DBMS PROJECT REPORT

DATABASE MANAGEMENT SYSTEM (DBMS)

We designed an ACTORS BOOKING DATABASE. The Actors Booking Database efficiently manages actor engagements, project schedules, and financial transactions in the entertainment industry. This report outlines the database schema, normalization, and key SQL queries demonstrating its robust functionality.

THE CELEBSCAPE DATABASE

COURSE CODE: CT-261

GROUP MEMBERS:

- Musfirah Saleem Shaikh [DT-009]
- Maryam Ashraff [DT-050]
- Daniya Karrar [DT-007]
- Taroob Anwer [DT-023]

COURSE TEACHER:

- **SIR ROHAIL QAMAR**

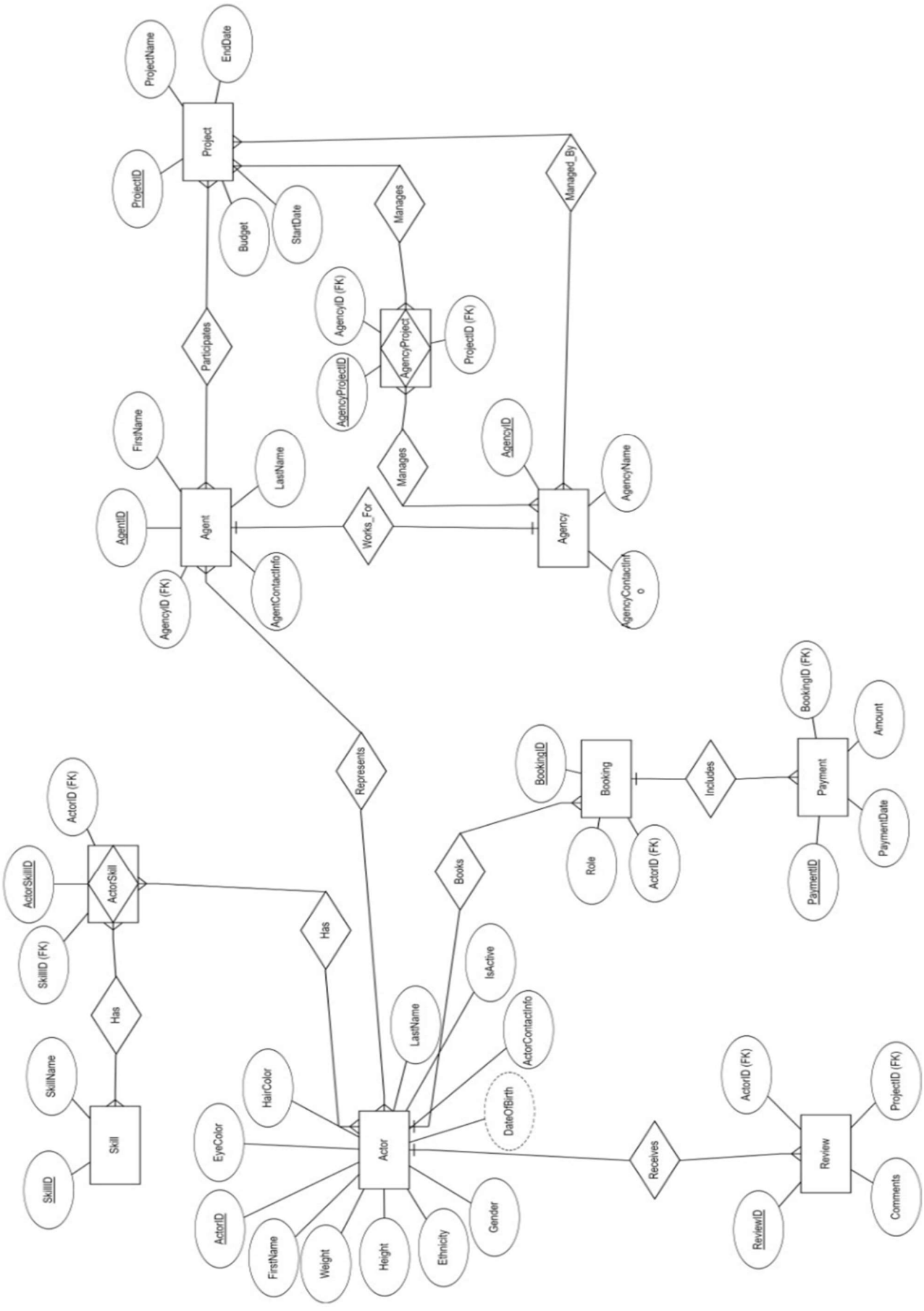
Contents

ABSTRACT:

The **CELEBSCAPE DATABASE** is designed to manage the booking, payment, and review processes of actors for various projects facilitated by multiple agencies. This database structure aims to streamline the operations involved in managing actors, agents, agencies, projects, and related transactions. The system ensures efficient data handling through normalization and supports complex queries for detailed reports.

ENTITY-RELATIONSHIP DIAGRAM (ERD):

An ERD visually represents the entities within the database and the relationships between them. Following is the ERD for the **CELEBSCAPE DATABASE**:


Project
ProjectName
EndDate
ProjectID
Budget
StartDate
Manages
Managed_By
Participates
AgencyID (FK)
AgencyProjectID
AgencyProject
ProjectID (FK)
Manages
FirstName
LastName
AgentID
Agent
Works_For
AgencyID
Agency
AgencyName
AgencyContactInfo
AgencyID (FK)
AgentContactInfo
Represents
ActorID (FK)
ActorSkillID
ActorSkill
SkillID (FK)
Has
Has
SkillName
Skill
SkillID
Books
BookingID
Booking
Role
ActorID (FK)
Includes
BookingID (FK)
Payment
Amount
PaymentDate
PaymentID
IsActive
LastName
ActorContactInfo
HairColor
DateOfBirth
EyeColor
Actor
Receives
ActorID (FK)
ProjectID (FK)
Review
ReviewID
Comments
ActorID
FirstName
Weight
Height
Ethnicity
Gender

NORMALIZATION TABLES:

Need for Normalization:

- **Reduces Redundancy: By separating skills and agents into different tables, we avoid repeating the same actor information multiple times.**
- **Improves Data Integrity: Changes in agent or skill information need to be made in only one place.**
- **Easier Maintenance: Data is easier to maintain and query because of its structured form.**

Example 1: Actor Table

Original Unnormalized Form:

a table that included actors with repetitive information about their skills and agents:

ActorID	FirstName	LastName	ContactInfo	DateOfBirth	SkillName	AgentID	AgentName	AgencyID	AgencyName
1	John	Doe	john@example.com	1980-01-01	Acting	101	Agent Smith	201	Top Talent Agency
1	John	Doe	john@example.com	1980-01-01	Dancing	101	Agent Smith	201	Top Talent Agency
2	Jane	Smith	jane@example.com	1985-02-02	Singing	102	Agent Jones	202	Star Agency

Normalized Form:

1. **Actor Table:**

```
CREATE TABLE Actor (
   ActorID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   ActorContactInfo VARCHAR(100),
   DateOfBirth DATE
);
```

ActorID	FirstName	LastName	ActorContactInfo	DateOfBirth
1	John	Doe	john@example.com	1980-01-01
2	Jane	Smith	jane@example.com	1985-02-02

2. **Skill Table:**

```
CREATE TABLE Skill (
   SkillID INT PRIMARY KEY,
   SkillName VARCHAR(50)
);
```

SkillID	SkillName
1	Acting
2	Dancing
3	Singing

3. **ActorSkill Table:**

```
CREATE TABLE ActorSkill (
   ActorSkillID INT PRIMARY KEY,
   ActorID INT,
   SkillID INT,
   FOREIGN KEY (ActorID) REFERENCES Actor(ActorID),
   FOREIGN KEY (SkillID) REFERENCES Skill(SkillID)
);
```

ActorSkillID	ActorID	SkillID
1	1	1
2	1	2
3	2	3

4. **Agent Table:**

```
CREATE TABLE Agent (
   AgentID INT PRIMARY KEY,
   AgentFirstName VARCHAR(50),
   AgentLastName VARCHAR(50),
   AgentContactInfo VARCHAR(100),
```

```
  AgencyID INT,
  FOREIGN KEY (AgencyID) REFERENCES Agency(AgencyID)
);
```

AgentID	AgentFirstName	AgentLastName	AgentContactInfo	AgencyID	AgencyName
101	Alice	Taylor	jane.newemail@example.com	201	Talent Agency
102	Chris	Anderson	chris.anderson@example.com	202	Star Agency

5. **Agency Table:**

```
CREATE TABLE Agency (
  AgencyID INT PRIMARY KEY,
  AgencyName VARCHAR(100),
  ContactInfo VARCHAR(100)
);
```

AgencyID	AgencyName	AgencyContactInfo
1	Top Talent Agency	newcontact@toptalent.com
2	Star Agency	info@staragency.com

Example 2: Booking Table

Original Unnormalized Form:

a table that included booking information with repetitive project and actor details:

BookingID	ActorID	FirstName	LastName	ProjectID	ProjectName	BookingDate	Role
1	1	John	Doe	1001	New Movie	2024-12-01	Supporting Actor
2	2	Jane	Smith	1002	TV Show	2024-03-01	Supporting Actor

Normalized Form:

1. **Booking Table:**

```
CREATE TABLE Booking (
   BookingID INT PRIMARY KEY,
   ActorID INT,
   ProjectID INT,
   BookingDate DATE,
   Role VARCHAR(50),
   FOREIGN KEY (ActorID) REFERENCES Actor(ActorID),
   FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
);
```

BookingID	ActorID	ProjectID	BookingDate	Role
1	1	1001	2024-02-01	Supporting Actor
2	2	1002	2024-03-01	Supporting Actor

2. **Actor Table:**

```
CREATE TABLE Actor (
   ActorID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
  ActorContactInfo VARCHAR(100),
   DateOfBirth DATE
);
```

ActorID	FirstName	LastName	ActorContactInfo	DateOfBirth
1	John	Doe	john@example.com	1980-01-01
2	Jane	Smith	jane@example.com	1985-02-02

3. **Project Table:**

```
CREATE TABLE Project (
   ProjectID INT PRIMARY KEY,
   ProjectName VARCHAR(100),
   StartDate DATE,
   EndDate DATE,
   Budget DECIMAL(10, 2)
);
```

ProjectID	ProjectName	StartDate	EndDate	Budget
1001	New Movie	2024-01-01	2024-12-31	1200000.00
1002	TV Show	2024-02-01	2024-11-30	2000000.00

Example 3: Review Table

Original Unnormalized Form:

a table that included review information with repetitive actor and project details:

ReviewID	ActorID	ActorName	ProjectID	ProjectName	Rating	Comments
1	1	John Doe	1001	New Movie	5	Amazing performance!
2	2	Jane Smith	1002	TV Show	4	Great job!
1	1	John Doe	1003	Commercial	3	Good effort.

Normalized Form:

1. **Review Table:**

```
CREATE TABLE Review (
   ReviewID INT PRIMARY KEY,
   ActorID INT,
   ProjectID INT,
   Rating INT,
   Comments TEXT,
   FOREIGN KEY (ActorID) REFERENCES Actor(ActorID),
   FOREIGN KEY (ProjectID) REFERENCES Project(ProjectID)
);
```

ReviewID	ActorID	ProjectID	Rating	Comments

1	1	1001	5	Amazing performance!
2	2	1002	4	Great job!
1	1	1003	3	Good effort.

2. **Actor Table:**

```
CREATE TABLE Actor (
   ActorID INT PRIMARY KEY,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   ContactInfo VARCHAR(100),
   DateOfBirth DATE
);
```

ActorID	FirstName	LastName	ActorContactInfo	DateOfBirth
1	John	Doe	john@example.com	1980-01-01
2	Jane	Smith	jane@example.com	1985-02-02

3. **Project Table:**

```
CREATE TABLE Project (
   ProjectID INT PRIMARY KEY,
   ProjectName VARCHAR(100),
   StartDate DATE,
   EndDate DATE,
   Budget DECIMAL(10, 2)
);
```

ProjectID	ProjectName	StartDate	EndDate	Budget
1001	New Movie	2024-01-01	2024-12-31	1200000.00
1002	TV Show	2024-02-01	2024-11-30	2000000.00

QUERIES:

DDL Queries:

1. **Create Table:**

```
CREATE TABLE Actor (
    ActorID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    ActorContactInfo VARCHAR(100),
    DateOfBirth DATE,
    Gender CHAR(1),
    Ethnicity VARCHAR(50),
    Height VARCHAR(50),
    Weight VARCHAR(50),
    EyeColor VARCHAR(20),
    HairColor VARCHAR(20),
    Region VARCHAR(50),
    City VARCHAR(50),
    IsActive BOOLEAN
);
```

2. **Drop Table:**

```
DROP TABLE IF EXISTS Actor;
```

3. **Alter Table:**

```
ALTER TABLE Actor ADD COLUMN MiddleName VARCHAR(50);
```

DML Queries:

1. **Insert Data:**

```
INSERT INTO Actor (ActorID, FirstName, LastName, DateOfBirth, Gender,
Ethnicity, Height, Weight, EyeColor, HairColor, Region, City, IsActive)
VALUES (1, 'John', 'Doe', '1985-02-15', 'M', 'Caucasian', '5ft 10in',
'160 lbs', 'Blue', 'Blonde', 'West', 'Los Angeles', TRUE);
```

2. **Update Data:**

```
UPDATE Actor SET ActorContactInfo = 'john.newemail@example.com' WHERE
ActorID = 1;
```

3. **Delete Data:**

```
DELETE FROM Actor WHERE ActorID = 1;
```

JOINS:

1. **Retrieve all bookings with actor and project details:**

```
SELECT b.BookingID, a.FirstName, a.LastName, p.ProjectName,
b.BookingDate, b.Role
FROM Booking b
JOIN Actor a ON b.ActorID = a.ActorID
JOIN Project p ON b.ProjectID = p.ProjectID;
```

2. **Retrieve all agents with their respective agency details:**

```
SELECT ag.AgentID, ag.FirstName, ag.LastName, ag.AgentContactInfo,
agy.AgencyName
FROM Agent ag
JOIN Agency agy ON ag.AgencyID = agy.AgencyID;
```

3. **Retrieve all reviews with actor and project details:**

```
SELECT r.ReviewID, a.FirstName, a.LastName, p.ProjectName, r.Rating,
r.Comments
FROM Review r
JOIN Actor a ON r.ActorID = a.ActorID
JOIN Project p ON r.ProjectID = p.ProjectID;
```

4. **Retrieve all payments with booking, actor, and project details:**

```
SELECT py.PaymentID, py.Amount, py.PaymentDate, b.BookingID,
a.FirstName, a.LastName, p.ProjectName
FROM Payment py
JOIN Booking b ON py.BookingID = b.BookingID
JOIN Actor a ON b.ActorID = a.ActorID
JOIN Project p ON b.ProjectID = p.ProjectID;
```

5. **Retrieve all bookings with actor details and their agency details:**

```
SELECT b.BookingID, a.FirstName, a.LastName, ag.AgentID, ag.FirstName
AS AgentFirstName, ag.LastName AS AgentLastName, agy.AgencyName
FROM Booking b
JOIN Actor a ON b.ActorID = a.ActorID
LEFT JOIN Agent ag ON a.ActorID = ag.AgentID
LEFT JOIN Agency agy ON ag.AgencyID = agy.AgencyID;
```

CONCLUSION:

The **CELEBSCAPE DATABASE** effectively manages data related to actors, agents, agencies, projects, bookings, reviews, payments, and skills. By following normalization principles, it ensures data integrity and efficiency. The use of various SQL queries for data manipulation and retrieval supports comprehensive data analysis and reporting, making it a robust system for managing actor bookings and related operations.

—THANKYOU—