NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

SECOND YEAR (3rd Semester) - 2023

DATABASE MANAGEMENT SYSTEM

Submitted By-

Sanket Vashisht-2022UCS1507

INDEX

- IMPLEMENTATION
  - XAMPP TABLES
  - APPLICATION WITH CODE
    - HTML, CSS, JAVASCRIPT, NODEJS, MYSQL, TRIGGERS
  - VIEW OF THE APPLICATION

- REFERENCES

INTRODUCTION

Introduction Of The Problem Area

The entertainment industry, including series production, is dynamic and multifaceted. With numerous production teams working concurrently on various projects, each with its own set of team members, timelines, and platforms for release, it becomes increasingly challenging to maintain a streamlined and efficient workflow. The need for a comprehensive information management system arises from the complexity of juggling diverse teams, talents, and series across multiple platforms. Without a centralized system, it's easy to lose track of resources, timelines, and the overall production landscape, leading to potential delays, mismanagement, and a decrease in overall productivity.

Problem Statement

A renowned series production house, is in the process of developing a comprehensive
system to manage and organize information about its diverse production teams, talented team members, popular released series like “Stranger Things” and “The Crown”, and the various platforms such as Amazon Prime, and Disney+ on which these series were released. This innovative system is designed to enable the production house to effectively track and manage its valuable production resources and assets, much like a GPS tracking system for a fleet of vehicles. This will ensure optimal utilization of resources, timely release of series, and ultimately, customer satisfaction.

Objective Of the Project

The primary objective of this project, “Series Production Management System”, is to create an innovative and centralized information management system for the renowned series production house. This system aims to streamline and enhance the organization of crucial data related to production teams, team members, and released series. By doing so, it seeks to provide the production house with a comprehensive tool to track and manage its valuable resources and assets efficiently. The ultimate goal is to optimize production workflows, ensure timely releases of series, and enhance overall customer satisfaction. Through this project, the production house aspires to bring about a GPS-like precision to the management of its dynamic and diverse production environment, fostering a more organized, productive, and successful series production process.

Requirement Analysis

Production house (production firm) database would be built considering the following needs and constraints:

- A series production firm has many sub-departments which play a crucial role in its production, promotion, and sales.

Example would be the genre it's going to belong (as different genres target different mass of audience), previous series released by the firm would set the benchmark for newer constraints and no other way than the rating system can be better in this.

- Script writers (based on which series is based) is the elementary step towards the production of a good series. Only one production house can work on a series written by a series writer. A production house can work on multiple series at a time.

- At a time only one release group will be allowed to release the series deciding on which platform it has to be released the best time of release when it's release would peak. Yet again a release group can release multiple series.

- A production team, which is the reason behind every service is a key role. At a time only one team will be in charge of producing the series.

- A production house needs many departments due to such a diverse range of skill sets required to create a series. Be it voice artists or actors. A production team has multiple departments working under it and a department can work for multiple projects.

- A department consists of multiple employees, but due to exceptional work that is expected out of their skill set, they don't swing in their areas of expertise.

HARDWARE REQUIREMENT

Hardware Tools

- Laptop or Computer
- Storage Devices
- Backup System
- Network Connection
- Firewall and Security Devices
- Power Supply

Software Tools

- Visual Studio Code
- MySQL Workbench
- Xampp
- Programming Languages like HTML, CSS, JavaScript, Python, Nodejs
- Web Browser

DESIGN

Entity Relationship Model

An ER model is a diagram containing entities or “items”, relationships among them, and the attributes of the entities and the relationships.

Entities for the series production management system are:

- productions
- departments
- Series
- produces
- Genre
- employees
- works_for
- has_departments
- Cast
- CrewMembers
- Script_writer
- Grievances

- ReleaseGroup

Attributes of the entities in the series production management system are:

Productions

- production_id (Primary Key)
- production_firm

departments

- department_id (Primary Key)
- department_name
- production_id (Foreign Key)

Series

- SeriesId (Primary Key)
- SeriesName
- Reviews
- Budget

produces

- Production_id (Foreign Key)
- SeriesId (Foreign Key)

Genre

- SeriesId (Foreign Key)
- genre_type

employees

- emp_id (Primary Key)
- first_name
- last_name
- middle_name
- DOB
- department_id (Foreign Key)

works_for

- Seriesid (Foreign Key)
- emp_id (Foreign Key)

has_departments

- department_id (Foreign Key)
- production_id (Foreign Key)

Cast

- cast_id (Primary Key)
- emp_id (Foreign Key)
- num_of_episodes
- name_in_series

CrewMembers

- crew_member_id (Primary Key)
- emp_id (Foreign Key)
- contract_duration
- designation

Script_writer

- script_id (Primary Key)
- srcipt_name
- emp_id (Foreign Key)

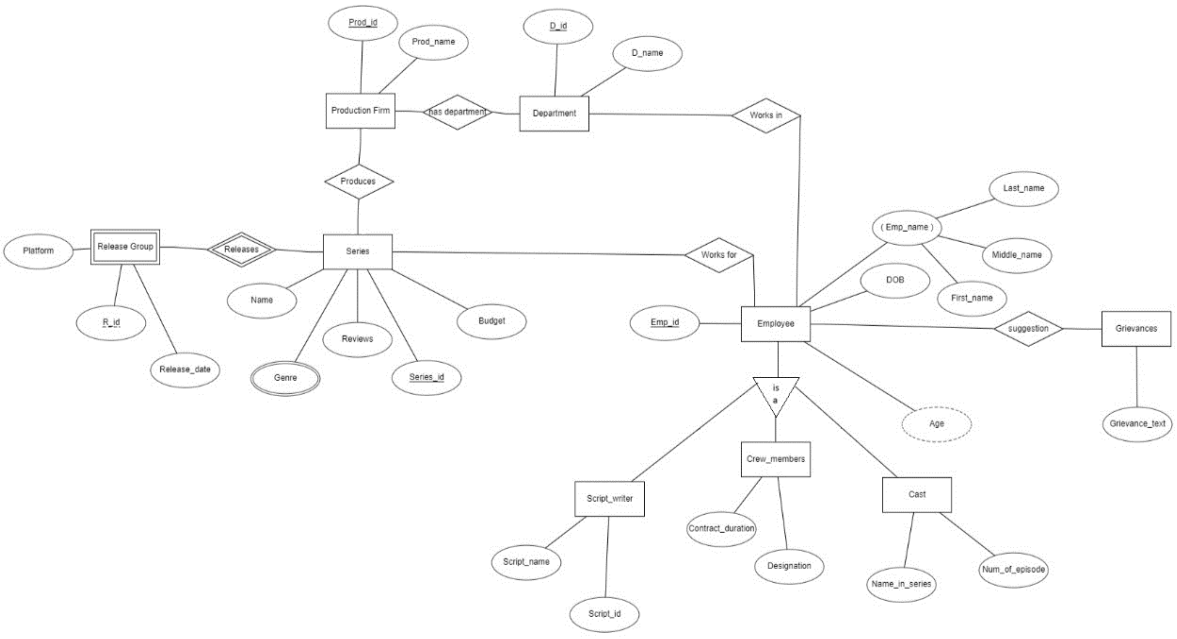
Grievances

- emp_id (Primary Key, Foreign Key)
- grievances_text

ReleaseGroup

- R_ID (Primary Key)
- Platform
- ReleaseDate
- SeriesID (Foreign Key)

ER DIAGRAM



ER To Schema

A schema refers to the logical structure of a database. It defines how the data is organized, how the different entities and their relationships are represented, and the rules and constraints that govern the data stored in the database. Database schema is essential for maintaining data consistency, enforcing data integrity, and ensuring that data is stored and retrieved accurately in a DBMS.

Table Name	Attributes	Foreign Keys
productions	production_id (PK), production_firm	
departments	department_id (PK), department_name	
has_departments	(PK) department_id, (PK) production_id	FK(department_id), FK(production_id)
Series	SeriesID (PK), SeriesName, Reviews, Budget	
produces	(PK) production_id, (PK) SeriesID	FK(production_id), FK(SeriesID)
Genre	(PK) SeriesID, (PK) genre_type	FK(SeriesID)
employees	emp_id (PK), first_name, last_name, middle_name, DOB, department_id	FK(department_id)

Functional Dependencies

Table	Functional Dependencies
productions	production_id → production_firm
departments	department_id → department_name
Series	SeriesID → SeriesName, Reviews, Budget
produces	(production_id, SeriesID) → {}
Genre	SeriesID, genre_type → {}
employees	emp_id → first_name, last_name, middle_name, DOB, department_id
works_for	(SeriesID, emp_id) → {}
has_departments	(department_id, production_id) → {}

Table	Functional Dependencies
Cast	cast_id → emp_id, num_of_episodes, name_in_series
CrewMembers	crew_member_id → emp_id, contract_duration, designation
Script_writer	script_id → script_name, emp_id
Grievances	emp_id → grievance_text
ReleaseGroup	R_ID → Platform, ReleaseDate, SeriesID

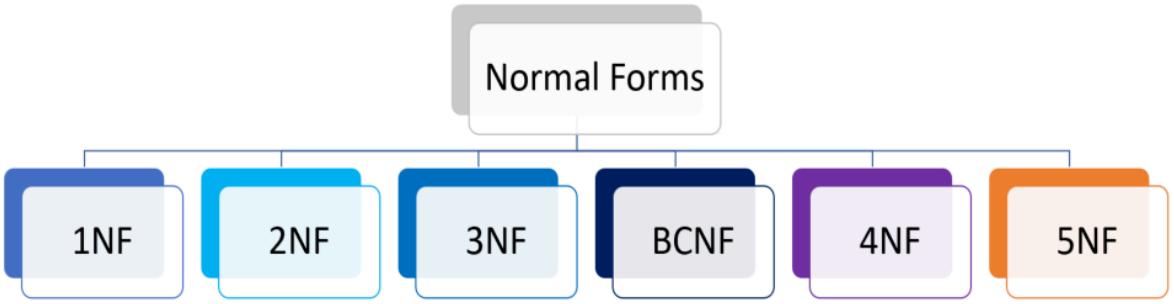
Relationship Between Tables

- The Relationship between Release Group and Series is **one to many** because one release group can release more than one particular series but one series can be released by only one release group.
- The Relationship between Series and Production firms is **many to many** because at a particular moment multiple production firms can produce multiple series.

- The Relationship between Production firm and Department is **many to many** because a Production firm can have multiple departments and also, a particular department of a particular production firm can work for another Production firm.
- The Relationship between department and employee is **one to many** because one department can have many employee to work under them but each individual employee can work for only one department.
- The Relationship between series and employee is **many to many** because each employee can individually work for different series.
- The relationship between employee and Grievance is **one to one** because each employee has its own grievance and each grievance is related to only one employee.

Normalization

- Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy.
- Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them.



1st Normal Form

In 1NF, a relation/table is said to be in this normal form if it satisfies the following conditions:

- It has a primary key that uniquely identifies each row.

- Each column in the table contains atomic values.

If a table does not satisfy 1NF, it can be converted to 1NF by splitting the columns with multiple values into separate columns, or by creating a new table for the repeating groups.

1NF ensures that a database schema is free from redundancy and that each attribute is atomic, which helps to eliminate the data anomalies during insert, delete, and update operations.

Here's an overview of the 1NF compliance of each table in the database:

1. The “Production” table is in 1NF, as each column has a unique name and contains atomic values.
2. The “Departments” table is in 1NF, as each column has a unique name and contains atomic values.
3. The “Series” table is in 1NF, as each column has a unique name and contains atomic values.
4. The “Produces” table is in 1NF, as each column has a unique name and contains atomic values.
5. The “Genre table” is in 1NF, as each column has a unique name and contains atomic values.
6. The “Employees” table is in 1NF, as each column has a unique name and contains atomic values.
7. The “works_for table” is in 1NF, as each column has a unique name and contains atomic values.
8. The “Has_departments” table is in 1NF, as each column has a unique name and contains atomic values.
9. The “Cast” table is in 1NF, as each column has a unique name and contains atomic values.
10. The “CrewMembers” table is in 1NF, as each column has a unique name and contains atomic values.
11. The “Script_wiriter” table is in 1NF, as each column has a unique name and contains atomic values.
12. The “Grievances” table is in 1NF, as each column has a unique name and contains atomic values.

13. The “Release Group” table is in 1NF, as each column has a unique name and contains atomic values.

In summary, all the tables in the database are in 1NF as they meet the conditions for unique name and atomic values.

2nd Normal Form

To be in 2NF, a relation/table must first be in 1NF (first normal form). Additionally, it must satisfy the following two conditions:

- It is 1NF.
- It must not have any partial dependencies: A partial dependency exists when a non-key attribute is functionally dependent on only a part of the primary key. In other words, if a non-key attribute depends on only a subset of the primary key, then it is in partial dependency. To remove this, we need to split the table into two separate tables where the partial dependent attributes are removed from the original table and put into a new table with the subset of the primary key.

Here's an explanation of 2NF and how each table in this database is in 2NF:

2NF (Second Normal Form) is a normal form that states that a table should not have any partial dependencies. In other words, a table is in 2NF if it is already in 1NF and does not have any non-prime attribute that is dependent on only a part of the primary key.

In this database, each table is already in 1NF, and none of the tables have partial dependencies. Therefore, all tables are already in 2NF.

Here's a brief explanation of why each table is in 2NF:

1. The “Productions” table has no partial dependency as production_firm depends on the Primary Key production_id. Therefore, this table is in 2NF.

2. The “Departments” table has no partial dependency as department_name and production_id depend on the primary key department_id. Therefore, this table is in 2NF.
3. The “Series” table has no partial dependency as SeriesName, Reviews, and Budget depend on the primary key SeriesID. Therefore, this table is in 2NF.
4. The “Produces” table has no partial dependency as both production_id and SeriesID together form the primary key. Therefore, this table is in 2NF.
5. The “Genre” table has no partial dependency as genre_type depends on the primary key SeriesID. Therefore, this table is in 2NF.
6. The “Employees” table has no partial dependency as first_name, last_name, middle_name, DOB, and department_id depend on the primary key emp_id. Therefore, this table is in 2NF.
7. The “works_for” table has no partial dependency as both Seriesid and emp_id together form the primary key. Therefore, this table is in 2NF.
8. The “has_departments” has no partial dependency as both department_id and production_id together form the primary key. Therefore, this table is in 2NF.
9. The “Cast” table has no partial dependency as num_of_episodes and name_in_series depend on the primary key cast_id. Therefore, this table is in 2NF.
10. The “CrewMembers” table has no partial dependency as contract_duration and designation depend on the primary key crew_member_id. Therefore, this table is in 2NF.
11. The “Script_writer” table has no partial dependency as script_name depends on the primary key script_id. Therefore, this table is in 2NF.
12. The “Grievances” table has no partial dependency as grievance_text depends on the primary key emp_id. Therefore, this table is in 2NF.
13. The “ReleaseGroup” table has no partial dependency as Platform and ReleaseDate depend on the primary key R_ID. Therefore, this table is in 2NF.

In summary, all the tables in the database are in 2NF as they meet the specified criteria.

3rd Normal Form

3NF stands for third normal form in database normalization. It is a property that a relation/table must satisfy in order to be considered well-designed and free from certain types of anomalies.

To be in 3NF, a relation/table must first be in 2NF (second normal form). Additionally, it must satisfy the following condition:

- It must not have any transitive dependencies between non-key attributes: A transitive dependency exists when a non-key attribute is functionally dependent on another non-key attribute. In other words, if a non-key attribute depends on another non-key attribute, which in turn depends on the primary key, then it is in transitive dependency. To remove this, we need to split the table into two separate tables, where the dependent attributes are removed from the original table and put into a new table with their respective attributes on which they are dependent.

The purpose of 3NF is to ensure that a table is free from anomalies that may occur due to transitive dependencies between non-key attributes. By removing these dependencies, we can ensure that each attribute in a relation/table is dependent only on the primary key, and hence the table is free from certain anomalies that may occur during data manipulation.

To meet the requirements of 3NF, the following conditions must be satisfied:
1. The table should be in 2NF.
2. No non-key attribute should be transitively dependent on the primary key.

In other words, no non-key attribute should be indirectly dependent on the primary key through another non-key attribute.
In this database, each table is already in 3NF, and none of the tables have transitive dependency for non-prime attributes. In other words, non-prime attributes should be functionally dependent only on the primary key. Therefore, all tables are already in 3NF.

Here's a brief explanation of why each table is in 3NF:

1. The “Productions” table has no transitive dependency, and production_firm depends only on the primary key production_id. Therefore, this table is in 3NF.
2. The “Departments” table has no transitive dependency, and department_name and production_id depend only on the primary key department_id. Therefore, this table is in 3NF.
3. The “Series” Table has no transitive dependency, and SeriesName, Reviews, and Budget depend only on the primary key SeriesID. Therefore, this table is in 3NF.
4. The “Produces” Table has no transitive dependency, and both production_id and SeriesID are part of the primary key. Therefore, this table is in 3NF.
5. The “Genre” Table has no transitive dependency, and genre_type depends only on the primary key SeriesID. Therefore, this table is in 3NF.
6. The “Employees” Table has no transitive dependency, and first_name, last_name, middle_name, DOB, and department_id depend only on the primary key emp_id. Therefore, this table is in 3NF.
7. The “works_for” Table has no transitive dependency, and both Seriesid and emp_id are part of the primary key. Therefore, this table is in 3NF.
8. The “has_departments” table has no transitive dependency, and both department_id and production_id are part of the primary key. Therefore, this table is in 3NF.
9. The “Cast” Table has no transitive dependency, and num_of_episodes and name_in_series depend only on the primary key cast_id. Therefore, this table is in 3NF.

10. The “CrewMembers” Table has no transitive dependency, and contract_duration and designation depend only on the primary key crew_member_id. Therefore, this table is in 3NF.
11. The “Script_writer” Table has no transitive dependency, and script_name depends only on the primary key script_id. Therefore, this table is in 3NF.
12. The “Grievances” Table has no transitive dependency, and grievance_text depends only on the primary key emp_id. Therefore, this table is in 3NF.
13. The “ReleaseGroup” Table has no transitive dependency, and Platform and ReleaseDate depend only on the primary key R_ID. Therefore, this table is in 3NF.

In summary, all the tables in the database are in 3NF as they meet the specified criteria.

BCNF

BCNF stands for Boyce-Codd Normal Form, which is a level of database normalization that is stronger than the third normal form (3NF). The main goal of BCNF is to eliminate the redundancy and anomalies that may arise due to the presence of functional dependencies.

A relation/table is said to be in BCNF if it satisfies the following conditions:

- It is in 3NF.
- For any non-trivial functional dependency (X -> Y), X must be a super key.

A relation is in BCNF if every non-trivial functional dependency in the relation is determined by a super key. This means that each

attribute in a relation depends only on the super keys and not on any other non-key attributes.

Achieving BCNF may require decomposing the original table into smaller tables with fewer attributes or creating new tables to represent the functional dependencies that are not satisfied by the original table.

BCNF ensures that a database schema is free of redundancy and that data is stored in a way that avoids anomalies during insertion, deletion, and update operations. It is important to note that not all relations can be decomposed into BCNF without losing information or introducing new dependencies, so achieving BCNF may not always be possible or necessary.

1. In “Productions” table production_firm is functionally dependent on the primary key production_id, which is a superkey. Therefore, it meets BCNF criteria.
2. In “Departments” table department_name is functionally dependent on the primary key (department_id), which is a superkey. production_id is functionally dependent on the primary key (department_id), which is a superkey. Therefore, it meets BCNF criteria.
3. In “Series” Table SeriesName, Reviews, and Budget are functionally dependent on the primary key SeriesID, which is a superkey. Therefore, it meets BCNF criteria.
4. In “Produces” table Both production_id and SeriesID are part of the primary key, so they are superkeys. Therefore, it meets BCNF criteria.
5. In “Genre” table genre_type is functionally dependent on the primary key SeriesID, which is a superkey. Therefore, it meets BCNF criteria.
6. In “Employees” Table first_name, last_name, middle_name, DOB, and department_id are functionally dependent on the primary key emp_id, which is a superkey.
7. In “works_for” table Both Seriesid and emp_id are part of the primary key, so they are superkeys. Therefore, it meets BCNF criteria.

8. In “has_departments” table Both department_id and production_id are part of the primary key, so they are superkeys. Therefore, it meets BCNF criteria.
9. In “cast” table num_of_episodes and name_in_series are functionally dependent on the primary key cast_id, which is a superkey. Therefore, it meets BCNF criteria.
10. In “CrewMembers” table contract_duration and designation are functionally dependent on the primary key crew_member_id, which is a superkey. Therefore, it meets BCNF criteria.
11. In “Script_writer” Table script_name is functionally dependent on the primary key script_id, which is a superkey. Therefore, it meets BCNF criteria.
12. In “Grievances” Table grievance_text is functionally dependent on the primary key emp_id, which is a superkey. Therefore, it meets BCNF criteria.
13. In “ReleaseGroup” table Platform and ReleaseDate are functionally dependent on the primary key R_ID, which is a superkey. SeriesID is functionally dependent on the primary key R_ID, which is a superkey. Therefore, it meets BCNF criteria.

In summary, all the tables in the database are in BCNF as they meet the specified criteria.

NATURAL JOIN TO PROVE LOSSLESS DECOMPOSITION

Natural join is a relational algebra operation that combines tables based on common attributes, and it implicitly selects the attributes that have the same name in both tables. Lossless decomposition is about reconstructing the original relation from its decomposed parts.

Natural Join for Productions and Has_Departments

```
SELECT p.production_id, p.production_firm, d.department_id,
d.department_name

FROM productions AS p

JOIN has_departments AS hd ON p.production_id =
hd.production_id

JOIN departments AS d ON hd.department_id = d.department_id;
```

Natural Join for Series, Produces, and Genre

```
SELECT s.SeriesID, s.SeriesName, s.Reviews, s.Budget,
g.genre_type

FROM Series AS s

JOIN produces AS pr ON s.SeriesID = pr.SeriesID

JOIN productions AS p ON pr.production_id = p.production_id

JOIN Genre AS g ON s.SeriesID = g.SeriesID;
```

Natural Join for Employees and Departments

```
SELECT e.emp_id, e.first_name, e.last_name, e.middle_name,
e.DOB, d.department_name

FROM employees AS e

JOIN departments AS d ON e.department_id = d.department_id;
```

Natural Join for Cast, Employees, Works_For, and Series

```
SELECT c.cast_id, e.first_name, e.last_name, c.num_of_episodes,
c.name_in_series, s.SeriesName

FROM Cast AS c

JOIN employees AS e ON c.emp_id = e.emp_id
```

```
JOIN works_for AS w ON e.emp_id = w.emp_id
JOIN Series AS s ON w.Seriesid = s.SeriesID;
```

Natural Join for Grievances and Employees

```
SELECT g.emp_id, e.first_name, e.last_name, g.grievance_text
FROM Grievances AS g
JOIN employees AS e ON g.emp_id = e.emp_id;
```

Natural Join for CrewMembers and Employees

```
SELECT cm.crew_member_id, e.first_name, e.last_name,
cm.contract_duration, cm.designation
FROM CrewMembers AS cm
JOIN employees AS e ON cm.emp_id = e.emp_id;
```

Natural Join for Script_Writer and Employees

```
SELECT sw.script_id, e.first_name, e.last_name, sw.script_name
FROM Script_writer AS sw
JOIN employees AS e ON sw.emp_id = e.emp_id;
```

Natural Join for ReleaseGroup and Series

```
SELECT rg.R_ID, rg.Platform, rg.ReleaseDate, s.SeriesName
FROM ReleaseGroup AS rg
JOIN Series AS s ON rg.SeriesID = s.SeriesID;
```

To prove the complete lossless decomposition, the natural join of the decomposed tables is said to be equal to the natural join of the original tables. This can be done by decomposing the original tables into smaller tables and then joining them back together.

The decomposed tables are:

1. Productions_Decomp (Productions, Has_Departments)
2. Series_Decomp (Series, Produces, Genre)
3. Employees_Decomp (Employees, Works_For)
4. Cast_Decomp (Cast, Employees)
5. Script_Writer_Decomp (Script_Writer, Employees)
6. ReleaseGroup_Decomp(ReleaseGroup)

The natural join on the decomposed tables are as follows:

Natural Join for Productions_Decomp

SELECT p.production_id, p.production_firm, hd.department_id, hd.production_id

FROM productions AS p

JOIN has_departments AS hd ON p.production_id = hd.production_id;

Natural Join for Series_Decomp

SELECT s.SeriesID, s.SeriesName, s.Reviews, s.Budget, g.genre_type, pr.production_id

FROM Series AS s

JOIN produces AS pr ON s.SeriesID = pr.SeriesID

JOIN Genre AS g ON s.SeriesID = g.SeriesID;

Natural Join for Employees_Decomp

SELECT e.emp_id, e.first_name, e.last_name, e.middle_name, e.DOB, e.department_id

FROM employees AS e

JOIN works_for AS w ON e.emp_id = w.emp_id;

Natural Join for Cast_Decomp

SELECT c.cast_id, e.first_name, e.last_name, c.num_of_episodes, c.name_in_series, w.Seriesid

FROM Cast AS c

JOIN employees AS e ON c.emp_id = e.emp_id

JOIN works_for AS w ON e.emp_id = w.emp_id;

Natural Join for Grievances_Decomp

SELECT g.emp_id, e.first_name, e.last_name, g.grievance_text

FROM Grievances AS g

JOIN employees AS e ON g.emp_id = e.emp_id;

Natural Join for CrewMembers_Decomp

SELECT cm.crew_member_id, e.first_name, e.last_name, cm.contract_duration, cm.designation

FROM CrewMembers AS cm

JOIN employees AS e ON cm.emp_id = e.emp_id;

Natural Join for Script_Writer_Decomp

```
SELECT sw.script_id, e.first_name, e.last_name, sw.script_name
FROM Script_writer AS sw
JOIN employees AS e ON sw.emp_id = e.emp_id;
```

Natural Join for ReleaseGroup_Decomp

```
SELECT rg.R_ID, rg.Platform, rg.ReleaseDate, rg.SeriesID
FROM ReleaseGroup AS rg;
```

Since the result of each decomposition's natural join is equal to the natural join of the original tables, then the decomposition is said to be lossless.

IMPLEMENTATION

Xampp Tables

	Table	Action	Rows	Type	Collation	Size	Overhead
☐	cast	Browse Structure Search Insert Empty Drop	26	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	crewmembers	Browse Structure Search Insert Empty Drop	23	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	departments	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	employees	Browse Structure Search Insert Empty Drop	54	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	genre	Browse Structure Search Insert Empty Drop	10	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	grievances	Browse Structure Search Insert Empty Drop	54	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	has_departments	Browse Structure Search Insert Empty Drop	16	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	produces	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	productions	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	releasegroup	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	48.0 KiB	-
☐	script_writer	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	32.0 KiB	-
☐	series	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	16.0 KiB	-
☐	works_for	Browse Structure Search Insert Empty Drop	66	InnoDB	utf8mb4_general_ci	48.0 KiB	-
	13 tables	Sum	279	InnoDB	utf8mb4_general_ci	416.0 KiB	0 B

Check all With selected:

CAST

	cast_id	emp_id	num_of_episodes	name_in_series
Edit Copy Delete	1	112	22	Chris Brown
Edit Copy Delete	2	364	28	Jessica Lee
Edit Copy Delete	3	976	23	David Garcia
Edit Copy Delete	4	481	26	Ashley Martinez
Edit Copy Delete	5	795	21	Matthew Lopez
Edit Copy Delete	6	653	27	Sarah Scott
Edit Copy Delete	7	164	29	Olivia Adams
Edit Copy Delete	8	742	19	Daniel Nguyen
Edit Copy Delete	9	558	24	Ava Rodriguez
Edit Copy Delete	10	351	31	Kevin Gonzalez
Edit Copy Delete	11	784	17	Lauren Taylor
Edit Copy Delete	12	416	20	Andrew Hernandez
Edit Copy Delete	13	799	28	Grace King
Edit Copy Delete	14	686	19	Liam Martin
Edit Copy Delete	15	238	23	Mia Moore
Edit Copy Delete	16	572	30	William Harris
Edit Copy Delete	17	816	15	Olivia Brown
Edit Copy Delete	18	494	18	James Davis
Edit Copy Delete	19	765	28	Charlotte Martinez
Edit Copy Delete	20	147	17	Henry Robinson
Edit Copy Delete	21	467	20	Ava Garcia
Edit Copy Delete	22	854	29	Noah Taylor
Edit Copy Delete	23	752	26	Amelia Williams
Edit Copy Delete	24	413	25	Brandon White
Edit Copy Delete	25	226	22	Hannah Moore

CREWMEMBERS

←T→	crew_member_id	emp_id	contract_duration	designation
Edit Copy Delete	1	214	12	Head of Production
Edit Copy Delete	2	501	10	Head of Production
Edit Copy Delete	3	789	11	Head of Production
Edit Copy Delete	4	327	9	Head of Finance
Edit Copy Delete	5	611	8	Head of Finance
Edit Copy Delete	6	748	13	Head of Cinematography
Edit Copy Delete	7	403	14	Head of Cinematography
Edit Copy Delete	8	175	10	Head of Cinematography
Edit Copy Delete	9	546	12	Head of Constume and Makeup
Edit Copy Delete	10	217	10	Head of Constume and Makeup
Edit Copy Delete	11	172	10	Head of Constume and Makeup
Edit Copy Delete	12	357	9	Head of Castig
Edit Copy Delete	13	614	8	Head of Castig
Edit Copy Delete	14	843	11	Head of Castig
Edit Copy Delete	15	268	8	Head of Castig
Edit Copy Delete	16	672	10	Head of Castig
Edit Copy Delete	17	621	16	Head of Set Design
Edit Copy Delete	18	971	10	Head of Set Design
Edit Copy Delete	19	465	11	Head of Set Design
Edit Copy Delete	20	873	8	Head of Set Design
Edit Copy Delete	21	286	12	Head of Editing
Edit Copy Delete	22	347	13	Head of Editing
Edit Copy Delete	23	586	9	Head of Editing

DEPARTMENTS

	department_id	department_name
Edit Copy Delete	1	Production
Edit Copy Delete	2	Finance
Edit Copy Delete	11	Scriptwriting
Edit Copy Delete	12	Casting
Edit Copy Delete	13	Cinematography
Edit Copy Delete	15	Costume and Makeup
Edit Copy Delete	16	Set Design
Edit Copy Delete	17	Editing

EMPLOYEES

	emp_id	first_name	last_name	middle_name	DOB	department_id
Edit Copy Delete	112	Matthew	Brown	E.	1993-09-28	12
Edit Copy Delete	147	Natalie	Hall	Y.	1995-04-05	12
Edit Copy Delete	164	Andrew	Harris	L.	1992-10-09	12
Edit Copy Delete	172	William	Clark	*NULL*	1981-06-12	15
Edit Copy Delete	175	Ashley	Martinez	F.	1991-03-12	13
Edit Copy Delete	200	Ella	Thompson	X.	1992-09-26	11
Edit Copy Delete	214	John	Doe	A.	1985-02-10	1
Edit Copy Delete	217	Sarah	Scott	H.	1980-12-28	15
Edit Copy Delete	226	Benjamin	Lopez	DD.	1996-07-23	12
Edit Copy Delete	238	Kevin	Evans	T.	1993-03-16	12
Edit Copy Delete	268	Michael	Williams	C.	1987-11-21	12
Edit Copy Delete	286	Grace	King	*NULL*	1994-09-10	17
Edit Copy Delete	327	Emily	Williams	C.	1995-01-30	2
Edit Copy Delete	347	Brandon	White	M.	1987-01-25	17
Edit Copy Delete	351	Amanda	Robinson	O.	1999-04-17	12
Edit Copy Delete	357	Daniel	Nguyen	I.	1996-10-22	12
Edit Copy Delete	364	Jennifer	Lee	F.	1985-07-19	12
Edit Copy Delete	403	David	Garcia	*NULL*	1986-09-25	13
Edit Copy Delete	413	Samantha	Young	CC.	1999-05-07	12
Edit Copy Delete	416	Elizabeth	Wright	Q.	1984-12-30	12
Edit Copy Delete	436	Henry	Robinson	EE.	1998-11-30	11
Edit Copy Delete	464	Sophie	Anderson	V.	1984-07-01	11
Edit Copy Delete	465	Lauren	Taylor	K.	1997-11-02	16
Edit Copy Delete	467	Robert	Scott	Z.	1988-05-19	12
Edit Copy Delete	481	Jessica	Wilson	H.	1994-03-02	12

GENRE

SeriesID	genre_type
1	Drama
2	Science Fiction
2	Drama
3	Fantasy
4	Drama
4	Comedy
5	Fantasy
5	Crime
3	Mystery
3	Adventure

GRIEVANCES

←T→				emp_id	grievance_text
☐	Edit	Copy	Delete	112	Health and wellness
☐	Edit	Copy	Delete	147	Employee turnover and retention issues
☐	Edit	Copy	Delete	164	Lack of clear company ethics
☐	Edit	Copy	Delete	172	Employee privacy concerns
☐	Edit	Copy	Delete	175	Lack of mental health programs
☐	Edit	Copy	Delete	200	Training requirement
☐	Edit	Copy	Delete	214	Salary concern
☐	Edit	Copy	Delete	217	Lack of company-wide training programs
☐	Edit	Copy	Delete	226	Ineffective time management
☐	Edit	Copy	Delete	238	Inadequate feedback mechanisms
☐	Edit	Copy	Delete	268	Workplace culture
☐	Edit	Copy	Delete	286	Inadequate job security measures
☐	Edit	Copy	Delete	327	Overtime problem
☐	Edit	Copy	Delete	347	Unresolved grievances
☐	Edit	Copy	Delete	351	Lack of work-related benefits
☐	Edit	Copy	Delete	357	Ethical dilemma
☐	Edit	Copy	Delete	364	Management issue
☐	Edit	Copy	Delete	403	Ineffective conflict mediation
☐	Edit	Copy	Delete	413	Lack of opportunities for career change
☐	Edit	Copy	Delete	416	Lack of innovation opportunities
☐	Edit	Copy	Delete	436	Safety concern
☐	Edit	Copy	Delete	464	Equipment malfunction
☐	Edit	Copy	Delete	465	Ineffective employee performance reviews
☐	Edit	Copy	Delete	467	Lack of clear project goals
☐	Edit	Copy	Delete	481	Career growth

HAS_DEPARTMENTS

department_id	production_id
1	1
2	1
11	1
12	1
13	1
15	1
16	1
17	1
1	2
2	2
11	2
12	2
13	2
15	2
16	2
17	2

PRODUCES

production_id	SeriesID
1	1
2	2
2	3
1	4
2	5

PRODUCTIONS

	production_id	production_firm
Edit Copy Delete	1	FilmCo Enterprises
Edit Copy Delete	2	GreatFilms Inc.

RELEASEGROUP

	R_ID	Platform	ReleaseDate	SeriesID
Edit Copy Delete	1	Netflix	2023-01-15	1
Edit Copy Delete	2	Netflix	2023-03-20	2
Edit Copy Delete	3	HBO	2023-05-10	3
Edit Copy Delete	4	AMC	2023-06-05	4
Edit Copy Delete	5	Netflix	2023-08-12	5

SCRIPT_WRITER

	script_id	script_name	emp_id
Edit Copy Delete	2	Stranger Things Script	464
Edit Copy Delete	3	Game of Thrones Script	577
Edit Copy Delete	4	Breaking Bad Script	200
Edit Copy Delete	6	Money Heist Script	493
Edit Copy Delete	11	Peaky Blinders Script	436

SERIES

	SeriesID	SeriesName	Reviews	Budget
Edit Copy Delete	1	Stranger Things	850	12000000.00
Edit Copy Delete	2	Game of Thrones	980	18000000.00
Edit Copy Delete	3	Breaking Bad	920	10000000.00
Edit Copy Delete	4	Money Heist	910	16000000.00
Edit Copy Delete	5	Peaky Blinders	920	10500000.00

WORKS_FOR

Seriesid	emp_id
1	357
2	614
3	843
4	268
5	672
1	214
2	214
3	789
4	501
5	789
1	327
2	611
3	611
4	327
5	611
1	748
2	403
3	175
4	403
5	175
1	172
2	217
3	546
4	546
5	217

Application With Code

```
const option = document.getElementById('option');
const main  = document.getElementById('main');

option.addEventListener('change', async () => {
    if(option.value == 0) {
        return;
    }
    if(option.value == 1) {
        main.innerHTML = '<input type="text" id="query"> <input type="button" id="btn" value="submit">';
        const query = document.getElementById('query');
        const btn = document.getElementById('btn');
        btn.addEventListener('click', async () => {
            const {data} = await axios.post("http://localhost:5000/run",{query:query.value});
            populateTable(data)
        })
        return
    }
    const {data} = await axios.get(`http://localhost:5000/${option.value}`);
    populateTable(data);
    return;
})

```

```
function populateTable(data) {
    main.innerHTML = `<table id="data-table"> </table>`
    const table = document.getElementById("data-table");
    const headers = table.createTHead();
    const headerRow = headers.insertRow(0);
    for (const key in data[0]) {
        const headerCell = document.createElement("th");
        headerCell.innerHTML = key;
        headerRow.appendChild(headerCell);
    }
    data.forEach((item) => {
        const row = table.insertRow();
        for (const key in item) {
            if (item.hasOwnProperty(key)) {
                const cell = row.insertCell();
                cell.innerHTML = item[key];
            }
        }
    });
}
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>DBMS Project</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h2>Select an Option:</h2>
        <select id="option">
            <option selected value="0">Select</option>
            <option value="production">production</option>
            <option value="series">series</option>
            <option value="department">department</option>
            <option value="cast">cast</option>
            <option value="grievances">grievances</option>
            <option value="crewmembers">crewmembers</option>
            <option value="writer">writer</option>
            <option value="release">release</option>
            <option value="genre">genre</option>
            <option value="employees">employees</option>
            <option value="1">Run</option>
        </select>
    </div>
    <main id="main">
    </main>

    <!-- javascript -->
    <script src="https://cdn.jsdelivr.net/npm/axios@1.1.2/dist/axios.min.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

```css
/* .container {
    margin: 20px;
    padding: 20px;
    border-radius: 8px;
    display: flex;
    justify-content: center;
    flex-direction: column;
}

select {
    padding: 10px;
    font-size: 16px;
}

#main {
    padding: 20px;
}

table {
    width: 100%;
    border-collapse: collapse;
}

th, td {
    padding: 12px 15px;
    text-align: left;
} */
.body{
    background-color: coral;
}
```

```css
.container {
    margin: 20px;
    padding: 40px;
    background-color: #b3e5fc;
    border-radius: 30px;
    box-shadow: 0 5px 20px rgba(13, 13, 13, 0.1);
    display: flex;
    justify-content: center;
    flex-direction: column;
}

select {
    padding: 25px;
    font-size: 16px;
    background-color: #e1f5fe;
    border-radius: 30px;
}

#main {
    padding: 30px;
}

table {
    width: 100%;
    border-collapse: collapse;
}

th, td {
    padding: 18px 22px;
    text-align: left;
}

```

```css
th {
    background-color: #014768;
    color: #ffffff;
}

tr:nth-child(even) {
    background-color: #e1f5fe;
}

tr:nth-child(odd) {
    background-color: #b3e5fc;
}
```

```json
{
  "name": "dbms",
  "version": "1.0.0",
  "description": "",
  "type": "module",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "start": "nodemon app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "mysql2": "^3.6.2"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

```
import mysql from 'mysql2/promise'
import dotenv from 'dotenv'
dotenv.config()
import express from 'express';
import cors from 'cors';
const app = express();

app.use(cors())
app.use(express.static('./public'));

// data parse middleware
app.use(express.json());

const pool = mysql.createPool({
  host: process.env.MYSQL_HOST,
  user: process.env.MYSQL_USER,
  password: process.env.MYSQL_PASSWORD,
  database: process.env.MYSQL_DATABASE
})

const queryResult = async (query) => {
    const [result] = await pool.query(query);
    return result
}

app.get('/production', async (req,res) => {
    const result = await queryResult(`SELECT p.production_id, p.production_firm, d.department_id, d.department_name
    FROM productions as p
    JOIN has_departments hd ON p.production_id = hd.production_id
    JOIN departments d ON hd.department_id = d.department_id`);
    res.json(result)
})
```

```
app.get('/series', async (req,res) => {
    res.json(await queryResult(`SELECT s.SeriesID, s.SeriesName, s.Reviews, s.Budget, g.genre_type
    FROM Series s
    LEFT JOIN Genre g ON s.SeriesID = g.SeriesID;`))
})

app.get('/department', async (req,res) => {
    res.json(await queryResult(`SELECT e.emp_id, e.first_name, e.last_name, e.middle_name, e.DOB, d.department_name
    FROM employees e
    JOIN departments d ON e.department_id = d.department_id;`))
})

app.get('/cast', async (req,res) => {
    res.json(await queryResult(`SELECT c.cast_id, e.first_name, e.last_name, c.num_of_episodes, c.name_in_series, s.SeriesName
    FROM Cast c
    JOIN employees e ON c.emp_id = e.emp_id
    JOIN works_for w ON e.emp_id = w.emp_id
    JOIN Series s ON w.Seriesid = s.SeriesID;`))
})

app.get('/grievances', async (req,res) => {
    res.json(await queryResult(`SELECT g.emp_id, e.first_name, e.last_name, g.grievance_text
    FROM Grievances g
    JOIN employees e ON g.emp_id = e.emp_id;`))
})

app.get('/crewmembers', async (req,res) => {
    res.json(await queryResult(`SELECT cm.crew_member_id, e.first_name, e.last_name, cm.contract_duration, cm.designation
    FROM CrewMembers cm
    JOIN employees e ON cm.emp_id = e.emp_id;`))
})
```

```
app.get('/writer', async (req,res) => {
    res.json(await queryResult(`SELECT sw.script_id, e.first_name, e.last_name, sw.script_name
    FROM Script_writer sw
    JOIN employees e ON sw.emp_id = e.emp_id;`))
})

app.get('/release', async (req,res) => {
    res.json(await queryResult(`SELECT rg.R_ID, rg.Platform, rg.ReleaseDate, s.SeriesName
    FROM ReleaseGroup rg
    JOIN Series s ON rg.SeriesID = s.SeriesID;`))
})

app.get('/genre', async (req,res) => {
    res.json(await queryResult(`SELECT SeriesID, GROUP_CONCAT(genre_type) AS Genres
    FROM Genre
    GROUP BY SeriesID;`))
})

app.get('/employees', async (req,res) => {
    res.json(await queryResult(`SELECT emp_id, first_name, last_name, DOB, YEAR(CURDATE()) - YEAR(DOB) AS age
    FROM employees;`))
})

app.post('/run',async (req,res) => {
    const {query} = req.body;
    res.json(await queryResult(query));
})

app.listen(process.env.PORT, () => {
    console.log("Listening to the port 5000");
})
```

```sql
CREATE DATABASE IF NOT EXISTS prod;
USE prod;

CREATE TABLE productions (
    production_id INT NOT NULL AUTO_INCREMENT,
    production_firm VARCHAR(255) NOT NULL,
    PRIMARY KEY (production_id)
);

CREATE TABLE departments (
    department_id INT NOT NULL AUTO_INCREMENT,
    department_name VARCHAR(255) NOT NULL,
    PRIMARY KEY (department_id)
);

CREATE TABLE Series (
  SeriesID INT NOT NULL AUTO_INCREMENT,
  SeriesName VARCHAR(255) NOT NULL,
  Reviews INT,
  Budget DECIMAL(10,2) NOT NULL,
  PRIMARY KEY (SeriesID)
);

CREATE TABLE produces (
 production_id INT NOT NULL,
 SeriesID INT NOT NULL ,
 FOREIGN KEY (production_id) REFERENCES productions(production_id),
 FOREIGN KEY (SeriesID) REFERENCES Series(SeriesID)
);

```

```sql
CREATE TABLE Genre (
  SeriesID INT,
  genre_type VARCHAR(255),
  FOREIGN KEY (SeriesID) REFERENCES Series(SeriesID)
);

CREATE TABLE employees (
    emp_id INT NOT NULL AUTO_INCREMENT,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    middle_name VARCHAR(255),
    DOB DATE,
    department_id INT NOT NULL,
    PRIMARY KEY (emp_id),
    FOREIGN KEY (department_id) REFERENCES departments (department_id)
);

CREATE TABLE works_for (
  Seriesid INT NOT NULL,
    emp_id INT NOT NULL,
    FOREIGN KEY (SeriesID) REFERENCES Series(SeriesID),
  FOREIGN KEY (emp_id) REFERENCES employees(emp_id)
);

CREATE TABLE has_departments (
    department_id INT NOT NULL,
    production_id INT NOT NULL,
    FOREIGN KEY (department_id) REFERENCES departments (department_id),
    FOREIGN KEY (production_id) REFERENCES productions (production_id)
);

```

```sql
CREATE TABLE Cast (
  cast_id INT NOT NULL AUTO_INCREMENT,
  emp_id INT NOT NULL,
  num_of_episodes INT NOT NULL,
  name_in_series VARCHAR(255) NOT NULL,
  PRIMARY KEY (cast_id),
  FOREIGN KEY (emp_id) REFERENCES employees (emp_id)
);

CREATE TABLE CrewMembers (
  crew_member_id INT NOT NULL AUTO_INCREMENT,
  emp_id INT NOT NULL,
  contract_duration INT NOT NULL,
  designation VARCHAR(255) NOT NULL,
  PRIMARY KEY (crew_member_id),
  FOREIGN KEY (emp_id) REFERENCES employees (emp_id)
);

CREATE TABLE Script_writer (
  script_id INT NOT NULL AUTO_INCREMENT,
  script_name VARCHAR(255) NOT NULL,
  emp_id INT NOT NULL,
  PRIMARY KEY (script_id),
   FOREIGN KEY (emp_id) REFERENCES employees (emp_id)
);

CREATE TABLE Grievances (
    emp_id INT NOT NULL,
    grievance_text TEXT,
    FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
    PRIMARY KEY (emp_id)
);
```


```sql
CREATE TABLE Cast (
  cast_id INT NOT NULL AUTO_INCREMENT,
  emp_id INT NOT NULL,
  num_of_episodes INT NOT NULL,
  name_in_series VARCHAR(255) NOT NULL,
  PRIMARY KEY (cast_id),
  FOREIGN KEY (emp_id) REFERENCES employees (emp_id)
);

CREATE TABLE CrewMembers (
  crew_member_id INT NOT NULL AUTO_INCREMENT,
  emp_id INT NOT NULL,
  contract_duration INT NOT NULL,
  designation VARCHAR(255) NOT NULL,
  PRIMARY KEY (crew_member_id),
  FOREIGN KEY (emp_id) REFERENCES employees (emp_id)
);

CREATE TABLE Script_writer (
  script_id INT NOT NULL AUTO_INCREMENT,
  script_name VARCHAR(255) NOT NULL,
  emp_id INT NOT NULL,
  PRIMARY KEY (script_id),
   FOREIGN KEY (emp_id) REFERENCES employees (emp_id)
);

CREATE TABLE Grievances (
    emp_id INT NOT NULL,
    grievance_text TEXT,
    FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
    PRIMARY KEY (emp_id)
);
```

```sql

CREATE TABLE ReleaseGroup (
    R_ID INT NOT NULL AUTO_INCREMENT,
    Platform VARCHAR(255),
    ReleaseDate DATE,
    SeriesID INT,
    PRIMARY KEY (R_ID),
    FOREIGN KEY (SeriesID) REFERENCES Series(SeriesID),
    UNIQUE KEY (R_ID, SeriesID)
);

INSERT INTO productions (production_firm,production_id) VALUES
('FilmCo Enterprises',1),
('GreatFilms Inc.',2);

INSERT INTO departments (department_name, department_id) VALUES
('Production', 1),
('Finance', 2),
('Scriptwriting', 11),
('Casting', 12),
('Cinematography', 13),
('Costume and Makeup', 15),
('Set Design', 16),
('Editing', 17);


```

```sql
INSERT INTO has_departments(department_id,production_id) VALUES
(1,1),
(2,1),
(11,1),
(12,1),
(13,1),
(15,1),
(16,1),
(17,1),
(1,2),
(2,2),
(11,2),
(12,2),
(13,2),
(15,2),
(16,2),
(17,2);


INSERT INTO Series (SeriesName, Reviews, Budget,SeriesID) VALUES
('Stranger Things', 850, 12000000.00,1),
('Game of Thrones', 980, 18000000.00,2),
('Breaking Bad', 920, 10000000.00,3),
('Money Heist', 910, 16000000.00,4),
('Peaky Blinders', 920, 10500000.00,5);


```

```sql
INSERT INTO produces (production_id, SeriesID) VALUES
(1, 1),
(2, 2),
(2, 3),
(1, 4),
(2, 5);


INSERT INTO Genre (SeriesID, genre_type) VALUES
(1, 'Drama'),
(2, 'Science Fiction'),
(2,'Drama'),
(3, 'Fantasy'),
(4, 'Drama'),
(4, 'Comedy'),
(5, 'Fantasy'),
(5, 'Crime');

```

```sql
INSERT INTO employees (first_name, last_name, middle_name, DOB, department_id,emp_id) VALUES
('John', 'Doe', 'A.', '1985-02-10', 1, 214),
('Jane', 'Smith', 'B.', '1990-07-15', 1, 501),
('Michael', 'Johnson', NULL, '1988-04-20', 1, 789),
('Emily', 'Williams', 'C.', '1995-01-30', 2, 327),
('Chris', 'Brown', 'D.', '1982-11-05', 2, 611),
('Sophie', 'Anderson', 'V.', '1984-07-01', 11, 464),
('Daniel', 'Hall', 'W.', '1997-05-14', 11, 577),
('Ella', 'Thompson', 'X.', '1992-09-26', 11, 200),
('Mia', 'Moore', 'Z.', '1988-12-03', 11, 493),
('Henry', 'Robinson', 'EE.', '1998-11-30', 11, 436),
('Daniel', 'Nguyen', 'I.', '1996-10-22', 12, 357),
('John', 'Smith', 'A.', '1995-08-15', 12, 614),
('Sarah', 'Johnson', 'B.', '1990-04-03', 12, 843),
('Michael', 'Williams', 'C.', '1987-11-21', 12, 268),
('Emily', 'Davis', 'D.', '1998-02-12', 12, 672),
('Matthew', 'Brown', 'E.', '1993-09-28', 12, 112),
('Jennifer', 'Lee', 'F.', '1985-07-19', 12, 364),
('Christopher', 'Miller', 'G.', '1991-06-14', 12, 976),
('Jessica', 'Wilson', 'H.', '1994-03-02', 12, 481),
('David', 'Anderson', 'J.', '1988-12-07', 12, 795),
('Melissa', 'Thompson', 'K.', '1997-01-25', 12, 653),
('Andrew', 'Harris', 'L.', '1992-10-09', 12, 164),
('Linda', 'Martinez', 'M.', '1986-05-31', 12, 742),
('William', 'Garcia', 'N.', '1989-08-26', 12, 558),
('Amanda', 'Robinson', 'O.', '1999-04-17', 12, 351),
('James', 'Clark', 'P.', '1996-11-04', 12, 784),
('Elizabeth', 'Wright', 'Q.', '1984-12-30', 12, 416),
('Daniel', 'Turner', 'R.', '1990-02-22', 12, 799),
('Maria', 'Adams', 'S.', '1997-07-08', 12, 686),
('Kevin', 'Evans', 'T.', '1993-03-16', 12, 238),
('Olivia', 'Perez', 'U.', '1998-09-12', 12, 572),
```

```
('Ryan', 'Jackson', 'V.', '1994-06-25', 12, 816),
('Stephanie', 'Hernandez', 'W.', '1991-01-03', 12, 494),
('Joseph', 'Moore', 'X.', '1987-10-17', 12, 765),
('Natalie', 'Hall', 'Y.', '1995-04-05', 12, 147),
('Robert', 'Scott', 'Z.', '1988-05-19', 12, 467),
('Megan', 'Baker', 'AA.', '1992-08-29', 12, 854),
('Thomas', 'King', 'BB.', '1986-12-13', 12, 752),
('Samantha', 'Young', 'CC.', '1999-05-07', 12, 413),
('Benjamin', 'Lopez', 'DD.', '1996-07-23', 12, 226),
('Emma', 'Adams', 'EE.', '1993-03-11', 12, 619),
('Jessica', 'Lee', 'E.', '1993-08-18', 13, 748),
('David', 'Garcia', NULL, '1986-09-25', 13, 403),
('Ashley', 'Martinez', 'F.', '1991-03-12', 13, 175),
('Matthew', 'Lopez', 'G.', '1998-06-07', 15, 546),
('William', 'Clark', NULL, '1981-06-12', 15, 172),
('Sarah', 'Scott', 'H.', '1980-12-28', 15, 217),
('Ava', 'Rodriguez', 'J.', '1992-02-17', 16, 621),
('Kevin', 'Gonzalez', NULL, '1983-07-09', 16, 971),
('Lauren', 'Taylor', 'K.', '1997-11-02', 16, 465),
('Andrew', 'Hernandez', 'L.', '1989-04-15', 16, 873),
('Grace', 'King', NULL, '1994-09-10', 17, 286),
('Brandon', 'White', 'M.', '1987-01-25', 17, 347),
('Hannah', 'Moore', 'N.', '1999-08-08', 17, 586);

INSERT INTO works_for (SeriesID, emp_id) VALUES
(1, 357),
(2, 614),
(3, 843),
(4, 268),
(5, 672),
(1, 214),
(2, 214),
(3, 789),
(4, 501),
(5, 789),
(1, 327),
(2, 611),
(3, 611),
(4, 327),
(5, 611),
(1, 748),
(2, 403),
(3, 175),
(4, 403),
(5, 175),
(1, 172),
(2, 217),
(3, 546),
(4, 546),
(5, 217),
(1, 621),
(2, 971),
(3, 465),
(4, 873),
```

```
  (5, 971),
  (1, 286),
  (2, 347),
  (3, 586),
  (4, 286),
  (5, 586),
  (1, 464),
  (2, 577),
  (3, 238),
  (4, 493),
  (5, 436),
  (1,112),
  (1,364),
  (1,976),
  (1,481),
  (2,795),
  (2,653),
  (2,164),
  (2,742),
  (2,558),
  (2,351),
  (2,784),
  (3,416),
  (3,799),
  (3,686),
  (4,238),
  (4,572),
  (4,816),
```

```
  (4,494),
  (4,765),
  (5,147),
  (5,467),
  (5,854),
  (5,752),
  (5,413),
  (5,226),
  (2,619);


```

```sql
INSERT INTO Cast (emp_id, num_of_episodes, name_in_series) VALUES
(112, 22, 'Chris Brown'),
(364, 28, 'Jessica Lee'),
(976, 23, 'David Garcia'),
(481, 26, 'Ashley Martinez'),
(795, 21, 'Matthew Lopez'),
(653, 27, 'Sarah Scott'),
(164, 29, 'Olivia Adams'),
(742, 19, 'Daniel Nguyen'),
(558, 24, 'Ava Rodriguez'),
(351, 31, 'Kevin Gonzalez'),
(784, 17, 'Lauren Taylor'),
(416, 20, 'Andrew Hernandez'),
(799, 28, 'Grace King'),
(686, 19, 'Liam Martin'),
(238, 23, 'Mia Moore'),
(572, 30, 'William Harris'),
(816, 15, 'Olivia Brown'),
(494, 18, 'James Davis'),
(765, 28, 'Charlotte Martinez'),
(147, 17, 'Henry Robinson'),
(467, 20, 'Ava Garcia'),
(854, 29, 'Noah Taylor'),
(752, 26, 'Amelia Williams'),
(413, 25, 'Brandon White'),
(226, 22, 'Hannah Moore'),
(619, 30, 'William Clark');
```

```sql
INSERT INTO CrewMembers (emp_id, contract_duration, designation) VALUES
(214, 12, 'Head of Production'),
(501, 10, 'Head of Production'),
(789, 11, 'Head of Production'),
(327, 9, 'Head of Finance'),
(611, 8, 'Head of Finance'),
(748, 13, 'Head of Cinematography'),
(403, 14, 'Head of Cinematography'),
(175, 10, 'Head of Cinematography'),
(546, 12, 'Head of Constume and Makeup'),
(217,10, 'Head of Constume and Makeup'),
(172, 10, 'Head of Constume and Makeup'),
(357,9,'Head of Castig'),
(614,8,'Head of Castig'),
(843,11,'Head of Castig'),
(268,8,'Head of Castig'),
(672,10,'Head of Castig'),
(621, 16, 'Head of Set Design'),
(971, 10, 'Head of Set Design'),
(465, 11, 'Head of Set Design'),
(873, 8, 'Head of Set Design'),
(286, 12, 'Head of Editing'),
(347, 13, 'Head of Editing'),
(586, 9, 'Head of Editing');


INSERT INTO Script_writer (script_name, script_id,emp_id) VALUES
('Stranger Things Script', 2,464),
('Game of Thrones Script', 3,577),
('Breaking Bad Script', 4,200),
('Money Heist Script', 6,493),
('Peaky Blinders Script', 11,436);
```

```sql


INSERT INTO Grievances (emp_id, grievance_text) VALUES
(214, 'Salary concern'),
(501, 'Workload issue'),
(789, 'Conflict with colleague'),
(327, 'Overtime problem'),
(611, 'Seeking leave'),
(464, 'Equipment malfunction'),
(577, 'Communication problem'),
(200, 'Training requirement'),
(493, 'Resource shortage'),
(436, 'Safety concern'),
(357, 'Ethical dilemma'),
(614, 'Job satisfaction'),
(843, 'Performance appraisal'),
(268, 'Workplace culture'),
(672, 'Diversity and inclusion'),
(112, 'Health and wellness'),
(364, 'Management issue'),
(976, 'Work-life balance'),
(481, 'Career growth'),
(795, 'Job security'),
(653, 'Inadequate conflict resolution processes'),
(164, 'Lack of clear company ethics'),
(742, 'Inadequate professional development opportunities'),
(558, 'Unfair termination or layoff'),
(351, 'Lack of work-related benefits'),
(784, 'Inadequate employee recognition programs'),
(416, 'Lack of innovation opportunities'),
```

```sql
(799, 'Ineffective employee onboarding'),
(686, 'Lack of support for mental health'),
(238, 'Inadequate feedback mechanisms'),
(572, 'Lack of alignment with company values'),
(816, 'Ineffective team communication'),
(494, 'Unresolved workplace conflicts'),
(765, 'Inadequate performance evaluation'),
(147, 'Employee turnover and retention issues'),
(467, 'Lack of clear project goals'),
(854, 'Inadequate recognition of extra hours worked'),
(752, 'Inadequate disability accommodations'),
(413, 'Lack of opportunities for career change'),
(226, 'Ineffective time management'),
(619, 'Inadequate company resources'),
(748, 'Lack of support for remote workers'),
(403, 'Ineffective conflict mediation'),
(175, 'Lack of mental health programs'),
(546, 'Inadequate diversity and inclusion efforts'),
(172, 'Employee privacy concerns'),
(217, 'Lack of company-wide training programs'),
(621, 'Inadequate work-related communication'),
(971, 'Lack of support for employee well-being'),
(465, 'Ineffective employee performance reviews'),
(873, 'Lack of clear company mission'),
(286, 'Inadequate job security measures'),
(347, 'Unresolved grievances'),
(586, 'Lack of support for remote meetings');


```

```sql
INSERT INTO ReleaseGroup (Platform, ReleaseDate, SeriesID) VALUES
('Netflix', '2023-01-15', 1),
('Netflix', '2023-03-20', 2),
('HBO', '2023-05-10', 3),
('AMC', '2023-06-05', 4),
('Netflix', '2023-08-12', 5);


INSERT INTO Series (SeriesID, SeriesName, Reviews, Budget) VALUES
(21, 'Mystery Quest', 0, 0.00);

INSERT INTO Genre (SeriesID, genre_type) VALUES
(3, 'Mystery'),
(3, 'Adventure');

SELECT p.production_id, p.production_firm, d.department_id, d.department_name
FROM productions as p
JOIN has_departments hd ON p.production_id = hd.production_id
JOIN departments d ON hd.department_id = d.department_id;

SELECT s.SeriesID, s.SeriesName, s.Reviews, s.Budget, g.genre_type
FROM Series s
LEFT JOIN Genre g ON s.SeriesID = g.SeriesID;

SELECT e.emp_id, e.first_name, e.last_name, e.middle_name, e.DOB, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;

```

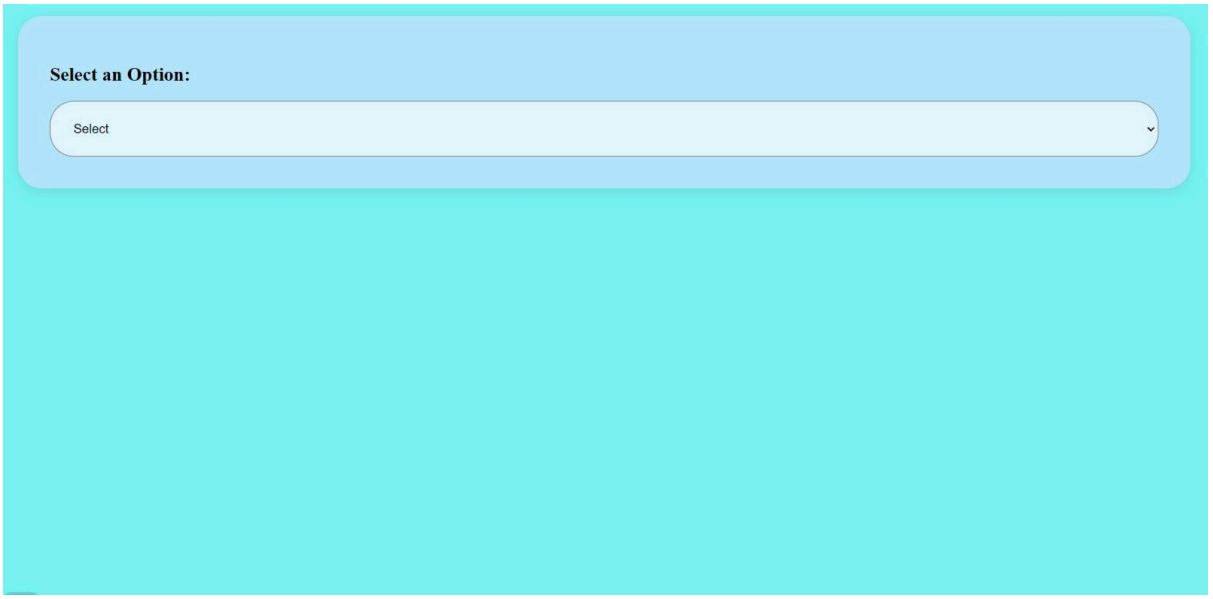
```sql
SELECT c.cast_id, e.first_name, e.last_name, c.num_of_episodes, c.name_in_series, s.SeriesName
FROM Cast c
JOIN employees e ON c.emp_id = e.emp_id
JOIN works_for w ON e.emp_id = w.emp_id
JOIN Series s ON w.Seriesid = s.SeriesID;

SELECT g.emp_id, e.first_name, e.last_name, g.grievance_text
FROM Grievances g
JOIN employees e ON g.emp_id = e.emp_id;

SELECT cm.crew_member_id, e.first_name, e.last_name, cm.contract_duration, cm.designation
FROM CrewMembers cm
JOIN employees e ON cm.emp_id = e.emp_id;

SELECT sw.script_id, e.first_name, e.last_name, sw.script_name
FROM Script_writer sw
JOIN employees e ON sw.emp_id = e.emp_id;

SELECT rg.R_ID, rg.Platform, rg.ReleaseDate, s.SeriesName
FROM ReleaseGroup rg
JOIN Series s ON rg.SeriesID = s.SeriesID;

SELECT SeriesID, GROUP_CONCAT(genre_type) AS Genres
FROM Genre
GROUP BY SeriesID;

SELECT emp_id, first_name, last_name, DOB, YEAR(CURDATE()) - YEAR(DOB) AS age
FROM employees;
```

```
MYSQL_HOST='127.0.0.1'
MYSQL_USER='root'
MYSQL_PASSWORD='yourpassword'
MYSQL_DATABASE='prod'
PORT='5000'
```

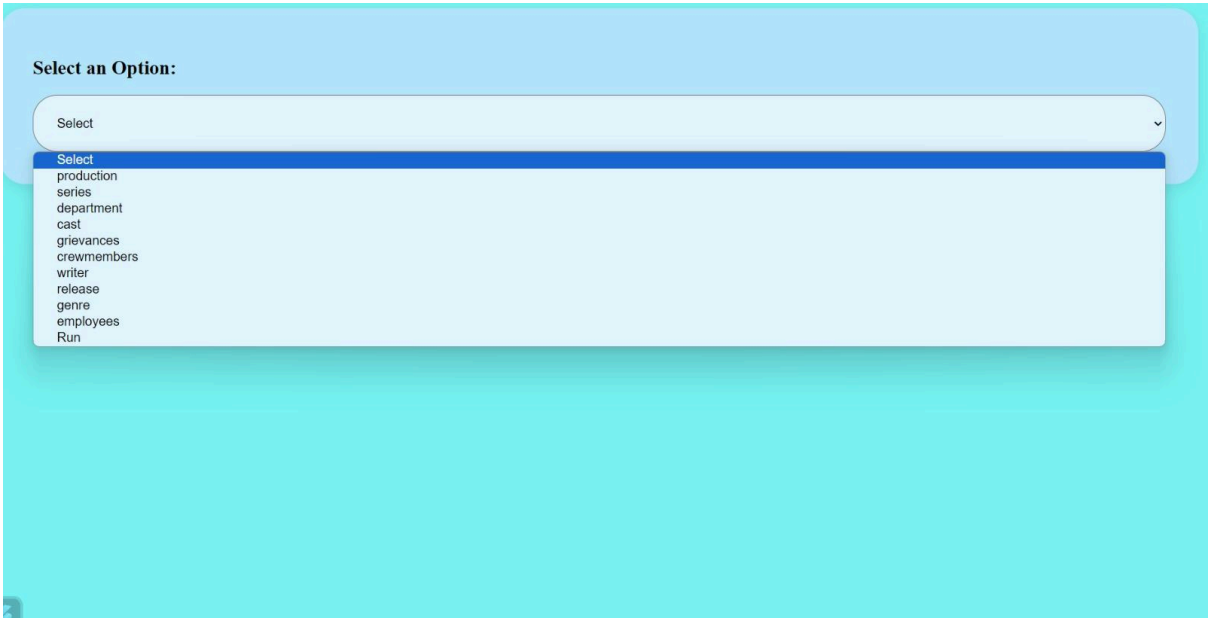
```sql
DELIMITER //

CREATE TRIGGER age_trigger BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
    DECLARE emp_age INT;
    SET emp_age = YEAR(CURDATE()) - YEAR(NEW.DOB);

    IF emp_age < 18 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Employees must be at least 18 years old.';
    END IF;
END;
//

CREATE TRIGGER budget_trigger BEFORE INSERT ON Series
FOR EACH ROW
BEGIN
    IF NEW.Budget < 1000000.00 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Series budget must be at least 1,000,000.00.';
    END IF;
END;
//

DELIMITER;
```

View Of the Application

Page view:



Selecting options:



Data present in production:

production_id	production_firm	department_id	department_name
1	FilmCo Enterprises	17	Editing
1	FilmCo Enterprises	16	Set Design
1	FilmCo Enterprises	15	Costume and Makeup
1	FilmCo Enterprises	13	Cinematography
1	FilmCo Enterprises	12	Casting
1	FilmCo Enterprises	11	Scriptwriting
1	FilmCo Enterprises	2	Finance
1	FilmCo Enterprises	1	Production
2	GreatFilms Inc.	17	Editing
2	GreatFilms Inc.	16	Set Design
2	GreatFilms Inc.	15	Costume and Makeup
2	GreatFilms Inc.	13	Cinematography
2	GreatFilms Inc.	12	Casting

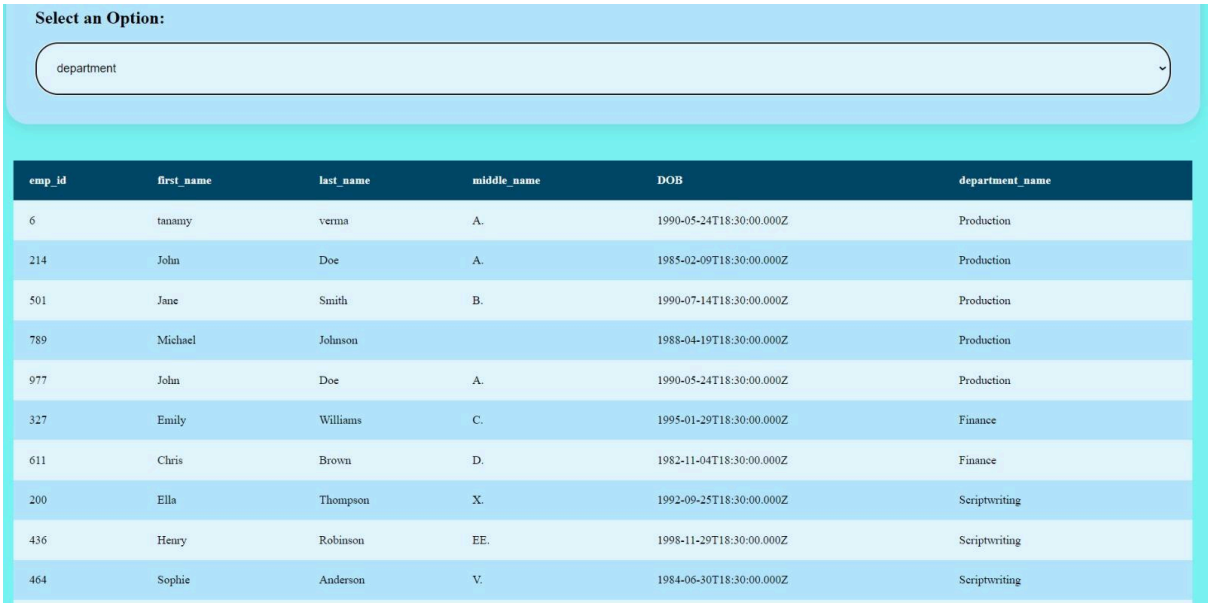
Data present in series:

Select an Option:

series

SeriesID	SeriesName	Reviews	Budget	genre_type
1	Stranger Things	850	12000000.00	Drama
2	Game of Thrones	980	18000000.00	Science Fiction
2	Game of Thrones	980	18000000.00	Drama
3	Breaking Bad	920	10000000.00	Fantasy
3	Breaking Bad	920	10000000.00	Mystery
3	Breaking Bad	920	10000000.00	Adventure
4	Money Heist	910	16000000.00	Drama
4	Money Heist	910	16000000.00	Comedy
5	Peaky Blinders	920	10500000.00	Fantasy
5	Peaky Blinders	920	10500000.00	Crime

Data present in department:

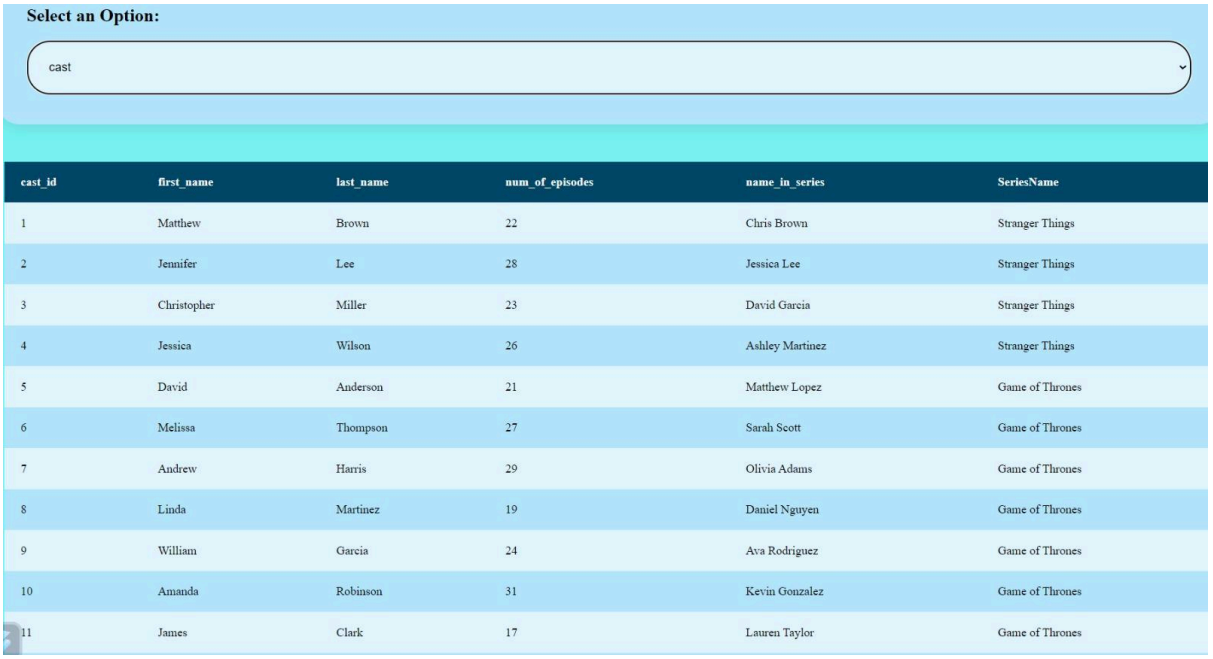


Select an Option:

department

emp_id	first_name	last_name	middle_name	DOB	department_name
6	tanamy	verma	A.	1990-05-24T18:30:00.000Z	Production
214	John	Doe	A.	1985-02-09T18:30:00.000Z	Production
501	Jane	Smith	B.	1990-07-14T18:30:00.000Z	Production
789	Michael	Johnson		1988-04-19T18:30:00.000Z	Production
977	John	Doe	A.	1990-05-24T18:30:00.000Z	Production
327	Emily	Williams	C.	1995-01-29T18:30:00.000Z	Finance
611	Chris	Brown	D.	1982-11-04T18:30:00.000Z	Finance
200	Ella	Thompson	X.	1992-09-25T18:30:00.000Z	Scriptwriting
436	Henry	Robinson	EE.	1998-11-29T18:30:00.000Z	Scriptwriting
464	Sophie	Anderson	V.	1984-06-30T18:30:00.000Z	Scriptwriting

Data present in cast:



Select an Option:

cast

cast_id	first_name	last_name	num_of_episodes	name_in_series	SeriesName
1	Matthew	Brown	22	Chris Brown	Stranger Things
2	Jennifer	Lee	28	Jessica Lee	Stranger Things
3	Christopher	Miller	23	David Garcia	Stranger Things
4	Jessica	Wilson	26	Ashley Martinez	Stranger Things
5	David	Anderson	21	Matthew Lopez	Game of Thrones
6	Melissa	Thompson	27	Sarah Scott	Game of Thrones
7	Andrew	Harris	29	Olivia Adams	Game of Thrones
8	Linda	Martinez	19	Daniel Nguyen	Game of Thrones
9	William	Garcia	24	Ava Rodriguez	Game of Thrones
10	Amanda	Robinson	31	Kevin Gonzalez	Game of Thrones
11	James	Clark	17	Lauren Taylor	Game of Thrones

Data present in grievances:

Select an Option:

grievances

emp_id	first_name	last_name	grievance_text
112	Matthew	Brown	Health and wellness
147	Piyush	Joshi	Employee turnover and retention issues
164	Andrew	Harris	Lack of clear company ethics
172	William	Clark	Employee privacy concerns
175	Ashley	Martinez	Lack of mental health programs
200	Ella	Thompson	Training requirement
214	John	Doe	Salary concern
217	Sarah	Scott	Lack of company-wide training programs
226	Benjamin	Lopez	Ineffective time management
238	Kevin	Evans	Inadequate feedback mechanisms
268	Michael	Williams	Workplace culture

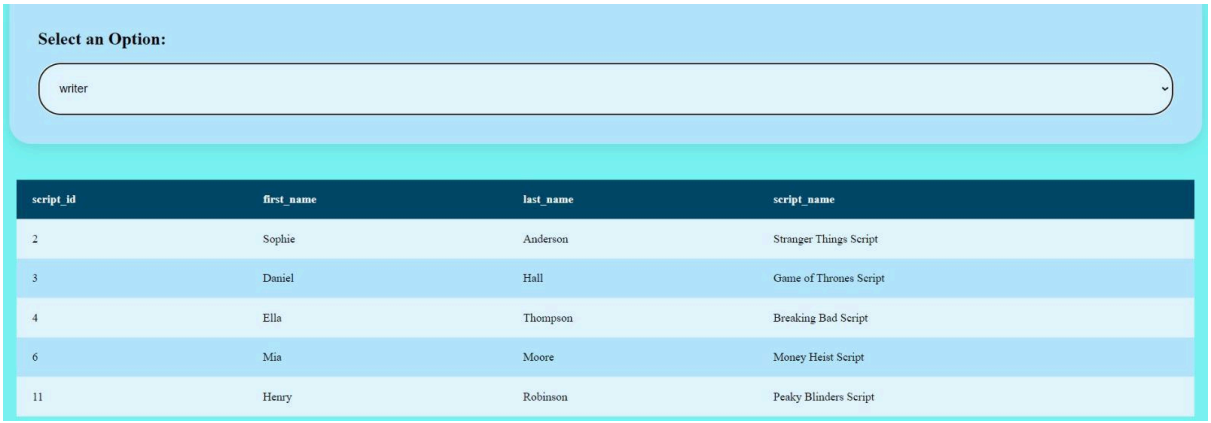
Data present in crewmembers:

Select an Option:

crewmembers

crew_member_id	first_name	last_name	contract_duration	designation
1	John	Doe	12	Head of Production
2	Jane	Smith	10	Head of Production
3	Michael	Johnson	11	Head of Production
4	Emily	Williams	9	Head of Finance
5	Chris	Brown	8	Head of Finance
6	Jessica	Lee	13	Head of Cinematography
7	David	Garcia	14	Head of Cinematography
8	Ashley	Martinez	10	Head of Cinematography
9	Matthew	Lopez	12	Head of Constume and Makeup
10	Sarah	Scott	10	Head of Constume and Makeup
11	William	Clark	10	Head of Constume and Makeup

Data present in writer:

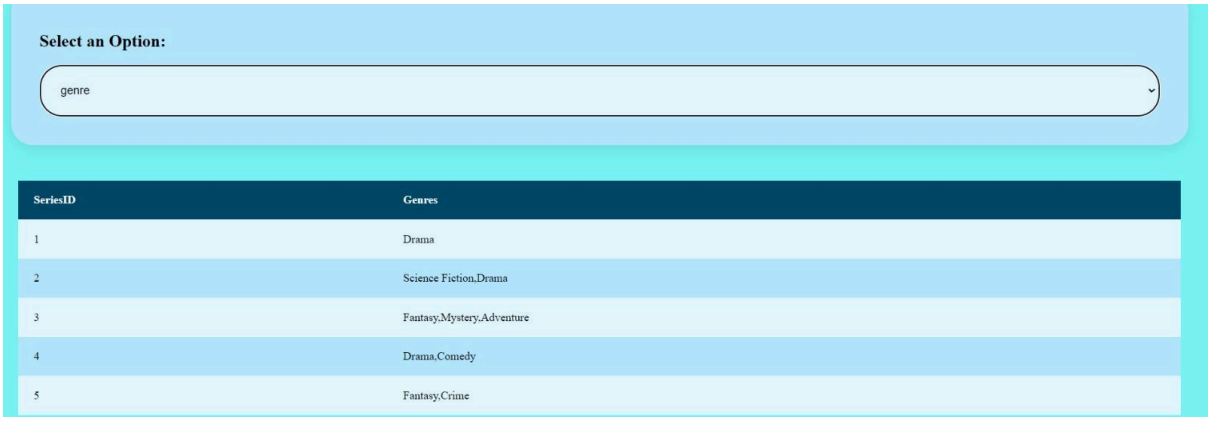


script_id	first_name	last_name	script_name
2	Sophie	Anderson	Stranger Things Script
3	Daniel	Hall	Game of Thrones Script
4	Ella	Thompson	Breaking Bad Script
6	Mia	Moore	Money Heist Script
11	Henry	Robinson	Peaky Blinders Script

Data present in release:

R_ID	Platform	ReleaseDate	SeriesName
1	Netflix	2023-01-14T18:30:00.000Z	Stranger Things
2	Netflix	2023-03-19T18:30:00.000Z	Game of Thrones
3	HBO	2023-05-09T18:30:00.000Z	Breaking Bad
4	AMC	2023-06-04T18:30:00.000Z	Money Heist
5	Netflix	2023-08-11T18:30:00.000Z	Peaky Blinders

Data present in genre:

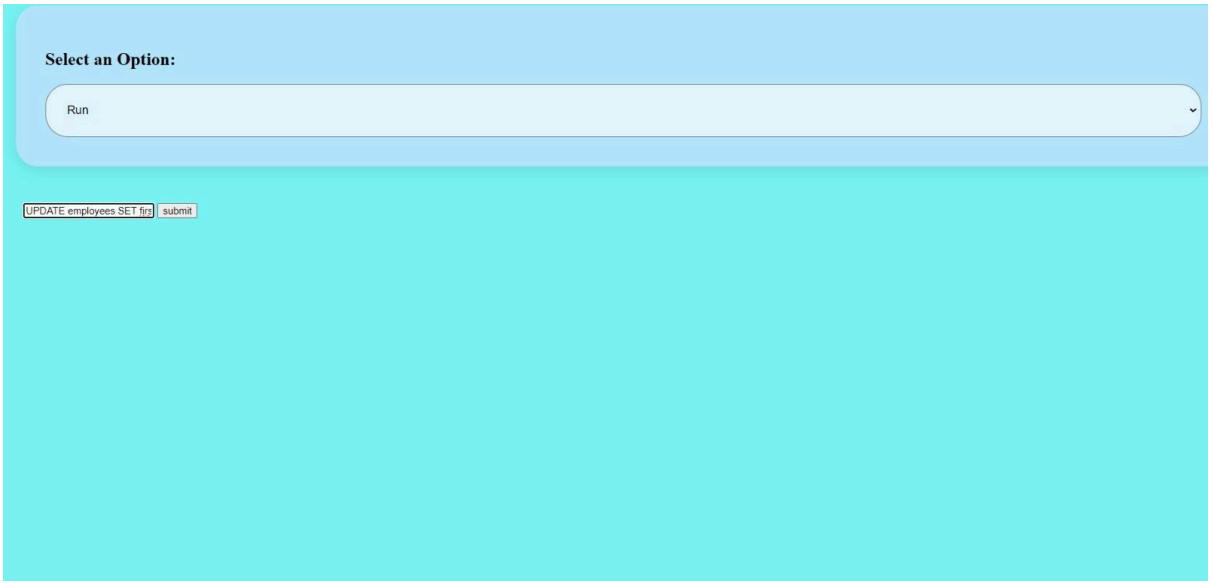


SeriesID	Genres
1	Drama
2	Science Fiction,Drama
3	Fantasy,Mystery,Adventure
4	Drama,Comedy
5	Fantasy,Crime

Data present in employees:

emp_id	first_name	last_name	DOB	age
6	tanamy	verma	1990-05-24T18:30:00.000Z	33
112	Matthew	Brown	1993-09-27T18:30:00.000Z	30
147	Piyush	Joshi	1992-08-14T18:30:00.000Z	31
164	Andrew	Harris	1992-10-08T18:30:00.000Z	31
172	William	Clark	1981-06-11T18:30:00.000Z	42
175	Ashley	Martinez	1991-03-11T18:30:00.000Z	32
200	Ella	Thompson	1992-09-25T18:30:00.000Z	31
214	John	Doe	1985-02-09T18:30:00.000Z	38
217	Sarah	Scott	1980-12-27T18:30:00.000Z	43
226	Benjamin	Lopez	1996-07-22T18:30:00.000Z	27
238	Kevin	Evans	1993-03-15T18:30:00.000Z	30

Updating employee details of emp_id=214



```
UPDATE employees
SET first_name = 'Sanket', last_name = 'Vashisht', DOB = '1990-01-15'
WHERE emp_id = 214;
```

Previous Data:

Select an Option:

employees

emp_id	first_name	last_name	DOB	age
6	tanamy	verma	1990-05-24T18:30:00.000Z	33
112	Matthew	Brown	1993-09-27T18:30:00.000Z	30
147	Piyush	Joshi	1992-08-14T18:30:00.000Z	31
164	Andrew	Harris	1992-10-08T18:30:00.000Z	31
172	William	Clark	1981-06-11T18:30:00.000Z	42
175	Ashley	Martinez	1991-03-11T18:30:00.000Z	32
200	Ella	Thompson	1992-09-25T18:30:00.000Z	31
214	John	Doe	1985-02-09T18:30:00.000Z	38
217	Sarah	Scott	1980-12-27T18:30:00.000Z	43
226	Benjamin	Lopez	1996-07-22T18:30:00.000Z	27
238	Kevin	Evans	1993-03-15T18:30:00.000Z	30

Updated Data:

emp_id	first_name	last_name	DOB	age
6	tanamy	verma	1990-05-24T18:30:00.000Z	33
112	Matthew	Brown	1993-09-27T18:30:00.000Z	30
147	Piyush	Joshi	1992-08-14T18:30:00.000Z	31
164	Andrew	Harris	1992-10-08T18:30:00.000Z	31
172	William	Clark	1981-06-11T18:30:00.000Z	42
175	Ashley	Martinez	1991-03-11T18:30:00.000Z	32
200	Ella	Thompson	1992-09-25T18:30:00.000Z	31
214	Sanket	Vashisht	1990-01-14T18:30:00.000Z	33
217	Sarah	Scott	1980-12-27T18:30:00.000Z	43

References

- Google.com
- PPT and PDFs
- College Lectures and Notes
- Book: Korth, Silbertz, Surdarshan,” Data Base Concepts”
- Youtube Lectures