**Movie** **Production** **Company**

**Abstract**

his database system orchestrates the intricate world of movie production,

managing diverse entities and their vital roles.

**At** **the** **heart** **stands** **the** **Production** **Company:** the driving force behind each

cinematic masterpiece. It's not just a name; it's a network of resources, talents, and

dreams. This entity houses the company's identity, financial standing, and workforce,

constantly striving to bring stories to life.

**Surrounding** **the** **company** **are** **its** **pillars** **of** **support:**

1. **Shareholders:** These invested individuals fuel the company's ambitions with their financial backing, ensuring a solid foundation for creativity to flourish.
2. **Employees:** The skilled crew and dedicated staff form the backbone of every film. From directing and lighting to accounting and distribution, their diverse expertise breathes life into each project.
3. **Crew** **Members:** These specialized artists bring their unique vision and technical prowess to the set. Cinematographers, editors, costume designers, and countless others weave their magic behind the scenes, transforming words into moving pictures.
4. **Staff:** The unsung heroes who keep the gears turning. From production assistants and accountants to marketing teams and legal eagles, they ensure the smooth running of every project, both on and off screen. But a company alone cannot make a movie; it thrives on collaboration.
5. **Films:** The stars of the show, each one a testament to countless hours of dedication and collaboration. These cinematic creations, born within the company's walls, connect with audiences across the globe, leaving an indelible mark on hearts and minds.
6. **Crew** **Members:** Their expertise extends beyond individual films; they form the core of a movie's production team, their skills and dedication woven into the very fabric of each project.
7. **Grants:** Sometimes, external support fuels the creative fire. Grants, offered by funding organizations, provide crucial financial injections, opening doors to ambitious projects and innovative storytelling.

**Relationships**

1. **Production Company - Shareholder Relationship:**

* A production company can have multiple shareholders.
* A shareholder is associated with a specific production company.
* The `production\_company\_shareholder` table facilitates this many-to-many relationship, linking production companies and shareholders. Each entry in this table represents a connection between a production company and a shareholder.

**Production - company - shareholder:**

* production\_company\_id (foreign key references production\_company)
* shareholder\_id (foreign key references shareholder)

2. **Production Company - Employee Relationship:**

* A production company can have multiple employees, including both Crew and Staff.
* An employee belongs to only one production company.
* The `employee` table represents this one-to-many relationship. The `employee\_type` field distinguishes between Crew and Staff members.

**employee:**

* employee\_id (primary key)
* production\_company\_id (foreign key references production\_company)
* employee\_type (e.g., Crew or Staff)
* other\_employee\_attributes

3. **Film - Crew Member Relationship:**

* A film is produced by a single production company.
* A film can have multiple crew members.
* The `film\_crew\_member` table manages this one-to-many relationship, connecting films and crew members.

**Film - crew - member:**

* film\_id (foreign key references film)
* crew\_member\_id (foreign key references crew\_member)

4. **Grant - Production Company Relationship:**

* A grant is associated with a specific production company.
* Multiple production companies can apply for the same grant.
* The `grant` table represents this one-to-many relationship. The `production\_company\_id` field connects the grant to the production company that applies for it.

**grant:**

* grant\_id (primary key)
* production\_company\_id (foreign key references production\_company)
* other\_grant\_attributes

5. **Employee - Crew/Staff Relationship:**

* An employee is either a crew or staff member.
* A crew or staff member is a specific type of employee.
* The `employee` table includes an `employee\_type` field to distinguish between Crew and Staff members. This is a straightforward one-to-many relationship.

**employee:**

* employee\_id (primary key)
* employee\_type (e.g., Crew or Staff)
* other\_employee\_attributes

6. **Grant - Production Company - Employee Relationship:**

* An employee from a production company is associated with a grant.
* A grant is associated with a specific production company.
* The `grant\_production\_company\_employee` table connects grants, production companies, and employees, allowing for tracking which employees from which production companies are associated with specific grants.

**Grant - production - company - employee:**

* grant\_id (foreign key references grant)
* production\_company\_id (foreign key references production\_company)
* employee\_id (foreign key references employee)

**Entity**

**Production Company:**

**Employs:** Provides a stage for skilled professionals like crew and staff.

**Produces:** Births movies, each a unique expression of the company's creative vision. **Receives:** Financial support through grants and investment from shareholders.

**Shareholders:**

**Invests:** Injects financial muscle into the company, powering its creative endeavors. **Benefits:** Raps the reward of successful productions through dividends and companygrowth.

**Relies:** On the company's leadership and vision to turn investments into impactful films.

**Employees:**

**Drives:** The engine of film production, contributing diverse expertise through crewand staff roles.

**Belongs:** Dedicated to a single production company, forming its core workforce. **Contributes:** To multiple films throughout their tenure, leaving their mark on various cinematic stories.

**Crew Members:**

**Brings:** Specialized skills and artistic vision to specific film projects.

**Collaborates:** With fellow crew members and staff to execute the director's vision. **Belongs:** To the broader pool of company-employed crew, their expertise availablefor future projects.

**Staff:**

**Supports:** Every stage of film production, ensuring smooth operations both on and off set.

**Serves:** The needs of crew members and the company as a whole, keeping the creative engine running.

**Specializes:** Within specific departments like accounting, marketing, or legal, contributing their essential skills to every project.

**Films:**

**Embodies:** The culmination of collective effort, showcasing the talent of crew, staff, and company leadership.

**Connects:** With audiences worldwide, sparking emotions and sharing unique perspectives.

**Relies:** On the production company's resources and expertise for its birth and release.

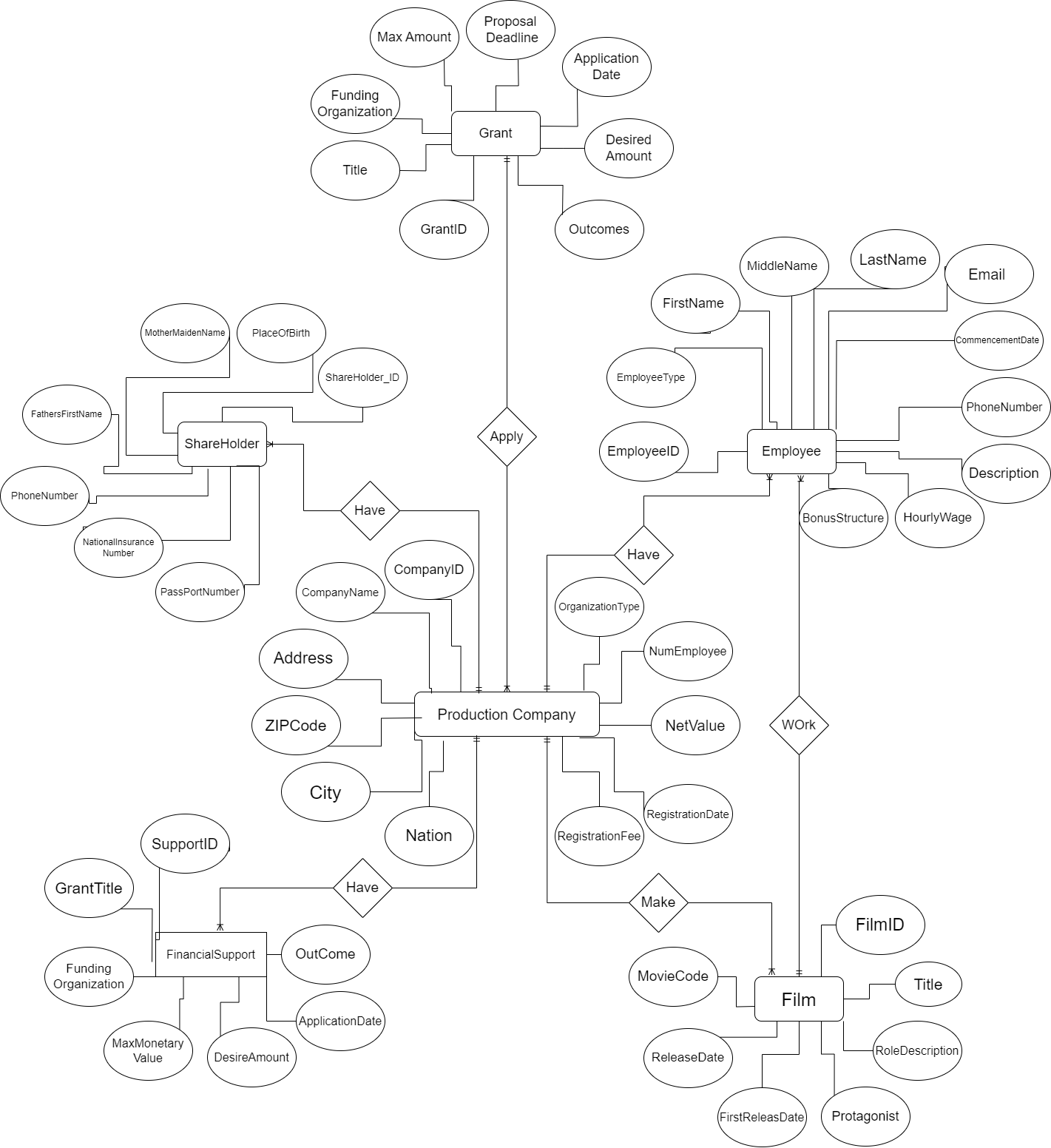
**Grants:**

**Boosts:** Specific films or company initiatives with financial injections.

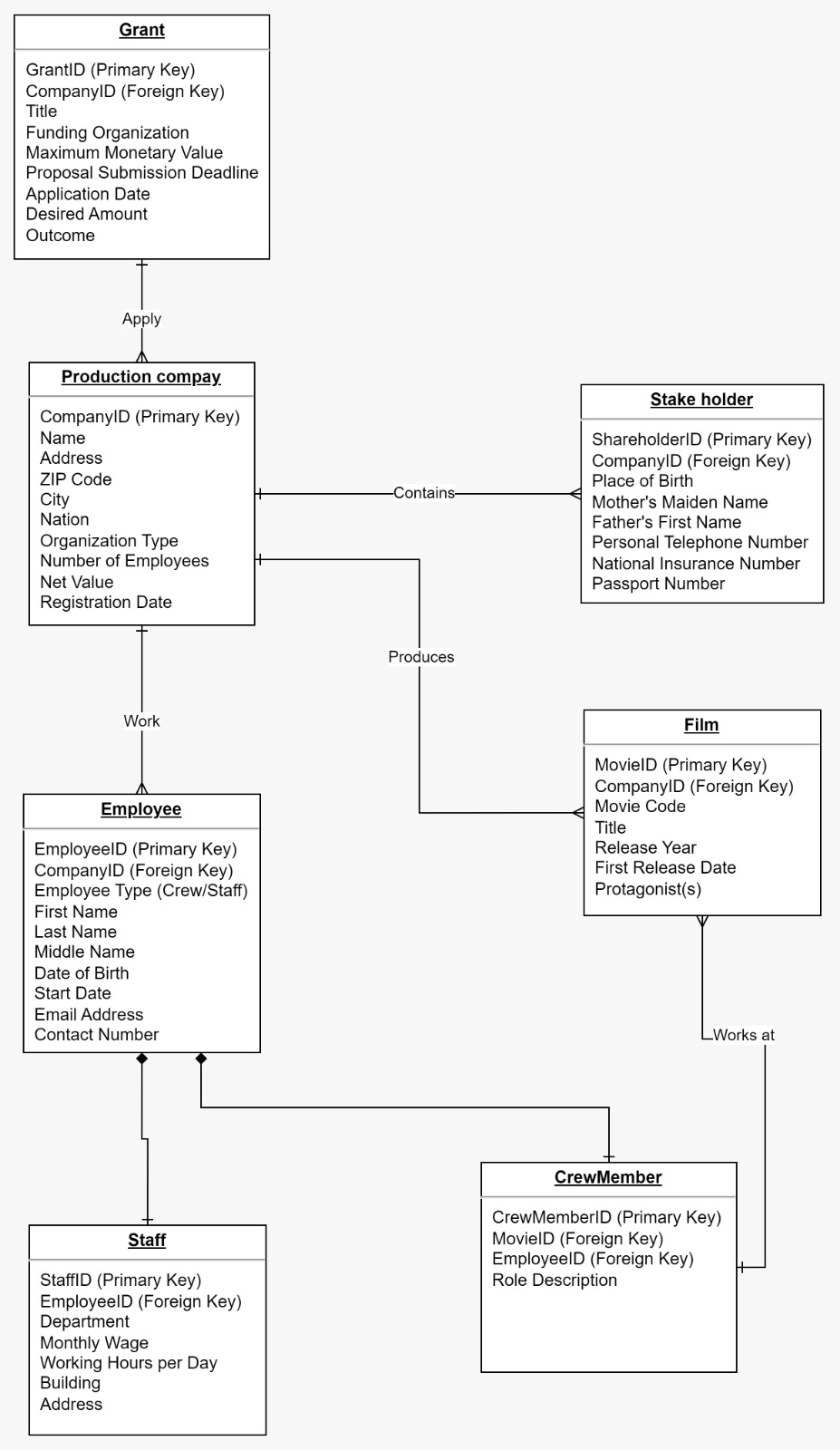
**Motivates:** Innovation and ambitious storytelling through targeted funding.

**Seeks:** Partnerships with production companies that align with its values and objectives.

**ERD (Chan’s Notation)**

****

**ERD**

****

**Normalization**

Normalization is a database design technique used to organize data in a relational database efficiently. It aims to minimize data redundancy and dependency by organizing fields and table of a database. The process involves dividing a large table into smaller tables and defining relationships between them. The primary goals of normalization are to reduce data redundancy, eliminate update anomalies, and ensure data integrity. There are several normal forms, each building upon the previous one. The most commonly used normal forms are:

**First Normal Form(1NF):**

Each table cell should contain a single value. Each record needs to be unique.

As our table do not contain any value for now so 1st Normalization is not Applied.Identify a unique column or set of columns (primary key) for each table.

**Second Normal Form (2NF):**

For a table to be in 2NF, it must first be in 1NF. Additionally, the table should not exhibit partial dependency, meaning that no proper subset of the candidate key should determine a non-prime attribute. The diagram illustrates separate entities after applying 2NF.

**Third Normal Form (3NF):**

To achieve 3NF, the table must already be in 2NF. Furthermore, there should be no transitive dependency for non-prime attributes, ensuring that non-prime attributes do not depend on other non-prime attributes. This normalization level aims to minimize data duplication, promoting data integrity. The diagram depicts the application of 3NF, with arrows indicating the normalization process.

Now, let's go through a step-by-step example to illustrate normalization:

Table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EmployeeID | EmployeeName | Companyname | CompanyLocation | GrantID | GrantName |
| 1 | John | IT | New York | 101 | lenovo |
| 2 | Jane | Gaming | Los Angeles | 102 | Biography |
| 3 | Bob | IT | New York | 101 | lenovo |

**First Normal Form (1NF):**

The table is already in 1NF as there are no repeating groups or arrays, and each cell contains a single value.

**Second Normal Form (2NF):**

The primary key is EmployeeID.

There are partial dependencies on the primary key. The non-prime attribute GrantName is dependent on the partial key {GrantID}.

Create a new table for the Grant information:

|  |  |
| --- | --- |
| GrantID | GrantName |
| 101 | lenovo |
| 102 | Biography |

Create a new table for the Company information:

|  |  |
| --- | --- |
| Companyname | CompanyLocation |
| IT | New York |
| Gaming | Los Angeles |

Create a new table for the Employee information:

|  |  |
| --- | --- |
| EmployeeID | EmployeeName |
| 1 | John |
| 2 | Jane |
| 3 | Bob |

**Third Normal Form (3NF):**

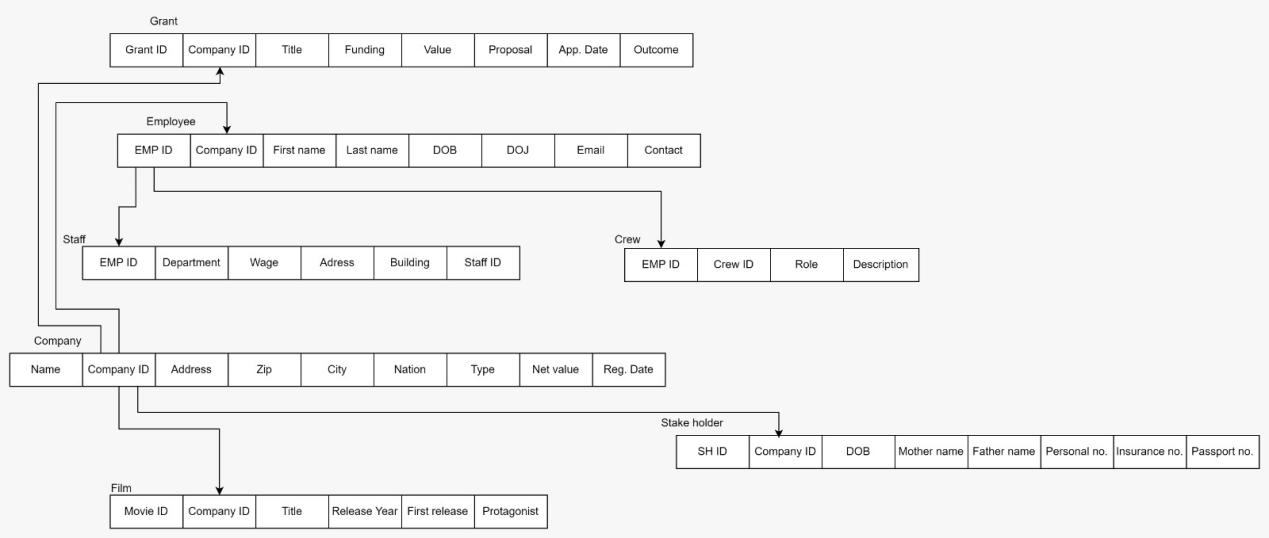
The primary key is EmployeeID.

There is a transitive dependency on the primary key. DepartmentLocation depends on Department, which is not the primary key.

Create a new table for Department information:

|  |  |  |  |
| --- | --- | --- | --- |
| EmployeeID | EmployeeName | Companyname | GrantID |
| 1 | John | IT | 101 |
| 2 | Jane | Gaming | 102 |
| 3 | Bob | IT | 101 |

Now, the table is in 3NF, and each table is free from transitive dependencies. This process helps in reducing redundancy and ensures that data is stored in a more efficient and maintainable manner. This is just a small part Example Complete for this scenario is given below :

****

**SQL Query**

* **ProductionCompany table**

CREATE TABLE production\_company\_table (

company\_id VARCHAR(255) PRIMARY KEY,

name VARCHAR(255),

address VARCHAR(255),

zip\_code VARCHAR(10),

city VARCHAR(255),

nation VARCHAR(255),

organization\_type VARCHAR(255),

employee\_count VARCHAR(10),

net\_value VARCHAR(255),

registration\_date VARCHAR(20)

);

* **Employee table**

CREATE TABLE employee\_table (

employee\_id SERIAL PRIMARY KEY,

company\_id VARCHAR(255) REFERENCES production\_company\_table(company\_id),

employee\_type VARCHAR(50) CHECK (employee\_type IN ('Crew', 'Staff')),

first\_name VARCHAR(255),

middle\_name VARCHAR(255),

last\_name VARCHAR(255),

dob DATE,

start\_date DATE,

email\_address VARCHAR(255),

contact\_number VARCHAR(15),

UNIQUE (company\_id, employee\_id)

);

* **Grant table**

CREATE TABLE grant\_table (

grant\_id SERIAL PRIMARY KEY,

company\_id VARCHAR(255) REFERENCES production\_company\_table(company\_id),

title VARCHAR(255),

funding\_organization VARCHAR(255),

max\_value INT CHECK (max\_value >= 0),

deadline DATE,

application\_date DATE,

desire\_amount INT CHECK (desire\_amount >= 0),

UNIQUE (company\_id, grant\_id)

);

* **Staff table**

CREATE TABLE staff\_table (

staff\_id VARCHAR(255) PRIMARY KEY,

employee\_id VARCHAR(255) REFERENCES employee\_table(employee\_id),

department VARCHAR(255),

monthly\_wage VARCHAR(255),

working\_hours VARCHAR(255),

building VARCHAR(255),

address VARCHAR(255),

CHECK (monthly\_wage >= 0),

UNIQUE (employee\_id)

);

* **Stakeholder table**

CREATE TABLE stakeholder\_table (

company\_id VARCHAR(255) REFERENCES production\_company\_table(company\_id),

stakeholder\_id VARCHAR(255),

place\_of\_birth VARCHAR(255),

mother\_name VARCHAR(255),

father\_name VARCHAR(255),

phone\_number VARCHAR(255),

insurance\_number VARCHAR(255),

passport\_number VARCHAR(255),

PRIMARY KEY (company\_id, stakeholder\_id),

UNIQUE (stakeholder\_id)

);

* **Movie table**

CREATE TABLE movie\_table (

movie\_id VARCHAR(255) PRIMARY KEY,

company\_id VARCHAR(255) REFERENCES production\_company\_table(company\_id),

movie\_code VARCHAR(255),

title VARCHAR(255),

release\_year VARCHAR(255),

release\_date VARCHAR(255),

protagonist VARCHAR(255),

UNIQUE (company\_id, movie\_id)

);

-- Crew table

CREATE TABLE crew\_table (

crew\_id VARCHAR(255) PRIMARY KEY,

movie\_id VARCHAR(255) REFERENCES movie\_table(movie\_id),

employee\_id VARCHAR(255) REFERENCES employee\_table(employee\_id),

role\_description VARCHAR(255),

UNIQUE (movie\_id, employee\_id)

);

**Data Types**

1. **VARCHAR(10):**

Similar to VARCHAR(255), but with a shorter maximum length of 10 characters. It's used for storing zip codes, which are typically short strings.

2. **DATE:**

The DATE data type is used to store date values, such as the date of birth (`dob`), start date (`start\_date`), deadline (`deadline`), application date (`application\_date`), and release date (`release\_date`).

3. **SERIAL:**

SERIAL is an auto-incrementing integer. It's commonly used for primary key columns to ensure each record has a unique identifier. In this schema, SERIAL is used for `employee\_id`, `grant\_id`, `staff\_id`, and `crew\_id`.

1. **INT:**

This data type is used for integer values. In the schema, it's used for `max\_value` and `desire\_amount`, representing integer values for funding grants.

5. **VARCHAR(20):**

Used for variable-length character strings with a maximum length of 20 characters. It's employed for `registration\_date`, which may store a date or timestamp related to the registration of a production company.

**Constraints**

1. **PRIMARY KEY:**

The PRIMARY KEY constraint is used to uniquely identify a record in a table. For example, `company\_id` is the primary key in the `production\_company\_table`, ensuring each company has a unique identifier.

2. **FOREIGN KEY:**

The FOREIGN KEY constraint establishes a link between two tables based on a related column. For instance, `company\_id` in the `employee\_table` references the `company\_id` in the `production\_company\_table`, creating a relationship between employees and production companies.

3. **CHECK:**

The CHECK constraint ensures that values in a column satisfy a specified condition. For instance, `employee\_type` must be either 'Crew' or 'Staff', and `max\_value` and `desire\_amount` must be non-negative.

4. **UNIQUE:**

The UNIQUE constraint ensures that all values in a column (or a combination of columns) are unique. For instance, the combination of `company\_id` and `employee\_id` must be unique in the `employee\_table`, preventing duplicate employee records within a company. The same principle applies to other unique constraints in the schema.

**Update Operations**

**Update Production Company's Address**

UPDATE production\_company\_table

SET address = 'New Address'

WHERE company\_id = 'XYZ123';

This SQL statement updates the address of a production company with company\_id 'XYZ123' to the new value 'New Address'.

**Update Employee's Email Address**

UPDATE employee\_table

SET email\_address = 'new\_email@example.com'

WHERE employee\_id = 1;

This SQL statement updates the email\_address of an employee with employee\_id 1 to the new value 'new\_email@example.com'.

**Delete Operations**

**Delete Production Company and associated data**

DELETE FROM production\_company\_table

WHERE company\_id = 'XYZ123';

This SQL statement deletes a production company with company\_id 'XYZ123'. It is assumed that this operation will also delete any associated data related to this production company.

**Delete Employee and associated data**

DELETE FROM employee\_table

WHERE employee\_id = 1;

This SQL statement deletes an employee with employee\_id 1. Similar to the previous operation, it's assumed that this operation will delete any associated data related to this employee.

**Add Operations**

**Insert a new Production Company**

INSERT INTO production\_company\_table (company\_id, name, address, zip\_code, city, nation, organization\_type, employee\_count, net\_value, registration\_date)

VALUES ('NEW789', 'New Production Company', 'New Address', '12345', 'New City', 'New Nation', 'Private', '50', '500000', '2023-01-01');

This SQL statement inserts a new production company with details such as company\_id, name, address, zip\_code, etc.

**Insert a new Crew Member**

INSERT INTO crew\_table (crew\_id, movie\_id, employee\_id, role\_description)

VALUES ('CREW456', 'MOV789', '2', 'Director');

This SQL statement inserts a new crew member with details such as crew\_id, movie\_id, employee\_id, role\_description, etc.

**Front-end interfaces**

Java is a general-purpose, high-level, and object-oriented programming language. Java is widely used in web development, enterprise applications, mobile app development (Android), and various other domains. I have Used Java to make Front End Interface .

**Components of the Java Program**

1. **Import Statements:**

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JTextField;

import java.awt.Color;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

These statements import necessary classes and packages from Java's Swing library for creating a graphical user interface (GUI) and JDBC for database connectivity.

1. **Class Declaration:**

public class Add\_Crew {

The program defines a public class named `Add\_Crew`.

3. **Main Method:**

public static void main(String[] s) {

This is the entry point of the program, where the main execution begins.

4. **Frame Initialization:**

JFrame f = new JFrame("Add Crew Member")

It creates a new JFrame (a top-level container) with the title "Add Crew Member".

5. **Labels and TextFields:**

JLabel l1 = new JLabel("Crew ID");

final JTextField tf1 = new JTextField();

JLabels are used to display text, and JTextFields allow the user to input text.

6. **Buttons:**

JButton b1 = new JButton("Add Crew Member");

JButton is used to create a clickable button. In this case, it's labeled "Add Crew Member".

7. **ActionListener Implementation:**

b1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// Action to be performed when the button is clicked

}

});

```

An ActionListener is implemented to define the action to be performed when the "Add Crew Member" button is clicked.

8. **Database Connection and Insertion:**

try (Connection connection = DriverManager.getConnection(url, user, password)) {

// Database connection

String query = "INSERT INTO crew\_table (crew\_id, movie\_id, employee\_id, role\_description) VALUES (?, ?, ?, ?)";

try (PreparedStatement preparedStatement = connection.prepareStatement(query)) {

// Set values and execute the query

}

} catch (SQLException ex) {

// Handle database connection errors

}

It establishes a connection to the MySQL database using JDBC and inserts data into the `crew\_table`.

9. **Message Dialogs:**

JOptionPane.showMessageDialog(f, "Crew member added successfully!");

JOptionPane is used to display dialog boxes, in this case, to show a message about the success or failure of the operation.

10. **Back Button and Navigation:**

JButton b2 = new JButton("Back");

b2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

// Action to be performed when the "Back" button is clicked

}

});

Another button is provided to go back to the previous screen (assuming there's a `Navigation` class with a `main` method).

11. **Frame Configuration:**

f.add(tf1);

f.add(l1);

// ... (similar lines for other components)

f.setSize(800, 600);

f.getContentPane().setBackground(Color.white);

f.setLayout(null);

f.setVisible(true);

Components are added to the frame, and the frame is configured with a size, background color, layout, and visibility.

12. **Closing the Frame:**

f.dispose();

Disposes the frame when the "Back" button is clicked.

13. **Class Closing:**

}

Closes the `main` method and class.

**Conclusion**

The presented movie production company database system, along with its front-end Java interface, encapsulates the intricacies of managing a dynamic film production environment. The entity-relationship diagram (ERD) captures the relationships between production companies, shareholders, employees, crew members, films, and grants, providing a comprehensive overview of the interconnected elements in the movie-making process. The normalization process ensures efficient data organization, minimizing redundancy and dependency, while the SQL queries illustrate how data is stored in the backend. The Java front-end interface offers a user-friendly means of interacting with the database, allowing for the addition of crew members with ease. The system supports essential operations such as updating, deleting, and adding records, fostering a robust and flexible foundation for movie production management. Overall, the integration of database design, normalization techniques, SQL, and Java programming enhances the efficiency and coherence of movie production company operations.