2024-10-14

DATABASES 2

LAB 4 – NORMALIZATION

Paulina Czarnota C21365726

EXERCISE 1: DESIGN

INTRODUCTION

This report presents a normalized database structure for Gill Art Gallery, focusing on managing information about customers, artists, paintings, and purchase transactions. The gallery sells paintings by various artists, with some paintings being sold multiple times, which requires careful data management to avoid redundancy and maintain data integrity.

PROBLEM DESCRIPTION

The database design must support the following requirements:

- **Customer Details**: Contact information and addresses for each customer.
- **Artist Information**: Names and unique identifiers for artists.
- **Painting Details**: Information about each painting, linked to the artist who created it.
- **Purchase Transactions**: Sales records, including sale dates and prices.

NORMALIZATION PROCESS

1. **First Normal Form (1NF)**:
   - Each attribute is atomic and there are no repeating groups.
   - The data is split into individual tables for each entity (Customer, Artist, Painting, and Purchase).
2. **Second Normal Form (2NF)**:
   - Ensures tables are in 1NF, with no partial dependencies on any part of a composite key.
3. **Third Normal Form (3NF)**:
   - Eliminates transitive dependencies, ensuring all non-key attributes depend only on the primary key.

FINAL DATABASE STRUCTURE IN 3NF

TABLE STRUCTURE

1. **City Table**
   - **Table Name**: City
   - **Columns**:

- ZipCode (VARCHAR(10), PRIMARY KEY)
- CityName (VARCHAR(100), NOT NULL)

2. **Customer Table**

   - **Table Name**: Customer

   - **Columns**:
     - CustomerID (SERIAL, PRIMARY KEY)
     - FirstName (VARCHAR(50), NOT NULL)
     - LastName (VARCHAR(50), NOT NULL)
     - Phone (VARCHAR(20), NOT NULL)
     - Street (VARCHAR(100), NOT NULL)
     - ZipCode (VARCHAR(10), FOREIGN KEY REFERENCES City(ZipCode) ON DELETE CASCADE)

3. **Artist Table**

   - **Table Name**: Artist

   - **Columns**:
     - ArtistID (SERIAL, PRIMARY KEY)
     - ArtistName (VARCHAR(100), NOT NULL)

4. **Painting Table**

   - **Table Name**: Painting

   - **Columns**:
     - PaintingID (SERIAL, PRIMARY KEY)
     - ArtistID (INT, FOREIGN KEY REFERENCES Artist(ArtistID) ON DELETE CASCADE)
     - PaintingCode (VARCHAR(20), NOT NULL)
     - Title (VARCHAR(100), NOT NULL)

   - **Unique Constraint**: Ensures uniqueness of painting codes per artist.

5. **Purchase Table**

   - **Table Name**: Purchase

   - **Columns**:
     - PurchaseID (SERIAL, PRIMARY KEY)

- CustomerID (INT, FOREIGN KEY REFERENCES Customer(CustomerID) ON DELETE CASCADE)
- PaintingID (INT, FOREIGN KEY REFERENCES Painting(PaintingID) ON DELETE CASCADE)
- PurchaseDate (DATE, NOT NULL)
- SalesPrice (DECIMAL(10, 2), NOT NULL)

POSTGRESQL IMPLEMENTATION

```
-- Set the schema to public
SET search_path TO public;


-- Drop any views that may exist
DROP VIEW IF EXISTS ranked_participants CASCADE;
DROP VIEW IF EXISTS best_irish_athletes CASCADE;
DROP VIEW IF EXISTS marathon_position CASCADE;
DROP VIEW IF EXISTS medal_table CASCADE;


-- Drop any unrelated tables if they exist
DROP TABLE IF EXISTS Purchase CASCADE;
DROP TABLE IF EXISTS Painting CASCADE;
DROP TABLE IF EXISTS Artist CASCADE;
DROP TABLE IF EXISTS Customer CASCADE;
DROP TABLE IF EXISTS City CASCADE;


-- Create City Table
CREATE TABLE City (
   ZipCode VARCHAR(10) PRIMARY KEY,
   CityName VARCHAR(100) NOT NULL
);


-- Create Customer Table
```

```sql
CREATE TABLE Customer (
    CustomerID SERIAL PRIMARY KEY,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Phone VARCHAR(20) NOT NULL,
    Street VARCHAR(100) NOT NULL,
    ZipCode VARCHAR(10) NOT NULL,
    FOREIGN KEY (ZipCode) REFERENCES City(ZipCode) ON DELETE CASCADE
);

-- Create Artist Table
CREATE TABLE Artist (
    ArtistID SERIAL PRIMARY KEY,
    ArtistName VARCHAR(100) NOT NULL
);

-- Create Painting Table
CREATE TABLE Painting (
    PaintingID SERIAL PRIMARY KEY,
    ArtistID INT NOT NULL,
    PaintingCode VARCHAR(20) NOT NULL,
    Title VARCHAR(100) NOT NULL,
    FOREIGN KEY (ArtistID) REFERENCES Artist(ArtistID) ON DELETE CASCADE,
    UNIQUE (ArtistID, PaintingCode) -- Ensures uniqueness of painting codes for each artist
);

-- Create Purchase Table
CREATE TABLE Purchase (
    PurchaseID SERIAL PRIMARY KEY,
```

```
    CustomerID INT NOT NULL,
    PaintingID INT NOT NULL,
    PurchaseDate DATE NOT NULL,
    SalesPrice DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (PaintingID) REFERENCES Painting(PaintingID) ON DELETE CASCADE
);

-- Verification step: List all tables in the public schema
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'public';
```

EXPLANATION

The provided SQL code performs several essential operations:

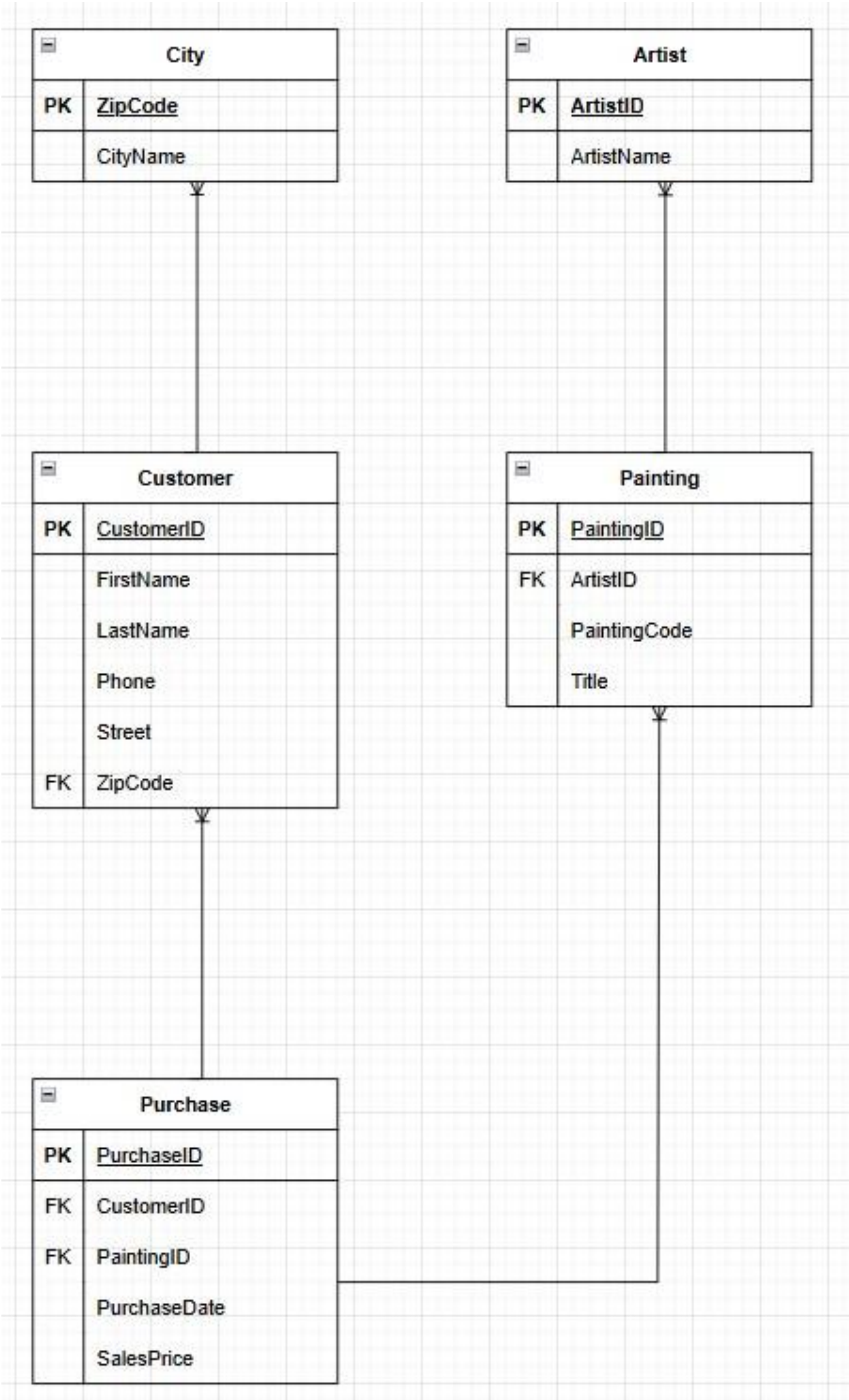
- **Schema Setting**: Sets the working schema to public to ensure the correct namespace.

- **Drop Statements**: Removes any existing tables or views that might conflict with the current setup.

- **Table Creation**: Defines tables (City, Customer, Artist, Painting, Purchase) with appropriate constraints.
  - Each table includes a **primary key** to uniquely identify records.
  - **Foreign keys** establish relationships between tables and enforce referential integrity.
  - A **UNIQUE constraint** on the Painting table ensures that each painting code is unique within an artist.

- **Verification Query**: Lists all tables in the public schema to confirm successful creation.

ER DIAGRAM


City
PK ZipCode
CityName
Artist
PK ArtistID
ArtistName
Customer
PK CustomerID
FirstName
LastName
Phone
Street
FK ZipCode
Painting
PK PaintingID
FK ArtistID
PaintingCode
Title
Purchase
PK PurchaseID
FK CustomerID
FK PaintingID
PurchaseDate
SalesPrice

ER DIAGRAM EXPLANATION

Entities and Keys

1. **City:**
   - **Primary Key:** ZipCode
   - **Attributes:** CityName
   - **Relationship:** Links to the Customer table via ZipCode.

2. **Customer:**
   - **Primary Key:** CustomerID
   - **Attributes:** FirstName, LastName, Phone, Street, ZipCode
   - **Foreign Key:** ZipCode (linked to City)
   - **Relationship:** Links to Purchase through CustomerID.

3. **Artist:**
   - **Primary Key:** ArtistID
   - **Attributes:** ArtistName
   - **Relationship:** Links to Painting through ArtistID.

4. **Painting:**
   - **Primary Key:** PaintingID
   - **Attributes:** PaintingCode, Title
   - **Foreign Key:** ArtistID (linked to Artist)
   - **Relationship:** Links to Purchase through PaintingID.

5. **Purchase:**
   - **Primary Key:** PurchaseID
   - **Attributes:** PurchaseDate, SalesPrice
   - **Foreign Keys:** CustomerID (linked to Customer), PaintingID (linked to Painting)

Relationships

- **City to Customer:** One-to-Many; one city can have multiple customers, each customer belongs to only one city.

- **Customer to Purchase:** One-to-Many; one customer can make multiple purchases, but each purchase involves only one customer.

- **Artist to Painting:** One-to-Many; one artist can create many paintings, but each painting is associated with only one artist.

- **Painting to Purchase:** One-to-Many; one painting can be sold multiple times (resold), each sale is recorded as a separate purchase.

EXERCISE 2: DESIGN AND IMPLEMENTATION

INTRODUCTION

This report presents the design and implementation of a normalized database schema for a student application system. The purpose of this exercise is to address the problems associated with a non-normalized table and to transform it into a more efficient and organized structure using principles of normalization. The original non-normalized table contains redundant data and violates various normalization forms, which can lead to inefficiencies and anomalies in data management.

PROBLEM DESCRIPTION

The initial table, Apps_NOT_Normalized, contains information related to student applications. However, it suffers from multiple issues:

- Redundant data entries for students and their addresses.
- Repeated reference information for referees.
- Multiple prior schools linked to each student without a clear structure.

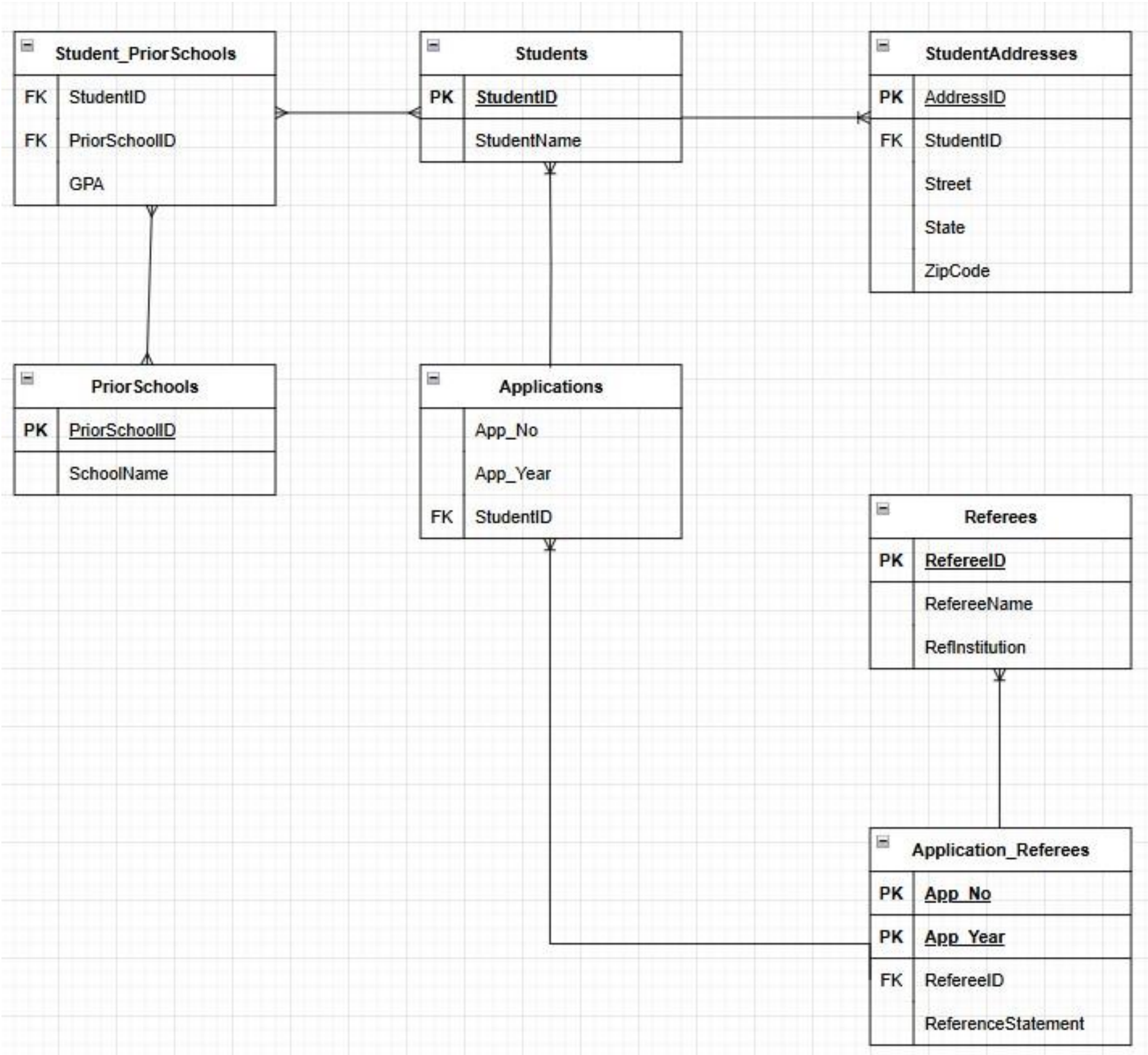
This scenario indicates that the data is not in a normalized form, particularly lacking proper relationships and separate entities for students, addresses, applications, referees, and prior schools. Our objective is to normalize this data into several related tables to eliminate redundancy and ensure data integrity.

TABLE STRUCTURES

The final implementation consists of the following tables, each representing a distinct entity:

1. **Students**: Stores unique student information.
2. **StudentAddresses**: Stores address details for each student.
3. **Applications**: Contains application details for students, including application number and year.
4. **Referees**: Stores unique referee information.
5. **Application_Referees**: Links applications to referees.
6. **PriorSchools**: Stores unique prior school names.
7. **Student_PriorSchools**: Links students to their prior schools and records their GPA.

ER DIAGRAM



Student_PriorSchools
FK StudentID
FK PriorSchoolID
GPA
Students
PK StudentID
StudentName
StudentAddresses
PK AddressID
FK StudentID
Street
State
ZipCode
PriorSchools
PK PriorSchoolID
SchoolName
Applications
App_No
App_Year
FK StudentID
Referees
PK RefereeID
RefereeName
RefInstitution
Application_Referees
PK App_No
PK App_Year
FK RefereeID
ReferenceStatement

ER DIAGRAM EXPLANATION

Entities and Attributes

1. **Students**

   - **Attributes**:
     - **StudentID (PK)**: Unique identifier for each student.
     - **StudentName**: Name of the student.

   - **Relationships**: One-to-many relationship with **Applications** and **StudentAddresses**.

2. **Referees**

   - **Attributes**:
     - **RefereeID (PK)**: Unique identifier for each referee.
     - **ReferenceName**: Name of the referee.
     - **RefInstitution**: Institution of the referee.

   - **Relationships**: One-to-many relationship with **Application_Referees**.

3. **Applications**

   - **Attributes**:
     - **App_No**: Application number.
     - **App_Year**: Year of application.
     - **StudentID (FK)**: Foreign key referencing **Students**.

   - **Relationships**: One-to-many relationship with **Application_Referees**.

4. **Application_Referees**

   - **Attributes**:
     - **App_No**: Application number (part of composite PK).
     - **App_Year**: Year of application (part of composite PK).
     - **RefereeID (FK)**: Foreign key referencing **Referees**.
     - **ReferenceStatement**: Statement from the referee.

   - **Relationships**: Junction table linking **Applications** and **Referees** (many-to-many relationship).

5. **StudentAddresses**

   - **Attributes**:

- **AddressID (PK)**: Unique identifier for each address.
- **StudentID (FK)**: Foreign key referencing **Students**.
- **Street**, **State**, **ZipCode**: Address details.

- **Relationships**: One-to-many relationship with **Students**.

6. **PriorSchools**

- **Attributes**:
    - **PriorSchoolID (PK)**: Unique identifier for each school.
    - **SchoolName**: Name of the prior school.

- **Relationships**: Many-to-many relationship with **Student_PriorSchools**.

7. **Student_PriorSchools**

- **Attributes**:
    - **StudentID (FK)**: Foreign key referencing **Students**.
    - **PriorSchoolID (FK)**: Foreign key referencing **PriorSchools**.
    - **GPA**: Grade point average at the prior school.

- **Relationships**: Many-to-many relationship between **Students** and **PriorSchools**.

POSTGRESQL IMPLEMENTATION

NON-NORMALIZED TABLE CREATION

To begin with, the non-normalized table is created and populated as follows:

```
-- Drop the existing non-normalized table if it exists

DROP TABLE IF EXISTS Apps_NOT_Normalized;


-- Create non-normalized table

CREATE TABLE Apps_NOT_Normalized (

  App_No INTEGER,

  StudentID INTEGER,

  StudentName VARCHAR(50),

  Street VARCHAR(100),
```

```
  State VARCHAR(30),
  ZipCode VARCHAR(7),
  App_Year INTEGER,
  ReferenceName VARCHAR(100),
  RefInstitution VARCHAR(100),
  ReferenceStatement VARCHAR(500),
  PriorSchoolId INTEGER,
  PriorSchoolAddr VARCHAR(100),
  GPA NUMERIC(4, 2)
);

-- Insert data into the non-normalized table
INSERT INTO Apps_NOT_Normalized VALUES(1,1,'Mark','Grafton Street','New York','NY234',2003,'Dr. Jones','Trinity College','Good guy',1,'Castleknock',65);
-- ... (additional insert statements)
```

NORMALIZED TABLES CREATION

The following SQL code creates the normalized tables as per the database schema described earlier.

```
-- Drop existing normalized tables if they exist
DROP TABLE IF EXISTS Application_Referees CASCADE;
DROP TABLE IF EXISTS PriorSchools CASCADE;
DROP TABLE IF EXISTS Applications CASCADE;
DROP TABLE IF EXISTS Referees CASCADE;
DROP TABLE IF EXISTS StudentAddresses CASCADE;
DROP TABLE IF EXISTS Students CASCADE;
DROP TABLE IF EXISTS Student_PriorSchools CASCADE;

-- Create Students table
```

```sql
CREATE TABLE Students (
    StudentID INTEGER PRIMARY KEY,
    StudentName VARCHAR(50) NOT NULL
);

-- Create StudentAddresses table
CREATE TABLE StudentAddresses (
    AddressID SERIAL PRIMARY KEY,
    StudentID INTEGER REFERENCES Students(StudentID) ON DELETE CASCADE,
    Street VARCHAR(100),
    State VARCHAR(30),
    ZipCode VARCHAR(7),
    UNIQUE(StudentID, Street, State, ZipCode)
);

-- Create Applications table
CREATE TABLE Applications (
    App_No INTEGER,
    StudentID INTEGER REFERENCES Students(StudentID) ON DELETE CASCADE,
    App_Year INTEGER,
    PRIMARY KEY (App_No, App_Year)
);

-- Create Referees table
CREATE TABLE Referees (
    RefereeID SERIAL PRIMARY KEY,
    ReferenceName VARCHAR(100),
    RefInstitution VARCHAR(100),
    UNIQUE(ReferenceName, RefInstitution)
```

```
);

-- Create Application_Referees table
CREATE TABLE Application_Referees (
    App_No INTEGER,
    App_Year INTEGER,
    RefereeID INTEGER REFERENCES Referees(RefereeID) ON DELETE CASCADE,
    ReferenceStatement VARCHAR(500),
    PRIMARY KEY (App_No, App_Year, RefereeID),
    FOREIGN KEY (App_No, App_Year) REFERENCES Applications(App_No, App_Year)
ON DELETE CASCADE
);

-- Create PriorSchools table
CREATE TABLE PriorSchools (
    PriorSchoolID SERIAL PRIMARY KEY,
    SchoolName VARCHAR(100) NOT NULL,
    UNIQUE(SchoolName)
);

-- Create Student_PriorSchools table
CREATE TABLE Student_PriorSchools (
    StudentID INTEGER REFERENCES Students(StudentID) ON DELETE CASCADE,
    PriorSchoolID INTEGER REFERENCES PriorSchools(PriorSchoolID) ON DELETE
CASCADE,
    GPA NUMERIC(4, 2),
    PRIMARY KEY (StudentID, PriorSchoolID)
);
```

DATA MIGRATION

After creating the normalized tables, we migrate the data from the non-normalized table into the normalized schema.

```
-- Data Migration

-- Insert data into Students
INSERT INTO Students (StudentID, StudentName)
SELECT DISTINCT StudentID, StudentName
FROM Apps_NOT_Normalized;

-- Insert data into StudentAddresses
INSERT INTO StudentAddresses (StudentID, Street, State, ZipCode)
SELECT DISTINCT StudentID, Street, State, ZipCode
FROM Apps_NOT_Normalized;

-- Insert data into Applications
INSERT INTO Applications (App_No, StudentID, App_Year)
SELECT DISTINCT App_No, StudentID, App_Year
FROM Apps_NOT_Normalized;

-- Insert data into Referees
INSERT INTO Referees (ReferenceName, RefInstitution)
SELECT DISTINCT ReferenceName, RefInstitution
FROM Apps_NOT_Normalized;

-- Insert data into Application_Referees
INSERT INTO Application_Referees (App_No, App_Year, RefereeID, ReferenceStatement)
SELECT DISTINCT a.App_No, a.App_Year, r.RefereeID, a.ReferenceStatement
```

FROM Apps_NOT_Normalized a

JOIN Referees r ON a.ReferenceName = r.ReferenceName AND a.RefInstitution = r.RefInstitution;

-- Insert data into PriorSchools

INSERT INTO PriorSchools (SchoolName)

SELECT DISTINCT PriorSchoolAddr

FROM Apps_NOT_Normalized;

-- Insert data into Student_PriorSchools

INSERT INTO Student_PriorSchools (StudentID, PriorSchoolID, GPA)

SELECT DISTINCT a.StudentID, p.PriorSchoolID, a.GPA

FROM Apps_NOT_Normalized a

JOIN PriorSchools p ON a.PriorSchoolAddr = p.SchoolName;

TABLE STRUCTURE

apps_not_normalized 1 | Statistics 1

SELECT * FROM Apps_NOT_Norm | Enter a SQL expression to filter results (use Ctrl+Space)

	app_no	studentid	studentname	street	state	zipcode	app_year	referencename	refinstitution	referencestatement	priorschoolid	priorschooladdr	gpa
1	1	1	Mark	Grafton Street	New York	NY234	2,003	Dr. Jones	Trinity College	Good guy	1	Castleknock	65
2	1	1	Mark	Grafton Street	New York	NY234	2,004	Dr. Jones	Trinity College	Good guy	1	Castleknock	65
3	2	1	Mark	White Street	Florida	Flo435	2,007	Dr. Jones	Trinity College	Good guy	1	Castleknock	65
4	2	1	Mark	White Street	Florida	Flo435	2,007	Dr. Jones	Trinity College	Good guy	2	Loreto College	87
5	3	1	Mark	White Street	Florida	Flo435	2,012	Dr. Jones	U Limerick	Very Good guy	1	Castleknock	65
6	3	1	Mark	White Street	Florida	Flo435	2,012	Dr. Jones	U Limerick	Very Good guy	2	Loreto College	87
7	2	2	Sarah	Green Road	California	Cal123	2,010	Dr. Byrne	DIT	Perfect	1	Castleknock	90
8	2	2	Sarah	Green Road	California	Cal123	2,010	Dr. Byrne	DIT	Perfect	3	St. Patrick	76
9	2	2	Sarah	Green Road	California	Cal123	2,011	Dr. Byrne	DIT	Perfect	1	Castleknock	90
10	2	2	Sarah	Green Road	California	Cal123	2,011	Dr. Byrne	DIT	Perfect	3	St. Patrick	76
11	2	2	Sarah	Green Road	California	Cal123	2,012	Dr. Byrne	UCD	Average	1	Castleknock	90
12	2	2	Sarah	Green Road	California	Cal123	2,012	Dr. Byrne	UCD	Average	3	St. Patrick	76
13	2	2	Sarah	Green Road	California	Cal123	2,012	Dr. Byrne	UCD	Average	4	DBS	66
14	2	2	Sarah	Green Road	California	Cal123	2,012	Dr. Byrne	UCD	Average	5	Harvard	45
15	1	3	Paul	Red Crescent	Carolina	Ca455	2,012	Dr. Jones	Trinity College	Poor	1	Castleknock	45
16	1	3	Paul	Red Crescent	Carolina	Ca455	2,012	Dr. Jones	Trinity College	Poor	3	St. Patrick	67
17	1	3	Paul	Red Crescent	Carolina	Ca455	2,012	Dr. Jones	Trinity College	Poor	4	DBS	23
18	1	3	Paul	Red Crescent	Carolina	Ca455	2,012	Dr. Jones	Trinity College	Poor	5	Harvard	67
19	3	3	Paul	Yellow Park	Mexico	Mex1	2,008	Prof. Cahill	UCC	Excellent	1	Castleknock	45
20	3	3	Paul	Yellow Park	Mexico	Mex1	2,008	Prof. Cahill	UCC	Excellent	3	St. Patrick	67
21	3	3	Paul	Yellow Park	Mexico	Mex1	2,008	Prof. Cahill	UCC	Excellent	4	DBS	23
22	3	3	Paul	Yellow Park	Mexico	Mex1	2,008	Prof. Cahill	UCC	Excellent	5	Harvard	67
23	1	4	Jack	Dartry Road	Ohio	Oh34	2,009	Prof. Lillis	DIT	Fair	3	St. Patrick	29
24	1	4	Jack	Dartry Road	Ohio	Oh34	2,009	Prof. Lillis	DIT	Fair	4	DBS	88
25	1	4	Jack	Dartry Road	Ohio	Oh34	2,009	Prof. Lillis	DIT	Fair	5	Harvard	66
26	2	5	Mary	Malahide Road	Ireland	IRE	2,009	Prof. Lillis	DIT	Good girl	3	St. Patrick	44
27	2	5	Mary	Malahide Road	Ireland	IRE	2,009	Prof. Lillis	DIT	Good girl	4	DBS	55
28	2	5	Mary	Malahide Road	Ireland	IRE	2,009	Prof. Lillis	DIT	Good girl	5	Harvard	66
29	2	5	Mary	Malahide Road	Ireland	IRE	2,009	Prof. Lillis	DIT	Good girl	1	Castleknock	74
30	1	5	Mary	Black Bay	Kansas	Kan45	2,005	Dr. Byrne	DIT	Perfect	3	St. Patrick	44
31	1	5	Mary	Black Bay	Kansas	Kan45	2,005	Dr. Byrne	DIT	Perfect	4	DBS	55
32	1	5	Mary	Black Bay	Kansas	Kan45	2,005	Dr. Byrne	DIT	Perfect	5	Harvard	66
33	3	6	Susan	River Road	Kansas	Kan45	2,011	Prof. Cahill	UCC	Messy	1	Castleknock	88
34	3	6	Susan	River Road	Kansas	Kan45	2,011	Prof. Cahill	UCC	Messy	3	St. Patrick	77
35	3	6	Susan	River Road	Kansas	Kan45	2,011	Prof. Cahill	UCC	Messy	4	DBS	56
36	3	6	Susan	River Road	Kansas	Kan45	2,011	Prof. Cahill	UCC	Messy	2	Loreto College	45

-- Verification

SELECT 'Data Migration Complete' AS Status;

VERIFICATION OF IMPLEMENTATION

After executing the above SQL code, the migration should complete successfully, as indicated by the output: