CSC 370 – Database Systems

Dr. Sean Chester

Movie Rating System

Project Sprint: 4

July 10th, 2024

Team27J

Alyssa Taylor V00987477

Karan Gosal V00979752

During Sprint 4, we were able to achieve most parts of **Level-3** proficiency in Data Modeling except the null values. Inheritance notions and weak entity sets were implemented into the system. The *Media* was added with subclasses *Movies* and *Shows*; *Shows* having the weak entity *episodes* attached via the weak relationship *Has_Episodes.* The addition of weak entities and relationships allows the database design to be more understandable and intuitive at a glance and also an aspect of generalization.

These weak entities and relationships facilitated the redesign of our ERD in order to incorporate the changes they brought. This allowed us to enhance various aspects of the ERD such as minimality, expressiveness, readability, self-explanation, extensibility, and normality.

Finally, due to the changes made to the system, we had to re-ensure that BCNF (and by extension 3NF) still holds for all dependencies. As it stands, both BCNF and 3NF are confirmed for the system. This is discussed in further depth later.

At the end of Sprint 5, we plan to achieve some proficiency in Data Modeling **Level-4**, specifically points 4 and 5, "Develops conceptual schemata that are compatible with multiple logical database models" and "Avoids over-engineering designs in favour of simplicity and satisfying only those requirements already known". We will also try to cover the **Level-4** of Data Analytics specifically point 3.

To do this we will look into several different concepts taught during lectures such as 4NF, triggers, constraint checks, null value use and also unit testing.

These concepts relate directly to the database's **current limitations**:

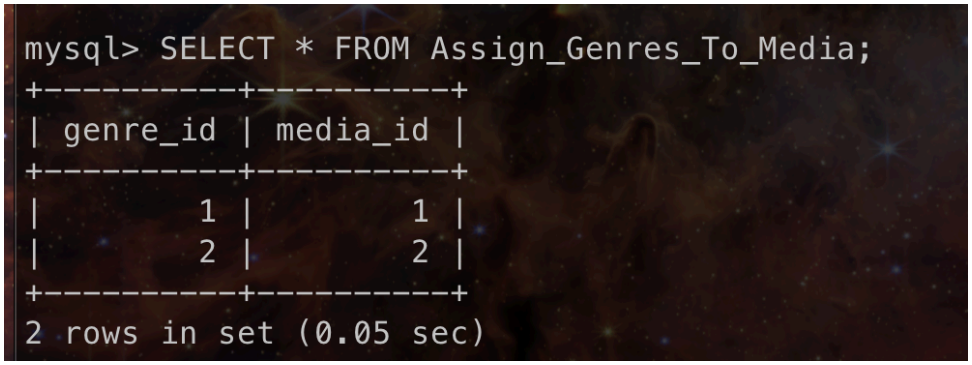
- 4NF is not ensured
    - Currently, the database is only confirmed to be in 3NF and BCNF, so it is not in the most rigorous normal form available yet, limiting our ability to fix anomalies relating to MVDs.
    - 4NF will be discussed by the team during Sprint 5 in order to ensure that the database is in the most rigorous form possible.
- Lack of triggers and Unchecked Constraints
    - Currently, constraints as discussed in Sprint 2 are left completely unchecked. During Sprint 5 we will look into all system constraints and discuss how to add checks for each.
    - Necessary triggers will be discussed and decided upon during Sprint 5, but a possible option is to add a limit to some actions that numerous users could

take. For example, adding a limit to how many times a show could be updated by admin users in a certain time period.

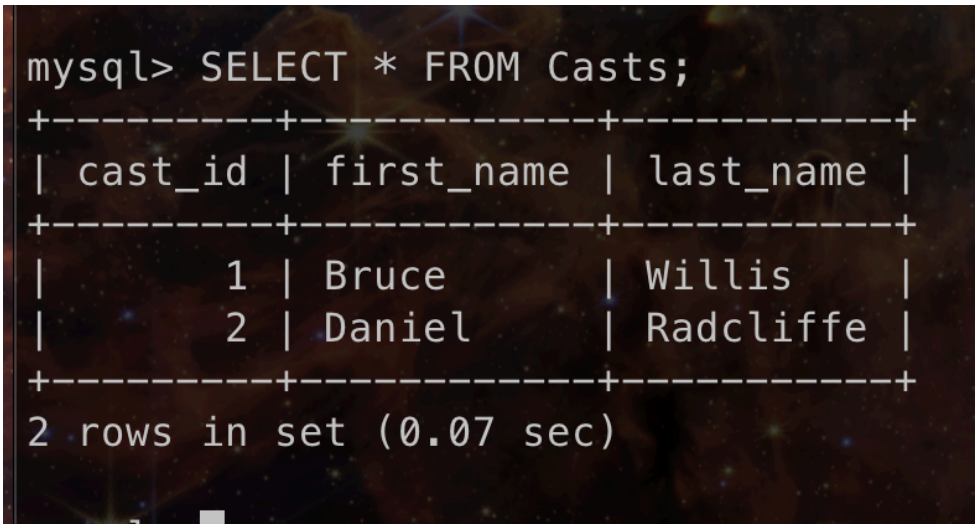
- No Null Value Support
    - As it stands, the database has little to no specified support for Null values within complex queries. During Sprint 5, the team will discuss Null values and how they can play into the database and make any changes necessary to ensure the integrity of tables, queries, and data containing Null.
- No Unit Tests
    - The database has yet to have any testing done on it, which could lead to major issues in a real world situation as problematic behaviours would go unnoticed.
    - During Sprint 5, an input domain model will be designed to facilitate the creation of robust unit tests for this system.
- Check further simplifying of Queries
    - We can explore our current queries and see if some of them can be simplified. This overall will improve the performance.

Users

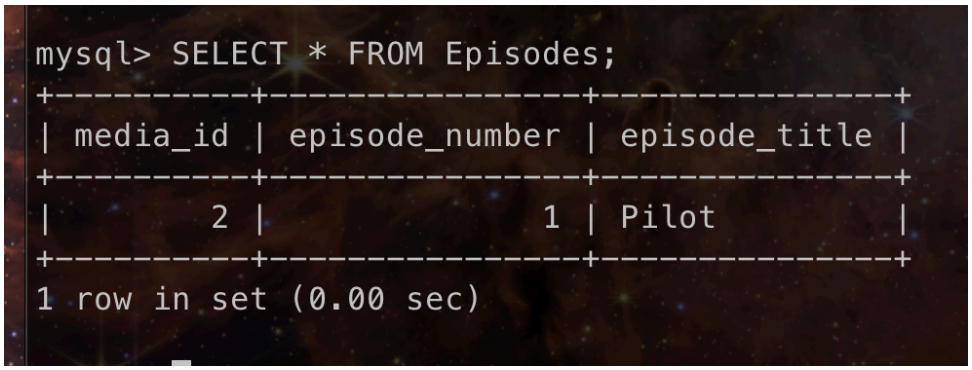
user_id	username	password	email	full_name	is_admin	created_at
1	johndoe	securepassword	johndoe@example.com	John Doe	FALSE	2024-06-01 12:34
2	sidhum	testpassword	sidhumoose@example.com	Sidhu Moose	TRUE	2024-06-01 12:50



```
mysql> SELECT * FROM Assign_Genres_To_Media;
+----------+----------+
| genre_id | media_id |
+----------+----------+
|        1 |        1 |
|        2 |        2 |
+----------+----------+
2 rows in set (0.05 sec)
```



```
mysql> SELECT * FROM Casts;
+---------+------------+-----------+
| cast_id | first_name | last_name |
+---------+------------+-----------+
|       1 | Bruce      | Willis    |
|       2 | Daniel     | Radcliffe |
+---------+------------+-----------+
2 rows in set (0.07 sec)
```



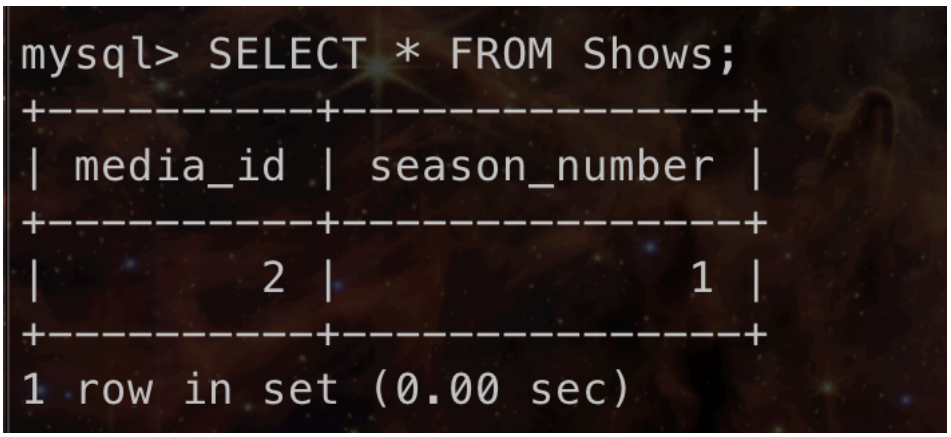
```
mysql> SELECT * FROM Episodes;
+----------+----------------+---------------+
| media_id | episode_number | episode_title |
+----------+----------------+---------------+
|        2 |              1 | Pilot         |
+----------+----------------+---------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM Genres;
+----------+------------+
| genre_id | genre_name |
+----------+------------+
|        1 | Action     |
|        2 | Adventure  |
+----------+------------+
2 rows in set (0.00 sec)
```

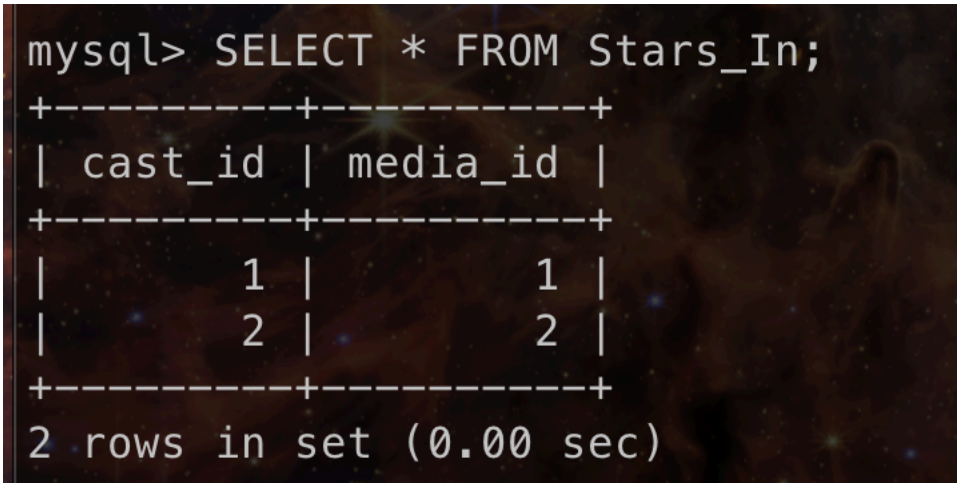
```
mysql> SELECT * FROM Media;
+----------+--------------------+-------------------------------+--------------+
| media_id | title              | synopsis                      | release_year |
+----------+--------------------+-------------------------------+--------------+
|        1 | Die Hard           | Movie about a police officer. |         2012 |
|        2 | The Big Bang Theory | The show about the scientists. |         2007 |
+----------+--------------------+-------------------------------+--------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Movies;
+----------+----------+
| media_id | duration |
+----------+----------+
|        1 |      120 |
+----------+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM Rates;
+---------+----------+--------+----------------+-----------+
| user_id | media_id | rating | review         | show_user |
+---------+----------+--------+----------------+-----------+
|       1 |        1 |      5 | Great movie!   |         1 |
|       2 |        2 |      4 | Enjoyable show. |         0 |
+---------+----------+--------+----------------+-----------+
2 rows in set (0.00 sec)
```



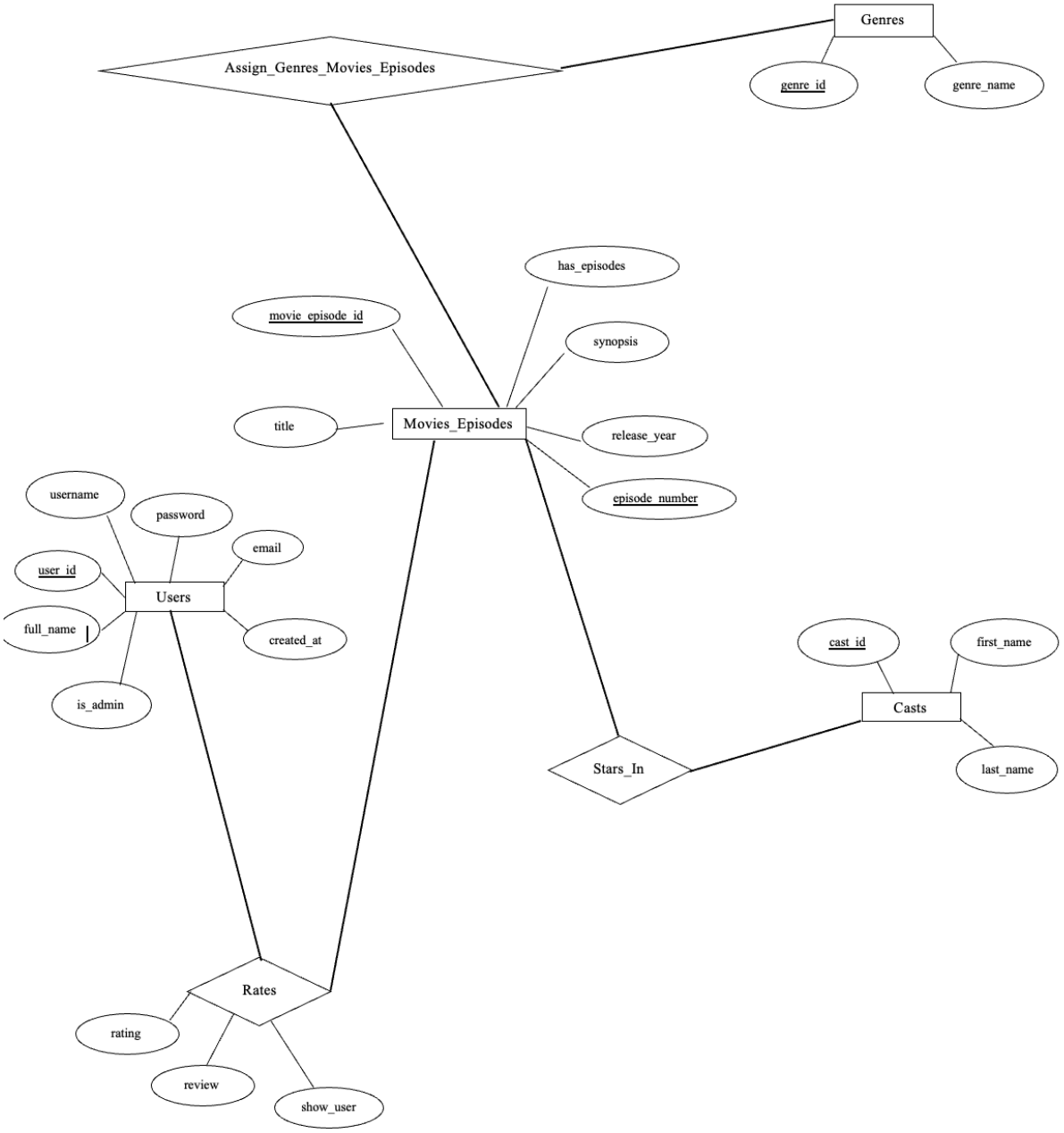
```
mysql> SELECT * FROM Shows;
+----------+---------------+
| media_id | season_number |
+----------+---------------+
|        2 |             1 |
+----------+---------------+
1 row in set (0.00 sec)
```



```
mysql> SELECT * FROM Stars_In;
+---------+----------+
| cast_id | media_id |
+---------+----------+
|       1 |        1 |
|       2 |        2 |
+---------+----------+
2 rows in set (0.00 sec)
```

All the tables above represent the tables that follow our updated ERD schema.

Original ER Diagram**:**



INHERITANCE AND WEAK ENTITY:

Inheritance and weak entity concepts are crucial for refining the database schema and improving overall design. We have identified areas where these concepts can be applied: the *Movie_Episodes* table. Currently, we are satisfied with the rest of our schema but again it can be changed in future introducing the weak entity and inheritance concept. The inheritance can be added by creating an entity called *Media* replacing the *Movies_Episode* table adding subclasses or special entities *Movies* and *Shows*. Then, we can have a weak entity *Episodes* in relation to *Shows* entity with a relationship *Has_Episodes*.

In the *Media* table, we can utilize inheritance to categorize media into two types that inherit from the base *Media* table. One type is *Movies*, distinguished by attributes such as *duration* indicating how long the movie is. The other type is *Shows*, characterized by a *season_number* attribute.

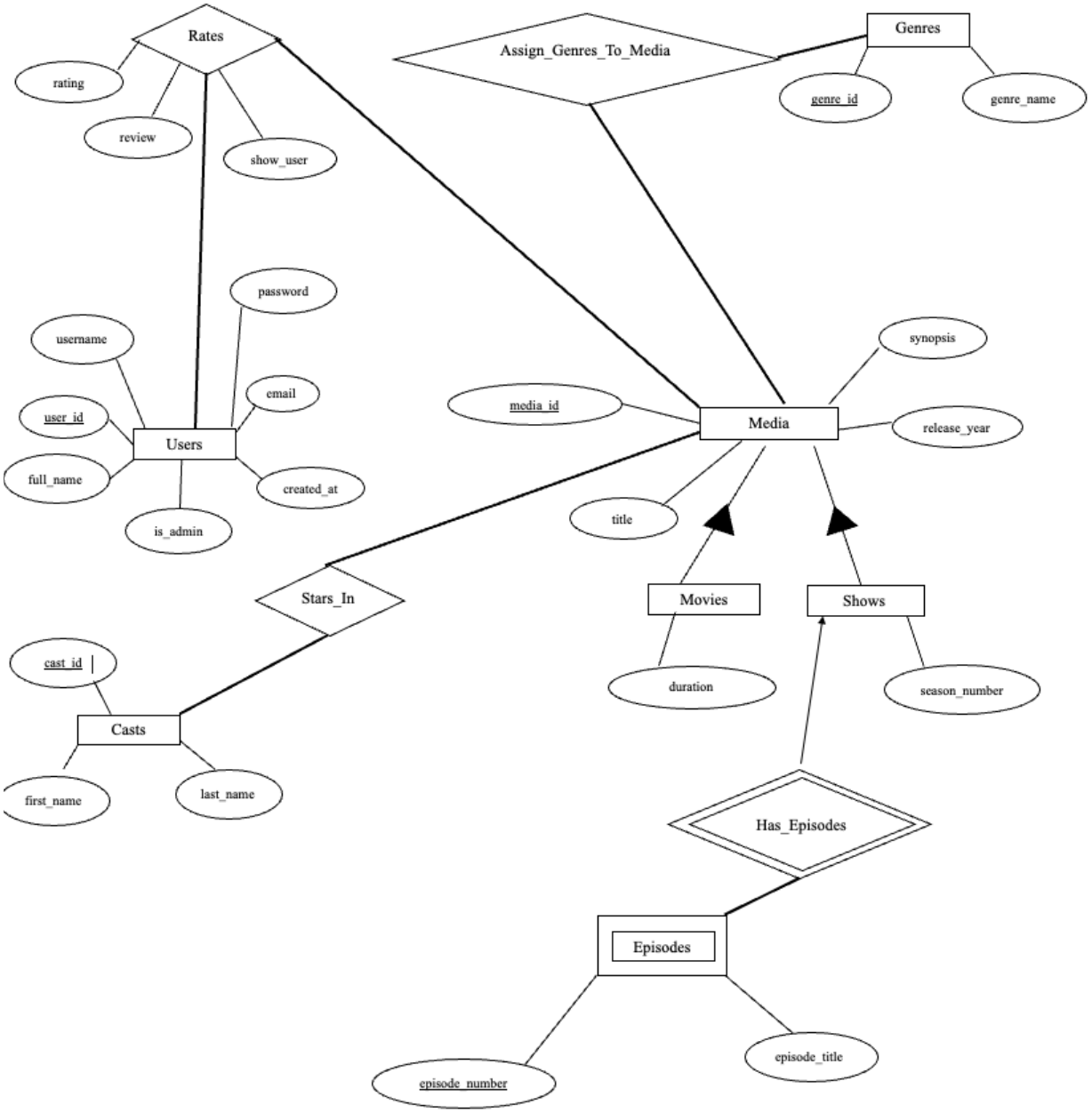
Similarly, the weak entity concept can be applied to Episode information for the shows. Shows can have one or more episodes. This constitutes a weak entity relationship because episodes exist only if the show exists. So, we have the weak entity relation as *Has_Episodes* with the weak entity *Episodes*.

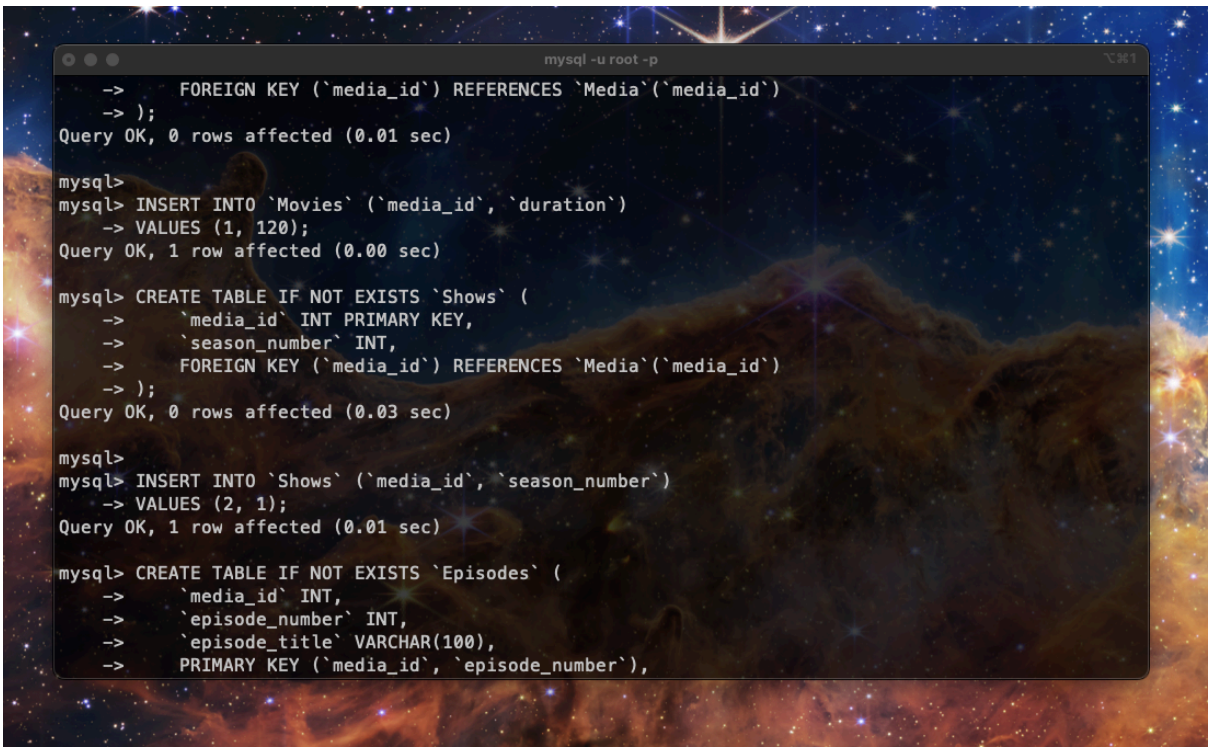
These approaches help maintain a cleaner database design by structuring media and episodes as a separate table.

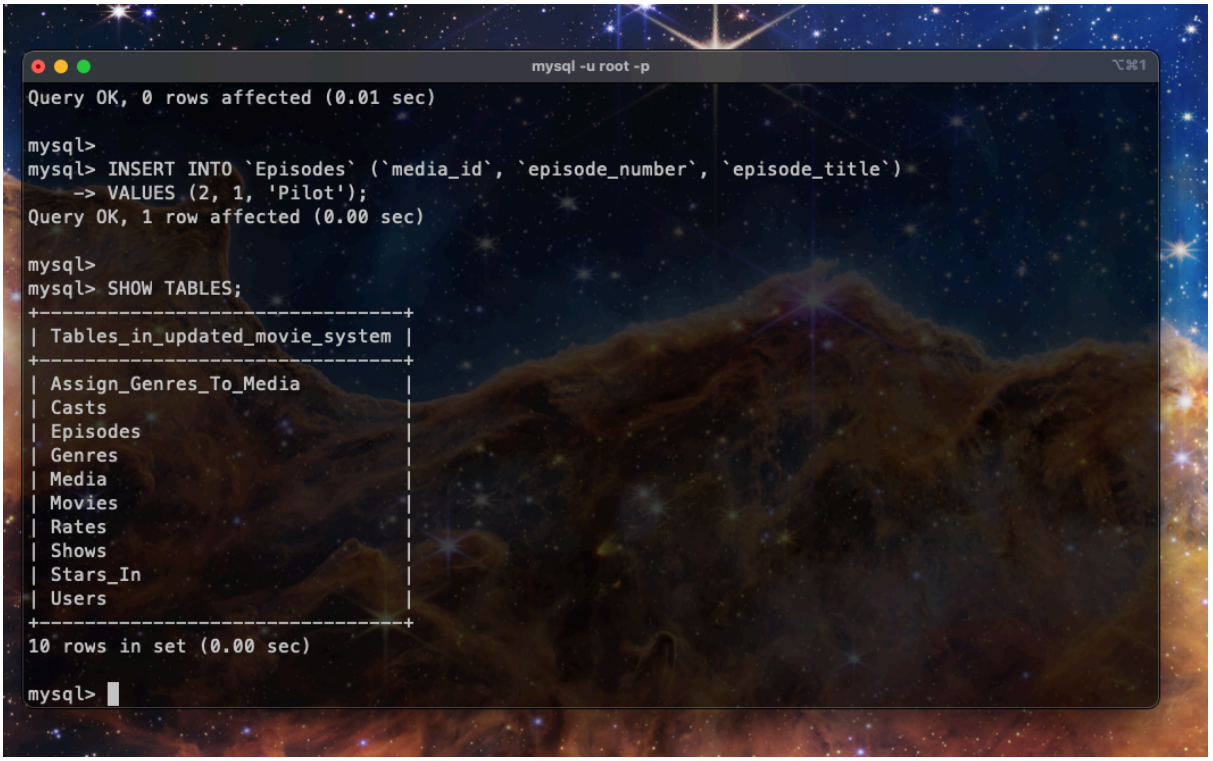
After changing the ERD Diagram, we now have the media entity with some inheritance for the Movies and Shows. Adding more than one season for the show as per our design, we need to create a separate *media_id* for it. This is done to reduce the complexity as otherwise we have to make the *season_number* and *media_id* as a composite key. If we didn't have movies, then it would have worked well but now it would make things complex.

Commands that will alter the existing database schema are in the *PS_4.sql* file in the repo.

Updated Entity-Relationship Diagram (ERD):




mysql -u root -p
->   FOREIGN KEY (`media_id`) REFERENCES `Media`(`media_id`)
->);
Query OK, 0 rows affected (0.01 sec)
mysql>
mysql> INSERT INTO `Movies` (`media_id`, `duration`)
-> VALUES (1, 120);
Query OK, 1 row affected (0.00 sec)
mysql> CREATE TABLE IF NOT EXISTS `Shows` (
->   `media_id` INT PRIMARY KEY,
->   `season_number` INT,
->   FOREIGN KEY (`media_id`) REFERENCES `Media`(`media_id`)
->);
Query OK, 0 rows affected (0.03 sec)
mysql>
mysql> INSERT INTO `Shows` (`media_id`, `season_number`)
-> VALUES (2, 1);
Query OK, 1 row affected (0.01 sec)
mysql> CREATE TABLE IF NOT EXISTS `Episodes` (
->   `media_id` INT,
->   `episode_number` INT,
->   `episode_title` VARCHAR(100),
->   PRIMARY KEY (`media_id`, `episode_number`),


mysql -u root -p
Query OK, 0 rows affected (0.01 sec)
mysql>
mysql> INSERT INTO `Episodes` (`media_id`, `episode_number`, `episode_title`)
-> VALUES (2, 1, 'Pilot');
Query OK, 1 row affected (0.00 sec)
mysql>
mysql> SHOW TABLES;
| Tables_in_updated_movie_system |
| Assign_Genres_To_Media |
| Casts |
| Episodes |
| Genres |
| Media |
| Movies |
| Rates |
| Shows |
| Stars_In |
| Users |
10 rows in set (0.00 sec)
mysql>

DESIGN QUALITY:

Each principle of database schema design quality - minimality, expressiveness, readability, self-explanation, extensibility, and normality holds an important role in a good database design and below is how our Movie Rating System exemplifies these qualities.

Minimality

- Focused Entities and Relationships: Our schema includes essential entities such as *Users, Media, Movies, Shows, Episodes, Genres and Casts*, each with attributes directly relevant to the Movie Rating System's requirements. **We have introduced subclasses for the Media Entity which provides even more refined entities and relations.** There are no redundant entities or unnecessary attributes that do not contribute to the system's functionality.

- Efficient Data Representation: By minimizing redundancy and ensuring each piece of information is represented only once where necessary, our schema optimizes data storage and retrieval efficiency without compromising on completeness. Here, in the new schema design, we have simplified the concept of shows by assigning a different *media ID* to each season of a show. This approach helps simplify the *Rates* and *Stars_In* tables, as it prevents empty spots in the relationship tables for movies, which do not have episodes.

Expressiveness

- Natural Representation of Entities: Entities such as *Users* and *Media* are named intuitively and structured logically, reflecting real-world concepts. For example, attributes like username, password, title, and release_year are straightforward and meaningful, making the schema easy to comprehend.

- Intuitive Relationship Definitions: Relationships like *Rates, Stars_In*, and *Assign_Genres_to_Media* are clearly defined and reflect common interactions within a movie rating system, enhancing understanding and usability.

Readability

- Aesthetic Schema Design: Visual representations such as ER diagrams or relational schemas adhere to readability principles. There are no line crossings and bends, and position entities and relationships are logically positioned (e.g., parents above children), which ensures clarity and ease of interpretation.

  NOTE: This was improved by updating our ER diagram as earlier our design lacks this. Examples can be *Media* entity (*Movies_Episodes* before) and its inherited entities *Movies* and *Shows*. Taking a look at the updated ER Diagram, we can say that the new schema after transformation is information-preserving mostly as information content is unchanged through inheritance. There is actually one extra attribute added which is the *season_number* to the shows which can be information changing specifically augmenting. However, the latter is a very minor change. Overall, we can say it is more information preserving.

- Clear and Concise Notation: Entity names, attribute labels, and relationship types are labelled explicitly and consistently.

Self-Explanation

- Comprehensive Annotations: Annotations within the schema (e.g., primary keys, foreign keys, constraints) provide sufficient detail to understand the schema's structure and behavior without extensive textual explanations.

- Intuitive Design Choices: The schema's design choices, such as using consistent naming conventions and standard relationship types, contribute to its self-explanatory nature. One can easily know how entities relate to each other and the system's operational logic.

Extensibility

- Modular Schema Architecture: The schema is designed in modular components (tables, relationships), facilitating localized changes and additions. **For instance,**

introducing new attributes (e.g., *season_number* in *Shows*) integrates seamlessly without disrupting existing schema elements.

- Scalable Relationship Management: Relationships are structured to accommodate future expansions and variations in data volume or complexity. This flexibility ensures that new requirements can be implemented without extensive redesign of the database schema. For example, **one can easily add a new subclass to the *Media* such as *Documentaries* or *Web series* without changing any other aspect.**

Normality

- Database Normalization: Tables are structured to adhere to standard normal forms (e.g., 1NF, 2NF, 3NF, BCNF). Currently, our database schema is normalized up to BCNF Normalization. This ensures data integrity, minimizes redundancy, and reduces anomalies such as data update issues or inconsistent dependencies.

- Efficient Data Handling: By following normalization principles, our schema optimizes database performance and storage efficiency, supporting smooth data operations and reliable information retrieval even after it grows in amount of data.

Since, we have updated our design, **we need to ensure if our tables are still in BCNF.**

Users

user_id -> username, password, full_name, email, is_admin, created_at
username -> user_id, password, full_name, email, is_admin, created_at
email -> username, password, full_name, user_id, is_admin, created_at

Candidate keys: {user_id}, {username}, {email}.
Primary key: {user_id}

Closures:

$\{user\_id\}^{+}$ = {username, password, full_name, email, is_admin, created_at, user_id}

$\{username\}^{+}$ = {username, password, full_name, email, is_admin, created_at, user_id}

$\{email\}^{+}$ = {username, password, full_name, email, is_admin, created_at, user_id}

Media

media_id -> title, synopsis, release_year

Closures:

$\{media\_id\}^{+}$ = {media_id , title, synopsis, release_year}

Candidate keys: {media_id}

Primary key: {media_id}

Movies

media_id -> duration

Closures:

$\{media\_id\}^{+}$ = {media_id, duration}

Since, the *Movies* table is inherited from the *Media* table, so it inherits all the attributes and itself has an attribute *duration*. Overall, the *media_id* is the primary key as well as foreign key.

Shows

media_id -> season_number

Closures:

$\{media\_id\}^{+} = \{media\_id, season\_number\}$

Since, the *Shows* table is inherited from the *Media* table, so it inherits all the attributes and itself has an attribute *season_number.* Overall, the *media_id* is the primary key as well as foreign key.

Genres

genre_id -> genre_name

genre_name -> genre_id

Closures:

$\{genre\_id\}^{+} = \{genre\_id, genre\_name\}$

$\{genre\_name\}^{+} = \{genre\_id, genre\_name\}$

Candidate keys: {genre_id}, {genre_name}

Primary key: {genre_id}

Casts

cast_id -> first_name, last_name

Closures:

$\{cast\_id\}^{+} = \{cast\_id, first\_name, last\_name\}$

Candidate keys: {cast_id}

Primary key: {cast_id}

Episodes

media_id, episode_number -> episode_title

Closures:

$\{ media\_id, episode\_number\}^{+} = \{media\_id, episode\_number , episode\_title\}$

Candidate keys: { media_id, episode_number }

Primary key: { media_id, episode_number }

As we can say left hand side of every FD is a candidate key and gets all the attributes for the table. Therefore, we can say our tables are in BCNF.

Based on the provided ERD and rationale on the design rules, we can affirm that our tables and database schema adhere well to good design principles and quality.

MINIMAL BASES AND PROJECTING FD'S

Basis of a set of FD's:

Two sets of functional dependencies are considered equivalent if they share the same closure, meaning that each set can be used to derive the same set of dependencies. When two sets of functional dependencies are equivalent, they essentially serve as a basis for each other. This implies that we can use either set to understand the constraints and relationships within the database, as they convey the same information about the dependencies among attributes. However, we will always prefer the minimal basis. We can first quickly explore the basis concept.

Casts

Original FD: Set A

{cast_id -> first_name, last_name}

New Constructed set of FD: Set B

{
  cast_id -> first_name
  cast_id -> last_name
}

Now, we can check if the above Set A and Set B are equivalent or are basis for each other.

Set A Closures excluding the empty set as we get nothing in it:

$\{ cast\_id \}^{+} = \{cast\_id, first\_name, last\_name\}$

$\{ first\_name \}^{+} = \{ first\_name \}$

$\{ last\_name \}^{+} = \{ last\_name \}$

$\{cast\_id, first\_name \}^{+} = \{cast\_id, first\_name, last\_name\}$

$\{cast\_id, last\_name \}^{+} = \{cast\_id, first\_name, last\_name\}$

{ first_name, last_name } $^{+}$ = { first_name, last_name}

{ cast_id , first_name, last_name } $^{+}$ = { cast_id , first_name, last_name}

Set B Closures excluding the empty set as we get nothing in it:

{ cast_id } $^{+}$ = {cast_id, first_name, last_name}

{ first_name } $^{+}$ = { first_name }

{ last_name } $^{+}$ = { last_name }

{cast_id, first_name } $^{+}$ = {cast_id, first_name, last_name}

{cast_id, last_name } $^{+}$ = {cast_id, first_name, last_name}

{ first_name, last_name } $^{+}$ = { first_name, last_name}

{ cast_id , first_name, last_name } $^{+}$ = { cast_id , first_name, last_name}

As, we can see both has the same closure set or are equivalent. Therefore, we these two FD's are basis for each other. This is just an example as these can be much more complex but this is just a proof of concept.

Minimal Basis:

As discussed in class, A set of functional dependencies *F* forms a **minimal basis** if it meets the following criteria; Here are the steps:

1) Each functional dependency has a right-hand side consisting of a single attribute.
2) Removing any functional dependency from the set would result in the set no longer being a basis.
3) Removing any attribute from the left-hand side of any functional dependency would also cause the set to cease being a basis.

We have the following relations:

Users(user_id, username, password, full_name, email, is_admin, created_at)

Media(media_id, title, synopsis, release_year)

Movies(media_id, duration)

Shows(media_id, season_number)

Genres(genre_id, genre_name)

Casts(cast_id, first_name, last_name)

Episodes(media_id, episode_number , episode_title)

We can test this on all of our FD's to see if they are minimal basis on each relation.

user_id -> username, password, full_name, email, is_admin, created_at

media_id -> title, synopsis, release_year

media_id -> duration

media_id -> season_number

genre_id -> genre_name

cast_id -> first_name, last_name

media_id, episode_number -> episode_title

Step 1

As looking at the set of FD's, we first need to get all the FD's as a singleton on the right hand side.

user_id -> username

user_id -> password

user_id -> full_name

user_id -> email

user_id -> is_admin

user_id -> created_at

media_id -> title

media_id -> synopsis

media_id -> release_year

media_id -> duration

media_id -> season_number

genre_id -> genre_name

cast_id -> first_name

cast_id -> last_name

media_id, episode_number -> episode_title

Step 2

Now, we can try removing FD from the set one by one. We test if we still have the basis otherwise, we need to keep the FD.

We will show on the first one:

user_id -> username

If we try to remove this, we get:

$\{user\_id\}^{+}$ = { password, full_name, email, is_admin, created_at, user_id}

we can say the *username* is out of the closure of *user_id*, so we need to keep this.

Similarly, we can test all the FD's. After testing all the FD's we found that we need to keep all the FD's.

Step 3

Now, we can test if we have any FD with a multi-attribute on the left-hand side of any FD and if we can remove one of its attributes.

media_id, episode_number -> episode_title

If we try to remove, *media_id,* we cannot get the *episode_title* as there are multiple episodes with the same number.
If we try to remove, *episode_number,* we cannot get the *episode_title* as there are multiple episodes in the shows.

After working through Step 1, 2 and 3, we can conclude, we have the minimal basis for the set of FD's.

Projecting FD's

Projecting functional dependencies (FDs) helps to find the FDs for a relation in case it is decomposed or a new relation is created. In our case, our table is already in BCNF, and as we proved earlier, it is a minimal basis. We will now showcase a proof of concept. Imagine that after the decomposition of some relation R, we obtain this cast relation and we want to determine its FDs.

Casts(cast_id, first_name, last_name)

We can find the closure of all the combination of attributes except the empty set.

{ cast_id }$^{+}$ = {cast_id, first_name, last_name}

{ first_name }$^{+}$ = { first_name }

{ last_name }$^{+}$ = { last_name }

{cast_id, first_name }$^{+}$ = {cast_id, first_name, last_name}

{cast_id, last_name }$^{+}$ = {cast_id, first_name, last_name}

{ first_name, last_name } $^{+}$ = { first_name, last_name}

{ cast_id , first_name, last_name } $^{+}$ = { cast_id , first_name, last_name}

We will take use the one which is in bold as it fulfils all the condition: it is minimal, cast_id is the primary key and it determines successfully all the attributes. We can discard the rest as some are superkeys and some are not able to determine all the attributes in the relation.

FD for Casts:

cast_id -> first_name, last_name

THIRD NORMAL FORM(3NF)

3NF is a database normalization form used to ensure that a database is free from certain types of redundancy and anomalies. A table is in 3NF if no non-prime attribute determines another non-prime attribute and the table is in 2NF. 3NF is useful in terms of dependency preserving.

Another way to think is if we have a relation *R* over attributes *C* with FD's *F* is in 3NF iff: Every FD *f* in *F* is in 3NF given *X*-> *Y*, then *X* is a superkey or Every attribute in *Y* is prime.

Our tables are already in BCNF and we know if a table is in BCNF, it is already in 3NF. However, we will still test to showcase and test if this is true.

user_id -> username, password, full_name, email, is_admin, created_at

Prime Attributes: user_id
Non-prime Attributes: username, password, full_name, email, is_admin, created_at

media_id -> title, synopsis, release_year

Prime Attributes: media_id
Non-prime Attributes: title, synopsis, release_year

media_id -> duration

Prime Attributes: media_id
Non-prime Attributes: duration

media_id -> season_number

Prime Attributes: media_id

Non-prime Attributes: season_number

genre_id -> genre_name

Prime Attributes: genre_id

Non-prime Attributes: genre_name

cast_id -> first_name, last_name

Prime Attributes: cast_id

Non-prime Attributes: first_name, last_name

media_id, episode_number -> episode_title

Prime Attributes: media_id, episode_number

Non-prime Attributes: episode_title

As we can see from above, all the prime attributes are determining the non-prime. In other words, left-hand side is a superkey. **Therefore, our tables are in 3NF as proved.**