UNIVERSITY INSTITUTE OF COMPUTING

PROJECT REPORT ON ONLINE GAMBLING MANAGEMENT SYSTEM

Program name: BCA

Subject name/Code: Data Management System (23CAT-251/23CAP-252)

Submitted by:

Name: Ashish

ID: 23BCA10458

Section: 23BCA- 4(B)

Submitted to:

Name: Mr. Arvinder Singh

ABSTRACT

The Gambling Module of the quiz application introduces a gamified element that enhances user engagement and interactivity. This module allows users to place bets using virtual coins with the opportunity to double their wagered amount based on a randomized outcome. By integrating this feature into the app, users are encouraged to interact more frequently, increasing retention and providing a dynamic experience.

The objective of this project is to simulate a basic betting mechanism where users can select from predefined coin values and bet against the app's randomized logic, which offers a 50% chance of winning. If the bet is successful, the user’s coin balance increases by the wagered amount; if unsuccessful, the coins are deducted. This gambling system is purely for entertainment purposes and is designed to motivate users to participate more in quizzes by earning and spending coins.

Key Features and Concepts:

- Allows users to gamble coins with a 50% win/loss probability.
- Gamification adds excitement and increases user interaction.
- Virtual coins system integrates with the overall quiz application.
- Dynamic feedback is provided using Flutter’s UI elements (e.g., SnackBars).
- Coin balance is updated in real-time based on betting results.
- Coin validation logic ensures that users cannot gamble more than they possess.
- Encourages users to participate in quizzes to earn coins for betting.

Through this project, we aimed to explore and implement concepts such as randomization, state management in Flutter, and secure backend integration with SQL databases. The successful development of this module contributes to an engaging and feature-rich quiz application that demonstrates how game mechanics can be employed to enhance learning apps.

Table of Content

Introduction

- **1.1 Overview of the Project**
    - The SQL file provides the structure for a database designed to manage the operations of a gambling website.
    - This database includes tables to handle user information, game details, betting activities, transaction records, and a support ticket system.
    - The system aims to streamline the management of data within an online gambling platform.

- **1.2 Types of Users**
    - The database is designed to accommodate two primary user types:
        - Regular Users: Individuals who participate in the games and use the platform's services. These users are recorded in the **Users** table.
        - Administrators: Personnel who manage the platform. The **Admins** table stores their information, with different roles such as **SuperAdmin**, **SupportManager**, and **GameManager** defined.

Objective

- **2.1 Purpose of the Project**
- The main purpose of this database project is to develop a system that can efficiently manage and organize the various data components of a gambling website.
- It seeks to replace manual or inefficient data handling methods with a centralized, database-driven approach.

- **2.2 Specific Goals**
- The project's goals, derived from the database structure and queries, include:
    - Efficiently storing and retrieving user account information.
    - Categorizing games and maintaining game-related data.
    - Recording and processing bet details and payouts.
    - Managing financial transactions (deposits and withdrawals).
    - Providing a system for users to submit and track support tickets.
    - Enabling data analysis to gain insights into game performance, user behavior, and financial trends.

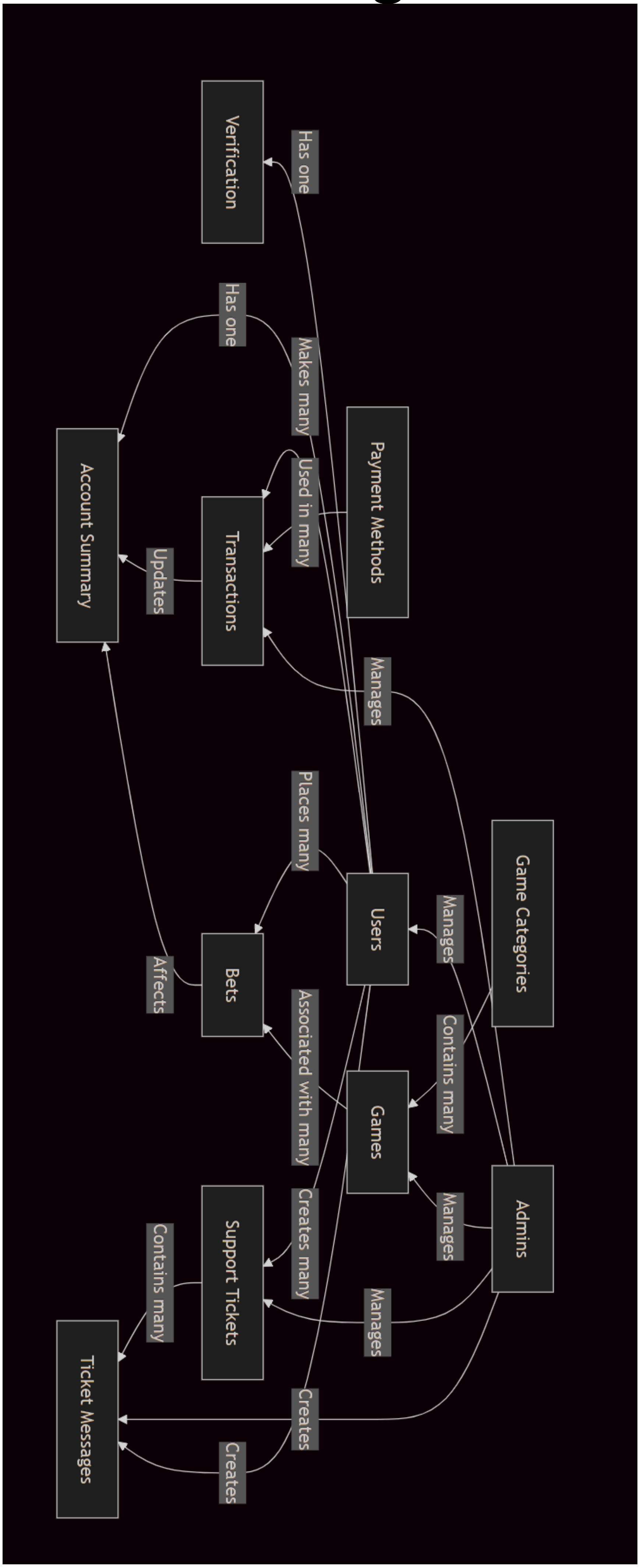
Designing

- **3.1 Entity-Relationship (ER) Diagram**
- The SQL file does not explicitly provide an ER diagram.
- However, the table structures imply the following entities and relationships:
    - Entities: **Users, Verification, GameCategories, Games, Bets, PaymentMethods, Transactions, AccountSummary, SupportTickets, TicketMessages, Admins.**
    - Relationships: For example, a **User** can place multiple **Bets**, a **Game** belongs to a **GameCategory**, and a **Transaction** is associated with a **User** and a **PaymentMethod**.

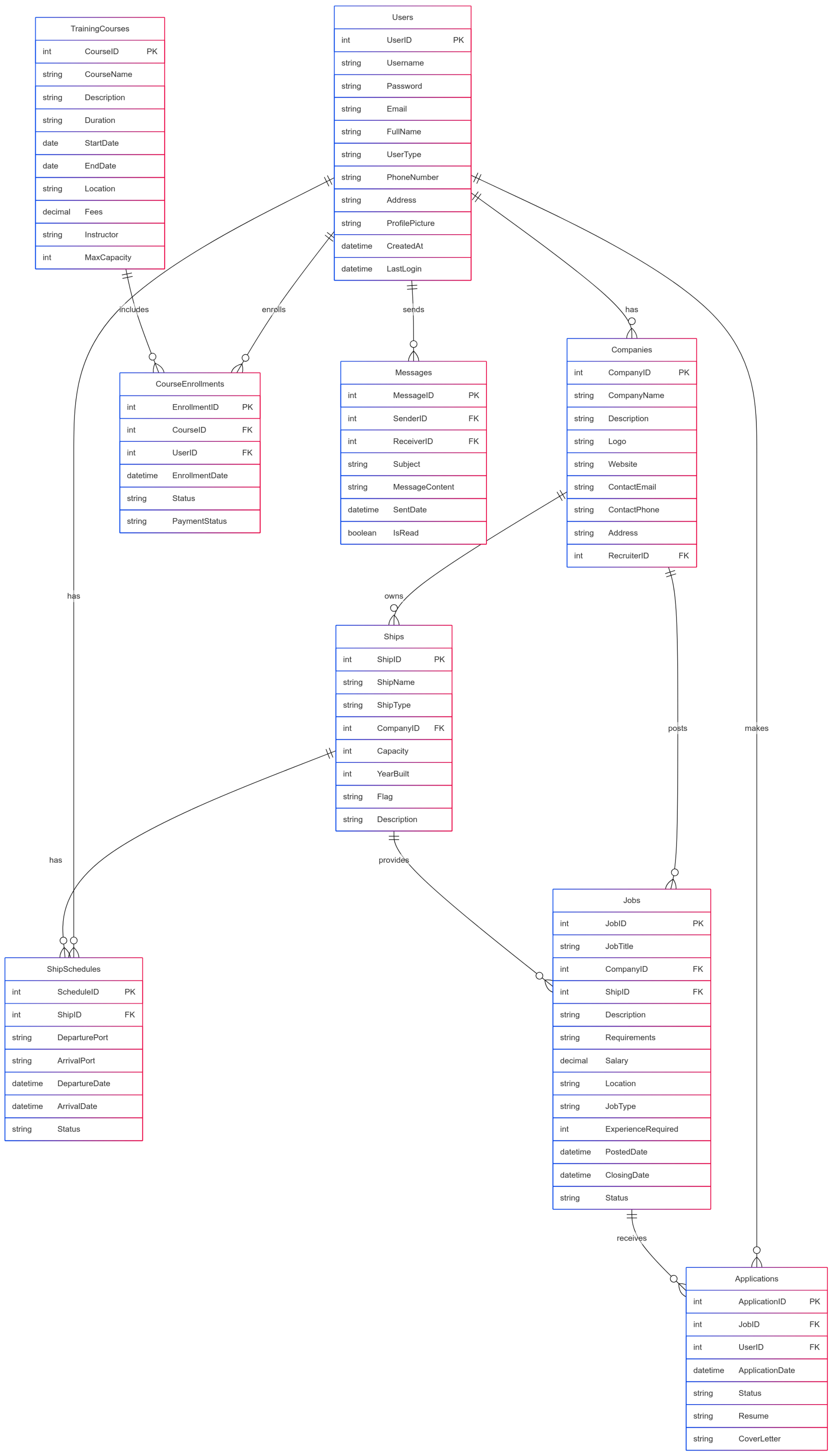
- **3.2 Relational Schema Diagram**
- A relational schema diagram is not present in the SQL file.
- The **CREATE TABLE** statements define the relational schema.

- **3.3 Relational Schema (Detailed Explanation)**
- The relational schema is defined by the **CREATE TABLE** statements.
- Each table consists of columns with specific data types and constraints:
    - **PRIMARY KEY**: Uniquely identifies each record in a table.
    - **FOREIGN KEY**: Establishes relationships between tables.
    - **NOT NULL**: Ensures that a column cannot contain a null value.
    - **UNIQUE**: Enforces uniqueness of values in a column.
    - **DEFAULT**: Specifies a default value for a column.

ER Diagram


Verification
Has one
Has one
Makes many
Payment Methods
Account Summary
Used in many
Transactions
Updates
Manages
Places many
Game Categories
Users
Manages
Bets
Affects
Associated with many
Contains many
Games
Admins
Manages
Support Tickets
Creates many
Contains many
Manages
Ticket Messages
Creates
Creates

Relational Schema Diagram



4. Database

The database is designed using the Relational Database Model to store and manage all the essential information for the gambling website. It consists of several tables, each representing a key entity within the system.

4.1 Tables with Sample Data

Here's a breakdown of the tables, their columns, data types, descriptions, and sample data:

1. Users

This table stores information about the registered users of the gambling website.

Coloumn Name	Data Type	Description
UserID	INT	Unique User ID (PK)
Username	VARCHAR(50)	VARCHAR(50)
Email	VARCHAR(100)	Email address
Phone	VARCHAR(20)	Phone number
PasswordHash	VARCHAR(255)	Hashed password
FirstName	VARCHAR(50)	First name
LastName	VARCHAR(50)	Last name
CountryCode	CHAR(2)	Country code
RegistrationDate	DATETIME	Registration date/time
LastLoginDate	DATETIME	Last login date/time

2. Verification

Coloumn Name	Data Type	Description
VerificationID	INT	Unique User ID (PK)
UserID	INT	User ID (FK)
PhoneVerified	BOOLEAN	Phone verified
OTPStatus	VARCHAR(20)	OTP status
IDType	VARCHAR(20)	Type of ID
IDNumber	VARCHAR(50)	ID number
IDVerified	BOOLEAN	ID verified
VerificationDate	DATETIME	Verification date

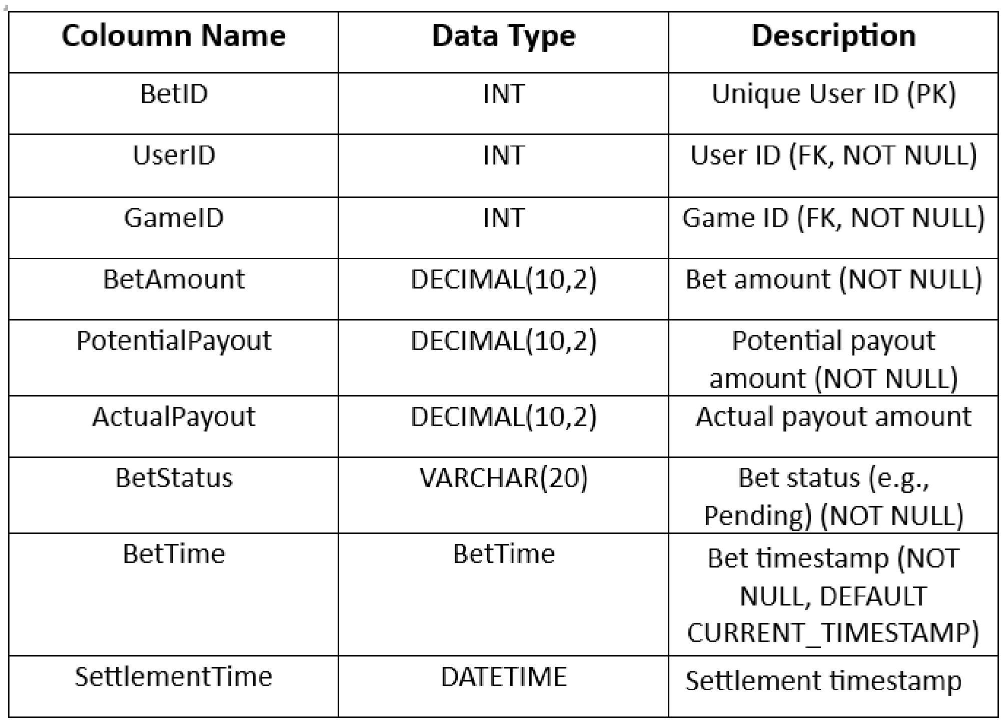
3. GameCategories

Coloumn Name	Data Type	Description
CategoryID	INT	Unique User ID (PK)
CategoryName	VARCHAR(50)	Name of the category (NOT NULL, UNIQUE)
Description	VARCHAR(255)	Category description

4. Games

Coloumn Name	Data Type	Description
GameID	INT	Unique User ID (PK)
CategoryID	INT	Category ID (FK, NOT NULL)
GameName	VARCHAR(100)	Name of the game (NOT NULL, UNIQUE)
Description	TEXT	Game description
MinBet	DECIMAL(10,2)	Minimum bet amount (NOT NULL)
MaxBet	DECIMAL(10,2)	Maximum bet amount (NOT NULL)
HouseEdge	DECIMAL(5,2)	House edge percentage (NOT NULL)
Status	VARCHAR(20)	Game status (e.g., Active) (NOT NULL, DEFAULT 'Active')

5. Bets



Coloumn Name	Data Type	Description
BetID	INT	Unique User ID (PK)
UserID	INT	User ID (FK, NOT NULL)
GameID	INT	Game ID (FK, NOT NULL)
BetAmount	DECIMAL(10,2)	Bet amount (NOT NULL)
PotentialPayout	DECIMAL(10,2)	Potential payout amount (NOT NULL)
ActualPayout	DECIMAL(10,2)	Actual payout amount
BetStatus	VARCHAR(20)	Bet status (e.g., Pending) (NOT NULL)
BetTime	BetTime	Bet timestamp (NOT NULL, DEFAULT CURRENT_TIMESTAMP)
SettlementTime	DATETIME	Settlement timestamp

5. Coding (Queries)

5.1 Table Creation Scripts

- The **CREATE TABLE** statements are used to create the database tables.
- These scripts define the schema of each table, including column names, data types, and constraints.

Code:-

```
-- Create database
CREATE DATABASE GamblingWebsite;
USE GamblingWebsite;

-- Users table
CREATE TABLE Users (
   UserID INT PRIMARY KEY AUTO_INCREMENT,
   Username VARCHAR(50) NOT NULL UNIQUE,
   Email VARCHAR(100) NOT NULL UNIQUE,
   Phone VARCHAR(20) NOT NULL UNIQUE,
 PasswordHash VARCHAR(255) NOT NULL,
 FirstName VARCHAR(50) NOT NULL,
 LastName VARCHAR(50) NOT NULL,
 CountryCode CHAR(2) NOT NULL,
 RegistrationDate DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
 LastLoginDate DATETIME,
 Status VARCHAR(20) NOT NULL DEFAULT 'Active' --
Active, Suspended, Banned
);

-- Verification table
CREATE TABLE Verification (
   VerificationID INT PRIMARY KEY AUTO_INCREMENT,
   UserID INT NOT NULL UNIQUE,
   PhoneVerified BOOLEAN NOT NULL DEFAULT 0,
   OTPStatus VARCHAR(20) DEFAULT 'Not Sent', -- Not
Sent, Sent, Verified
   IDType VARCHAR(20), -- Aadhar, Passport, Driver's
License, etc.
   IDNumber VARCHAR(50),
   IDVerified BOOLEAN NOT NULL DEFAULT 0,
   VerificationDate DATETIME,
   FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- Game Categories table
CREATE TABLE GameCategories (
   CategoryID INT PRIMARY KEY AUTO_INCREMENT,
   CategoryName VARCHAR(50) NOT NULL UNIQUE,
   Description VARCHAR(255)
);

-- Games table
CREATE TABLE Games (
   GameID INT PRIMARY KEY AUTO_INCREMENT,
   CategoryID INT NOT NULL,
 GameName VARCHAR(100) NOT NULL UNIQUE,
 Description TEXT,
 MinBet DECIMAL(10, 2) NOT NULL,
 MaxBet DECIMAL(10, 2) NOT NULL,
 HouseEdge DECIMAL(5, 2) NOT NULL, -- Percentage
 Status VARCHAR(20) NOT NULL DEFAULT 'Active', --
Active, Maintenance, Retired
 FOREIGN KEY (CategoryID) REFERENCES
GameCategories(CategoryID)
);

-- Bets table
CREATE TABLE Bets (
   BetID INT PRIMARY KEY AUTO_INCREMENT,
   UserID INT NOT NULL,
   GameID INT NOT NULL,
   BetAmount DECIMAL(10, 2) NOT NULL,
   PotentialPayout DECIMAL(10, 2) NOT NULL,
   ActualPayout DECIMAL(10, 2),
   BetStatus VARCHAR(20) NOT NULL, -- Pending, Won,
Lost, Cancelled
   BetTime DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
   SettlementTime DATETIME,
   FOREIGN KEY (UserID) REFERENCES Users(UserID),
   FOREIGN KEY (GameID) REFERENCES
Games(GameID)
);

-- Payment Methods table
CREATE TABLE PaymentMethods (
   PaymentMethodID INT PRIMARY KEY
AUTO_INCREMENT,
   MethodName VARCHAR(50) NOT NULL UNIQUE,
   Description VARCHAR(255)
);
```

```sql
-- Transactions table
CREATE TABLE Transactions (
 TransactionID INT PRIMARY KEY AUTO_INCREMENT,
 UserID INT NOT NULL,
 PaymentMethodID INT NOT NULL,
 TransactionType VARCHAR(20) NOT NULL, -- Deposit,
Withdrawal
 Amount DECIMAL(10, 2) NOT NULL,
 Status VARCHAR(20) NOT NULL, -- Pending,
Completed, Failed, Cancelled
 TransactionDate DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
 CompletionDate DATETIME,
 ReferenceNumber VARCHAR(100),
 FOREIGN KEY (UserID) REFERENCES Users(UserID),
 FOREIGN KEY (PaymentMethodID) REFERENCES
PaymentMethods(PaymentMethodID)
);

-- Account Summary table
CREATE TABLE AccountSummary (
    SummaryID INT PRIMARY KEY AUTO_INCREMENT,
    UserID INT NOT NULL UNIQUE,
    Balance DECIMAL(10, 2) NOT NULL DEFAULT 0,
    TotalWagered DECIMAL(10, 2) NOT NULL DEFAULT 0,
    TotalWon DECIMAL(10, 2) NOT NULL DEFAULT 0,
    TotalLost DECIMAL(10, 2) NOT NULL DEFAULT 0,
    LastUpdated DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- Support Tickets table
CREATE TABLE SupportTickets (
    TicketID INT PRIMARY KEY AUTO_INCREMENT,
    UserID INT NOT NULL,
 Subject VARCHAR(100) NOT NULL,
 Category VARCHAR(50) NOT NULL,
 Status VARCHAR(20) NOT NULL DEFAULT 'Open', --
Open, Closed, Pending
 Priority VARCHAR(20) NOT NULL DEFAULT 'Medium', --
Low, Medium, High, Urgent
 CreatedDate DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
 ClosedDate DATETIME,
 FOREIGN KEY (UserID) REFERENCES Users(UserID)
);

-- Ticket Messages table
CREATE TABLE TicketMessages (
    MessageID INT PRIMARY KEY AUTO_INCREMENT,
    TicketID INT NOT NULL,
    SenderType VARCHAR(10) NOT NULL, -- User,
Support, Admin
    SenderID INT NOT NULL, -- UserID or AdminID
    MessageText TEXT NOT NULL,
    SentDate DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    FOREIGN KEY (TicketID) REFERENCES
SupportTickets(TicketID)
);

-- Admins table
CREATE TABLE Admins (
    AdminID INT PRIMARY KEY AUTO_INCREMENT,
    Username VARCHAR(50) NOT NULL UNIQUE,
    PasswordHash VARCHAR(255) NOT NULL,
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL UNIQUE,
    Role VARCHAR(50) NOT NULL, -- SuperAdmin,
GameManager, SupportManager, etc.
 Status VARCHAR(20) NOT NULL DEFAULT 'Active', --
Active, Inactive
 LastLoginDate DATETIME
);
```

5.2 Data Insertion Queries

- The **INSERT INTO** statements are used to insert sample data into the tables.
- These queries populate the database with initial data for testing and demonstration purposes.

Values:-

```
-- Insert users (Mix of Indian and American names)
INSERT INTO Users (Username, Email, Phone, PasswordHash, FirstName,
LastName, CountryCode) VALUES
('john_smith', 'john.smith@email.com', '+1234567890', 'hash1', 'John', 'Smith',
'US'),
('priya_patel', 'priya.patel@email.com', '+919876543210', 'hash2', 'Priya', 'Patel',
'IN'),
('mike_johnson', 'mike.j@email.com', '+1987654321', 'hash3', 'Michael',
'Johnson', 'US'),
('raj_kumar', 'raj.kumar@email.com', '+919876543211', 'hash4', 'Raj', 'Kumar',
'IN'),
('sarah_wilson', 'sarah.w@email.com', '+1234567891', 'hash5', 'Sarah', 'Wilson',
'US'),
('anita_sharma', 'anita.s@email.com', '+919876543212', 'hash6', 'Anita', 'Sharma',
'IN'),
('david_brown', 'david.b@email.com', '+1234567892', 'hash7', 'David', 'Brown',
'US'),
('amit_verma', 'amit.v@email.com', '+919876543213', 'hash8', 'Amit', 'Verma',
'IN'),
('emma_davis', 'emma.d@email.com', '+1234567893', 'hash9', 'Emma', 'Davis',
'US'),
('neha_gupta', 'neha.g@email.com', '+919876543214', 'hash10', 'Neha', 'Gupta',
'IN');
```

```sql
-- Insert verification records
INSERT INTO Verification (UserID, PhoneVerified, OTPStatus, IDType, IDNumber)
VALUES
(1, 1, 'Verified', 'Passport', 'US123456'),
(2, 1, 'Verified', 'Aadhar', '1234-5678-9012'),
(3, 1, 'Verified', 'Driver License', 'DL789012'),
(4, 1, 'Verified', 'Aadhar', '2345-6789-0123'),
(5, 0, 'Sent', 'Passport', 'US234567'),
(6, 1, 'Verified', 'Aadhar', '3456-7890-1234'),
(7, 1, 'Verified', 'Driver License', 'DL890123'),
(8, 1, 'Verified', 'Aadhar', '4567-8901-2345'),
(9, 0, 'Not Sent', 'Passport', 'US345678'),
(10, 1, 'Verified', 'Aadhar', '5678-9012-3456');

-- Insert game categories
INSERT INTO GameCategories (CategoryName, Description) VALUES
('Slots', 'Virtual slot machine games'),
('Poker', 'Various poker variants'),
('Roulette', 'Classic roulette games'),
('Blackjack', 'Traditional blackjack games'),
('Teen Patti', 'Popular Indian card game'),
('Andar Bahar', 'Traditional Indian card game'),
('Baccarat', 'Classic casino card game'),
('Craps', 'Dice-based casino game'),
('Lottery', 'Various lottery games'),
('Sports Betting', 'Betting on sports events');

-- Insert games
INSERT INTO Games (CategoryID, GameName, Description, MinBet, MaxBet,
HouseEdge) VALUES
(1, 'Lucky Sevens', 'Classic slot machine', 1.00, 100.00, 3.50),
(2, 'Texas Hold''em', 'Popular poker variant', 5.00, 500.00, 2.50),
(3, 'European Roulette', 'Single-zero roulette', 1.00, 1000.00, 2.70),
(4, 'Classic Blackjack', 'Traditional blackjack', 5.00, 200.00, 0.50),
(5, 'Teen Patti Pro', 'Modern teen patti', 2.00, 200.00, 2.00),
(6, 'Quick Andar Bahar', 'Fast-paced card game', 1.00, 100.00, 2.50),
(7, 'Speed Baccarat', 'Fast baccarat variant', 5.00, 500.00, 1.20),
(8, 'Street Craps', 'Classic dice game', 2.00, 200.00, 1.40),
(9, 'Power Ball', 'Lottery game', 1.00, 50.00, 5.00),
(10, 'Cricket Betting', 'Cricket match betting', 1.00, 1000.00, 4.00);
```

```sql
-- Insert payment methods
INSERT INTO PaymentMethods (MethodName, Description) VALUES
('Credit Card', 'Major credit cards accepted'),
('UPI', 'Indian UPI payments'),
('PayPal', 'PayPal payment system'),
('Net Banking', 'Indian bank transfers'),
('Google Pay', 'Google payment service'),
('Venmo', 'US digital wallet'),
('PhonePe', 'Indian digital wallet'),
('Bank Transfer', 'Direct bank transfers'),
('Cryptocurrency', 'Bitcoin and other cryptos'),
('Apple Pay', 'Apple payment service');

-- Insert bets
INSERT INTO Bets (UserID, GameID, BetAmount, PotentialPayout, BetStatus)
VALUES
(1, 1, 10.00, 20.00, 'Pending'),
(2, 5, 50.00, 95.00, 'Won'),
(3, 2, 25.00, 45.00, 'Lost'),
(4, 6, 15.00, 30.00, 'Pending'),
(5, 3, 100.00, 190.00, 'Won'),
(6, 4, 20.00, 38.00, 'Lost'),
(7, 7, 75.00, 140.00, 'Pending'),
(8, 8, 30.00, 55.00, 'Won'),
(9, 9, 5.00, 25.00, 'Lost'),
(10, 10, 200.00, 380.00, 'Pending');

-- Insert transactions
INSERT INTO Transactions (UserID, PaymentMethodID, TransactionType, Amount,
Status) VALUES
(1, 1, 'Deposit', 100.00, 'Completed'),
(2, 2, 'Deposit', 500.00, 'Completed'),
(3, 3, 'Withdrawal', 200.00, 'Pending'),
(4, 4, 'Deposit', 1000.00, 'Completed'),
(5, 5, 'Withdrawal', 300.00, 'Completed'),
(6, 6, 'Deposit', 250.00, 'Completed'),
(7, 7, 'Withdrawal', 150.00, 'Pending'),
(8, 8, 'Deposit', 400.00, 'Completed'),
(9, 9, 'Deposit', 600.00, 'Failed'),
(10, 10, 'Withdrawal', 450.00, 'Completed');
```

```sql
-- Insert account summaries
INSERT INTO AccountSummary (UserID, Balance, TotalWagered, TotalWon,
TotalLost) VALUES
(1, 100.00, 50.00, 30.00, 20.00),
(2, 450.00, 200.00, 150.00, 50.00),
(3, 75.00, 100.00, 25.00, 75.00),
(4, 1000.00, 300.00, 200.00, 100.00),
(5, 250.00, 150.00, 100.00, 50.00),
(6, 180.00, 120.00, 60.00, 60.00),
(7, 325.00, 200.00, 125.00, 75.00),
(8, 400.00, 250.00, 200.00, 50.00),
(9, 600.00, 300.00, 150.00, 150.00),
(10, 550.00, 400.00, 250.00, 150.00);

-- Insert support tickets
INSERT INTO SupportTickets (UserID, Subject, Category, Priority) VALUES
(1, 'Withdrawal Issue', 'Payment', 'High'),
(2, 'Game Frozen', 'Technical', 'Medium'),
(3, 'Account Verification', 'Account', 'Low'),
(4, 'Bonus Not Received', 'Promotion', 'Medium'),
(5, 'Login Problems', 'Technical', 'High'),
(6, 'Game Rules Query', 'Support', 'Low'),
(7, 'Payment Failed', 'Payment', 'High'),
(8, 'Account Locked', 'Account', 'High'),
(9, 'Deposit Issue', 'Payment', 'Medium'),
(10, 'Game Feedback', 'Support', 'Low');

-- Insert ticket messages
INSERT INTO TicketMessages (TicketID, SenderType, SenderID, MessageText)
VALUES
(1, 'User', 1, 'My withdrawal is stuck for 2 days'),
(2, 'Support', 1, 'Please clear cache and try again'),
(3, 'User', 3, 'Need help with verification'),
(4, 'Support', 2, 'Checking bonus eligibility'),
(5, 'User', 5, 'Cannot access my account'),
(6, 'Support', 3, 'Here are the detailed rules'),
(7, 'User', 7, 'Payment transaction failed twice'),
(8, 'Support', 4, 'Account unlocked now'),
(9, 'User', 9, 'Deposit not showing in balance'),
(10, 'Support', 5, 'Thank you for your feedback');
```

```
-- Insert admins
INSERT INTO Admins (Username, PasswordHash, FirstName, LastName, Email,
Role) VALUES
('admin_john', 'hash1', 'John', 'Anderson', 'john.a@gambling.com', 'SuperAdmin'),
('admin_priya', 'hash2', 'Priya', 'Reddy', 'priya.r@gambling.com', 'SupportManager'),
('admin_mike', 'hash3', 'Michael', 'Clark', 'mike.c@gambling.com', 'GameManager'),
('admin_raj', 'hash4', 'Raj', 'Malhotra', 'raj.m@gambling.com', 'PaymentManager'),
('admin_sarah', 'hash5', 'Sarah', 'Thompson', 'sarah.t@gambling.com',
'SecurityManager'),
('admin_amit', 'hash6', 'Amit', 'Singh', 'amit.s@gambling.com', 'SupportAgent'),
('admin_lisa', 'hash7', 'Lisa', 'Brown', 'lisa.b@gambling.com', 'GameManager'),
('admin_vikram', 'hash8', 'Vikram', 'Mehta', 'vikram.m@gambling.com',
'PaymentAgent'),
('admin_emma', 'hash9', 'Emma', 'Wilson', 'emma.w@gambling.com',
'SupportAgent'),
('admin_rahul', 'hash10', 'Rahul', 'Sharma', 'rahul.s@gambling.com',
'SecurityAgent');
```

- **5.3 SELECT and JOIN Queries**
- The SQL file contains a set of **SELECT** queries that demonstrate how to retrieve and analyze data from the database.
- These queries use **JOIN** operations to combine data from multiple tables, filter data using **WHERE** clauses, group data using **GROUP BY**, and order data using **ORDER BY**.

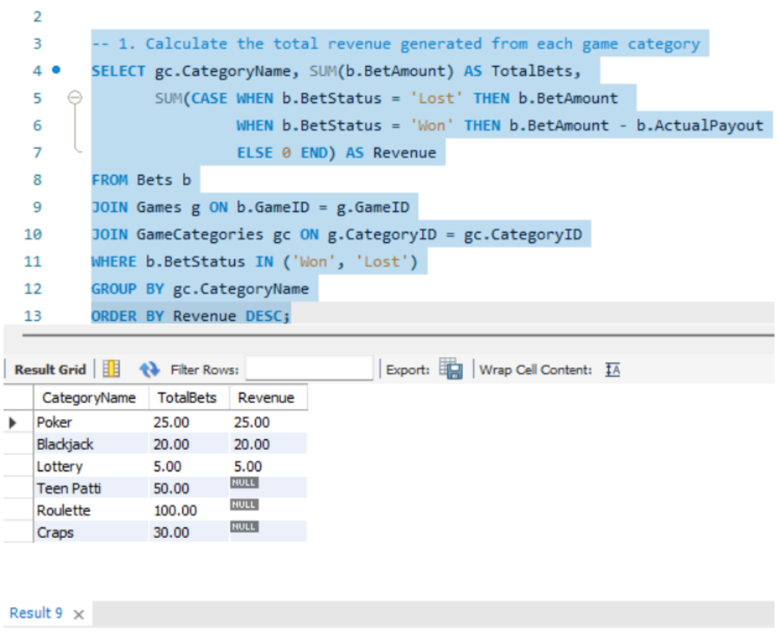
```
USE GamblingWebsite;

-- 1. Calculate the total revenue generated from each game category
SELECT gc.CategoryName, SUM(b.BetAmount) AS TotalBets,
    SUM(CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
          WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
          ELSE 0 END) AS Revenue
FROM Bets b
JOIN Games g ON b.GameID = g.GameID
JOIN GameCategories gc ON g.CategoryID = gc.CategoryID
WHERE b.BetStatus IN ('Won', 'Lost')
GROUP BY gc.CategoryName
ORDER BY Revenue DESC;
```

```
-- 2. Get the most active games based on bet count in the last 30 days
SELECT g.GameName, COUNT(b.BetID) AS BetCount, SUM(b.BetAmount) AS
TotalWagered
FROM Games g
JOIN Bets b ON g.GameID = b.GameID
WHERE b.BetTime >= DATE_SUB(CURRENT_TIMESTAMP, INTERVAL 30 DAY)
GROUP BY g.GameName
ORDER BY BetCount DESC
LIMIT 5;

-- 3. Get all open support tickets with high priority and their latest message
SELECT st.TicketID, u.Username, st.Subject, st.Priority, st.CreatedDate,
      (SELECT tm.MessageText
       FROM TicketMessages tm
       WHERE tm.TicketID = st.TicketID
       ORDER BY tm.SentDate DESC
       LIMIT 1) AS LatestMessage
FROM SupportTickets st
JOIN Users u ON st.UserID = u.UserID
WHERE st.Status = 'Open' AND st.Priority IN ('High', 'Urgent')
ORDER BY st.Priority, st.CreatedDate;

-- 4. Find payment methods with the highest transaction volumes
SELECT pm.MethodName,
      COUNT(t.TransactionID) AS TransactionCount,
      SUM(t.Amount) AS TotalVolume,
      SUM(CASE WHEN t.TransactionType = 'Deposit' THEN t.Amount ELSE 0 END)
AS TotalDeposits,
      SUM(CASE WHEN t.TransactionType = 'Withdrawal' THEN t.Amount ELSE 0
END) AS TotalWithdrawals
FROM PaymentMethods pm
JOIN Transactions t ON pm.PaymentMethodID = t.PaymentMethodID
WHERE t.Status = 'Completed'
GROUP BY pm.MethodName
ORDER BY TotalVolume DESC;
```

```
-- 5. Get a summary of admin activities by role
SELECT a.Role, COUNT(DISTINCT a.AdminID) AS AdminCount,
      COUNT(tm.MessageID) AS SupportMessages
FROM Admins a
LEFT JOIN TicketMessages tm ON a.AdminID = tm.SenderID AND tm.SenderType
= 'Admin'
GROUP BY a.Role
ORDER BY SupportMessages DESC;

-- 6. Find games with the highest house edge and their performance
SELECT g.GameName, g.HouseEdge,
      COUNT(b.BetID) AS TotalBets,
      SUM(b.BetAmount) AS TotalWagered,
      SUM(CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
              WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
              ELSE 0 END) AS ActualProfit,
      (SUM(CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
               WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
               ELSE 0 END) / SUM(b.BetAmount)) * 100 AS ActualMargin
FROM Games g
LEFT JOIN Bets b ON g.GameID = b.GameID
WHERE b.BetStatus IN ('Won', 'Lost')
GROUP BY g.GameName, g.HouseEdge
ORDER BY g.HouseEdge DESC;

-- 7. Find users who have verification issues (phone verified but ID not verified)
SELECT u.UserID, u.Username, u.Email, u.Phone, u.CountryCode,
      v.PhoneVerified, v.OTPStatus, v.IDType, v.IDVerified
FROM Users u
JOIN Verification v ON u.UserID = v.UserID
WHERE v.PhoneVerified = 1 AND v.IDVerified = 0
ORDER BY u.RegistrationDate;

-- 8. Get a comprehensive user activity report
SELECT u.Username, u.CountryCode,
      a.Balance, a.TotalWagered, a.TotalWon, a.TotalLost,
      COUNT(DISTINCT b.BetID) AS BetCount,
      COUNT(DISTINCT t.TransactionID) AS TransactionCount,
 COUNT(DISTINCT st.TicketID) AS TicketCount,
 MAX(b.BetTime) AS LastBetDate,
 MAX(t.TransactionDate) AS LastTransactionDate,
 u.LastLoginDate
FROM Users u
LEFT JOIN AccountSummary a ON u.UserID = a.UserID
LEFT JOIN Bets b ON u.UserID = b.UserID
LEFT JOIN Transactions t ON u.UserID = t.UserID
LEFT JOIN SupportTickets st ON u.UserID = st.UserID
GROUP BY u.Username, u.CountryCode, a.Balance, a.TotalWagered, a.TotalWon,
a.TotalLost, u.LastLoginDate
ORDER BY a.TotalWagered DESC;

-- 9. Identify country-specific game preferences
SELECT u.CountryCode, gc.CategoryName, COUNT(b.BetID) AS BetCount,
       ROUND((COUNT(b.BetID) * 100.0 /
             (SELECT COUNT(*) FROM Bets b2
              JOIN Users u2 ON b2.UserID = u2.UserID
              WHERE u2.CountryCode = u.CountryCode)), 2) AS
PercentageOfCountryBets
FROM Users u
JOIN Bets b ON u.UserID = b.UserID
JOIN Games g ON b.GameID = g.GameID
JOIN GameCategories gc ON g.CategoryID = gc.CategoryID
GROUP BY u.CountryCode, gc.CategoryName
HAVING COUNT(b.BetID) > 0
ORDER BY u.CountryCode, BetCount DESC;

-- 10. Analyze the relationship between game house edge and actual profit
margin
SELECT
    g.HouseEdge AS TheoreticalEdge,
    ROUND(AVG((CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
                    WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
                    ELSE 0 END) / b.BetAmount * 100), 2) AS ActualMargin,
    ROUND(AVG((CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
                    WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
                    ELSE 0 END) / b.BetAmount * 100) - g.HouseEdge, 2) AS
```

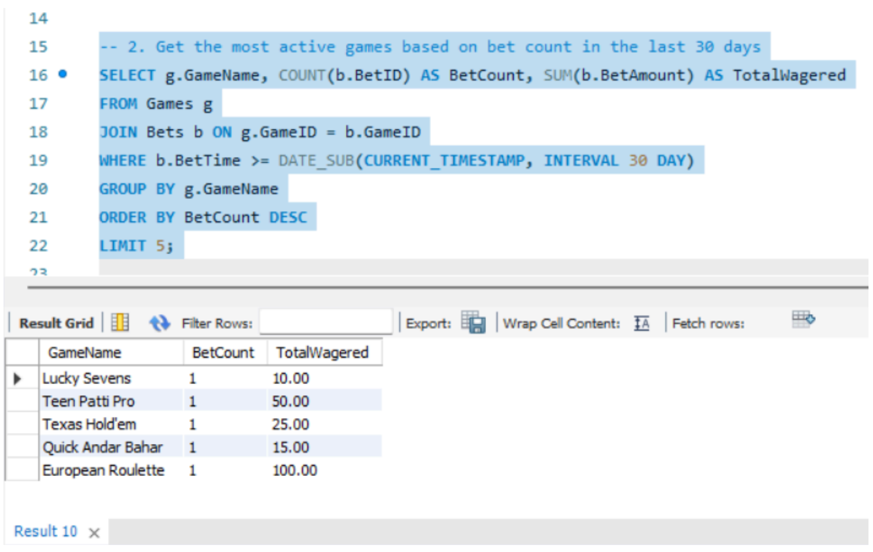
```
MarginDifference,
 COUNT(DISTINCT g.GameID) AS GameCount,
 SUM(b.BetAmount) AS TotalWagered
FROM Games g
JOIN Bets b ON g.GameID = b.GameID
WHERE b.BetStatus IN ('Won', 'Lost')
GROUP BY g.HouseEdge
ORDER BY g.HouseEdge;

-- 11. Calculate the average response and resolution time for support tickets by
category
SELECT
    st.Category,
    COUNT(st.TicketID) AS TotalTickets,
    AVG(TIMESTAMPDIFF(MINUTE, st.CreatedDate,
        (SELECT MIN(tm.SentDate)
         FROM TicketMessages tm
         WHERE tm.TicketID = st.TicketID AND tm.SenderType IN ('Support',
'Admin')))) AS AvgFirstResponseMinutes,
    AVG(CASE WHEN st.Status = 'Closed'
            THEN TIMESTAMPDIFF(HOUR, st.CreatedDate, st.ClosedDate)
            ELSE NULL END) AS AvgResolutionHours,
    COUNT(CASE WHEN st.Status = 'Closed' THEN 1 ELSE NULL END) AS
ClosedTickets,
    (COUNT(CASE WHEN st.Status = 'Closed' THEN 1 ELSE NULL END) * 100.0 /
COUNT(st.TicketID)) AS ClosureRate
FROM SupportTickets st
GROUP BY st.Category
HAVING COUNT(st.TicketID) > 0
ORDER BY AvgFirstResponseMinutes;

-- 12. Find the most profitable time periods for the platform (hourly analysis)
SELECT
    HOUR(b.BetTime) AS HourOfDay,
    COUNT(b.BetID) AS TotalBets,
    SUM(b.BetAmount) AS TotalWagered,
    SUM(CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
            WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
            ELSE 0 END) AS Revenue,
(SUM(CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
 WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
 ELSE 0 END) / SUM(b.BetAmount) * 100) AS ProfitMargin,
 COUNT(DISTINCT u.UserID) AS UniqueUsers
FROM Bets b
JOIN Users u ON b.UserID = u.UserID
WHERE b.BetStatus IN ('Won', 'Lost')
GROUP BY HOUR(b.BetTime)
ORDER BY Revenue DESC;

-- 13. Analyze cross-selling opportunities (users who play multiple game
categories)
WITH UserCategoryCounts AS (
    SELECT
        u.UserID,
        u.Username,
        COUNT(DISTINCT g.CategoryID) AS CategoryCount,
        GROUP_CONCAT(DISTINCT gc.CategoryName ORDER BY gc.CategoryName
SEPARATOR ', ') AS Categories
    FROM Users u
    JOIN Bets b ON u.UserID = b.UserID
    JOIN Games g ON b.GameID = g.GameID
    JOIN GameCategories gc ON g.CategoryID = gc.CategoryID
    GROUP BY u.UserID, u.Username
)

SELECT
    CategoryCount,
    COUNT(UserID) AS UserCount,
    (COUNT(UserID) * 100.0 / (SELECT COUNT(DISTINCT UserID) FROM Bets)) AS
PercentageOfUsers,
    Categories
FROM UserCategoryCounts
GROUP BY CategoryCount, Categories
ORDER BY CategoryCount DESC;
```

Outputs



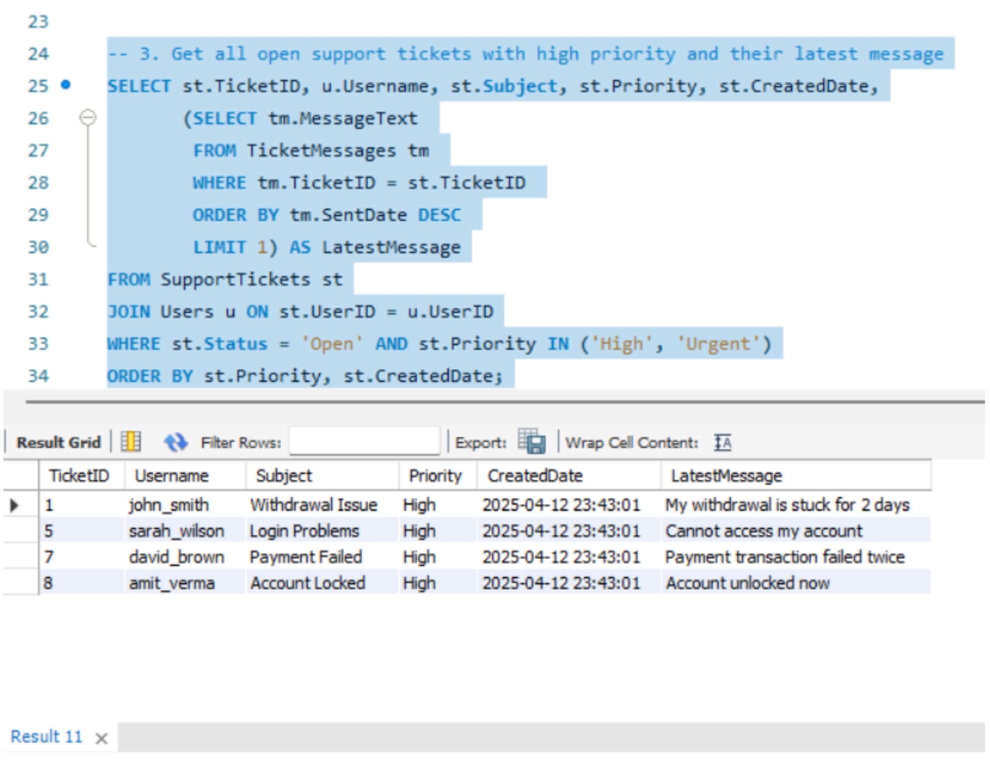
```sql
-- 1. Calculate the total revenue generated from each game category
SELECT gc.CategoryName, SUM(b.BetAmount) AS TotalBets,
       SUM(CASE WHEN b.BetStatus = 'Lost' THEN b.BetAmount
                WHEN b.BetStatus = 'Won' THEN b.BetAmount - b.ActualPayout
                ELSE 0 END) AS Revenue
FROM Bets b
JOIN Games g ON b.GameID = g.GameID
JOIN GameCategories gc ON g.CategoryID = gc.CategoryID
WHERE b.BetStatus IN ('Won', 'Lost')
GROUP BY gc.CategoryName
ORDER BY Revenue DESC;
```

CategoryName	TotalBets	Revenue
Poker	25.00	25.00
Blackjack	20.00	20.00
Lottery	5.00	5.00
Teen Patti	50.00	NULL
Roulette	100.00	NULL
Craps	30.00	NULL



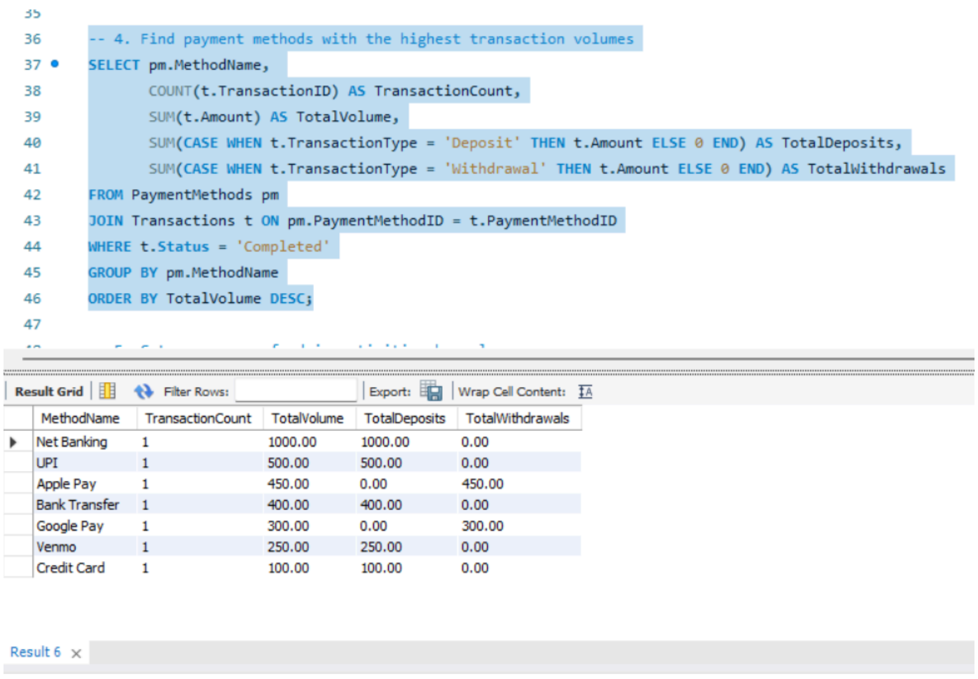
```sql
-- 2. Get the most active games based on bet count in the last 30 days
SELECT g.GameName, COUNT(b.BetID) AS BetCount, SUM(b.BetAmount) AS TotalWagered
FROM Games g
JOIN Bets b ON g.GameID = b.GameID
WHERE b.BetTime >= DATE_SUB(CURRENT_TIMESTAMP, INTERVAL 30 DAY)
GROUP BY g.GameName
ORDER BY BetCount DESC
LIMIT 5;
```

GameName	BetCount	TotalWagered
Lucky Sevens	1	10.00
Teen Patti Pro	1	50.00
Texas Hold'em	1	25.00
Quick Andar Bahar	1	15.00
European Roulette	1	100.00



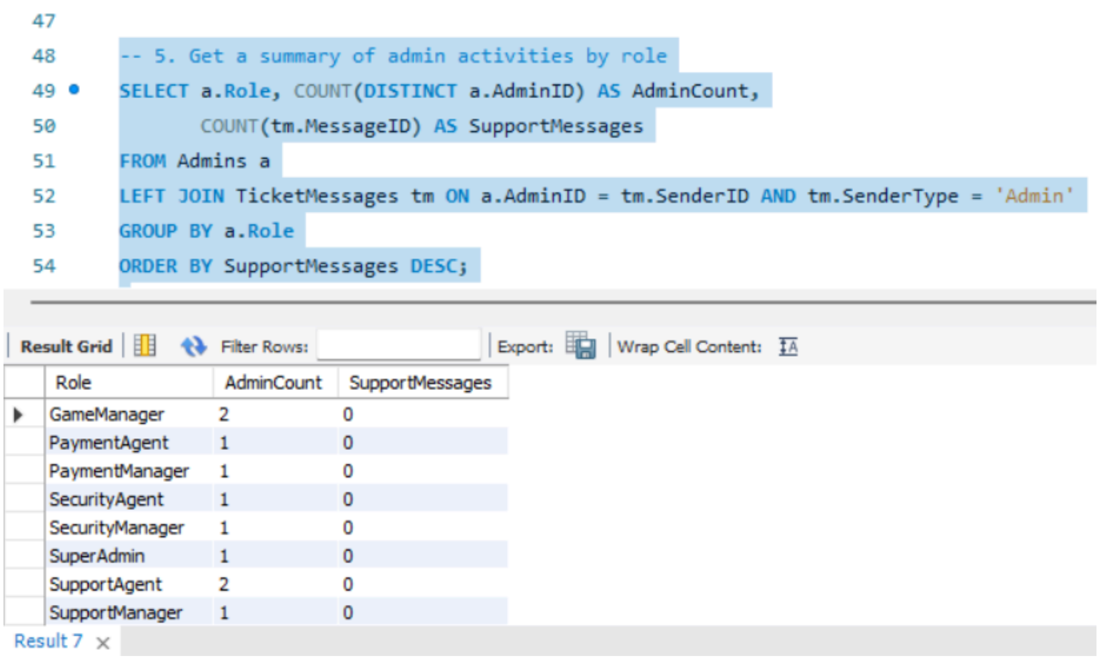
```sql
-- 3. Get all open support tickets with high priority and their latest message
SELECT st.TicketID, u.Username, st.Subject, st.Priority, st.CreatedDate,
       (SELECT tm.MessageText
        FROM TicketMessages tm
        WHERE tm.TicketID = st.TicketID
        ORDER BY tm.SentDate DESC
        LIMIT 1) AS LatestMessage
FROM SupportTickets st
JOIN Users u ON st.UserID = u.UserID
WHERE st.Status = 'Open' AND st.Priority IN ('High', 'Urgent')
ORDER BY st.Priority, st.CreatedDate;
```

TicketID	Username	Subject	Priority	CreatedDate	LatestMessage
1	john_smith	Withdrawal Issue	High	2025-04-12 23:43:01	My withdrawal is stuck for 2 days
5	sarah_wilson	Login Problems	High	2025-04-12 23:43:01	Cannot access my account
7	david_brown	Payment Failed	High	2025-04-12 23:43:01	Payment transaction failed twice
8	amit_verma	Account Locked	High	2025-04-12 23:43:01	Account unlocked now



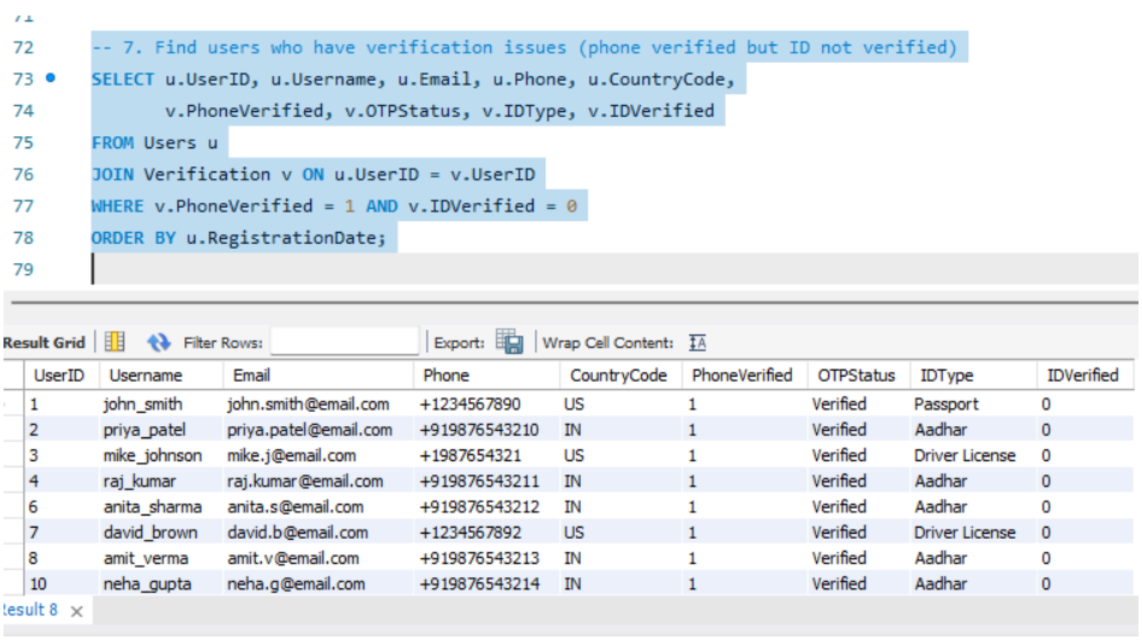
```sql
-- 4. Find payment methods with the highest transaction volumes
SELECT pm.MethodName,
       COUNT(t.TransactionID) AS TransactionCount,
       SUM(t.Amount) AS TotalVolume,
       SUM(CASE WHEN t.TransactionType = 'Deposit' THEN t.Amount ELSE 0 END) AS TotalDeposits,
       SUM(CASE WHEN t.TransactionType = 'Withdrawal' THEN t.Amount ELSE 0 END) AS TotalWithdrawals
FROM PaymentMethods pm
JOIN Transactions t ON pm.PaymentMethodID = t.PaymentMethodID
WHERE t.Status = 'Completed'
GROUP BY pm.MethodName
ORDER BY TotalVolume DESC;
```

MethodName	TransactionCount	TotalVolume	TotalDeposits	TotalWithdrawals
Net Banking	1	1000.00	1000.00	0.00
UPI	1	500.00	500.00	0.00
Apple Pay	1	450.00	0.00	450.00
Bank Transfer	1	400.00	400.00	0.00
Google Pay	1	300.00	0.00	300.00
Venmo	1	250.00	250.00	0.00
Credit Card	1	100.00	100.00	0.00



```sql
-- 5. Get a summary of admin activities by role
SELECT a.Role, COUNT(DISTINCT a.AdminID) AS AdminCount,
       COUNT(tm.MessageID) AS SupportMessages
FROM Admins a
LEFT JOIN TicketMessages tm ON a.AdminID = tm.SenderID AND tm.SenderType = 'Admin'
GROUP BY a.Role
ORDER BY SupportMessages DESC;
```

Role	AdminCount	SupportMessages
GameManager	2	0
PaymentAgent	1	0
PaymentManager	1	0
SecurityAgent	1	0
SecurityManager	1	0
SuperAdmin	1	0
SupportAgent	2	0
SupportManager	1	0



```sql
-- 7. Find users who have verification issues (phone verified but ID not verified)
SELECT u.UserID, u.Username, u.Email, u.Phone, u.CountryCode,
       v.PhoneVerified, v.OTPStatus, v.IDType, v.IDVerified
FROM Users u
JOIN Verification v ON u.UserID = v.UserID
WHERE v.PhoneVerified = 1 AND v.IDVerified = 0
ORDER BY u.RegistrationDate;
```

UserID	Username	Email	Phone	CountryCode	PhoneVerified	OTPStatus	IDType	IDVerified
1	john_smith	john.smith@email.com	+1234567890	US	1	Verified	Passport	0
2	priya_patel	priya.patel@email.com	+919876543210	IN	1	Verified	Aadhar	0
3	mike_johnson	mike.j@email.com	+1987654321	US	1	Verified	Driver License	0
4	raj_kumar	raj.kumar@email.com	+919876543211	IN	1	Verified	Aadhar	0
6	anita_sharma	anita.s@email.com	+919876543212	IN	1	Verified	Aadhar	0
7	david_brown	david.b@email.com	+1234567892	US	1	Verified	Driver License	0
8	amit_verma	amit.v@email.com	+919876543213	IN	1	Verified	Aadhar	0
10	neha_gupta	neha.g@email.com	+919876543214	IN	1	Verified	Aadhar	0

6. Result

- 6.1 Query Output Demonstration
- This section would present the results of executing the SQL queries from section 5.3.
- It would show the tables returned by the SELECT statements, demonstrating the data retrieval and analysis capabilities of the database.

- **6.2 Functional Output Summary**
- This section would provide a summary of the functionality achieved through the queries.
- For example:
- Queries to analyze game revenue and identify top-performing games.
- Queries to track user activity and identify trends.
- Queries to generate reports on financial transactions and support tickets.

7. Summary

- The Gambling Management System is a structured database project designed to manage and track data related to gambling events, participants, employees, betting activities, and associated financial transactions. The system aims to streamline operations by providing a centralized platform to store, retrieve, and manage information in a secure and efficient manner.

- Built using SQL and following systematic database design principles, the project incorporates key elements such as table relationships, primary and foreign keys, normalization, and query optimization. It includes tables for users, employees, customers, games, bets, and results, enabling detailed tracking of every aspect of the gambling process

- This system allows for the management of user roles (admin, manager, employee), smooth handling of betting transactions, recording of outcomes, and analysis of data for performance and accountability. It is especially useful for organizations managing legal gambling events and looking to maintain transparency and compliance with regulations.

Key Highlights:

- ER diagram and relational schema for clear database structure.
- Multiple interconnected tables to handle various entities in the gambling system.
- SQL scripts for table creation, data insertion, and advanced queries.
- Realistic sample data to demonstrate functionality.
- Normalization and relationship constraints to ensure data integrity.
- JOIN queries to fetch meaningful insights from multiple tables.

7. Conclusion

The Gambling Management System project has been successfully developed to offer an efficient and structured solution for managing gambling-related data. By employing a relational database model, the system ensures that all entities—such as users, employees, customers, games, and bets—are interlinked in a meaningful and organized way. The use of SQL for database creation and manipulation has provided a strong foundation for managing large volumes of data with ease and precision.

The project emphasizes data integrity, security, and efficiency through normalization and the use of primary and foreign keys. Queries have been constructed to handle real-world operations, including tracking user activity, monitoring bet outcomes, and generating insightful reports. This ensures transparency and supports responsible gambling practices.

In conclusion, the Gambling Management System not only meets the goals set out at the beginning of the project but also demonstrates the practical application of database principles in real-life scenarios. It serves as a scalable and reliable solution that can be adapted to more complex systems in the future.