Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 10
Art Gallery Database Final Report

Usba Gohir 500978708
Yusra Farooqi 500978708

Table of Contents

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 1

Application: Art Gallery Database Management System - ArtisticView

Overview:
ArtisticView is an art gallery/convention center that plays a crucial role in fostering culture, creativity and community engagement. Here, artists are able to showcase their work and connect with their audience. In addition, ArtisticView hosts a wide range of events, from corporate conferences to cultural gatherings, serving as a social hub for the community.

Functions:
This DBMS aims to facilitate the organization, retrieval, and management of data in a structured manner. For the purpose of this art gallery, the DBMS serves as a critical tool for efficiently storing and managing information about artwork, artists, events, employees, and attendees. The key tables include artwork, artist, employee, ticket sale, and artwork purchase. The name for every entry is the primary key for most of these tables. The functionality of the "Artwork" table is that it adds new artwork, updates artwork details, and removes sold artwork from the database. Details about the artist of each artwork can be found in the "Artist" table. The "Artist" table adds new arts, updates artist information, and removes old artists from the database. The "Event" table assigns and manages art exhibitions while tracking their dates and capacity levels. Each event showcases art from a specific genre. To view which art pieces will be featured in each event, the "genre" section of the "Artwork" table can be searched. To track that the capacity of each event is not exceeded, the "Ticket Sale" table is used. The "Ticket Sale" has a "ticket number" attribute which is the primary key for this table. The table records ticket sales for events and tracks guest list and purchase dates. To see the cost of each ticket or more information about the event it is intended for, the "Event" table is used. The "Artwork Purchase" table has a "Collector ID" attribute, being the primary key for the table. This table also tracks information on the Bid for each artwork sold as well as the payment method and purchase date. Artwork ID is also listed but for more information on each piece, the "Artwork" table is intended to be used.

Entities:

ARTWORK

Artist	Title	Year	Medium	Dimensions (cm)	Genre	Price (CAD)	Artwork ID
Vincent van Gogh	The Starry Night	1889	Oil paint	74x92	Landscape painting	100,000,000	00051

Usba Gohir 500978708
Yusra Farooqi 500978708

Leonardo da Vinci	Mona Lisa	1503	Oil paint	77x53	Portrait painting	860,000,000	00213
Pablo Picasso	Still Life with Chair-Caning	1912	Mixed media	29x37	Still life	179,400,000	00542

ARTIST

Name	Birthdate (YYYY/MM/DD)	Nationality
Vincent van Gogh	1853/03/30	Dutch
Leonardo da Vinci	1452/04/15	Italian
Pablo Picasso	1881/10/25	Spanish

EVENT

Event Name	Featured Genre	Date (YYYY/MM/DD)	Ticket Price (CAD)	Visitor Capacity
Surreal Dreams	Still life	2021/04/20	30	100
Art Gala	*all*	2022/09/30	45	150
Children's Art Workshop	Landscape painting	2023/05/13	20	40

TICKET SALE

Ticket Number	Event Name	First Name	Last Name	Purchase Date (YYYY/MM/DD)
001	Surreal Dreams	Dan	Humphrey	2021/04/15
020	Art Gala	Chuck	Bass	2022/09/25
05	Children's Art Workshop	Blair	Waldorf	2023/05/13

Usba Gohir 500978708
Yusra Farooqi 500978708

ARTWORK PURCHASE

Collector ID	Artwork ID	Purchase Date (YYYY/MM/DD)	Bid (CAD)	Payment Method
0001	00051	2023/01/20	1 200 000	Credit
0021	00213	2023/07/28	995 000	Credit
0012	00542	2022/12/13	2 570 000	Debit

Other Potential Entities:

- Art Collector - (Attributes: Collector ID, Name, Address, Phone, Age, Username, and Password for Services)
- Exhibition - (Attributes: Exhibition ID, Title, Start Date, End Date, Description, Location)
- Exhibition Artwork - (Attributes: Exhibition ID, Artwork ID, Position [Wall, Floor, Pedestal], Status [On Display, Sold], Price when on display)
- Artwork Purchase - (Attributes: Collector ID, Artwork ID, Purchase Date, Bid, Payment Method)
- Inventory - (Attributes: Inventory ID, Name, Description, Quantity, Location)
- Sponsor - (Attributes: Sponsor ID, Name, Address, Phone, Email, Contribution Amount)
- Feedback - (Attributes: Feedback ID, Collector ID, Exhibition ID, Event ID, Rating [1-5], Comments)

Relationships Among Tables and Other Potential Entities:

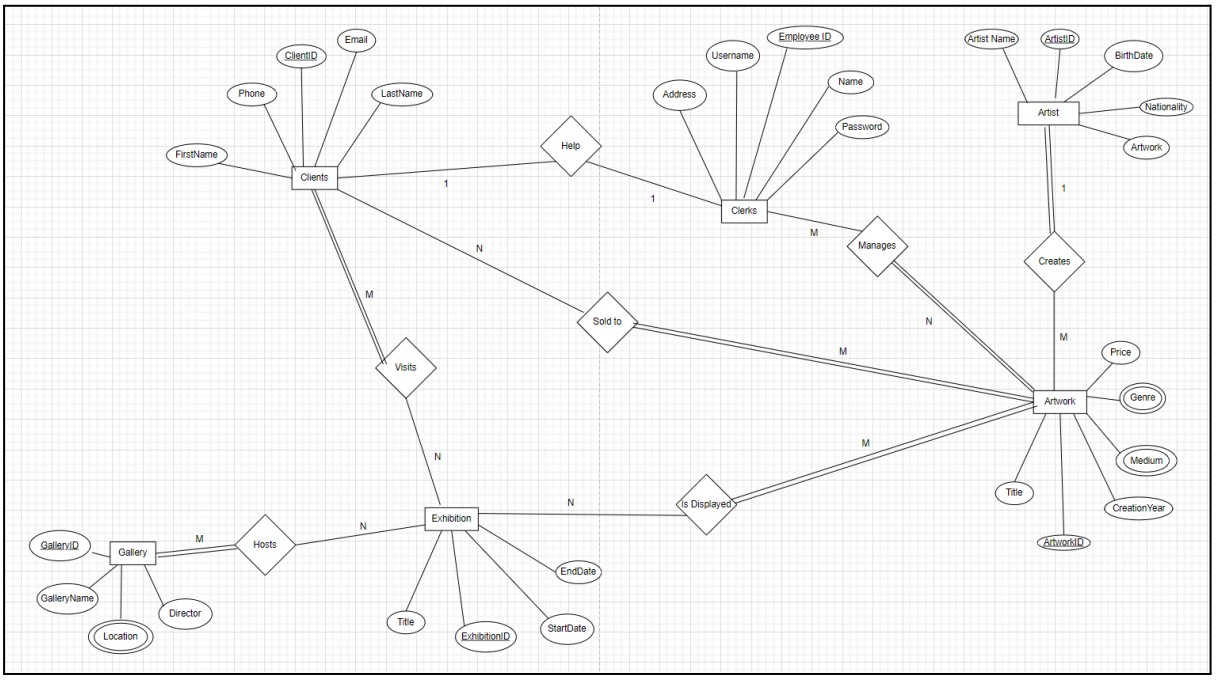
- Artist -> Creates -> Artwork
- Artwork -> Part of -> Exhibition Artwork
- Exhibition -> Features -> Exhibition Artwork
- Art Collector -> Collects -> Artwork (if a collector owns the artwork)
- Art Collector -> Attends -> Exhibition
- Art Collector -> Makes -> Purchase
- Event -> Features -> Exhibition (if an event is part of an exhibition)
- Event -> Sells -> Ticket
- Art Collector -> Buys -> Ticket (when collectors purchase tickets)
- Inventory -> Contains -> Artwork (to track which artworks are in inventory)
- Exhibition -> Sponsored by -> Sponsor (to indicate sponsorships for exhibitions)
- Collector -> Provides -> Feedback (to capture feedback from collectors about exhibitions or events)

Changes:

We revised the overview, entities, and relationships to align with the implemented SQL code. Consequently, we eliminated any unused elements, such as CDs and Audiobooks, from the inventory.

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 2



Changes:

Unused attributes, including employeeID and room availability, were removed. This adjustment enabled us to utilize the username as the primary key for both. Additionally, we addressed the issue of hour of use, ensuring it is no longer treated as a multivalued attribute.

Assignment 3

```
-- Create the Artist table
CREATE TABLE Artist (
    ArtistID NUMBER PRIMARY KEY,      -- Unique artist identifier
    ArtistName VARCHAR2(255) NOT NULL,
    BirthDate DATE,
    Nationality VARCHAR2(255)
);

-- Create the Artwork table
CREATE TABLE Artwork (
    ArtworkID NUMBER PRIMARY KEY,    -- Unique artwork identifier
    Title VARCHAR2(255) NOT NULL,
    CreationYear NUMBER,
    Medium VARCHAR2(255),
    Dimensions VARCHAR2(255),
    Price NUMBER
);
```

Usba Gohir 500978708
Yusra Farooqi 500978708

```
-- Create the Exhibition table
CREATE TABLE Exhibition (
    ExhibitionID NUMBER PRIMARY KEY, -- Unique exhibition identifier
    Title VARCHAR2(255) NOT NULL,
    StartDate DATE,
    EndDate DATE,
    Description VARCHAR2(1000)
);

-- Create the Customer table
CREATE TABLE Customer (
    CustomerID NUMBER PRIMARY KEY,  -- Unique customer identifier
    FirstName VARCHAR2(255) NOT NULL,
    LastName VARCHAR2(255) NOT NULL,
    Email VARCHAR2(255),
    Phone VARCHAR2(15)
);

-- Create the Gallery table
CREATE TABLE Gallery (
    GalleryID NUMBER PRIMARY KEY,      -- Unique gallery identifier
    GalleryName VARCHAR2(255) NOT NULL,
    Location VARCHAR2(255),
    Director VARCHAR2(255)
);

-- Create the Client table
CREATE TABLE Client (
    ClientID NUMBER PRIMARY KEY,    -- Unique client identifier
    Phone VARCHAR2(15),
    Email VARCHAR2(255),
    FirstName VARCHAR2(255),
    LastName VARCHAR2(255)
);

-- Create the Artist-Artwork relationship table
CREATE TABLE ArtistArtwork (
    ArtistID NUMBER,               -- Foreign key for the artist
    ArtworkID NUMBER,              -- Foreign key for the artwork
    PRIMARY KEY (ArtistID, ArtworkID), -- Combined primary key
    FOREIGN KEY (ArtistID) REFERENCES Artist(ArtistID), -- Reference to
the Artist table
```

```
    FOREIGN KEY (ArtworkID) REFERENCES Artwork(ArtworkID) -- Foreign
key; define relationship tables (establish relationships between
entities).
);

-- Create the Artwork-Exhibition relationship table
CREATE TABLE ArtworkExhibition (
    ArtworkID NUMBER,
    ExhibitionID NUMBER,
    PRIMARY KEY (ArtworkID, ExhibitionID), -- Combined primary key
    FOREIGN KEY (ArtworkID) REFERENCES Artwork(ArtworkID),     --
Reference to the Artwork table
    FOREIGN KEY (ExhibitionID) REFERENCES Exhibition(ExhibitionID) --
Reference to the Exhibition table
);

-- Create the Artwork-Customer relationship table
CREATE TABLE ArtworkCustomer (
    ArtworkID NUMBER,
    CustomerID NUMBER,
    PRIMARY KEY (ArtworkID, CustomerID), -- Combined primary key
    FOREIGN KEY (ArtworkID) REFERENCES Artwork(ArtworkID),     --
Reference to the Artwork table
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)   --
Reference to the Customer table
);

-- Create the Gallery-Exhibition relationship table
CREATE TABLE GalleryExhibition (
    GalleryID NUMBER,
    ExhibitionID NUMBER,
    PRIMARY KEY (GalleryID, ExhibitionID),                     --
Combined primary key
    FOREIGN KEY (GalleryID) REFERENCES Gallery(GalleryID),     --
Reference to the Gallery table
    FOREIGN KEY (ExhibitionID) REFERENCES Exhibition(ExhibitionID) --
Reference to the Exhibition table
);

-- Create the Customer-Exhibition relationship table
CREATE TABLE CustomerExhibition (
    CustomerID NUMBER,
    ExhibitionID NUMBER,
```

```
    PRIMARY KEY (CustomerID, ExhibitionID),                          --
Combined primary key
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),        --
Reference to the Customer table
    FOREIGN KEY (ExhibitionID) REFERENCES Exhibition(ExhibitionID)   --
Reference to the Exhibition table
);

-- Create the Client-Exhibition relationship table
CREATE TABLE ClientExhibition (
    ClientID NUMBER,
    ExhibitionID NUMBER,
    PRIMARY KEY (ClientID, ExhibitionID),
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID),              --
Reference to the Client table
    FOREIGN KEY (ExhibitionID) REFERENCES Exhibition(ExhibitionID)   --
Reference to the Exhibition table
);

-- Create the Client-Artwork relationship table
CREATE TABLE ClientArtwork (
    ClientID NUMBER,
    ArtworkID NUMBER,
    PRIMARY KEY (ClientID, ArtworkID),                               --
Combined primary key
    FOREIGN KEY (ClientID) REFERENCES Client(ClientID),              --
Reference to the Client table
    FOREIGN KEY (ArtworkID) REFERENCES Artwork(ArtworkID)            --
Reference to the Artwork table
);
```

RA Notation:

1. Artist table:
   π_{ArtistID, ArtistName, BirthDate, Nationality}(Artist)

2. Artwork table:
   π_{ArtworkID, Title, CreationYear, Medium, Dimensions, Price}(Artwork)

3. Exhibition table:
   π_{ExhibitionID, Title, StartDate, EndDate, Description}(Exhibition)

Usba Gohir 500978708
Yusra Farooqi 500978708

4. Customer table:
   π_{CustomerID, FirstName, LastName, Email, Phone}(Customer)

5. Gallery table:
   π_{GalleryID, GalleryName, Location, Director}(Gallery)

6. Client table:
   π_{ClientID, Phone, Email, FirstName, LastName}(Client)

7. Artist-Artwork relationship table:
   π_{ArtistID, ArtworkID}(ArtistArtwork)

8. Artwork-Exhibition relationship table:
   π_{ArtworkID, ExhibitionID}(ArtworkExhibition)

9. Artwork-Customer relationship table:
   π_{ArtworkID, CustomerID}(ArtworkCustomer)

10. Gallery-Exhibition relationship table:
    π_{GalleryID, ExhibitionID}(GalleryExhibition)

11. Customer-Exhibition relationship table:
    π_{CustomerID, ExhibitionID}(CustomerExhibition)

12. Client-Exhibition relationship table:
    π_{ClientID, ExhibitionID}(ClientExhibition)

13. Client-Artwork relationship table:
    π_{ClientID, ArtworkID}(ClientArtwork)

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 4 Part 1

Query 1:

Description: Retrieve a list of unique artwork mediums:

Query:
```
SELECT DISTINCT ArtworkMedium
FROM Artworks;
```

Output/Results:

	ARTWORKMEDIUM
1	Oil paint
2	Mixed media

$\pi ArtworkMedium(Artworks)$

Query 2:

Description: List all artists and their nationalities, ordered by nationality:

Query:
```
SELECT ArtistName, Nationality
FROM Artists
ORDER BY Nationality;
```

Output/Results:

	ARTISTNAME	NATIONALITY
1	Vincent van Gogh	Dutch
2	Leonardo da Vinci	Italian
3	Pablo Picasso	Spanish

$\pi_{ArtistName,\ Nationality}(\sigma true(Artists) \bowtie Artists'_{ArtistID,\ Nationality'})$

Query 3:

Description: Retrieve the top 3 most expensive artworks:

Query:
```
SELECT *
FROM (SELECT ArtworkTitle, Price FROM Artworks ORDER BY Price DESC)
WHERE ROWNUM <= 3;
```

Output/Results:

	ARTWORKTITLE	PRICE
1	Mona Lisa	860000000
2	Still Life	179400000
3	The Starry Night	100000000

$\pi_{ArtworkTitle,\ Price}(\sigma ROWNUM \leq 3(\sigma true(\rho_{ArtworkTitle,\ Price}(\sigma_{true}(Artworks) \bowtie ROWNUM))))$

Query 4:

Description: Count the number of events for each artwork medium and display them in descending order:

Query:
SELECT ArtworkMedium, COUNT(*) AS EventCount
FROM Artworks
JOIN Events ON Artworks.ArtworkID = Events.ArtworkID
GROUP BY ArtworkMedium –groups rows that have the same values into summary rows
ORDER BY EventCount DESC;

Output/Results:

	ARTWORKMEDIUM	EVENTCOUNT
1	Oil paint	2
2	Mixed media	1

$\pi ArtworkMedium,\ COUNT(*)\ as\ EventCount(\gamma ArtworkMedium,\ COUNT(*)\ as\ EventCount(Artworks \bowtie Eve$

Query 5:

Description: Find the average event price for each artist:

Query:
SELECT Artists.ArtistName, AVG(Events.EventPrice) AS AvgEventPrice
FROM Artists
LEFT JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
LEFT JOIN Events ON Artworks.ArtworkID = Events.ArtworkID
GROUP BY Artists.ArtistName;

Output/Results:

	ARTISTNAME	AVGEVENTPRICE
1	Leonardo da Vinci	45
2	Pablo Picasso	30
3	Vincent van Gogh	(null)

$\pi Artists.ArtistName,\ AVG(Events.EventPrice)\ as\ AvgEventPrice(\gamma Artists.ArtistName (Artists \bowtie Artworks \bowtie Events))$

Usba Gohir 500978708
Yusra Farooqi 500978708

Query 6:

Description: select distinct customer information from a table named "Customers." This query will retrieve unique rows based on the combination of the columns CustomerID, FirstName, LastName, Email, and Phone from the Customers table.

Query:
SELECT DISTINCT CustomerID, FirstName, LastName, Email, Phone
FROM Customers;

Output/Results:

	CUSTOMERID	FIRSTNAME	LASTNAME	EMAIL	PHONE
1	450	Chuck	Bass	cbass@gmail.com	416-777-9999
2	451	Dan	Humphrey	danh@yahoo.ca	647-765-2357
3	452	Blair	Waldorf	waldorfb@outlook.com	905-543-1356

$\pi CustomerID,\ FirstName,\ LastName,\ Email,\ Phone(Customers)$

Query 7:

Description: Show all artwork by the Artist Leonardo da Vinci
Query:
SELECT DISTINCT CustomerID, FirstName, LastName, Email, Phone
FROM Customers;

Output/Results:

	ARTWORKID	ARTWORKTITLE
1	52	Mona Lisa

$\pi CustomerID,\ FirstName,\ LastName,\ Email,\ Phone(\delta(Customers))$

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 4 Part 2

VIEW 1: ExpensiveArtworks

Description: The "ExpensiveArtworks" view contains information about high-priced artworks. Specifically, it includes the title of the artwork and its price, but only includes artworks with a price exceeding 100,000,000.00 (100 million) units of the currency used in the database.

Query:

```
CREATE VIEW ExpensiveArtworks AS
SELECT  ArtworkTitle, Price
FROM Artworks
WHERE Price > 100000000.00;
```

Output/Results:

	ARTWORKTITLE	PRICE
1	Mona Lisa	860000000
2	Still Life	179400000
3	David	230000000
4	Beaches	540000000
5	Past	240000000

$$ExpensiveArtworks \leftarrow \pi ArtworkTitle,\ Price(\sigma Price\ >\ 100000000.00(Artworks))$$

VIEW 2: UpcomingEventDates

Description: The "UpcomingEventDates" view is created to display upcoming events with their titles, event prices, and start dates. It filters events from the "Events" table, showing only those with a start date on or after October 16, 2023.

Query:

```
CREATE VIEW UpcomingEventDates AS
SELECT Title, EventPrice, StartDate
FROM Events
WHERE StartDate >= '2023-10-16';
```

Output/Results:

Usba Gohir 500978708
Yusra Farooqi 500978708

	TITLE	EVENTPRICE	STARTDATE
1	Palette Paradiso	45	23-10-16
2	Tegan and Sara	34	23-11-06
3	With Love	23	23-11-23
4	Finding Light	15	23-12-05

$UpcomingEventDates \leftarrow \pi Title,\ EventPrice,\ StartDate(\sigma StartDate\ >=\ '2023-10-16'(Events))$

VIEW 3:

Description: creates a view named "ArtistTotalValue" that combines data from the "Artists" and "Artworks" tables. It calculates the total value of artworks for each artist by summing the "Price" column from the "Artworks" table, grouping the results by artist using their unique ID.

VIEW:

```
CREATE VIEW ArtistTotalValue AS
SELECT Artists.ArtistID, Artists.ArtistName, SUM(Artworks.Price) AS TotalValue
FROM Artists
LEFT JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistID, Artists.ArtistName;
```

% groups rows that have the same values
% returns all records from the left table, and the matching records from the right table
%union same as inner join
Output/Results:

	ARTISTID	ARTISTNAME	TOTALVALUE
1	1201	Leonardo da Vinci	860000000
2	1203	Vincent van Gogh	100000000
3	1204	Micoangellou	1010000000
4	1202	Pablo Picasso	179400000

$ArtistTotalValue \leftarrow \pi Artists.ArtistID,\ Artists.ArtistName,\ SUM(Artworks.Price)\ as\ TotalValue(\gamma Artists.ArtistID,\ Artists.ArtistName(Artists \ltimes Artworks))Here:$

Query 1:

Description: This query provides a list of artist names along with the average price of their artworks, allowing you to see which artist's works, on average, are priced higher or lower.

Query:

```
SELECT Artists.ArtistName, AVG(Artworks.Price) AS AveragePrice
```

```
FROM Artists
JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistName;
```

Output/Results:

	ARTISTNAME	AVERAGEPRICE
1	Leonardo da Vinci	860000000
2	Micoangellou	336666666.66666666666666666666666667
3	Pablo Picasso	179400000
4	Vincent van Gogh	100000000

$ArtistTotalValue \leftarrow \pi Artists.ArtistID,\ Artists.ArtistName,\ SUM(Artworks.Price)\ as\ TotalValue\ (\gamma Artists.ArtistID,\ Artists.ArtistName(Artists \ltimes Artworks))$

Query 2:

Description: Counts the number of artworks by each artist:

Query:

```
SELECT Artists.ArtistName, COUNT(Artworks.ArtworkID) AS TotalArtworks
FROM Artists
JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistName;
```

Output/Results:

	ARTISTNAME	TOTALARTWORKS
1	Leonardo da Vinci	1
2	Micoangellou	3
3	Pablo Picasso	1
4	Vincent van Gogh	1

$ArtistTotalArtworks \leftarrow \pi Artists.ArtistName,\ COUNT(Artworks.ArtworkID)\ as\ TotalArtworks(\gamma Artists.ArtistName(Artists \ltimes Artworks))$

Query 3:

Description: Calculates the total value of artworks by each artist and order the result by total value in descending order.

Query:

```
SELECT Artists.ArtistName, SUM(Artworks.Price) AS TotalValue
```

```
FROM Artists
LEFT JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistName
ORDER BY TotalValue DESC;
```

Output/Results:

	ARTISTNAME	TOTALVALUE
1	Micoangellou	1010000000
2	Leonardo da Vinci	860000000
3	Pablo Picasso	179400000
4	Vincent van Gogh	100000000

$$ArtistTotalValue \leftarrow \pi Artists.ArtistName,\ SUM(Artworks.Price)\ as\ TotalValue\ (\gamma Artists.ArtistName(Artists \ltimes Artworks) \ltimes \sigma TotalValue\ is\ not\ NULL)$$

Query 4:

Description: List all customers who purchased tickets, along with the events they attended, the artists whose artworks were exhibited, and the event titles.

Query:

```
SELECT Customers.FirstName, Customers.LastName, Events.Title AS EventTitle,
Artists.ArtistName
FROM Customers
JOIN Tickets ON Customers.CustomerID = Tickets.CustomerID
JOIN Events ON Tickets.EventID = Events.EventID
JOIN Artworks ON Events.ArtworkID = Artworks.ArtworkID
JOIN Artists ON Artworks.ArtistID = Artists.ArtistID;
```

Output/Results:

	FIRSTNAME	LASTNAME	EVENTTITLE	ARTISTNAME
1	Chuck	Bass	Surreal Dreams	Pablo Picasso
2	Dan	Humphrey	Art Gala	Leonardo da Vinci
3	Blair	Waldorf	Childrens Art Workshop	Vincent van Gogh

$$\pi Customers.FirstName,\ Customers.LastName,\ Events.Title\ as\ EventTitle,\ Artists.ArtistName\ (Customers \bowtie Tickets \bowtie Events \bowtie Artworks \bowtie Artists)$$

Query 5:

Description: List all customers who purchased tickets, along with the events they attended, the artists whose artworks were exhibited, and the event titles. Include the event start and end dates.

Usba Gohir 500978708
Yusra Farooqi 500978708

Query:

```
SELECT Customers.FirstName, Customers.LastName, Events.Title AS EventTitle,
Artists.ArtistName, Events.StartDate, Events.EndDate
FROM Customers
JOIN Tickets ON Customers.CustomerID = Tickets.CustomerID
JOIN Events ON Tickets.EventID = Events.EventID
JOIN Artworks ON Events.ArtworkID = Artworks.ArtworkID
JOIN Artists ON Artworks.ArtistID = Artists.ArtistID;
```

Output/Results:

	FIRSTNAME	LASTNAME	EVENTTITLE	ARTISTNAME	STARTDATE	ENDDATE
1	Chuck	Bass	Surreal Dreams	Pablo Picasso	21-04-20	21-04-30
2	Dan	Humphrey	Art Gala	Leonardo da Vinci	22-09-30	22-10-01
3	Blair	Waldorf	Childrens Art Workshop	Vincent van Gogh	23-05-13	22-05-13

$$\pi Customers.FirstName,\ Customers.LastName,\ Events.Title\ as\ EventTitle,\ Artists.ArtistName,\ Events.StartDate,\ Events.EndDate(Customers \bowtie Tickets \bowtie Events \bowtie Artworks \bowtie Artists)$$

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 5

-

VIEW 1: ExpensiveArtworks

Description: The "ExpensiveArtworks" view contains information about high-priced artworks. Specifically, it includes the title of the artwork and its price, but only includes artworks with a price exceeding 100,000,000.00 (100 million) units of the currency used in the database.

Query:

```
CREATE VIEW ExpensiveArtworks AS
SELECT  ArtworkTitle, Price
FROM Artworks
HAVING COUNT(Price) > 100000000.00;
```

Output/Results:

Artwork Title	Price
Mona Lisa	860,000,000
Still Life	179,400,000
David	230,000,000
Beaches	540,000,000
Past	240,000,000

$ExpensiveArtworks \leftarrow \pi ArtworkTitle,\ Price(\sigma Price\ >\ 100000000.00(Artworks))$

VIEW 2:

Description: creates a view named "ArtistTotalValue" that combines data from the "Artists" and "Artworks" tables. It calculates the total value of artworks for each artist by summing the "Price" column from the "Artworks" table, grouping the results by artist using their unique ID.

VIEW:

```
CREATE VIEW ArtistTotalValue AS
SELECT Artists.ArtistID, Artists.ArtistName, SUM(Artworks.Price) AS TotalValue
FROM Artists
LEFT JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistID, Artists.ArtistName;
```

Output/Results:

Artist ID	Artist Name	Total Value
1201	Leonardo da Vinci	860,000,000
1203	Vincent van Gogh	100,000,000
1204	Micoangellou	1,010,000,000
1202	Pablo Picasso	179,400,000

$\pi ArtworkTitle,\ Price(\sigma Price\ >\ 100000000.00(Artworks))$

Usba Gohir 500978708
Yusra Farooqi 500978708

VIEW 3: UpcomingEventDates

Description: The "UpcomingEventDates" view is created to display upcoming events with their titles, event prices, and start dates. It filters events from the "Events" table, showing only those with a start date on or after October 16, 2023.

Query:

```
CREATE VIEW UpcomingEventDates AS
SELECT Title, EventPrice, StartDate
FROM Events
WHERE StartDate >= '2023-10-16';
```

Output/Results:

$UpcomingEventDates \leftarrow \pi Title,\ EventPrice,\ StartDate(\sigma StartDate\ >=\ '2023-10-16'(Events))Here$:

VIEW 4: EventOccupancy

Description: creates a view named "EventOccupancy" contains information regarding Event details. Specifically, it includes the Event ID, Title, and Capacity, but only for intimate events which contain less than 50 people.

VIEW:

```
CREATE VIEW EventOccupancy AS
SELECT EventID, Title, EventCapacity
FROM Events
WHERE EXISTS (EventCapacity<=50);
```

Output/Results:

Event ID	Event Title	Event Capacity
352	Childrens Art Workshop	40
353	Palette Paradiso	40
354	Tegan and Sara	40
355	With Love	40
356	Finding Light	40

$EventOccupancy \leftarrow \pi EventID,\ Title,\ EventCapacity(\sigma EventCapacity\ <=\ 50(Events))$

Query 1:

Description: This query provides a list of artist names along with the average price of their artworks, allowing you to see which artist's works, on average, are priced higher or lower.

Query:

```
SELECT Artists.ArtistName, AVG(Artworks.Price) AS AveragePrice
FROM Artists
JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
```

GROUP BY Artists.ArtistName;

Output/Results:

Artist Name	Average Price
Leonardo da Vinci	860,000,000.00
Micoangellou	336,666,666.67
Pablo Picasso	179,400,000.00
Vincent van Gogh	100,000,000.00

$\pi Artists.ArtistName,\ AVG(Artworks.Price)\ as\ AveragePrice(\gamma Artists.ArtistName(Artists \bowtie Artworks))$

Query 2:

Description: Counts the number of artworks by each artist:

Query:

```
SELECT Artists.ArtistName, COUNT(Artworks.ArtworkID) AS TotalArtworks
FROM Artists
JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistName;
```

Output/Results:

Artist Name	Total Artworks
Leonardo da Vinci	1
Micoangellou	3
Pablo Picasso	1
Vincent van Gogh	1

$SELECT\ Artists.ArtistName,\ COUNT(Artworks.ArtworkID)\ AS\ TotalArtworksFROM\ ArtistsJOIN\ Artworks\ ON\ Artists.ArtistID\ =\ Artworks.ArtistIDGROUP\ BY\ Artists.ArtistName;$

Query 3:

Description: Calculates the total value of artworks by each artist and order the result by total value in descending order.

Query:

```
SELECT Artists.ArtistName, SUM(Artworks.Price) AS TotalValue
FROM Artists
LEFT JOIN Artworks ON Artists.ArtistID = Artworks.ArtistID
GROUP BY Artists.ArtistName
ORDER BY TotalValue DESC;
```

Output/Results:

First Name	Last Name	Event Title	Artist Name
Dan	Humphrey	Art Gala	Leonardo da Vinci
Chuck	Bass	Surreal Dreams	Pablo Picasso
Blair	Waldorf	Childrens Art Workshop	Vincent van Gogh

$ArtistTotalValue \leftarrow \pi Artists.ArtistName,\ SUM(Artworks.Price)\ as\ TotalValue(\gamma Artists.ArtistName(Artists \ltimes Artworks))$

Usba Gohir 500978708
Yusra Farooqi 500978708

Query 4:

Description: The query combines data from the "Customers" and "Artists" tables and labels each row with its type (either "Customer" or "Artist"). It retrieves customer information from the "Customers" table and artist information from the "Artists" table using a UNION operator.

Query:

```
SELECT 'Customer' AS Type, CustomerID AS ID, FirstName AS Name, LastName AS Desc
FROM Customers
UNION
SELECT 'Artist' AS Type, ArtistID AS ID, ArtistName AS Name, Nationality AS Desc FROM
Artists;
```

Output/Results:

Type	ID	Name	Desc
Artist	1201	Leonardo da Vinci	Italian
Artist	1202	Pablo Picasso	Spanish
Artist	1203	Vincent van Gogh	Dutch
Artist	1204	Micoangellou	English
Customer	450	Chuck	Bass
Customer	451	Dan	Humphrey
Customer	452	Blair	Waldorf

$Result \leftarrow \pi$'$Customer$' $as\ Type,\ CustomerID\ as\ ID,\ FirstName\ as\ Name,\ LastName\ as\ Desc(Customers) \cup \pi$'$Artist$' $as\ Type,\ ArtistID\ as\ ID,\ ArtistName\ as\ Name,\ Nationality\ as\ Desc(Artists)$

Query 5:

Description: The MINUS query retrieves a list of customer IDs from the "Customers" table that are not found in the list of customer IDs who have made art purchases from the "ArtPurchases" table. In other words, it identifies customers who have not purchased any artworks.

Query:

```
SELECT CustomerID
FROM Customers
MINUS
SELECT DISTINCT CustomerID
FROM ArtPurchases;
```

Output/Results:

Customer ID
451

$Result \leftarrow \pi CustomerID(Customers) - \pi CustomerID(ArtPurchases)$

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 6

Customers:

	CUSTOMERID	FIRSTNAME	LASTNAME	EMAIL	PHONE
1	450	Chuck	Bass	cbass@gmail.com	416-777-9999
2	451	Dan	Humphrey	danh@yahoo.ca	647-765-2357
3	452	Blair	Waldorf	waldorfb@outlook.com	905-543-1356

- Primary Key: CustomerID
- Functional Dependencies:
    - FirstName, LastName, Email, and Phone are functionally dependent on CustomerID

Artists:

	ARTISTID	ARTISTNAME	BIRTHDATE	NATIONALITY
1	1200	Vincent van Gogh	53-03-30	Dutch
2	1201	Leonardo da Vinci	52-04-15	Italian
3	1202	Pablo Picasso	81-10-25	Spanish

- Primary Key: ArtistID
- Functional Dependencies:
    - ArtistName, BirthDate, and Nationality are functionally dependent on ArtistID.

Artworks:

	ARTWORKID	ARTWORKTITLE	CREATIONYEAR	ARTWORKMEDIUM	DIMENSIONS	PRICE	ARTISTID
1	51	The Starry Night	1889	Oil paint	74x92	100000000	(null)
2	52	Mona Lisa	1503	Oil paint	77x53	860000000	1201
3	53	Still Life	1912	Mixed media	29x37	179400000	1202

- Primary Key: ArtworkID
- Functional Dependencies:
    - ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, and ArtistID are functionally dependent on ArtworkID.

Events:

	EVENTID	TITLE	STARTDATE	ENDDATE	EVENTCAPACITY	EVENTPRICE	ARTWORKID
1	350	Surreal Dreams	21-04-20	21-04-30	100	30	53
2	351	Art Gala	22-09-30	22-10-01	150	45	52
3	352	Childrens Art Workshop	23-05-13	22-05-13	40	20	51

- Primary Key: EventID
- Functional Dependencies:
    - Title, StartDate, EndDate, EventCapacity, EventPrice, and ArtworkID are functionally dependent on EventID.

Tickets:

Usba Gohir 500978708
Yusra Farooqi 500978708

	TICKETID	TICKETPURCHASEDATE	CUSTOMERID	EVENTID
1	1	21-04-15	450	350
2	2	22-09-25	451	351
3	3	23-05-13	452	352

- Primary Key: TicketID
- Functional Dependencies:
    - TicketPurchaseDate, CustomerID, and EventID are functionally dependent on TicketID.

ArtPurchases:

	PURCHASEID	PAYMENTMETHOD	CUSTOMERID	ARTWORKID
1	583	Credit	450	51
2	584	Cash	450	52
3	585	Debit	452	53

- Primary Key: PurchaseID
- Functional Dependencies:
    - PaymentMethod, CustomerID, and ArtworkID are functionally dependent on PurchaseID.

- describes the relationship between attributes (columns) in a relational database table. It specifies how the values of one or more attributes determine the values of other attributes.
- defines a rule or constraint that tells you when certain attributes are dependent on others within a table.

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 7

A relational schema is **3NF** if it is in first (1NF) and second normal form (2NF) and every non-key attribute is non-transitively dependent only on the primary key. (A transitive dependency is when one non-key attribute is functionally dependent on another non-key attribute, rather than directly on the primary key).

- A relational schema is **2NF** if all non-key attributes are functionally dependent on the primary key. One attribute is functionally dependent on another if, for every unique value of the determining attribute, there is exactly one corresponding value for the dependent attribute.
    - In 2NF, there should be no partial dependencies
    - Functional dependency ensures that the values of the non-key attributes can be uniquely determined by the values of the primary key, and there is no duplication of information
- A relational schema is **1NF** if each attribute contains atomic values (meaning that the value can not further be divided into smaller pieces)

Customers:

	CUSTOMERID	FIRSTNAME	LASTNAME	EMAIL	PHONE
1	450	Chuck	Bass	cbass@gmail.com	416-777-9999
2	451	Dan	Humphrey	danh@yahoo.ca	647-765-2357
3	452	Blair	Waldorf	waldorfb@outlook.com	905-543-1356

- Primary Key: CustomerID
- Functional Dependencies:
    - {CustomerID} -> FirstName, LastName, Email, Phone
- This table is in 3NF
    - It is 1NF because each attribute contains atomic values
        - CustomerID is a unique identifier for each artist
        - FirstName, LastName, Email, and Phone are its atomic values
    - It is 2NF because all non-key attributes are functionally dependent on the primary key.
        - FirstName is functionally dependent on CustomerID because for every unique CustomerID, there is exactly one corresponding FirstName
        - LastNameis functionally dependent on CustomerID because for every unique CustomerID, there is exactly one corresponding LastName

        - Email is functionally dependent on CustomerID because for every unique CustomerID, there is exactly one corresponding Email
        - Phone is functionally dependent on CustomerID because for every unique CustomerID, there is exactly one corresponding Phone
    - To be 3NF, all non-key attributes should not be transitively dependent on the primary key
        - Here, FirstName, LastName, Email, and Phone are all directly dependent on CustomerID, and there are no transitive dependencies
        - Therefore, the table is in 3NF

Artists:

	ARTISTID	ARTISTNAME	BIRTHDATE	NATIONALITY
1	1200	Vincent van Gogh	53-03-30	Dutch
2	1201	Leonardo da Vinci	52-04-15	Italian
3	1202	Pablo Picasso	81-10-25	Spanish

- Primary Key: ArtistID
- Functional Dependencies:
    - {ArtistID} -> ArtistName, BirthDate, Nationality
- This table is in 3NF
    - It is 1NF because each attribute contains atomic values
        - ArtistID is a unique identifier for each artist
        - ArtistName, BirthDate, and Nationality are its atomic values
    - It is 2NF because all non-key attributes are functionally dependent on the primary key.
        - ArtistName is functionally dependent on ArtistID because for every unique ArtistID, there is exactly one corresponding ArtistName
        - BirthDate is functionally dependent on ArtistID because for every unique ArtistID, there is exactly one corresponding BirthDate
        - Nationality is functionally dependent on ArtistID because for every unique ArtistID, there is exactly one corresponding Nationality
    - To be 3NF, all non-key attributes should not be transitively dependent on the primary key

- Here, FirstName, LastName, Email, and Phone are all directly dependent on CustomerId, and there are no transitive dependencies
- Therefore, the table is in 3NF

Artworks:

	ARTWORKID	ARTWORKTITLE	CREATIONYEAR	ARTWORKMEDIUM	DIMENSIONS	PRICE	ARTISTID
1	51	The Starry Night	1889	Oil paint	74x92	100000000	(null)
2	52	Mona Lisa	1503	Oil paint	77x53	860000000	1201
3	53	Still Life	1912	Mixed media	29x37	179400000	1202

- Primary Key: ArtworkID
- Functional Dependencies:
  - {ArtworkID} -> ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID
  -
- This table is in 3NF
  - It is 1NF because each attribute contains atomic values
    - ArtworkID is a unique identifier for each artist
    - ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, and ArtistID are its atomic values
  - It is 2NF because all non-key attributes are functionally dependent on the primary key.
    - ArtworkTitle is functionally dependent on ArtworkID because for every unique ArtworkID, there is exactly one corresponding ArtworkTitle
    - CreationYear is functionally dependent on ArtworkID because for every unique ArtworkID, there is exactly one corresponding CreationYear
    - ArtworkMedium is functionally dependent on ArtworkID because for every unique ArtworkID, there is exactly one corresponding ArtworkMedium
    - Dimension is functionally dependent on ArtworkID because for every unique ArtworkID, there is exactly one corresponding Dimensions
    - Price is functionally dependent on ArtworkID because for every unique ArtworkID, there is exactly one corresponding Price
    - ArtistID is functionally dependent on ArtworkID because for every unique ArtworkID, there is exactly one corresponding ArtistID

- To check for 3NF, there should be no transitive dependencies. Because ArtistID is referenced as a foreign key and there are separate tables for Artists and Artwork, this removes the transitive relationship and ensures that ArtistID is dependent on the primary key ArtworkID, and Artist-specific information is stored in a separate table.

Events:

	EVENTID	TITLE	STARTDATE	ENDDATE	EVENTCAPACITY	EVENTPRICE	ARTWORKID
1	350	Surreal Dreams	21-04-20	21-04-30	100	30	53
2	351	Art Gala	22-09-30	22-10-01	150	45	52
3	352	Childrens Art Workshop	23-05-13	22-05-13	40	20	51

- Primary Key: EventID
- Functional Dependencies:
  - {EventID) -> Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID
- This table is in 3NF
  - It is 1NF because each attribute contains atomic values
    - EventID is a unique identifier for each artist
    - Title, StartDate, EndDate, EventCapacity, EventPrice, and ArtworkID are its atomic values
  - It is 2NF because all non-key attributes are functionally dependent on the primary key.
    - Title is functionally dependent on EventID because for every unique EventID, there is exactly one corresponding Title
    - StartDate is functionally dependent on EventID because for every unique EventID, there is exactly one corresponding StartDate
    - EndDate is functionally dependent on EventID because for every unique EventID, there is exactly one corresponding EndDate
    - EventPrice is functionally dependent on EventID because for every unique EventID, there is exactly one corresponding EventPrice
    - ArtworkID is functionally dependent on EventID because for every unique EventID, there is exactly one corresponding ArtworkID
  - To check for 3NF, there should be no transitive dependencies. Because ArtworkID is referenced as a foreign key and there are separate tables for Artwork and Events, this removes the transitive relationship and ensures that ArtworkID is dependent on the

primary key EventID, and Artwork-specific information is stored in a separate table.

Tickets:

	TICKETID	TICKETPURCHASEDATE	CUSTOMERID	EVENTID
1	1	21-04-15	450	350
2	2	22-09-25	451	351
3	3	23-05-13	452	352

- Primary Key: TicketID
- Functional Dependencies:
    - {TicketID} -> TicketPurchaseDate, CustomerID, EventID
- This table is in 3NF
    - It is 1NF because each attribute contains atomic values
        - TicketID is a unique identifier for each artist
        - TicketPurchaseDate, CustomerID, and EventID are its atomic values
    - It is 2NF because all non-key attributes are functionally dependent on the primary key.
        - TicketPurchaseDate is functionally dependent on TicketID because for every unique TicketID, there is exactly one corresponding TicketPurchaseDate
        - CustomerID is functionally dependent on TicketID because for every unique TicketID, there is exactly one corresponding CustomerID
        - EventID is functionally dependent on TicketID because for every unique TicketID, there is exactly one corresponding EventID
    - To check for 3NF, there should be no transitive dependencies. Because CustomerID and EventID are referenced as foreign keys and there are separate tables for Customers, Events, and Tickets, this removes the transitive relationship and ensures that CustomerID and EventID are dependent on the primary key TicketID, and Customer-specific information as well as Event-specific information are stored in separate tables.

Usba Gohir 500978708
Yusra Farooqi 500978708

ArtPurchases:

	PURCHASEID	PAYMENTMETHOD	CUSTOMERID	ARTWORKID
1	583	Credit	450	51
2	584	Cash	450	52
3	585	Debit	452	53

- Primary Key: PurchaseID
- Functional Dependencies:
    - {PurchaseID} -> PaymentMethod, CustomerID, ArtworkID
- This table is in 3NF
    - It is 1NF because each attribute contains atomic values
        - PurchaseID is a unique identifier for each artist
        - PaymentMethod, CustomerID, and ArtworkID are its atomic values
    - It is 2NF because all non-key attributes are functionally dependent on the primary key.
        - PaymentMethod is functionally dependent on PurchaseID because for every unique PurchaseID, there is exactly one corresponding PaymentMethod
        - CustomerID is functionally dependent on PurchaseID because for every unique PurchaseID, there is exactly one corresponding CustomerID
        - ArtworkID is functionally dependent on PurchaseID because for every unique PurchaseID, there is exactly one corresponding ArtworkID
    - To check for 3NF, there should be no transitive dependencies. Because CustomerID and ArtworkID are referenced as foreign keys and there are separate tables for Customers, Artworks, and Tickets, this removes the transitive relationship and ensures that CustomerID and ArtworkID are dependent on the primary key PurchaseID, and Customer-specific information as well as Artwork-specific information are stored in separate tables.

Usba Gohir 500978708
Yusra Farooqi 500978708

Assignment 8

FDs List:

Functional Dependencies for Artists Table:

- **ArtistID -> ArtistName, BirthDate, Nationality**
  - (ArtistID uniquely determines ArtistName, BirthDate, and Nationality)

Functional Dependencies for Artworks Table:

- **ArtworkID -> ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID**
  - (ArtworkID uniquely determines all other attributes)

- **ArtistID -> ArtistName, BirthDate, Nationality**
  - (ArtistID uniquely determines ArtistName, BirthDate, Nationality)
  - This is a transitive dependency through ArtistID in the Artworks table.

Functional Dependencies for Events Table:

- **EventID -> Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID**
  - (EventID uniquely determines all other attributes)

- **ArtworkID -> ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID**
  - (ArtworkID uniquely determines all other attributes)
  - This is a transitive dependency through ArtworkID in the Events table.

Functional Dependencies for Tickets Table:

- **TicketID -> TicketPurchaseDate, CustomerID, EventID**
  - (TicketID uniquely determines all other attributes)

- **CustomerID -> FirstName, LastName, Email, Phone**
  - (CustomerID uniquely determines all other attributes)
  - This is a transitive dependency through CustomerID in the Tickets table.

- **EventID -> Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID**
  - (EventID uniquely determines all other attributes)
  - This is a transitive dependency through EventID in the Tickets table.

Functional Dependencies for Customers Table:

- **CustomerID -> FirstName, LastName, Email, Phone**
  - (CustomerID uniquely determines all other attributes)

Functional Dependencies for ArtPurchases Table:

- **PurchaseID -> PaymentMethod, CustomerID, ArtworkID**

  - (PurchaseID uniquely determines all other attributes)

- **CustomerID -> FirstName, LastName, Email, Phone**
  - (CustomerID uniquely determines all other attributes)
  - This is a transitive dependency through CustomerID in the ArtPurchases table.

- **ArtworkID -> ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID**
  - (ArtworkID uniquely determines all other attributes)
  - This is a transitive dependency through ArtworkID in the ArtPurchases table.

Note:
- Transitive dependencies have been indicated where applicable.
- This set of functional dependencies provides the basis for further normalization using the Bernstein and BCNF algorithms.

Algorithm

- step-by-step process for applying the Bernstein and BCNF algorithms to ensure that the tables are in 3NF/BCNF
- address transitive and partial dependencies

Applying the Bernstein Algorithm:

Artists Table:
- No transitive or partial dependencies are present.
- The table is already in 3NF.

Artworks Table:
1. **Closure of Attributes:**
   - Start with the closure of individual attributes.
   - Closure of `ArtistID`: {ArtistID, ArtistName, BirthDate, Nationality}
   - Closure of `ArtworkID`: {ArtworkID, ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID}

2. **Remove Redundant Attributes:**
   - Remove redundant attributes to ensure that each attribute determines a key.
   - Artists Table doesn't have redundant attributes.
   - Artworks Table is already in 3NF.

Events Table:
1. **Closure of Attributes:**
   - Start with the closure of individual attributes.
   - Closure of `EventID`: {EventID, Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID}
   - Closure of `ArtworkID`: {ArtworkID, ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID}

2. **Remove Redundant Attributes:**
   - Remove redundant attributes to ensure that each attribute determines a key.

- Events Table is already in 3NF.

Tickets Table:
1. **Closure of Attributes:**
   - Start with the closure of individual attributes.
   - Closure of `TicketID`: {TicketID, TicketPurchaseDate, CustomerID, EventID}
   - Closure of `CustomerID`: {CustomerID, FirstName, LastName, Email, Phone}
   - Closure of `EventID`: {EventID, Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID}

2. **Remove Redundant Attributes:**
   - Remove redundant attributes to ensure that each attribute determines a key.
   - Tickets Table is already in 3NF.

Customers Table:
- No transitive or partial dependencies are present.
- The table is already in 3NF.

ArtPurchases Table:
1. **Closure of Attributes:**
   - Start with the closure of individual attributes.
   - Closure of `PurchaseID`: {PurchaseID, PaymentMethod, CustomerID, ArtworkID}
   - Closure of `CustomerID`: {CustomerID, FirstName, LastName, Email, Phone}
   - Closure of `ArtworkID`: {ArtworkID, ArtworkTitle, CreationYear, ArtworkMedium, Dimensions, Price, ArtistID}

2. **Remove Redundant Attributes:**
   - Remove redundant attributes to ensure that each attribute determines a key.
   - ArtPurchases Table is already in 3NF.

Applying the BCNF Algorithm:
ALL TABLES ARE ALREADY IN BCNF

Conclusion:
All tables (Artists, Artworks, Events, Tickets, Customers, ArtPurchases) are in 3NF/BCNF. The FDs have been considered, and the tables have been normalized to eliminate transitive and partial dependencies.

Example if we need to convert using Algorithm IF events table was not already BCNF:

EventsTable

EventID	Title	StartDate	EndDate	EventCapacity	EventPrice	ArtworkID	ArtistName	Nationality
1	Art Show	2022-01-01	2022-01-10	100	20.00	101	Artist1	Nationality1
2	Music Concert	2022-02-15	2022-02-16	150	30.00	102	Artist2	Nationality2
3	Workshop	2022-03-20	2022-03-22	50	15.00	103	Artist3	Nationality3

In this table, there's a transitive dependency (EventID -> ArtworkID -> ArtistName, Nationality) through the ArtworkID.

Applying the Bernstein Algorithm:
Closure of Attributes:

Closure of EventID: {EventID, Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID, ArtistName, Nationality}
Closure of ArtworkID: {ArtworkID, ArtistName, Nationality}
Closure of EventID, ArtworkID: {EventID, Title, StartDate, EndDate, EventCapacity, EventPrice, ArtworkID, ArtistName, Nationality}
Identify Redundant Attributes:

The closure of EventID, ArtworkID includes all other attributes, making EventID, ArtworkID a superkey.

Decompose into 3NF:

Create a new table for EventsTable with the following structure:

EventsTable_New

EventID	Title	StartDate	EndDate	EventCapacity	EventPrice	ArtworkID
1	Art Show	2022-01-01	2022-01-10	100	20.00	101
2	Music Concert	2022-02-15	2022-02-16	150	30.00	102
3	Workshop	2022-03-20	2022-03-22	50	15.00	103

So finally:

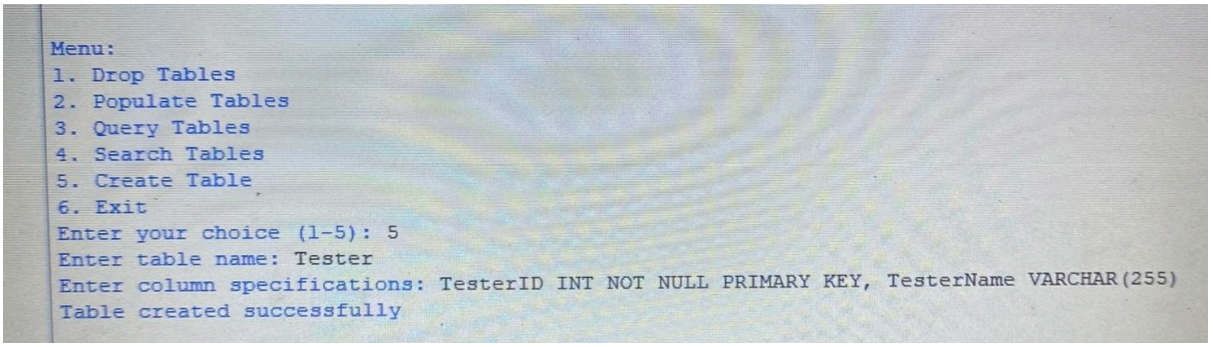
ArtistsTable

ArtworkID	ArtistName	Nationality
101	Artist1	Nationality1
102	Artist2	Nationality2
103	Artist3	Nationality3

Each table represents a distinct entity, and there are no transitive dependencies. The ArtistsTable is related to EventsTable through the ArtworkID. This ensures that the FDs are preserved, and the tables are now in 3NF.

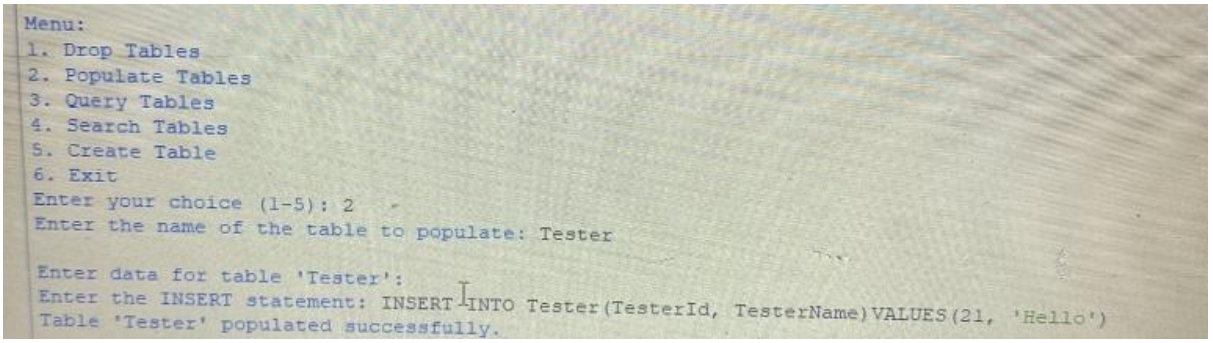
Assignment 9:

For this lab, we created a UI on Python. It simply asks users to select an option from the menu, gives prompts, and executes the user's input.
Create table:



Usba Gohir 500978708
Yusra Farooqi 500978708

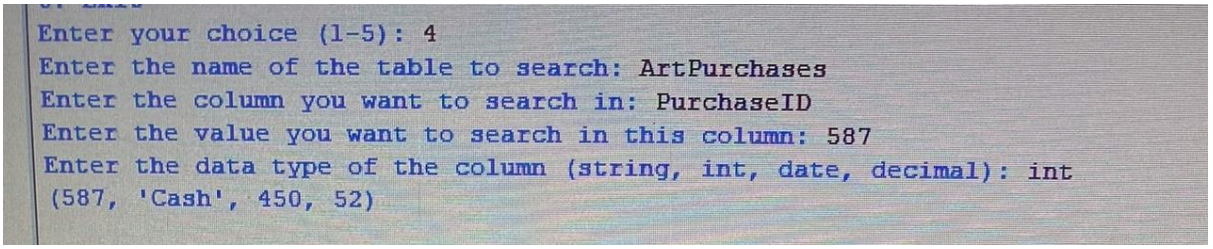
Populate table:



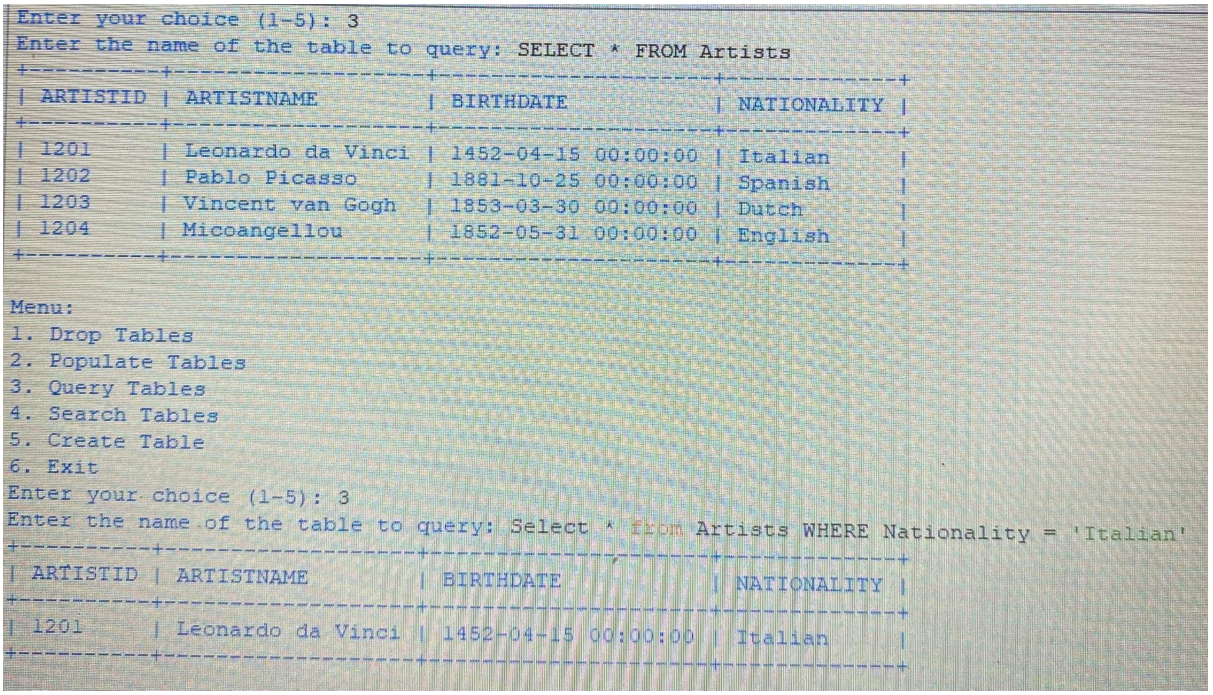
Drop table:

```
Menu:
1. Drop Tables
2. Populate Tables
3. Query Tables
4. Search Tables
5. Create Table
6. Exit
Enter your choice (1-5): 1
Enter the name of the table to drop: Tester
Table dropped successfully
```

Search table:



Query table:



Exit program:

```
Menu:
1. Drop Tables
2. Populate Tables
3. Query Tables
4. Search Tables
5. Create Table
6. Exit
Enter your choice (1-5): 6
Exiting program.
>>>
```