10/21/2024

MY SQL PROJECT

ER DIAGRAMS

K0THAPALLI THRIVENI
BESANT TECHNOLOGIES

Project Context: ER Diagrams in SQL

This project focuses on the use of Entity-Relationship (ER) diagrams to design and model databases in SQL. ER diagrams help visualize the structure of a database by representing entities (like tables) and the relationships between them.

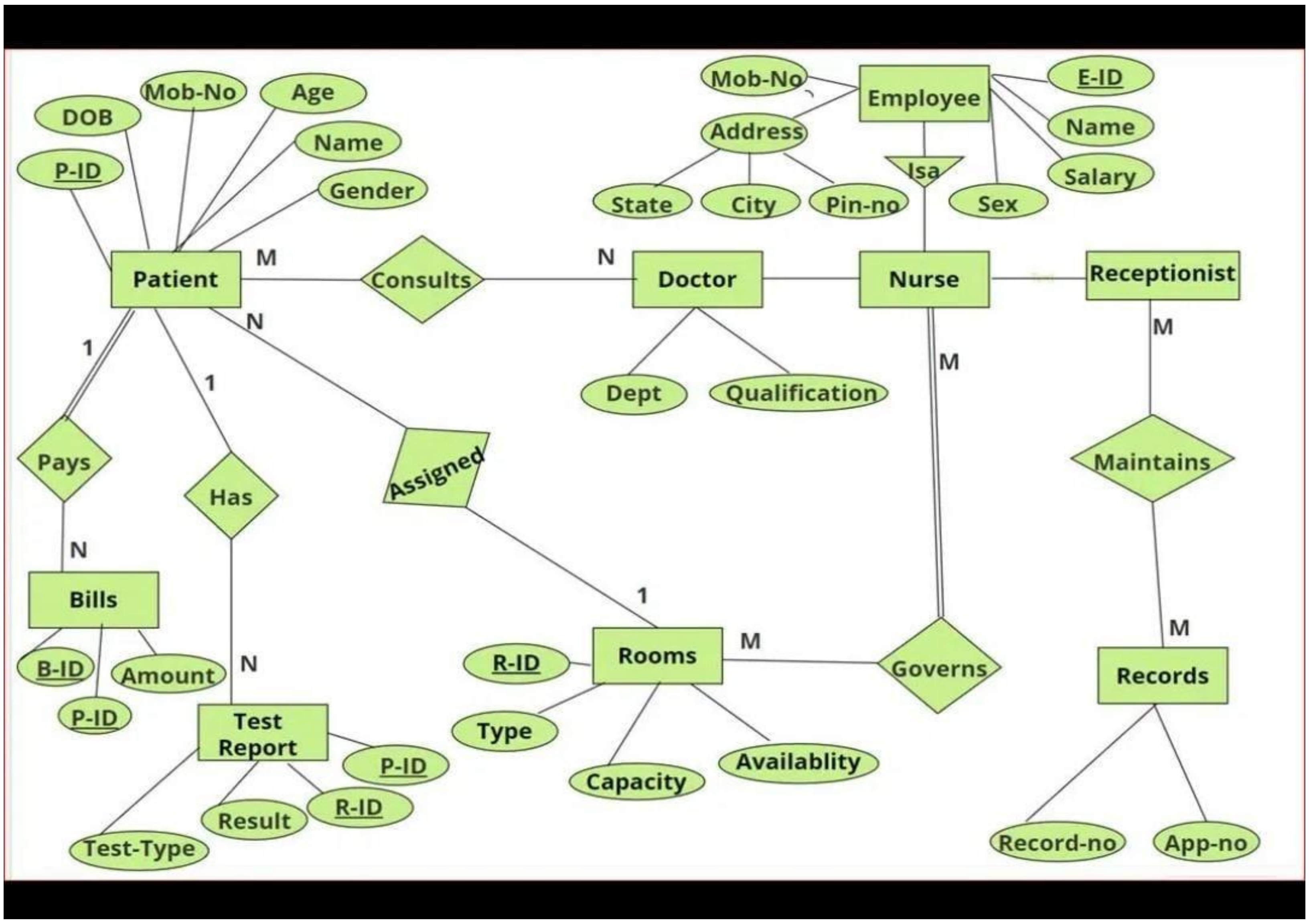
The goal of this project is to demonstrate how a real-world system, such as a hospital management system, can be translated into a database using ER diagrams. This involves:

- Identifying key entities (e.g., Patient, Doctor, Procedure).
- Defining the relationships between these entities (e.g., a patient undergoes procedures).
- Implementing the ER diagram in SQL by creating tables, setting primary and foreign keys, and ensuring data integrity.

This approach ensures a well-structured database that is efficient and easy to maintain.

SQL-PROJECT DOCUMENTATION

Title: Hospital Management System ER Diagram



To create tables for the entities in the ER diagram, here is a syntax in SQL based on the structure of the ER diagram:

1.Patient Table:

```
CREATE TABLE Patient (
   P_ID INT PRIMARY KEY,
   Name VARCHAR(100),
   Gender VARCHAR(10),
   Age INT,
   DOB DATE,
   Mob_No VARCHAR(15)
);
```

2. Doctor Table:

```
CREATE TABLE Doctor
```

```
    E_ID INT PRIMARY KEY,  -- Assuming Employee ID is used for doctors too
    Dept VARCHAR(50),
    FOREIGN KEY (E_ID) REFERENCES Employee(E_ID)
);
```

3. Nurse Table:

```
CREATE TABLE Nurse (
    E_ID INT PRIMARY KEY,  -- Assuming Employee ID is used for nurses too
    Qualification VARCHAR(100),
    FOREIGN KEY (E_ID) REFERENCES Employee(E_ID)
);
```

4. Employee Table:

```
CREATE TABLE Employee (
    E_ID INT PRIMARY KEY,
    Name VARCHAR(100),
    Mob_No VARCHAR(15),
    Address VARCHAR(255),
    State VARCHAR(50),
    City VARCHAR(50),
    Pin_no VARCHAR(10),
    Salary DECIMAL(10, 2),
    Sex VARCHAR(10)
);
```

5. Receptionist Table:

```
CREATE TABLE Receptionist (
    E_ID INT PRIMARY KEY,  -- Assuming Employee ID is used for receptionists too
    FOREIGN KEY (E_ID) REFERENCES Employee(E_ID)
);
```

6. Bills Table:

```
CREATE TABLE Bills (
    B_ID INT PRIMARY KEY,
    Amount DECIMAL(10, 2),
    P_ID INT,
    FOREIGN KEY (P_ID) REFERENCES Patient(P_ID)
);
```

7. Test Report Table:

```
CREATE TABLE Test_Report (
    R_ID INT PRIMARY KEY,
    Test_Type VARCHAR(100),
    Result VARCHAR(255),
    P_ID INT,
    FOREIGN KEY (P_ID) REFERENCES Patient(P_ID)
);
```

8. Rooms Table:

sql

```
CREATE TABLE Rooms (
    R_ID INT PRIMARY KEY,
    Type VARCHAR(50),
    Capacity INT,
    Availability BOOLEAN
);
```

9. Records Table:

```
CREATE TABLE Records (
    Record_No INT PRIMARY KEY,
    App_No INT
);
```

10. Relationships:

10.1 Consults Relationship (between Patient and Doctor):

```
CREATE TABLE Consults (
    P_ID INT,
    E_ID INT,
    FOREIGN KEY (P_ID) REFERENCES Patient(P_ID),
    FOREIGN KEY (E_ID) REFERENCES Doctor(E_ID),
    PRIMARY KEY (P_ID, E_ID)
);
```

10.2 Assigned Relationship (between Doctor and Rooms):

```
CREATE TABLE Assigned (
    E_ID INT,
    R_ID INT,
    FOREIGN KEY (E_ID) REFERENCES Doctor(E_ID),
    FOREIGN KEY (R_ID) REFERENCES Rooms(R_ID),
    PRIMARY KEY (E_ID, R_ID)
);
```

10.3 Pays Relationship (between Patient and Bills):

```
CREATE TABLE Pays (
    P_ID INT,
    B_ID INT,
    FOREIGN KEY (P_ID) REFERENCES Patient(P_ID),
    FOREIGN KEY (B_ID) REFERENCES Bills(B_ID),
    PRIMARY KEY (P_ID, B_ID)
);
```

10.4 Has Relationship (between Bills and Test Report):

```
CREATE TABLE Has (
    B_ID INT,
    R_ID INT,
    FOREIGN KEY (B_ID) REFERENCES Bills(B_ID),
    FOREIGN KEY (R_ID) REFERENCES Test_Report(R_ID),
    PRIMARY KEY (B_ID, R_ID)
);
```

10.5 Governs Relationship (between Nurse and Rooms):

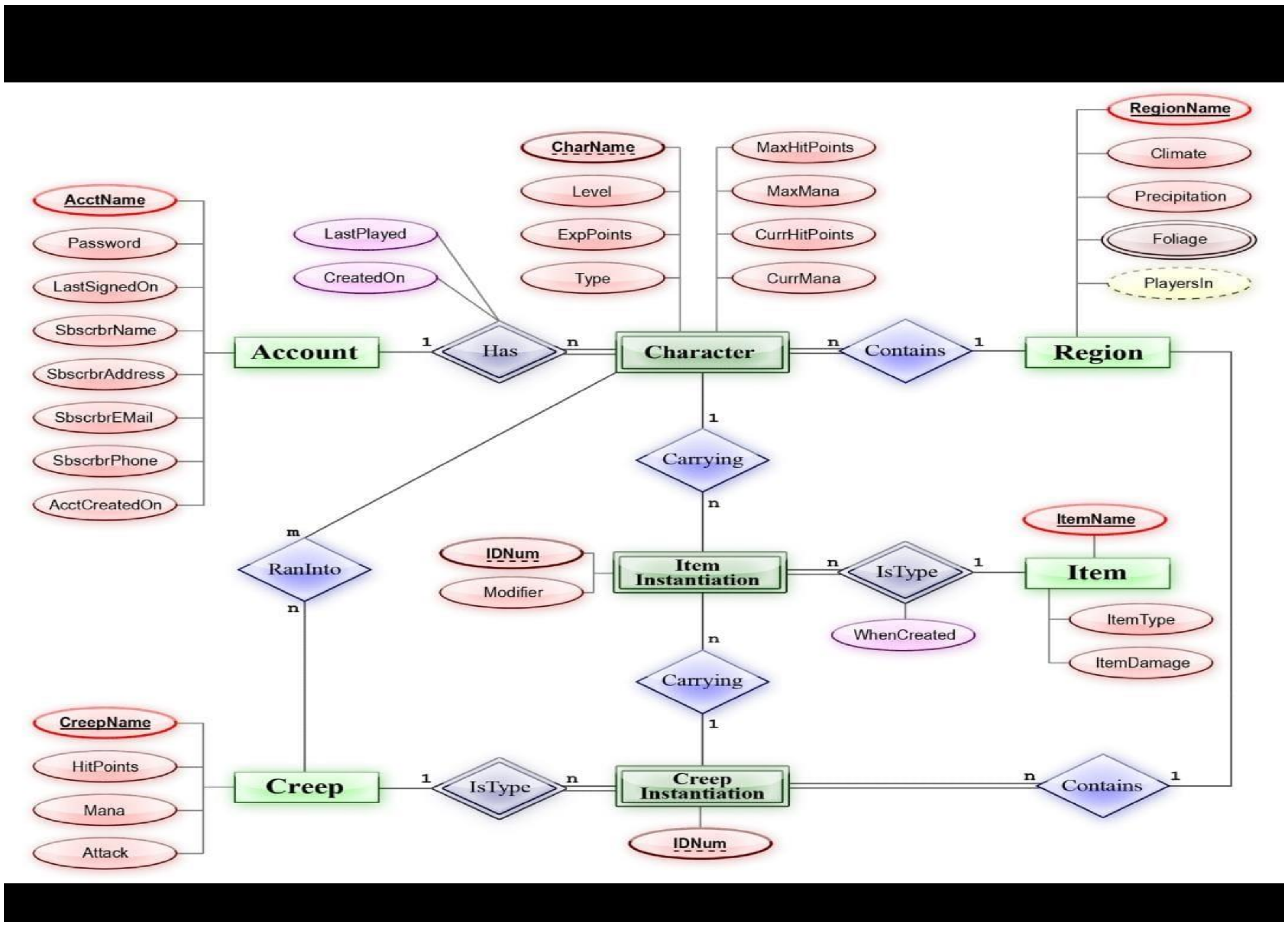
```
CREATE TABLE Governs (
    E_ID INT,
    R_ID INT,
    FOREIGN KEY (E_ID) REFERENCES Nurse(E_ID),
    FOREIGN KEY (R_ID) REFERENCES Rooms(R_ID),
    PRIMARY KEY (E_ID, R_ID)
);
```

10.6 Maintains Relationship (between Receptionist and Records):

```
CREATE TABLE Maintains (
    E_ID INT,
    Record_No INT,
    FOREIGN KEY (E_ID) REFERENCES Receptionist(E_ID),
    FOREIGN KEY (Record_No) REFERENCES Records(Record_No),
    PRIMARY KEY (E_ID, Record_No)
);
```

This syntax outlines how the entities and relationships from the ER diagram would translate into SQL tables with appropriate foreign keys and primary keys for establishing the connections between them.

Title: Game Management System ER Diagram



Here’s the SQL syntax based on the entities and relationships from the game system ER diagram:

1. Account Table:

```
CREATE TABLE Account (
   AcctName VARCHAR(50) PRIMARY KEY,
   Password VARCHAR(100),
   LastSignedOn DATE,
   SbscrbrName VARCHAR(100),
   SbscrbrAddress VARCHAR(255),
   SbscrbrEMail VARCHAR(100),
   SbscrbrPhone VARCHAR(15),
   AcctCreatedOn DATE
);
```

2. Character Table:

```
CREATE TABLE Character (
   CharName VARCHAR(50) PRIMARY KEY,
   Level INT,
   ExpPoints INT,
   Type VARCHAR(50),
   MaxHitPoints INT,
   MaxMana INT,
   CurrHitPoints INT,
   CurrMana INT,
   LastPlayed DATE,
   CreatedOn DATE,
   AcctName VARCHAR(50),
   FOREIGN KEY (AcctName) REFERENCES Account(AcctName)
);
```

3. Region Table:

```
CREATE TABLE Region (
   RegionName VARCHAR(50) PRIMARY KEY,
   Climate VARCHAR(50),
   Precipitation VARCHAR(50),
   Foliage VARCHAR(50)
);
```

4. Item Table:

```
CREATE TABLE Item (
   ItemName VARCHAR(50) PRIMARY KEY,
   ItemType VARCHAR(50),
   ItemDamage INT
);
```

5. Creep Table:

```
CREATE TABLE Creep (
   CreepName VARCHAR(50) PRIMARY KEY,
   HitPoints INT,
   Mana INT,
   Attack INT
);
```

6. Item Instantiation Table:

```
CREATE TABLE Item_Instantiation (
   IDNum INT PRIMARY KEY,
   Modifier VARCHAR(50),
   ItemName VARCHAR(50),
   WhenCreated DATE,
   FOREIGN KEY (ItemName) REFERENCES Item(ItemName)
);
```

7. Creep Instantiation Table:

```
CREATE TABLE Creep_Instantiation (
   IDNum INT PRIMARY KEY,
   CreepName VARCHAR(50),
   FOREIGN KEY (CreepName) REFERENCES Creep(CreepName)
);
```

8. Relationships:

8.1 Has Relationship (between Account and Character):

```
CREATE TABLE Has (
   AcctName VARCHAR(50),
 CharName VARCHAR(50),
   FOREIGN KEY (AcctName) REFERENCES Account(AcctName),
   FOREIGN KEY (CharName) REFERENCES Character(CharName),
   PRIMARY KEY (AcctName, CharName)
);
```

8.2 *Contains Relationship (between Character and Region):*

```
CREATE TABLE Contains_Character_Region (
    CharName VARCHAR(50),
    RegionName VARCHAR(50),
    FOREIGN KEY (CharName) REFERENCES Character(CharName),
    FOREIGN KEY (RegionName) REFERENCES Region(RegionName),
    PRIMARY KEY (CharName, RegionName)
);
```

8.3 Carrying Relationship (between Character and Item Instantiation):

```
CREATE TABLE Carrying_Character_Item (
    CharName VARCHAR(50),
    IDNum INT,
    FOREIGN KEY (CharName) REFERENCES Character(CharName),
    FOREIGN KEY (IDNum) REFERENCES Item_Instantiation(IDNum),
    PRIMARY KEY (CharName, IDNum)
);
```

8.4 RanInto Relationship (between Character and Creep):

```
CREATE TABLE RanInto (
    CharName VARCHAR(50),
    CreepName VARCHAR(50),
    FOREIGN KEY (CharName) REFERENCES Character(CharName),
    FOREIGN KEY (CreepName) REFERENCES Creep(CreepName),
    PRIMARY KEY (CharName, CreepName)
);
```

8.5 *Carrying Relationship (between Item Instantiation and Creep Instantiation):*

```
CREATE TABLE Carrying_Creep_Item (
    IDNum_Creep INT,
    IDNum_Item INT,
    FOREIGN KEY (IDNum_Creep) REFERENCES Creep_Instantiation(IDNum),
    FOREIGN KEY (IDNum_Item) REFERENCES Item_Instantiation(IDNum),
    PRIMARY KEY (IDNum_Creep, IDNum_Item)
);
```

8.6 IsType Relationship (between Creep and Creep Instantiation):

```
CREATE TABLE IsType_Creep (
    CreepName VARCHAR(50),
    IDNum INT,
    FOREIGN KEY (CreepName) REFERENCES Creep(CreepName),
    FOREIGN KEY (IDNum) REFERENCES Creep_Instantiation(IDNum),
    PRIMARY KEY (CreepName, IDNum)
);
```

8.7 IsType Relationship (between Item and Item Instantiation):

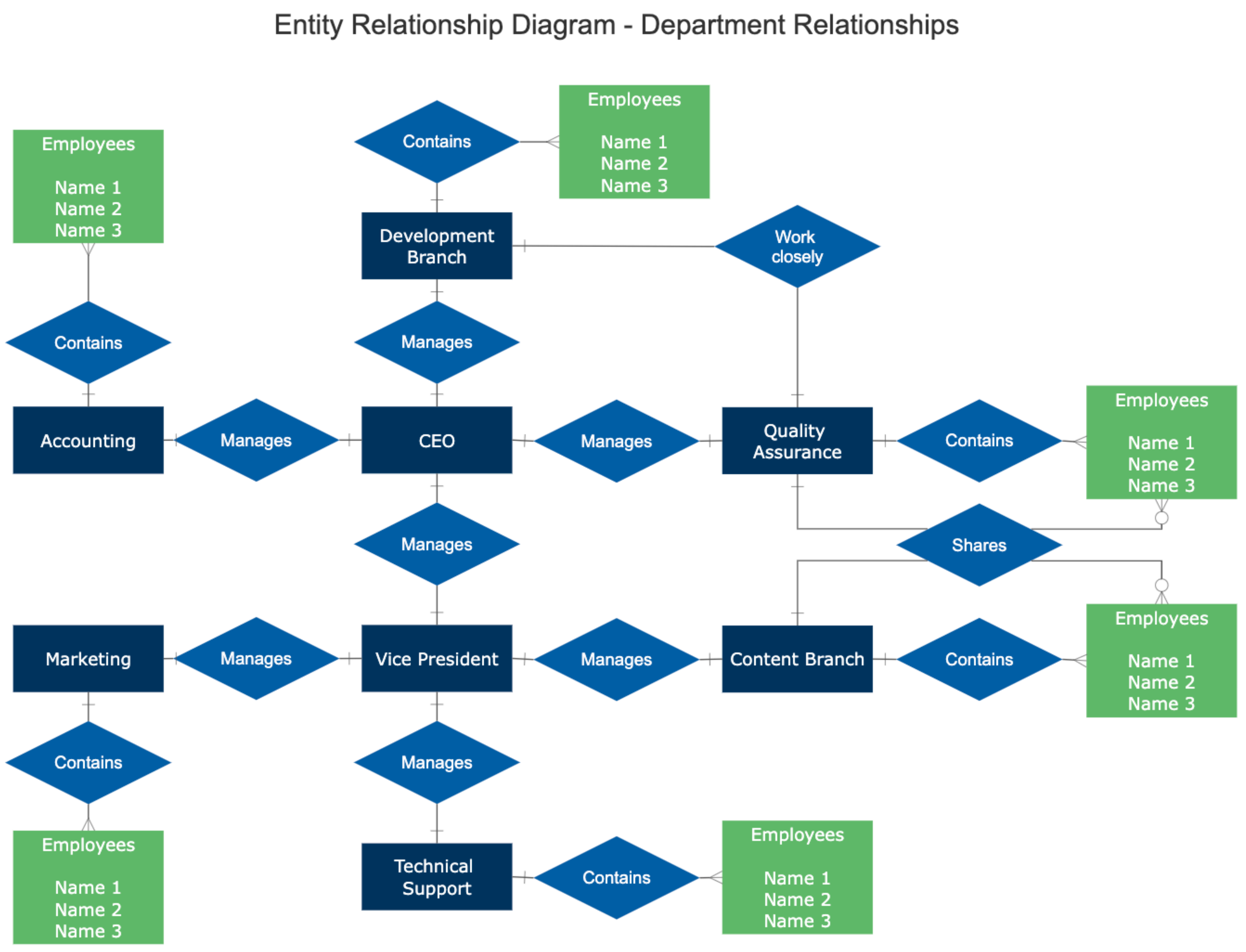
```
CREATE TABLE IsType_Item (
    ItemName VARCHAR(50),
    IDNum INT,
    FOREIGN KEY (ItemName) REFERENCES Item(ItemName),
    FOREIGN KEY (IDNum) REFERENCES Item_Instantiation(IDNum),
    PRIMARY KEY (ItemName, IDNum)
);
```

8.8 Contains Relationship (between Region and Creep Instantiation):

```
CREATE TABLE Contains_Region_Creep (
    RegionName VARCHAR(50),
    IDNum INT,
    FOREIGN KEY (RegionName) REFERENCES Region(RegionName),
    FOREIGN KEY (IDNum) REFERENCES Creep_Instantiation(IDNum),
    PRIMARY KEY (RegionName, IDNum)
);
```

This syntax defines the structure for the entities and relationships from the game system ER diagram in SQL format, with appropriate foreign keys and primary keys to manage the connections between tables.

Title: Department Relationships ER Diagram


Entity Relationship Diagram - Department Relationships
Employees Name 1 Name 2 Name 3
Contains
Development Branch
Work closely
Employees Name 1 Name 2 Name 3
Contains
Manages
Accounting
Manages
CEO
Manages
Quality Assurance
Contains
Employees Name 1 Name 2 Name 3
Manages
Shares
Marketing
Manages
Vice President
Manages
Content Branch
Contains
Employees Name 1 Name 2 Name 3
Contains
Manages
Employees Name 1 Name 2 Name 3
Technical Support
Contains
Employees Name 1 Name 2 Name 3

Based on the ER diagram you provided for department relationships, here is the SQL syntax for creating the necessary tables and their columns.

1.Tables to Create

2.Department

3.Employee

SQL Syntax for Creating Tables

1. Department Table:

```
CREATE TABLE Department (
   department_id INT PRIMARY KEY AUTO_INCREMENT,
   department_name VARCHAR(100) NOT NULL
);
```

department_id: Unique identifier for each department.

department_name: The name of the department (e.g., Accounting, Development Branch, etc.).

2. Employee Table:

```
CREATE TABLE Employee (
   employee_id INT PRIMARY KEY AUTO_INCREMENT,
   employee_name VARCHAR(100) NOT NULL,
   department_id INT,
   FOREIGN KEY (department_id) REFERENCES Department(department_id)
);
```

employee_id: Unique identifier for each employee.

employee_name: Name of the employee (e.g., Name 1, Name 2).

department_id: Foreign key linking employees to the department they belong to.

3. Management Table:

This table records the relationship where certain employees manage specific departments.

```
CREATE TABLE Management (
   manager_id INT,
   department_id INT,
   PRIMARY KEY (manager_id, department_id),
   FOREIGN KEY (manager_id) REFERENCES Employee(employee_id),
   FOREIGN KEY (department_id) REFERENCES Department(department_id)
);
```

manager_id: Employee who manages the department (likely CEO, Vice President, etc.).

department_id: The department being managed.

4. Department Relationships Table:

This table is for capturing relationships like "Work closely" and "Shares" between departments.

```
CREATE TABLE DepartmentRelationships (
       department1_id INT,
       department2_id INT,
       relationship_type VARCHAR(50) NOT NULL,
       PRIMARY KEY (department1_id, department2_id),
  FOREIGN KEY (department1_id) REFERENCES Department(department_id),
   FOREIGN KEY (department2_id) REFERENCES Department(department_id)
);
```

department1_id and department2_id: Represent two departments involved in relationship.

Relationship_type : Defines the type of relationship ("Work closely", "Shares").

Inserting Data into the Tables

Example Insertion for Departments

```
INSERT INTO Department (department_name) VALUES
('Accounting'),
('Development Branch'),
('Quality Assurance'),
('Content Branch'),
('Technical Support'),
('Marketing');
```

Example Insertion for Employees

```
INSERT INTO Employee (employee_name, department_id) VALUES
('Name 1', 1),
('Name 2', 1),
('Name 3', 1),
('Name 1', 2),
('Name 2', 2),
('Name 3', 2),
('Name 1', 3),
('Name 2', 3),
('Name 3', 3);
```

Example Insertion for Management

```
INSERT INTO Management (manager_id, department_id) VALUES
(1, 1),  -- CEO manages Accounting
(2, 2),  -- CEO manages Development Branch
(3, 3);  -- Vice President manages Quality Assurance
```

Example Insertion for Department Relationship

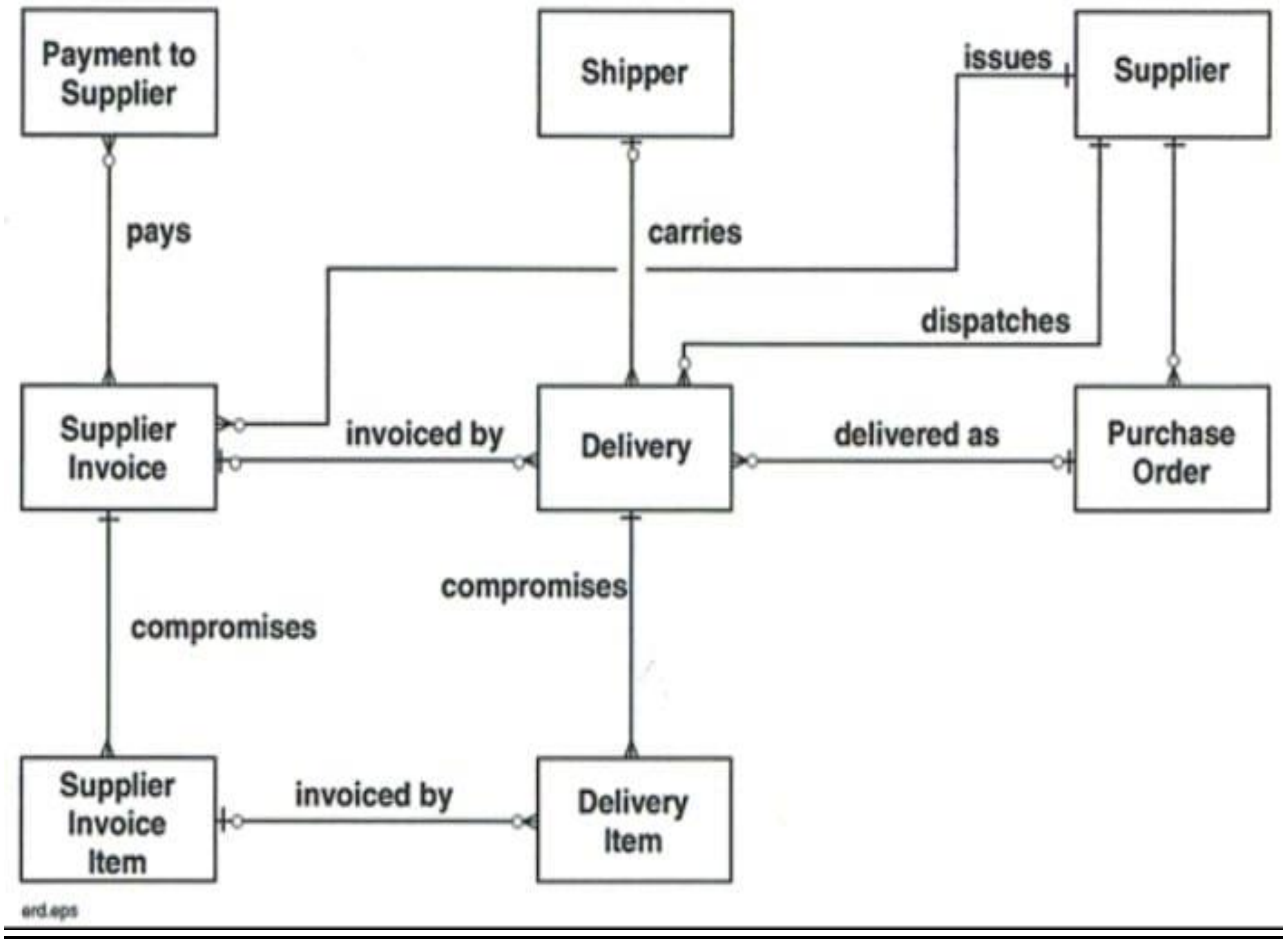
INSERT INTO DepartmentRelationships (department1_id, department2_id, relationship_type) VALUES

(2, 3, 'Work closely'), -- Development Branch works closely with QA

(3, 4, 'Shares');   -- QA shares resources with Content Branch

This structure represents the department relationships and the hierarchical management described in the ER diagram.

Title: Supply Chain Management Entity-Relationship Diagram



Below is the SQL syntax to create the tables based on the ER diagram provided. The diagram depicts the relationships between suppliers, invoices, deliveries, purchase orders, and associated items.

SQL Table Creation Syntax:

1.Table for Supplier

```
CREATE TABLE Supplier (
   SupplierID INT PRIMARY KEY,
   SupplierName VARCHAR(100),
   Address VARCHAR(255)
);
```

2. Table for Shipper

```
CREATE TABLE Shipper (
    ShipperID INT PRIMARY KEY,
    ShipperName VARCHAR(100)
);
```

3.Table for Purchase Order

```
CREATE TABLE PurchaseOrder (
    PurchaseOrderID INT PRIMARY KEY,
    SupplierID INT,
    OrderDate DATE,
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID)
);
```

4.Table for Delivery

```
CREATE TABLE Delivery (
    DeliveryID INT PRIMARY KEY,
    PurchaseOrderID INT,
    DeliveryDate DATE,
    ShipperID INT,
    FOREIGN KEY (PurchaseOrderID) REFERENCES PurchaseOrder(PurchaseOrderID),
    FOREIGN KEY (ShipperID) REFERENCES Shipper(ShipperID)
);
```

5. Table for DeliveryItem

```
CREATE TABLE DeliveryItem (
    DeliveryItemID INT PRIMARY KEY,
    DeliveryID INT,
    ItemDescription VARCHAR(255),
    Quantity INT,
    FOREIGN KEY (DeliveryID) REFERENCES Delivery(DeliveryID)
);
```

6. Table for Supplier Invoice

```
CREATE TABLE SupplierInvoice (
    InvoiceID INT PRIMARY KEY,
    DeliveryID INT,
    InvoiceDate DATE,
    TotalAmount DECIMAL(10, 2),
    FOREIGN KEY (DeliveryID) REFERENCES Delivery(DeliveryID)
);
```

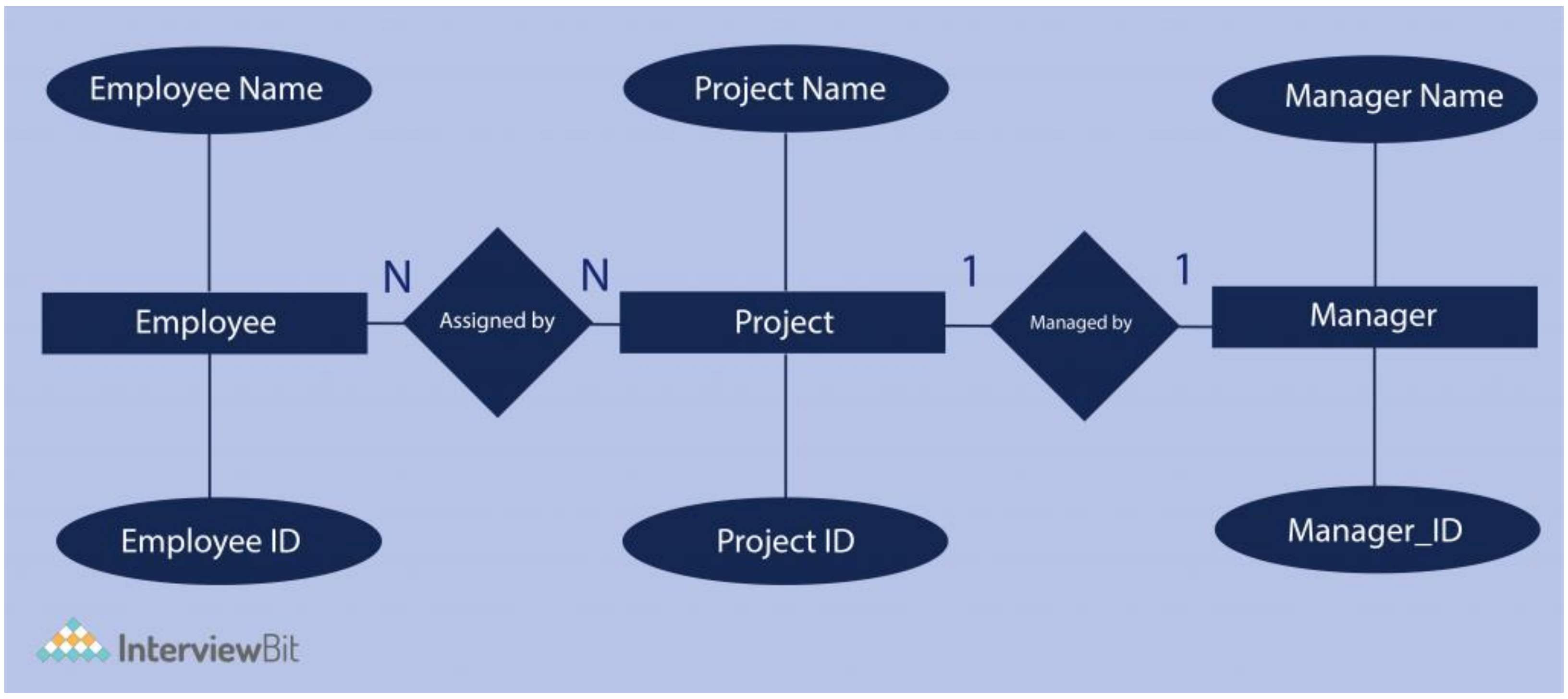
7. Table for Supplier Invoice Item

```
CREATE TABLE SupplierInvoiceItem (
    InvoiceItemID INT PRIMARY KEY,
    InvoiceID INT,
    Description VARCHAR(255),
    Quantity INT,
    UnitPrice DECIMAL(10, 2),
    FOREIGN KEY (InvoiceID) REFERENCES SupplierInvoice(InvoiceID)
);
```

8. Table for Payment to Supplier

```
CREATE TABLE PaymentToSupplier (
    PaymentID INT PRIMARY KEY,
    InvoiceID INT,
    PaymentDate DATE,
    AmountPaid DECIMAL(10, 2),
    FOREIGN KEY (InvoiceID) REFERENCES SupplierInvoice(InvoiceID)
);
```

Title: Employee-Project-Manager Relationship Database Schema



SQL Syntax for Creating Tables

1. Employee Table

```
CREATE TABLE Employee (
   employee_id INT PRIMARY KEY AUTO_INCREMENT,
   employee_name VARCHAR(100) NOT NULL
);
```

employee_id: Unique identifier for each employee.

employee_name: The name of the employee.

2. Project Table

```
CREATE TABLE Project (
   project_id INT PRIMARY KEY AUTO_INCREMENT,
   project_name VARCHAR(100) NOT NULL
);
```

project_id: Unique identifier for each project.

project_name: The name of the project.

3. Manager Table

```
CREATE TABLE Manager (
   manager_id INT PRIMARY KEY AUTO_INCREMENT,
    manager_name VARCHAR(100) NOT NULL
);
```

manager_id: Unique identifier for each manager.

manager_name: The name of the manager.

4. EmployeeProject Table (To handle the many-to-many relationship between Employee and Project)

```
CREATE TABLE EmployeeProject (
    employee_id INT,
    project_id INT,
    PRIMARY KEY (employee_id, project_id),
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id),
    FOREIGN KEY (project_id) REFERENCES Project(project_id)
);
```

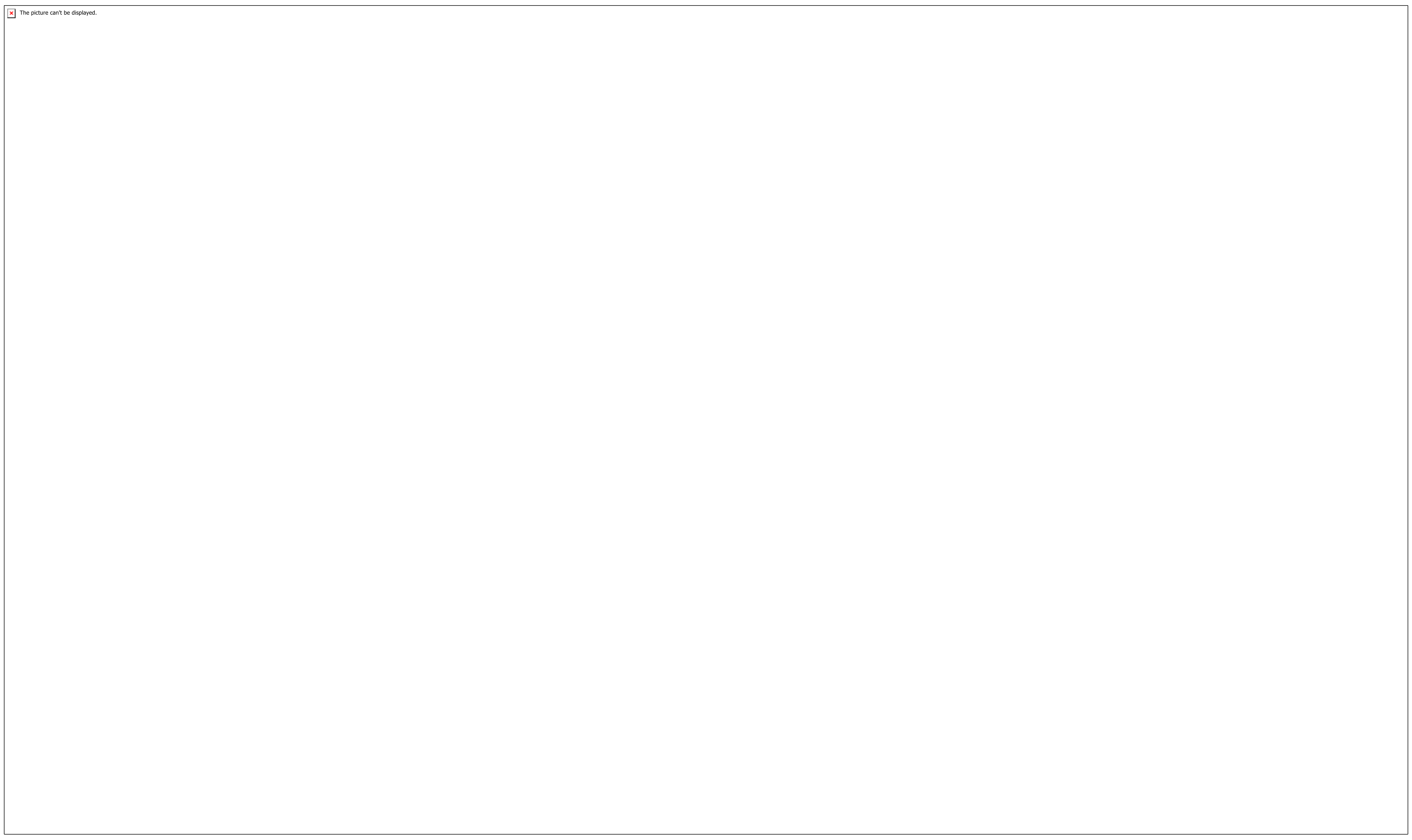
employee_id: Refers to an employee working on a project.

project_id: Refers to the project assigned to the employee.

5. ProjectManager Table (To handle the one-to-one relationship between Project and Manager)

```
CREATE TABLE ProjectManager (
    project_id INT,
    manager_id INT,
    PRIMARY KEY (project_id),
    FOREIGN KEY (project_id) REFERENCES Project(project_id),
    FOREIGN KEY (manager_id) REFERENCES Manager(manager_id)
);
```

Title: E-Commerce Database Schema



SQL Syntax for Table Creation:

1. Table: User

```
CREATE TABLE User (
   u_id INT PRIMARY KEY,
   name VARCHAR(100),
   email VARCHAR(100)
);
```

2. Table: Order

```
CREATE TABLE Order (
   order_no INT PRIMARY KEY,
   order_amount DECIMAL(10, 2),
   order_date DATE,
   u_id INT,
   FOREIGN KEY (u_id) REFERENCES User(u_id)
);
```

3.Table: Product

```
CREATE TABLE Product (
  p_id INT PRIMARY KEY,
   name VARCHAR(100),
   price DECIMAL(10, 2),
   description TEXT,
   c_id INT,
   FOREIGN KEY (c_id) REFERENCES Product_Category(c_id)
);
```

4.Table: Product_Category

```
CREATE TABLE Product_Category (
   c_id INT PRIMARY KEY,
   name VARCHAR(100)
);
```

5. Table: Cart

```
CREATE TABLE Cart (
   cart_id INT PRIMARY KEY,
   u_id INT,
   FOREIGN KEY (u_id) REFERENCES User(u_id)
);
```

6. Table: Payment

```
CREATE TABLE Payment (
   p_id INT PRIMARY KEY,
   method VARCHAR(50),
   amount DECIMAL(10, 2),
   order_no INT,
   FOREIGN KEY (order_no) REFERENCES Order(order_no)
);
```

7. Table: Address

```
CREATE TABLE Address (
   a_id INT PRIMARY KEY,
   city VARCHAR(100),
   state VARCHAR(100),
   country VARCHAR(100),
  u_id INT,
   FOREIGN KEY (u_id) REFERENCES User(u_id)
);
```

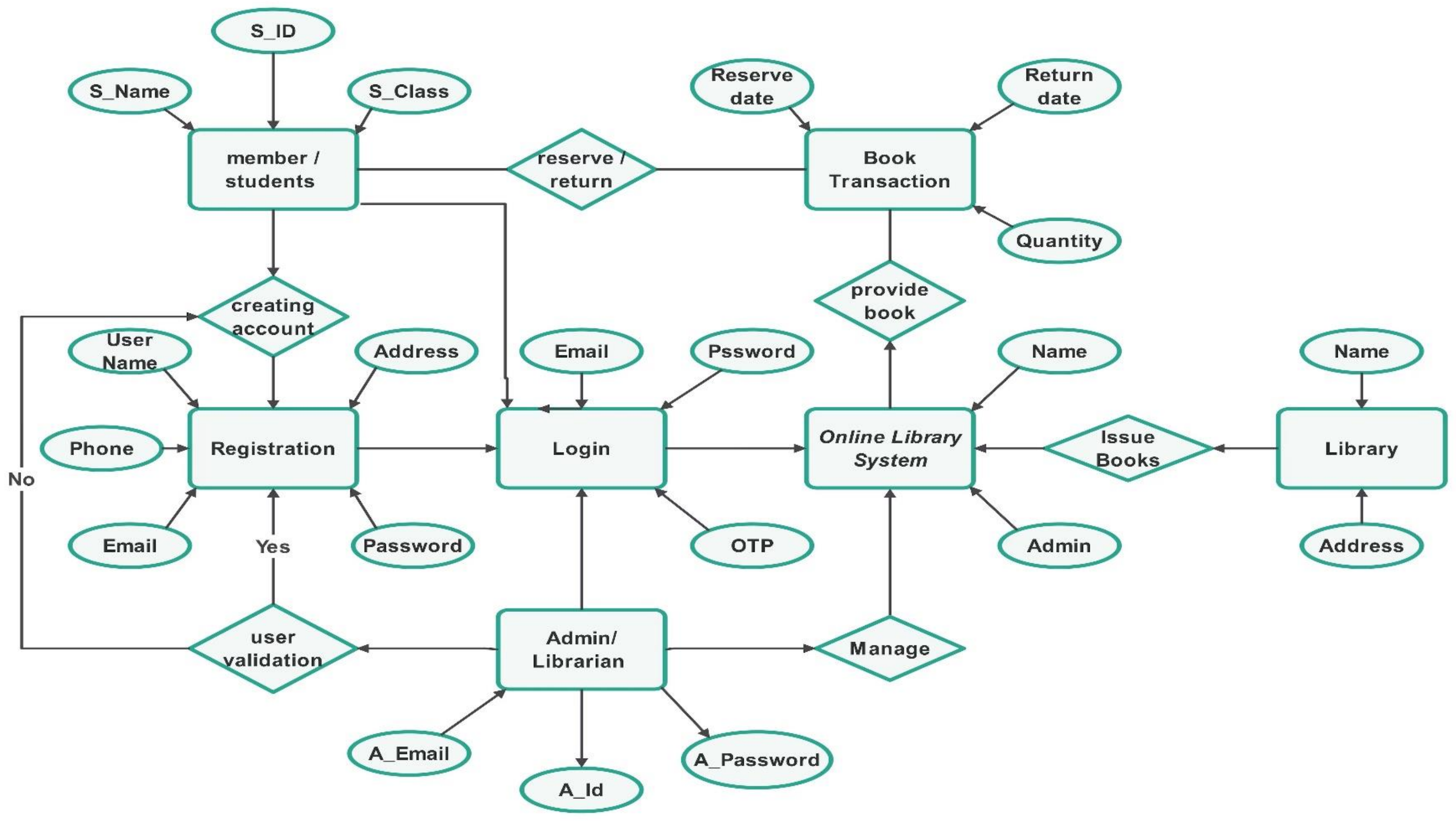
8. Table: Tracking_Detail

```
CREATE TABLE Tracking_Detail (
   t_id INT PRIMARY KEY,
   status VARCHAR(100),
   order_no INT,
   FOREIGN KEY (order_no) REFERENCES Order(order_no)
);
```

9. Table: Cart_Product_Relation

```
CREATE TABLE Cart_Product_Relation (
   cart_id INT,
   p_id INT,
   FOREIGN KEY (cart_id) REFERENCES Cart(cart_id),
   FOREIGN KEY (p_id) REFERENCES Product(p_id),
   PRIMARY KEY (cart_id, p_id)
);
```

<u>Title: Online Library Management System Database Schema</u>



SQL Syntax for Creating Tables:

1. Table: Students

```
CREATE TABLE Students (
   s_id INT PRIMARY KEY,
   s_name VARCHAR(100),
   s_class VARCHAR(50)
);
```

2. Table: Registration

```
CREATE TABLE Registration (
   user_name VARCHAR(100),
   phone VARCHAR(15),
   email VARCHAR(100),
   password VARCHAR(50),
   address VARCHAR(255),
   PRIMARY KEY (email)
);
```

3. Table: Login

```
CREATE TABLE Login (
   email VARCHAR(100),
   password VARCHAR(50),
   otp INT,
   PRIMARY KEY (email),
   FOREIGN KEY (email) REFERENCES Registration(email)
);
```

4. Table: Admin_Librarian

```
CREATE TABLE Admin_Librarian (
   a_id INT PRIMARY KEY,
   a_email VARCHAR(100),
   a_password VARCHAR(50)
);
```

5. Table: Library

```
CREATE TABLE Library (
   name VARCHAR(100),
   address VARCHAR(255)
);
```

6. Table: Book_Transaction

```
CREATE TABLE Book_Transaction (
   transaction_id INT PRIMARY KEY,
   reserve_date DATE,
   return_date DATE,
   quantity INT,
   s_id INT,
   FOREIGN KEY (s_id) REFERENCES Students(s_id)
);
```

7. Table: Online_Library_System

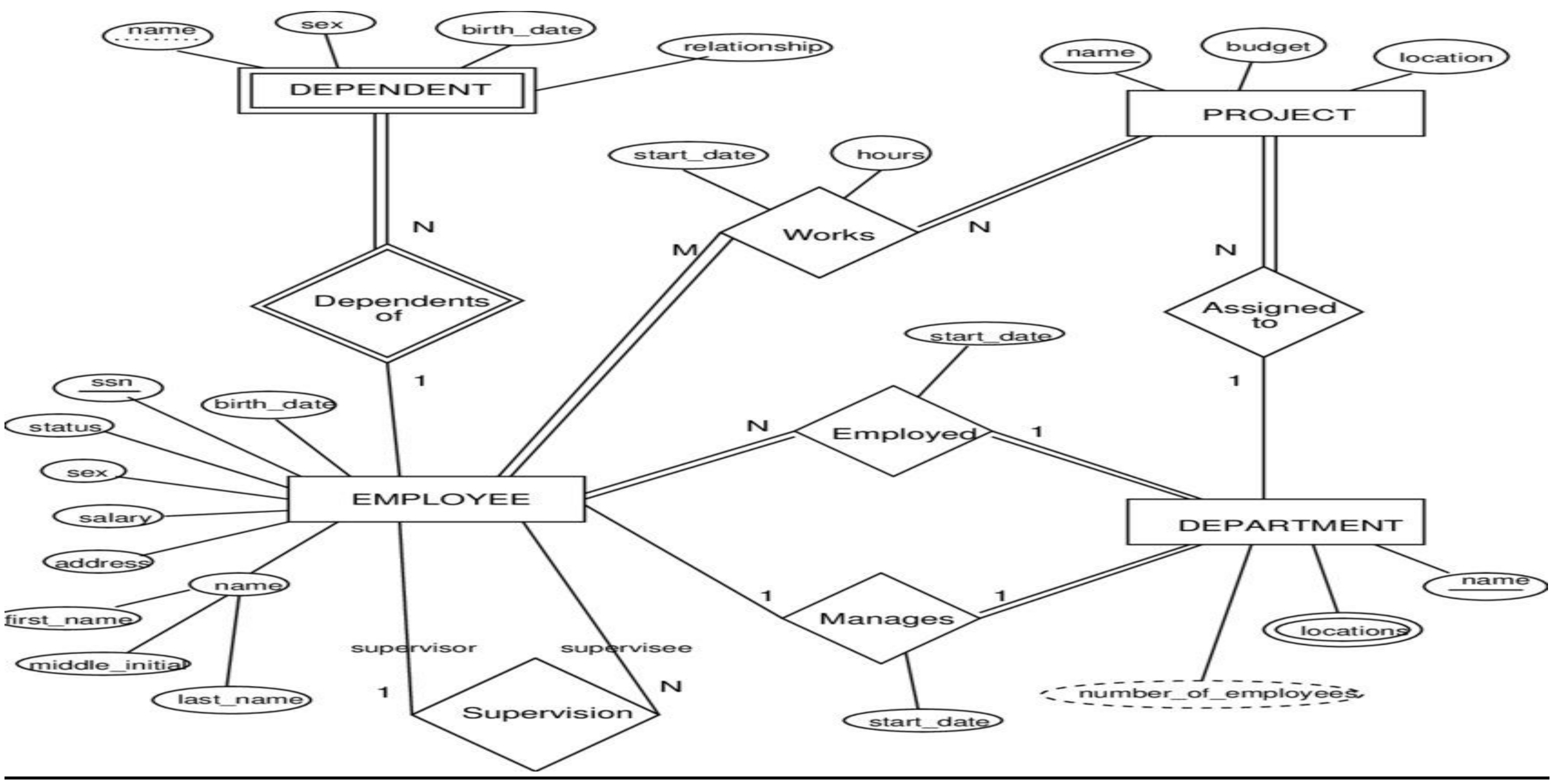
```
CREATE TABLE Online_Library_System (
   transaction_id INT,
   admin_id INT,
   PRIMARY KEY (transaction_id, admin_id),
   FOREIGN KEY (transaction_id) REFERENCES Book_Transaction(transaction_id),
   FOREIGN KEY (admin_id) REFERENCES Admin_Librarian(a_id)
);
```

8. Table: Issue_Books

```
CREATE TABLE Issue_Books (
   transaction_id INT,
   admin_id INT,
   library_name VARCHAR(100),
   PRIMARY KEY (transaction_id, admin_id, library_name),
   FOREIGN KEY (transaction_id) REFERENCES Book_Transaction(transaction_id),
   FOREIGN KEY (admin_id) REFERENCES Admin_Librarian(a_id),
   FOREIGN KEY (library_name) REFERENCES Library(name)
);
```

____

Title: "Railway Ticket Reservation System - Entity Relationship Diagram (ERD)"

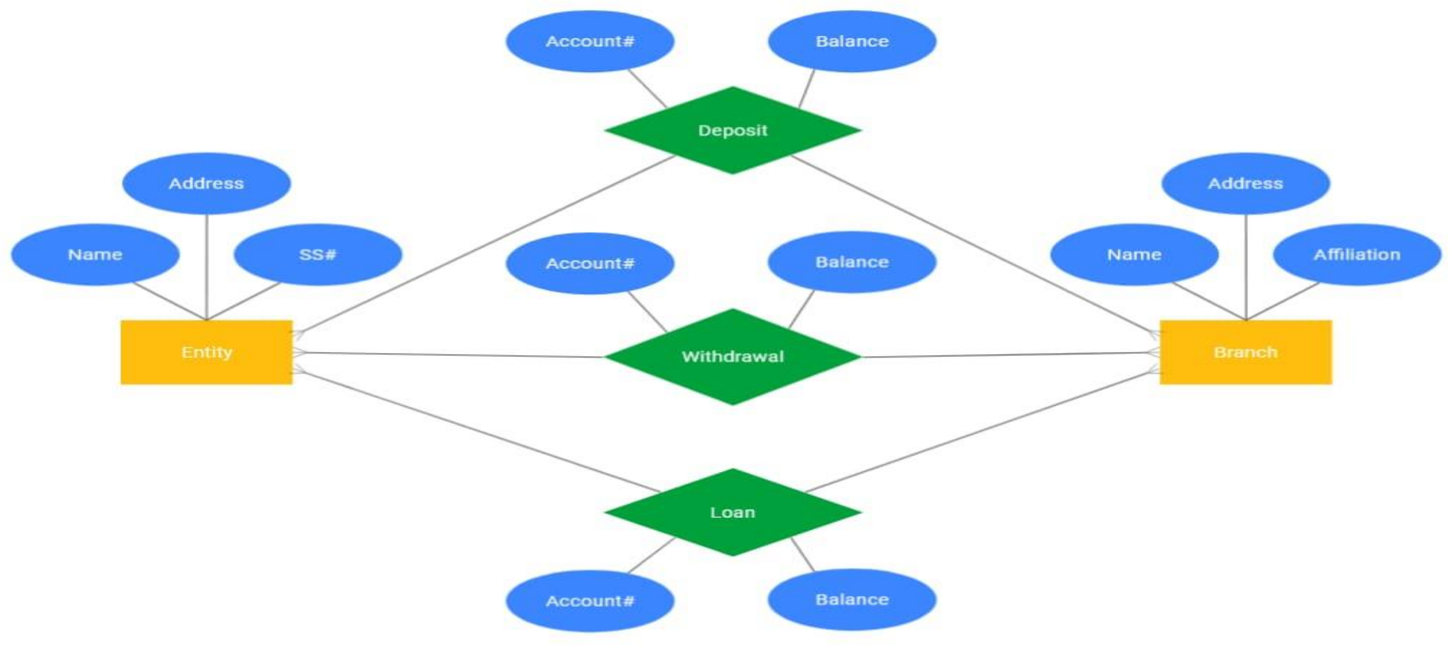


SQL Syntax for Table Creation:

```
CREATE TABLE Ticket_Reservation (
    PNR_no VARCHAR(20) PRIMARY KEY,
    From_station VARCHAR(50),
    To_station VARCHAR(50),
    From_date DATE,
    To_date DATE,
    Train_code VARCHAR(10),
    Class_id VARCHAR(10),
    From_km INT,
    To_km INT
);
CREATE TABLE Passenger_Info (
    PAX_info_id INT PRIMARY KEY,
    PNR_no VARCHAR(20),
    PAX_name VARCHAR(50),
    PAX_age INT
  PAX_sex CHAR(1),
    Seat_no VARCHAR(10),
    FOREIGN KEY (PNR_no) REFERENCES Ticket_Reservation(PNR_no)
);
CREATE TABLE Zone (
    Zone_id VARCHAR(10) PRIMARY KEY,
    Zone_name VARCHAR(50),
    Zone_code VARCHAR(10)
);
CREATE TABLE Station (
    Station_id VARCHAR(10) PRIMARY KEY,
    Station_code VARCHAR(10),
    Station_name VARCHAR(50),
    Zone_id VARCHAR(10),
    FOREIGN KEY (Zone_id) REFERENCES Zone(Zone_id)
);
CREATE TABLE Class (
    Class_id VARCHAR(10) PRIMARY KEY,
    Class_code VARCHAR(10),
    Class_name VARCHAR(20)
);
CREATE TABLE Seat_Availability (
    Train_code VARCHAR(10),
    Class_id VARCHAR(10),
    No_of_seats INT,
    Seat_per_coach INT,
    PRIMARY KEY (Train_code, Class_id),
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code),
    FOREIGN KEY (Class_id) REFERENCES Class(Class_id)
);
CREATE TABLE Train (
    Train_code VARCHAR(10) PRIMARY KEY,
    Train_name VARCHAR(50),
    Distance INT,
    Start_time TIME,
    End_time TIME,
    Frequency VARCHAR(100)
);
CREATE TABLE Train_Fare (
    Fare_id INT PRIMARY KEY,
    Train_code VARCHAR(10),
    Class_id VARCHAR(10),
    From_km INT,
    To_km INT,
    From_date DATE,
    To_date DATE,
    Fare DECIMAL(10, 2),
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code),
    FOREIGN KEY (Class_id) REFERENCES Class(Class_id)
);
CREATE TABLE Payment_Info (
    Payment_id INT PRIMARY KEY,
    PNR_no VARCHAR(20),
    Pay_date DATE,
    Amount DECIMAL(10, 2),
    Pay_mode VARCHAR(20),
    Inst_type VARCHAR(20),
    Inst_amt DECIMAL(10, 2),
    FOREIGN KEY (PNR_no) REFERENCES Ticket_Reservation(PNR_no)
);
CREATE TABLE Via_Details (
    Details_id INT PRIMARY KEY,
    Train_code VARCHAR(10),
    Via_station_code VARCHAR(10),
    Via_station_name VARCHAR(50),
    Km_from_origin INT,
    Reach_time TIME,
    FOREIGN KEY (Train_code) REFERENCES Train(Train_code)
);
CREATE TABLE Login_Credential (
    login_id VARCHAR(20) PRIMARY KEY,
    password VARCHAR(100)
);

CREATE TABLE Refund_Rule (
    Rule_id INT PRIMARY KEY,
    From_time TIME,
    To_time TIME,
    Refundable_amt DECIMAL(10, 2)
);
```

Title: "Banking System - ER Diagram"



SQL Syntax for Table Creation:

1. Table for Entity (Could be customer or other related entities)

```
CREATE TABLE Entity (
    Entity_id INT PRIMARY KEY AUTO_INCREMENT,
 Name VARCHAR(100),
    SSN VARCHAR(15) UNIQUE,
    Address VARCHAR(255)
);
```

2. Table for Branch

```
CREATE TABLE Branch (
    Branch_id INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Address VARCHAR(255),
    Affiliation VARCHAR(100)
);
```

3. Table for Deposit

```
CREATE TABLE Deposit (
    Deposit_id INT PRIMARY KEY AUTO_INCREMENT,
    Account_Number VARCHAR(20),
    Entity_id INT,
    Branch_id INT,
    Balance DECIMAL(10, 2),
    FOREIGN KEY (Entity_id) REFERENCES Entity(Entity_id),
    FOREIGN KEY (Branch_id) REFERENCES Branch(Branch_id)
);
```

4. Table for Withdrawal

```
CREATE TABLE Withdrawal (
    Withdrawal_id INT PRIMARY KEY AUTO_INCREMENT,
    Account_Number VARCHAR(20),
    Entity_id INT,
    Branch_id INT,
    Balance DECIMAL(10, 2),
    FOREIGN KEY (Entity_id) REFERENCES Entity(Entity_id),
    FOREIGN KEY (Branch_id) REFERENCES Branch(Branch_id)
);
```

5. Table for Loan

```
CREATE TABLE Loan (
   Loan_id INT PRIMARY KEY AUTO_INCREMENT,
   Account_Number VARCHAR(20),
   Entity_id INT,
   Branch_id INT,
   Balance DECIMAL(10, 2),
   FOREIGN KEY (Entity_id) REFERENCES Entity(Entity_id),
   FOREIGN KEY (Branch_id) REFERENCES Branch(Branch_id)
);
```

Sample Stored Procedures:

1. Procedure to Insert a New Entity (Customer)

```
DELIMITER $$
CREATE PROCEDURE AddNewEntity (
   IN p_name VARCHAR(100),
   IN p_ssn VARCHAR(15),
   IN p_address VARCHAR(255)
)
BEGIN
   INSERT INTO Entity (Name, SSN, Address)
   VALUES (p_name, p_ssn, p_address);
END$$
DELIMITER ;
```

2. Procedure to Insert a New Branch

```
DELIMITER $$
CREATE PROCEDURE AddNewBranch (
   IN p_name VARCHAR(100),
   IN p_address VARCHAR(255),
   IN p_affiliation VARCHAR(100)
)
BEGIN
   INSERT INTO Branch (Name, Address, Affiliation)
   VALUES (p_name, p_address, p_affiliation);
```

```
END$$
DELIMITER ;
```

3. Procedure to Add a Deposit

```
DELIMITER $$
CREATE PROCEDURE AddDeposit (
   IN p_account_number VARCHAR(20),
   IN p_entity_id INT,
   IN p_branch_id INT,
   IN p_balance DECIMAL(10, 2)
)
BEGIN
   INSERT INTO Deposit (Account_Number, Entity_id, Branch_id, Balance)
   VALUES (p_account_number, p_entity_id, p_branch_id, p_balance);
END$$
DELIMITER ;
```

4. Procedure to Make a Withdrawal

```
DELIMITER $$
CREATE PROCEDURE MakeWithdrawal (
   IN p_account_number VARCHAR(20),
   IN p_entity_id INT,
   IN p_branch_id INT,
   IN p_withdrawal_amount DECIMAL(10, 2)
)
BEGIN
   Check if balance is sufficient
   DECLARE current_balance DECIMAL(10, 2);
   SELECT Balance INTO current_balance
   FROM Withdrawal
   WHERE Account_Number = p_account_number
   AND Entity_id = p_entity_id
   AND Branch_id = p_branch_id;
  IF current_balance >= p_withdrawal_amount THE
```

```
 UPDATE Withdrawal
        SET Balance = Balance - p_withdrawal_amount
        WHERE Account_Number = p_account_number
        AND Entity_id = p_entity_id
        AND Branch_id = p_branch_id;
    ELSE
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Insufficient funds for withdrawal.';
    END IF;
END$$
DELIMITER ;
```

5. Procedure to Add a Loan

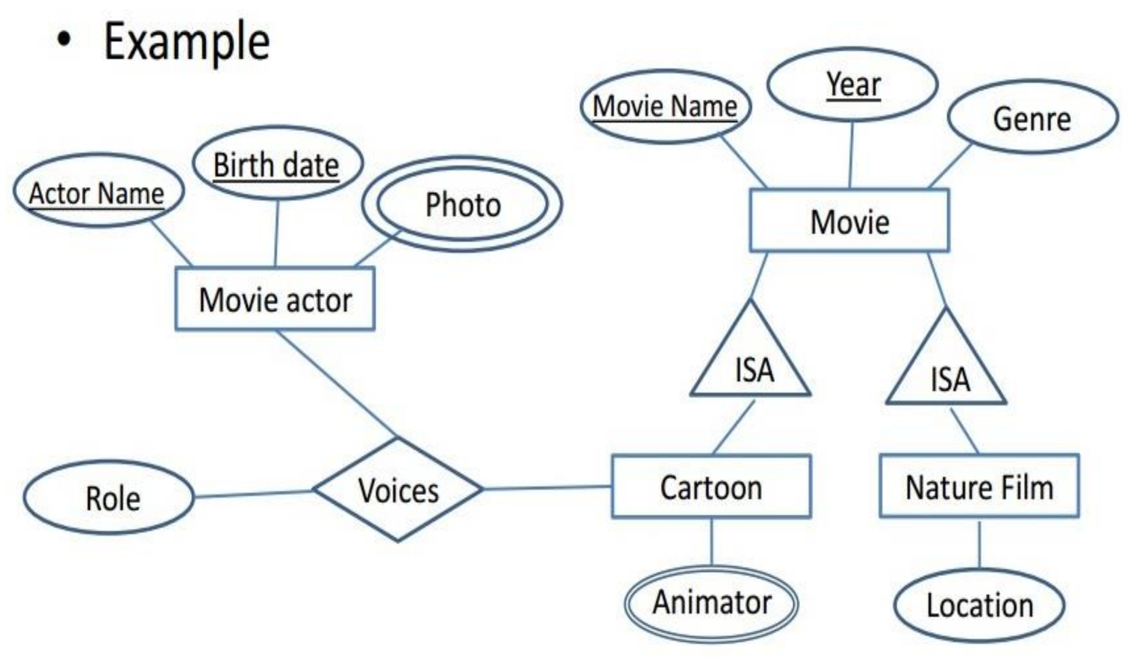
```
DELIMITER $$
CREATE PROCEDURE AddLoan (
    IN p_account_number VARCHAR(20),
    IN p_entity_id INT,
    IN p_branch_id INT,
    IN p_loan_amount DECIMAL(10, 2)
)
BEGIN
    INSERT INTO Loan (Account_Number, Entity_id, Branch_id, Balance)
    VALUES (p_account_number, p_entity_id, p_branch_id, p_loan_amount);
END$$
DELIMITER ;
```

6. Procedure to Update Loan Balance (Repayment)

```
DELIMITER $$
CREATE PROCEDURE UpdateLoanBalance (
    IN p_account_number VARCHAR(20),
    IN p_entity_id INT,
    IN p_branch_id INT,
    IN p_repayment_amount DECIMAL(10, 2)
)
```

```
BEGIN
    UPDATE Loan
    SET Balance = Balance - p_repayment_amount
    WHERE Account_Number = p_account_number
    AND Entity_id = p_entity_id
    AND Branch_id = p_branch_id;
     Ensure balance is not negative
    IF (SELECT Balance FROM Loan WHERE Account_Number = p_account_number) < 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Loan balance cannot be negative.';
    END IF;
END$$
DELIMITER ;
```

Title: Entity-Relationship Diagram for Movie Actor and Movie



Here's the SQL syntax to create tables based on the ER diagram:

Table for Movie Actor

```
CREATE TABLE MovieActor (
    ActorID INT PRIMARY KEY AUTO_INCREMENT,
    ActorName VARCHAR(255) NOT NULL,
    BirthDate DATE,
  Photo BLOB
);
```

Table for Movie

```
CREATE TABLE Movie (
    MovieID INT PRIMARY KEY AUTO_INCREMENT,
    MovieName VARCHAR(255) NOT NULL,
    ReleaseYear INT,
    Genre VARCHAR(100)
);
```

Relationship table for Actor's Voices Role in Movies

```
CREATE TABLE Voices (
    VoiceID INT PRIMARY KEY AUTO_INCREMENT,
    ActorID INT,
    MovieID INT,
    Role VARCHAR(255),
    FOREIGN KEY (ActorID) REFERENCES MovieActor(ActorID),
    FOREIGN KEY (MovieID) REFERENCES Movie(MovieID)
);
```

Table for Cartoon (subtype of Movie)

```
CREATE TABLE Cartoon (
    MovieID INT PRIMARY KEY,
    Animator VARCHAR(255),
    FOREIGN KEY (MovieID) REFERENCES Movie(MovieID)
);
```

Table for Nature Film (subtype of Movie)

```
CREATE TABLE NatureFilm (
    MovieID INT PRIMARY KEY,
    Location VARCHAR(255),
    FOREIGN KEY (MovieID) REFERENCES Movie(MovieID)
);
```

1. **Trigger to Ensure Consistent Insertions Between Movie and Subtypes (Cartoon/NatureFilm)**

```
2. CREATE TRIGGER ensure_movie_subtype
AFTER INSERT ON Movie
FOR EACH ROW
BEGIN
  IF NOT EXISTS (SELECT 1 FROM Cartoon WHERE MovieID = NEW.MovieID)
  AND NOT EXISTS (SELECT 1 FROM NatureFilm WHERE MovieID = NEW.MovieID) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'A movie must be either a Cartoon or a Nature Film';
  END IF;
END;
```

2. Trigger to Automatically Delete Related Records

```
CREATE TRIGGER delete_movie_dependencies
AFTER DELETE ON Movie
FOR EACH ROW
BEGIN
  DELETE FROM Voices WHERE MovieID = OLD.MovieID;
  DELETE FROM Cartoon WHERE MovieID = OLD.MovieID;
  DELETE FROM NatureFilm WHERE MovieID = OLD.MovieID;
END;
```

3. Trigger to Automatically Log Actor Insertion

```
CREATE TABLE ActorLog (
  LogID INT PRIMARY KEY AUTO_INCREMENT,
  ActorID INT,
  ActorName VARCHAR(255),
  InsertionDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Trigger to log the insertion of a new actor

```
CREATE TRIGGER log_actor_insertion
AFTER INSERT ON MovieActor
FOR EACH ROW
BEGIN
  INSERT INTO ActorLog (ActorID, ActorName)
```

```
    VALUES (NEW.ActorID, NEW.ActorName);
END;
```

4. Trigger to Ensure Only One Subtype (Cartoon or Nature Film) is Inserted for a Movie

```
CREATE TRIGGER prevent_duplicate_subtype
BEFORE INSERT ON Cartoon
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT 1 FROM NatureFilm WHERE MovieID = NEW.MovieID) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'This movie is already categorized as a Nature Film.';
    END IF;
END;
CREATE TRIGGER prevent_duplicate_subtype_naturefilm
BEFORE INSERT ON NatureFilm
FOR EACH ROW
BEGIN
    IF EXISTS (SELECT 1 FROM Cartoon WHERE MovieID = NEW.MovieID) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'This movie is already categorized as a Cartoon.';
    END IF;
END;
```

5. Trigger to Auto-Update Actor’s Role in the Voices Table

```
CREATE TRIGGER update_actor_in_voices
AFTER UPDATE ON MovieActor
FOR EACH ROW
BEGIN
    UPDATE Voices
    SET Role = CONCAT('Voiced by ', NEW.ActorName)
    WHERE ActorID = OLD.ActorID;
END;
```