Samuel Bernstein
Zachary Earl
CS 340 Final Project
http://flip1.engr.oregonstate.edu:5559/movies

CS 340 Updated Database Project Proposal

Our database design will represent an online movie streaming service. The end user of such a service will typically only be able to see movies, genres, and actors in order to search through the content. The database will hold all of the aforementioned information as well as the connections between these different entities based on the relationships defined below.

Entities:

1. User account – user id, first/last name, payment info, email
2. Movies – the actual movie titles and corresponding info
3. Movie genres – classification for different types of movies
4. Actors – will contain actor id along with first/last name

Relationships:

1. User can only have one account – this is a one-to-one relationship as each user may only have one account
2. Movies will have a genre – this is a many-to-one relationship because movies will only be classified by a single genre to avoid overpopulating the different genres with the same movies
3. Movies will have actors – this is a many-to-many relationship because a movie can have many actors and actors can be in many movies
4. Users will view movies – this is a many-to-many relationship because there will be many users and many movies, each of which can have multiple relationships with members of the other entity

Outline

Our database holds information akin to a company providing online streaming services to its customers. As one may suspect, the idea for this database was derived from Netflix, a company that started out as a DVD rental by mail business and eventually became a conglomerate by expanding its business into a platform specializing in viewing this media via online distribution. Hence, our database will mainly revolve around the users and the movies they choose to view. Of course there are a massive amount of movies comprised of several different entities.

These include the different genres a movie is classified as, the user may not like horror movies and therefore would want to exclude them in their search for a movie to watch. Movies also include many actors, the can see what actor played in which movie. Maybe the user wants to restrict a search for a specific movie title, this desired ability alone makes a database essential for online streaming services.

Special Note: Our database represents somewhat of a hybrid between the front end and back end of an online movie streaming database. On the backend there is access to users along with their personal information and movie viewing records, which typically would not be available to anyone else using the service. On the frontend, there is access to movies, genres, actors, and the relationships between them. This is more typical of what you would find as a user of an online movie streaming service.

Database Outline

Our database has four main entities: user, movies, genres, and actors. The user account table holds rows of users who have paid to use our media streaming services. The users can view, edit, and delete their accounts as well as add new ones. This table includes the following attributes: an auto incrementing id, f_name, l_name, and credit card. We must identify our customers that we automatically bill each month, therefore it's imperative that each account must have credit card information and name of the customer; these attributes can't be null. The primary key is the id.

Each movie must have an auto incrementing id, a title, and a genre. These are required attributes and cannot be null. The user can view, add, edit, and delete movies from their services. Some movies include the year it was produced, however we do not know the year all our movies were produced and so this is an optional attribute that could be null.. The genre_id is a foreign key referenced by genre's primary key, id, this will be discussed at length in the next paragraph. The primary key is the id.

As noted, each of our movies must be categorized into only one of many genres. Each genre must have an auto-incrementing id and name and thus neither can be null. However, we may not have a movie of a particular genre though. For example, a user could want a thriller but none are available and therefore this makes it a many to one relationship with total participation from movies. Therefore, if a user edits or adds a movie, they must select a genre from the dropdown menu. Though, a user can still add or edit genres, but they cannot delete them. The primary key is the id.

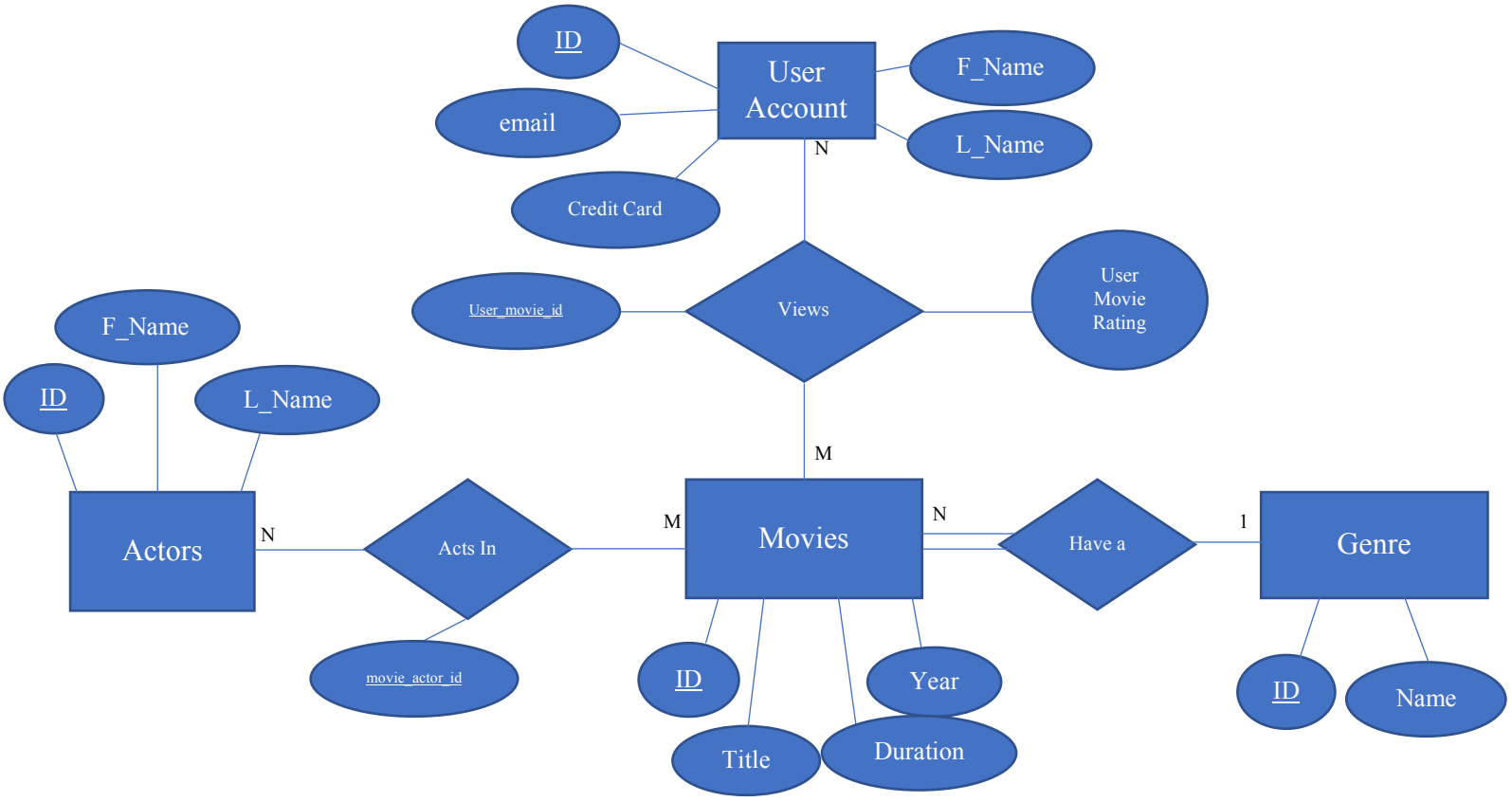
Some of our movies have participating actors, but there's a chance we may no longer offer a movie with a particular actor at this time. The user can add or edit actors, but not delete them. We must identify each actor and therefore they must have an auto incrementing id, f_name, and l_name; none of these can be null. The primary key is id.

There are two relationship tables in our database: 'acts_in' and 'views'. The 'acts_in' table describes which actor plays a part in which movie. As noted, there's a chance we may no longer offer a movie with a particular actor at this time and so our customers can still view information about an actor but not view his movies making this a many to many relationship. The only option the user has is to delete a relationship from the table, they cannot add or edit this relationship. This table features three rows including movie_id for movie id and actor_id for actor id. These are foreign keys that reference

movie.id and actor.id respectively and delete/update on cascade. It also includes the auto incrementing primary key which is movie_actor_id.

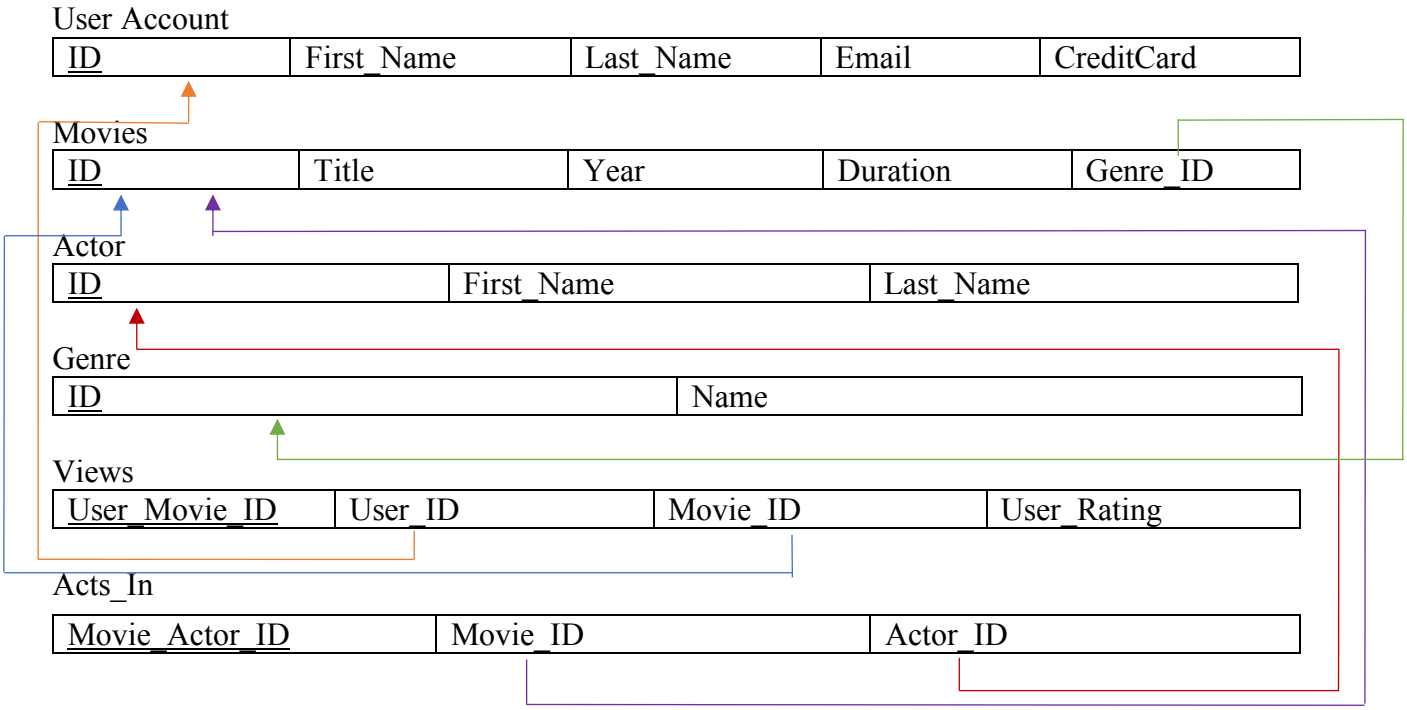
The ‘views’ table describes which user viewed which movie and the optional user rating. Our customers can choose whether to view and rate any of our available movies. They can add, edit, or delete views. This table features four rows: user_id, movie_id, user_rating, and user_movie_id. The rating can be null at any time, the user can rate a movie between 1-5, select the user and movie via a dropdown menu. This relationship table describes a many to many relationship since a user can exist without viewing a movie and a movie can exist without ever being viewed. The user_id and the movie_id are foreign keys that reference user.id and movie.id respectively and delete/update on cascade. The primary key is user_movie_id.

ERD



- Every user account can view zero or more movies (an account may exist without having viewed a movie)
- Movies can be viewed by zero or more user accounts (a movie can have zero views)
- A user account has the option to give a rating to any viewed movie in the database
- Every movie has exactly one genre
- A genre can have zero or more movies (movies can be removed from the database and the genre can still exist without any movies belonging to it)
- Each movie has zero or more actors
- Each actor can be in zero or more movies (if a movie is removed from the database, the actor can still exist without belonging to a movie in the database)

Schema



Data Definition Queries

```
CREATE TABLE user (
   id INT AUTO_INCREMENT,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   email VARCHAR(255) NOT NULL,
   credit_card VARCHAR(16) NOT NULL,
   PRIMARY KEY (id)
) ENGINE=InnoDB;

CREATE TABLE genre (
```

```
   id INT AUTO_INCREMENT,
   name VARCHAR(255) NOT NULL,
   PRIMARY KEY (id)
) ENGINE=InnoDB;

CREATE TABLE movies (
   id INT AUTO_INCREMENT,
   title VARCHAR(255) NOT NULL,
   year INT(4),
   duration INT(3),
   genre_id INT NOT NULL,
   PRIMARY KEY (id),
   FOREIGN KEY (genre_id) REFERENCES genre(id)
) ENGINE=InnoDB;

CREATE TABLE actors (
   id INT AUTO_INCREMENT,
   first_name VARCHAR(255) NOT NULL,
   last_name VARCHAR(255) NOT NULL,
   PRIMARY KEY (id)
) ENGINE=InnoDB;

CREATE TABLE views (
   user_movie_id INT AUTO_INCREMENT,
   user_id INT,
   movie_id INT,
   rating INT(1),
   FOREIGN KEY (user_id) REFERENCES user(id)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (movie_id) REFERENCES movies(id)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   PRIMARY KEY (user_movie_id)
) ENGINE=InnoDB;

CREATE TABLE acts_in (
   movie_actor_id INT AUTO_INCREMENT,
   movie_id INT,
   actor_id INT,
   FOREIGN KEY (movie_id) REFERENCES movies(id)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
   FOREIGN KEY (actor_id) REFERENCES actors(id)
      ON UPDATE CASCADE,
   PRIMARY KEY (movie_actor_id)
) ENGINE=InnoDB;
```

Data Manipulation Queries

```
--Adding data--

--Movies
INSERT INTO movies(title, year, duration, genre_id)
VALUES([movie],[year],[duration],[genre]);
--Users
INSERT INTO user(first_name, last_name, email, credit_card)
VALUES([firstName],[lastName],[email],[creditCard]);
--Genres
INSERT INTO genre(name) VALUES([genre]);
--Actors
INSERT INTO actors(first_name, last_name) VALUES([firstName],[lastName]);
--Views
INSERT INTO views(user_id, movie_id, rating) VALUES([user],[movie],[rating]);
--Acts In
INSERT INTO acts_in(actor_id, movie_id) VALUES([actor],[movie]);

--Editing Data--

--Movies
UPDATE movies m
INNER JOIN genre g
ON m.genre_id = g.id
SET title = [title], year = [year], duration = [duration], genre_id = [genre]
WHERE m.id = [movie];

--Users
UPDATE user
SET first_name = [firstName], last_name = [lastName], email = [email], credit_card =
[creditCard]
WHERE id = [user];

--Genres
UPDATE genre
SET name = [name]
WHERE id = [genre];

--Actors
UPDATE actors
SET first_name = [firstName], last_name = [lastName]
WHERE id = [actor];

--Views
UPDATE views
```

```
SET user_id= [user], movie_id=[movie], rating=[rating]
WHERE user_movie_id = [userMovie];

--Removing Data--

--Movies
DELETE FROM movies WHERE id = [removeMovie];

--Users
DELETE FROM user WHERE id = [removeUser];

--Views
DELETE FROM views WHERE user_movie_id = [removeView];

--Acts In
UPDATE acts_in ai
INNER JOIN actors a
ON ai.actor_id=a.id
INNER JOIN movies m
ON ai.movie_id=m.id
SET a.first_name = [fistName], a.last_name = [lastName], title = [title]
WHERE ai.movie_actor_id = [movieActor];

--Viewing Data--

--Movies
SELECT m.id, title, year, duration, name
FROM movies m
INNER JOIN genre g ON m.genre_id = g.id
WHERE m.id = [movie];

--Users
SELECT id, first_name, last_name, email, credit_card
FROM user
WHERE id = [user];

--Genres
SELECT id, name
FROM genre
WHERE id = [genre];

--Actors
SELECT id, first_name, last_name
FROM actors
WHERE id = [actor];

--Views
SELECT v.user_movie_id, u.first_name, u.last_name, m.title, v.rating
```

```
FROM views v
INNER JOIN movies m
ON v.movie_id = m.id
INNER JOIN user u
ON v.user_id = u.id
WHERE v.user_movie_id = [view];

--Acts In
SELECT movie_actor_id, first_name, last_name, title
FROM actors a
INNER JOIN acts_in ai
ON a.id = ai.actor_id
INNER JOIN movies m
ON ai.movie_id = m.id
WHERE movie_actor_id = [actsIn];

--User searches for movie by title
SELECT title, year, duration, name
FROM movies m
INNER JOIN genre g
ON m.genre_id = g.id
WHERE title=[movieTitle];
```