependency** happens when a **non-key attribute** depends on only part of a **composite primary key**, not the entire key. This can lead to redundant data.

For example, in a table where the primary key is made up of two columns, if one of those columns can uniquely identify another attribute (like a product name), but the other column doesn't need to, that's a partial dependency. This causes repetition because the product name will be stored multiple times for each order, even if the product itself hasn't changed.

To eliminate partial dependencies, we move the dependent attribute to another table. This is part of the process of **Second Normal Form (2NF)**.

In 2NF, we ensure that **every non-key attribute depends on the entire composite key**, not just a part of it. This way, we avoid repeating data. So, if a product name only depends on the product ID (and not on the order ID), we move the product name into its own table where the product ID is the key.

- How do foreign keys help maintain data integrity?

Answer:

A **foreign key** is a column (or a set of columns) in one table that **refers to the primary key** of another table. It essentially links two tables together by creating a relationship between them.

Foreign keys ensure that the relationship between tables is consistent. For example, if a record in one table refers to a record in another table, the foreign key ensures that the referenced record **actually exists**. This prevents situations where we might accidentally reference non-existing data.

Example: If we have a Books table with a PublisherID as a foreign key, this ensures that every book is linked to an actual publisher from the Publishers table. If you try to add a book with a PublisherID that doesn’t exist in the Publishers table, the database will reject the action.

It ensures that only valid data is entered, foreign keys reduce the risk of **data anomalies** like **inconsistent data** or **mislinked records**. This helps maintain the overall accuracy of the data within the database.

Conclusions:

In this laboratory activity, we explored **Second Normal Form (2NF)** and its importance in **database normalization**. We learned that 2NF eliminates **partial dependencies** by ensuring that all non-key attributes depend on the entire primary key, reducing redundancy and improving data integrity. By splitting tables, such as moving the Publisher data to a separate table, we enhanced data organization and minimized repetition. This normalization process not only prevents anomalies but also improves database efficiency by optimizing storage and operations. Overall, the activity demonstrated how 2NF leads to more **organized**, **consistent**, and **efficient** databases.