Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU34041 Information Management II

Database Design Project

Alice Doherty, 19333356

25th March 2022

Table of Contents

Section A: Description of Database Application Area and ER Model

Application Description

I have chosen to model the Academy Awards, commonly referred to as the Oscars. Individuals are nominated in various categories for their performance or work on a particular film released in the previous year. The award show itself is the night in which the winners are announced and include performances, as well as a variety of celebrities to present the awards. The main use case of this application would be to query across the nominations stored for a specific category, individual, or film. This is similar to the Academy Awards Database[1] available online. Organisers may also find having the details of all performers and presenters stored useful, for example for payroll. To help describe the application, I will briefly describe each table represented by the system.

Academy Award Show

This entity represents the award show itself and the includes details about the show like where it will be held, and who the official broadcaster is. The show host refers to the individual who hosts the overall show and is distinct from the various presenters who present the awards. Note that the last few Oscars have not had a host and so this attribute is optional. The ceremony year refers to the year the show is held and is the primary key as there will always only be a single show each year which determines the values of all other attributes of the show.

Performer

This represents an individual performer at the show and includes details about how much they are getting paid, and what song(s) they will perform. Note that multiple individuals may perform the same song together (for example, a duet), and this can be represented in the Song table by having two PerformerID associated with the same song name.

[1] https://awardsdatabase.oscars.org/

Presenter

Each award is presented by one or more individuals. Details of which award category the individual is presenting as well as how much they are getting paid are represented by this entity.

Voter

The Academy elects individuals to nominate and vote for the Oscar winners. Each of these voters belongs to one of 17 branches[2], each of which represent a certain discipline such as production, acting, sound, etc. I have also chosen to store the voter's race and gender as the Oscars have been subject to controversy regarding the lack of diversity within their voters[3], which translates to a lack of diversity in nominees. Although, the data I am using in my SQL is far from accurate, I thought these fields would create some interesting queries and would be something I would be interested to see in a real-world application.

Nominee

This represents an individual that is nominated for an award. The profession attribute indicates whether they are an actor, cinematographer, animator, etc. Note that the same nominee entity is used throughout all years. For example, a nominee with the ID 1 may have been nominated for different awards in both 2021 and 2022. Both nominations would be associated with the nominee with ID 1. The only time a name should be repeated in the nominee table is if there are multiple people with the same name.

Film

This represents an individual film and stores its title, director, genre, and release year. Note that the release year is not the same as the ceremony year as a film must be released in the year before the ceremony to qualify for an Oscar.

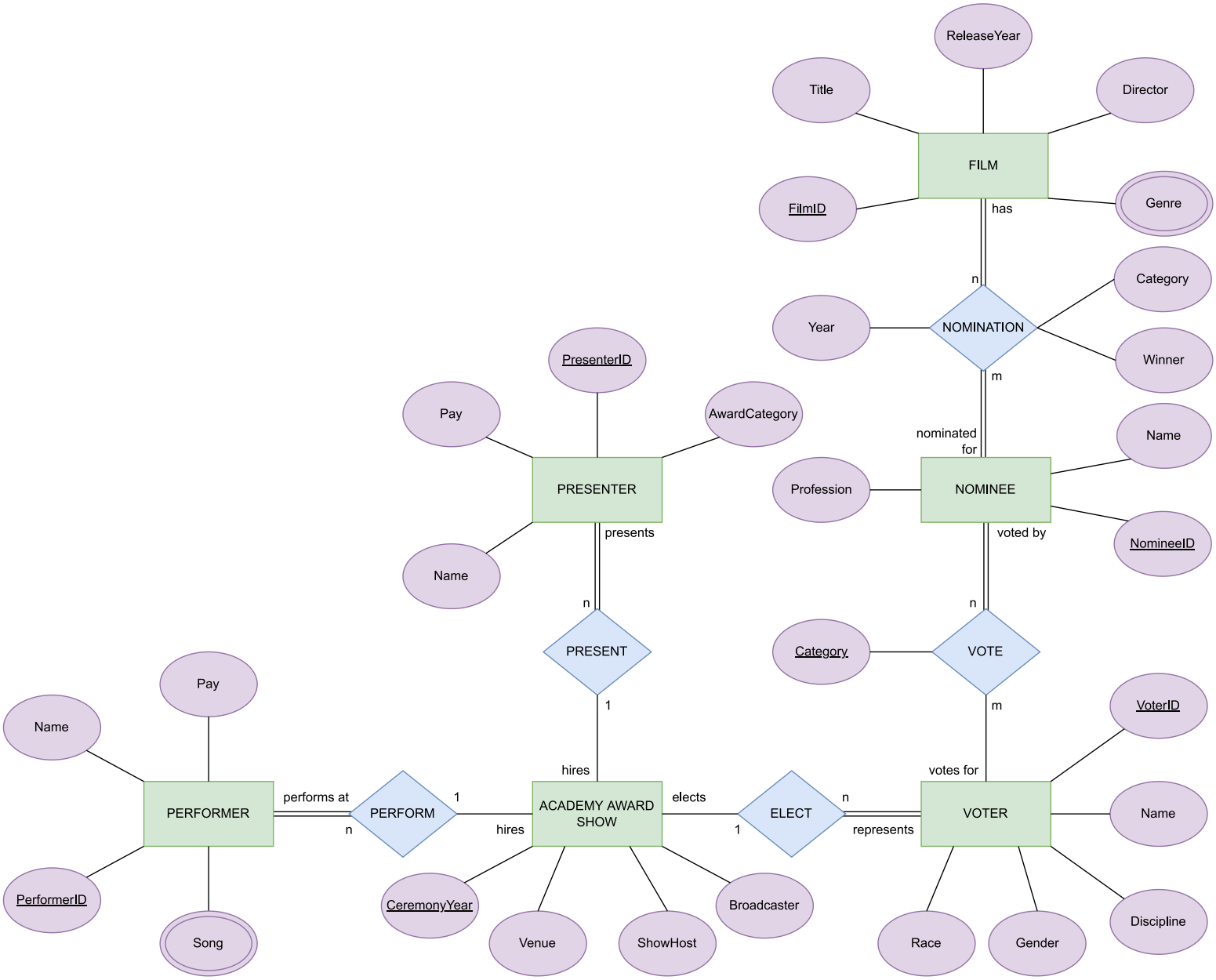
[2] More details on how the Oscar voting works can be found here: https://variety.com/feature/who-votes-on-oscars-academy-awards-how-voting-works-1203490944/

[3] A 2012 study by the Los Angeles Times found that 94% of voters were Caucasian and 77% were male. Source: https://www.latimes.com/entertainment/envelope/oscars/la-et-unmasking-oscar-academy-project-html-htmlstory.html

Nomination

This entity represents a relationship between a film and nominee. It also stores the category of the nomination, year of nomination, and whether that nominee won or not. If the award show has not happened yet, the boolean winner attribute will be NULL. Year of nomination is the same as the ceremony year (represented as a foreign key) and is not the same as the film release year as stated above. More detail on the keys and constraints for this relationship will be described later in the report.

Entity Relationship Diagram



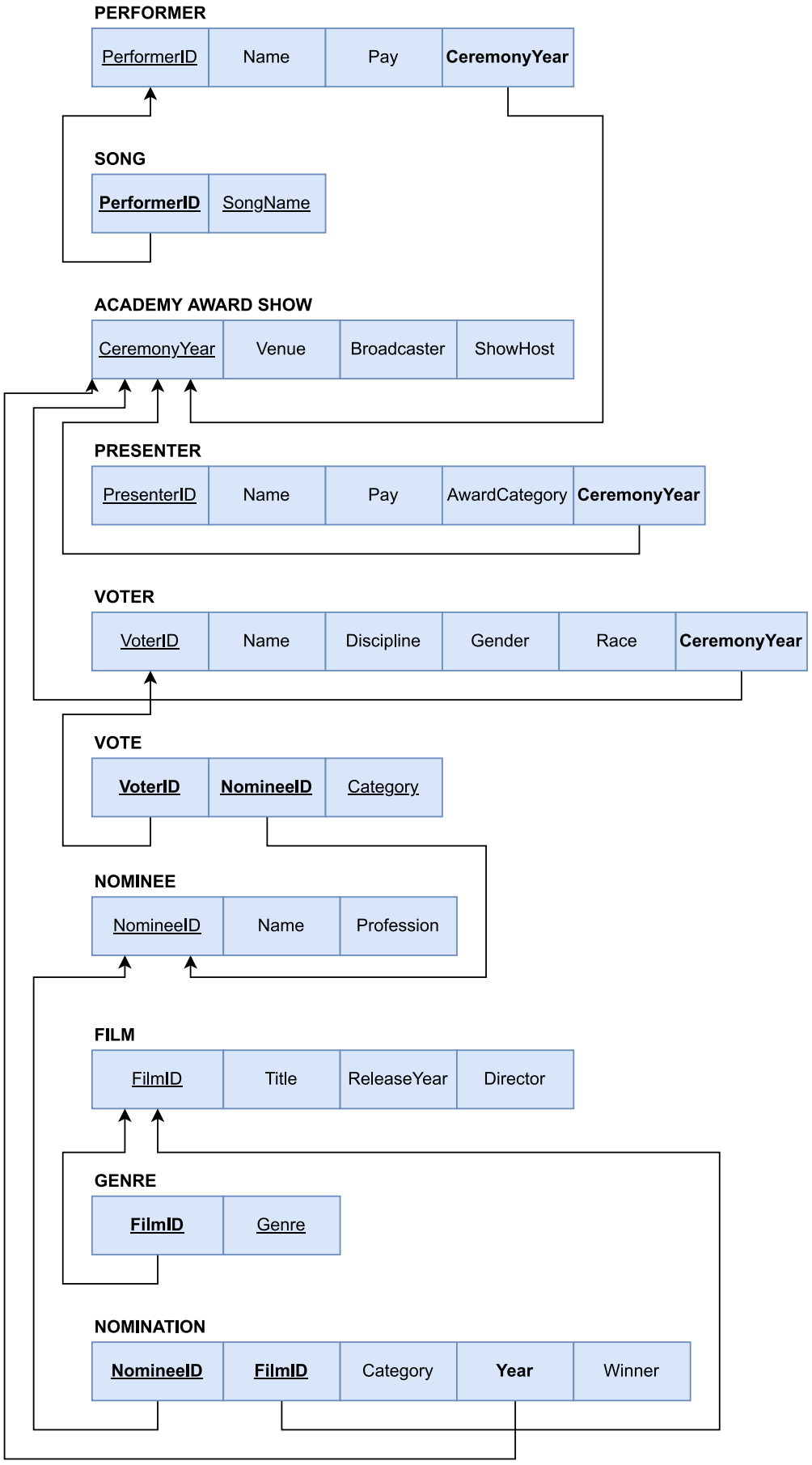
The entity relationship (ER) diagram provides a high-level view of the system in terms of entities, relationships, and attributes. To read the roles and cardinalities between relationships/entities, I have followed the convention such that anything above a line should be read left to right, and anything below the line right to left. For vertical lines, anything right of the line should be read from top to bottom, and anything left of the line bottom to top. A single line represents partial participation, and a double line represents total participation.

Performer, Presenter, and Voter all have a many-to-one relationship with Academy Award Show. This translates to each award show having multiple performers, presenters, and voters. Additionally, there is total participation on the Performer/Presenter/Voter side as none of these entities would exist if the Academy Award Show did not exist.

There is a many-to-many relationship between both Voter and Nominee. This translates to a voter being able to vote for multiple nominees across several categories and a nominee being voted for by multiple voters. Nominee and Film also has a many-to-many relationship as a nominee can be nominated for multiple awards for different films, and a film having multiple individuals who worked on that film nominated. More details of the constraints associated with these relationships will be discussed when looking at the relational schema.

Mapping to Relational Schema

This step represents a move from the conceptual ER model to a logical database design. I used the rules covered in the lecture material to map the one-to-many and many-to-many relationships to the relational schema. For the many-to-many relationships, new tables are created (Vote and Nomination). An arrow is drawn from a foreign key (in bold) to its corresponding primary key (underlined).

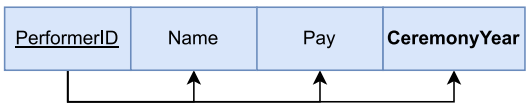


Underlined Attributes = Primary Key

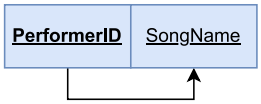
Bold Attributes = Foreign Key

Functional Dependency Diagrams

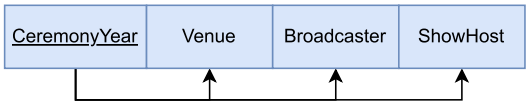
PERFORMER



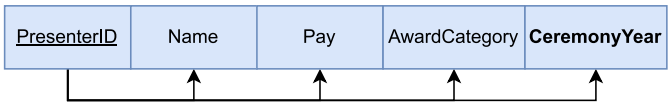
SONG



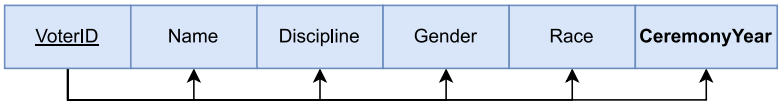
ACADEMY AWARD SHOW



PRESENTER

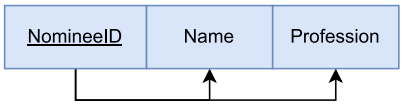


VOTER

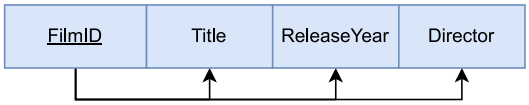


VOTE

NOMINEE

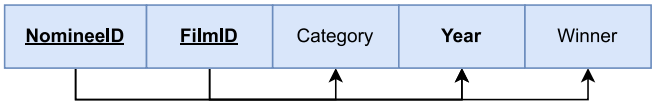


FILM



GENRE

NOMINATION



Underlined Attributes = Primary Key

Bold Attributes = Foreign Key

This diagram shows the functional dependencies between the attributes in a particular table, and which attributes determine what. By following the design guidelines outlined in lecture materials, this database has been designed so that it is in Boyce-Codd Normal Form (i.e all attributes in the relation should be dependent on the key, and nothing but the key).

Assumptions & Further Discussion

- There is only one Academy Award show every year.
- Each performer can perform more than one song.
- More than one performer can perform the same song (e.g a duet).
- For a film to be in the database it must be nominated for something.
- A film can only be nominated in one Oscar show and the film's release year must be the year before the ceremony year.
- An actor *cannot* be nominated more than once in the *same* category (e.g an actor cannot be nominated twice for Best Actor for Film X and Film Y)[4].
- An actor *cannot* be nominated more than once (across different categories) for the *same* performance (e.g an actor cannot be nominated for Best Actor and Best Supporting Actor for their performance in Film X)[5].
- An actor *can* be nominated more than once (across different categories) for *different* performances (e.g an actor can be nominated for Best Actor and Best Supporting Actor for their performance in Film X and Film Y respectively).

Vote Relation

The primary key for Vote is made up of all three attributes: VoterID, NomineeID, and Category. There are several reasons all three attributes are needed to make up the primary key. The VoterID and NomineeID are not enough as a voter can vote for the same nominee across multiple categories (e.g if they are nominated for Best Actor and Best Supporting Actor for *different* films). VoterID and Category are not enough as a voter can vote for multiple

[4] This assumption is based on the official Academy rules. Source: https://www.oscars.org/sites/oscars/files/94aa_rules.pdf
[5] See footnote 4.

nominees in the same category[6]. Finally, NomineeID and Category are also not enough to be the primary key as many different voters could vote for the same nominee-category combination.

Nomination Relation

This relation was initially mapped so that NomineeID, FilmID, and Category all made up the primary key. However, upon learning the rules associated with multiple nominations (see the last three assumptions listed above), I have updated my model so that only the NomineeID and FilmID make up the primary key. The category no longer needs to be part of the composite primary key because a nominee can only ever be nominated once for their performance in a particular film (i.e NomineeID and FilmID will always be unique and determine the other attributes). These constraints are upheld in the SQL code below.

Section B: Explanation of Data and SQL Code

Creating Database Tables

```
CREATE TABLE performer (
    performer_id INTEGER NOT NULL AUTO_INCREMENT,
    performer_name VARCHAR(255) NOT NULL,
    pay INTEGER NOT NULL,
    ceremony_year YEAR NOT NULL,
    PRIMARY KEY (performer_id),
    FOREIGN KEY (ceremony_year)
          REFERENCES academy_award_show(ceremony_year),
    CHECK (pay > 0)
);
```

The CREATE TABLE statement is used in SQL is create a new table in the database. The above creates a new table called “performer” with the listed attributes. The highlighted keyword after the attribute name indicates the type of that attribute (e.g “performer_id” must be an integer). The AUTO_INCREMENT field generates a new number for “performer_id” when a

[6] The Academy uses preferential voting for Best Picture as it has more contenders and rather than voting for one person per category, the nominees are ranked.

row is inserted into the table. This means the “performer_id” (primary key) is automatically generated and increments after each insertion. This makes the code more reliable and less prone to user error.

This table also contains several constraints. The NOT NULL constraint ensures that a particular column can not contain NULL values. The PRIMARY KEY constraint is used to identify the primary key of the table and therefore must have UNIQUE values, that cannot be NULL. The FOREIGN KEY constraint prevents the user from accidentally destroying links between tables and prevents data that isn’t contained in the parent table to be inserted. Finally, there is a CHECK constraint which ensures that any value inserted into the “pay” column must be greater than 0. This models the real world where someone cannot be paid a negative amount of money, or nothing.

Altering Tables

```
ALTER TABLE performer
ADD CHECK (pay < 1000000);

ALTER TABLE presenter
ADD CHECK (pay < 1000000);
```

The ALTER TABLE statement is used to make changes to an existing table. In the above example, an extra constraint is being added to the “performer” table. The table currently has a check that ensures “pay” is greater than zero but after executing the above code, another constraint will be added ensuring “pay” is less than a million.

```
ALTER TABLE film
ADD producer VARCHAR(255);

-- Dropping it so it the tables align
-- with ER model
ALTER TABLE film
DROP COLUMN producer;
```

Currently the “film” table only stores the director’s name but, for example, if you wanted to add a column for the producer’s name it could be done with the code above. The DROP

command is used afterwards to remove it once again, so that my tables align with the data model I've created (as adding the "producer" column was just for demonstration purposes).

Trigger Operations

A trigger is a sequence of commands that runs if a certain event occurs. Although some constraints were included when creating the database tables, I have used triggers to implement further checks.

```
DELIMITER $$

CREATE TRIGGER category_check
BEFORE INSERT
    ON nomination FOR EACH ROW
    BEGIN
        IF EXISTS (SELECT * FROM nomination
            WHERE (nominee_id = NEW.nominee_id)
            AND (category = NEW.category)
            AND (nomination_year = NEW.nomination_year)) THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'The same nominee cannot be nominated
                  more than once in the same category (in the same year).';
        END IF;
    END $$


CREATE TRIGGER film_check
BEFORE INSERT
    ON nomination FOR EACH ROW
    BEGIN
        IF EXISTS (SELECT * FROM nomination
            WHERE (nominee_id = NEW.nominee_id)
            AND (film_id = NEW.film_id)) THEN
            SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'The same nominee cannot be nominated
                  more than once (across different categories) for the same
                  performance/film.';
        END IF;
    END $$

DELIMITER ;
```

The first trigger defined above ensures that the same nominee cannot be nominated more than once in the same category. This trigger performs the check before an INSERT statement is executed and will only insert the data if the trigger is executed without throwing an error.

It checks if the new tuple being added has the same “nominee_id”, “category”, and “nomination_year” as a row already in the “nomination” table. If a tuple already exists in the table with the same values, a SQL error is thrown with the indicated message and the insert fails.

The second trigger ensures that the same nominee cannot be nominated more than once (across different) categories for the same performance (film). It performs its checks in a similar way to the first trigger but instead checks there is not a row in the table with the same “nominee_id” and “film_id” to the tuple being added (the “nomination_year” isn’t needed here because a film will never be nominated in more than one year).

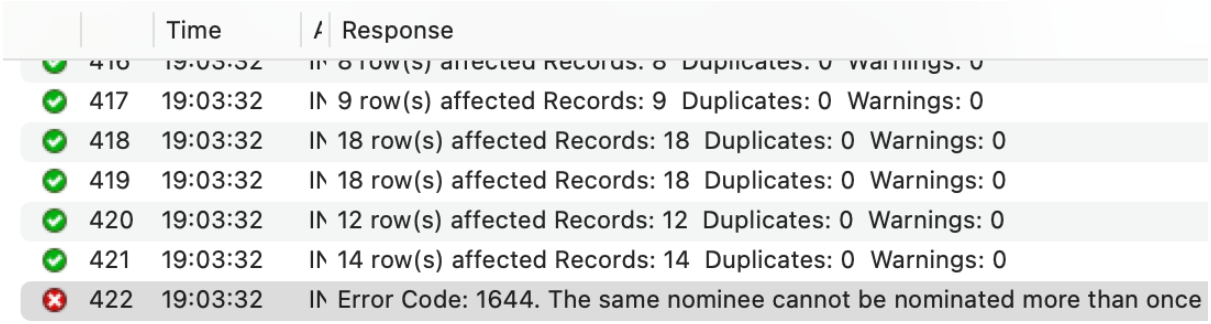


Figure 1: The error message shown in MySQL Workbench when the user tries to insert a nominee more than once with the same category.

The purpose of the DELIMETER $$ statement is to temporarily change the delimiter to “$$”. The delimiter needs to be changed when writing a trigger statement so that we can use “;” to define multiple statements inside the trigger without telling SQL it is the end of the statement. The delimiter is then changed back to its default after we have defined all our trigger statements using DELIMITER ;.

Creation of Views

A view is like a virtual table. Statements and functions can be defined within a view to specify the data we want from one or more tables. The view can then be queried as if it was its own table.

```
CREATE VIEW best_actress_nominees_2021 (nominee, film, winner) AS
SELECT nominee.nominee_name, film.title, nomination.winner
FROM nominee, film, nomination
WHERE (nominee.nominee_id = nomination.nominee_id)
    AND (film.film_id = nomination.film_id)
    AND (nomination.category = "Best Actress")
    AND (nomination.nomination_year = "2021");
```

The above code creates a view that returns all the nominees for Best Actress in 2021. Running the query `SELECT * FROM academy_awards.best_actress_nominees_2021;` will return the table below.

nominee	film	winner
Frances McDormand	Nomadland	1
Viola David	Ma Rainey's Black Bottom	0
Carey Mulligan	Promising Young Woman	0

Figure 2: The nominees for Best Actress in 2021 with the winner indicated.

Note that there are usually five nominees for Best Actress but for simplicity I only entered three nominees per category. The “1” in the “winner” column indicates that Frances McDormand won Best Actress in 2021 for her performance in Nomadland. The full SQL code submitted alongside this contains more views to return the nominees for other categories.

```
CREATE VIEW performer_payroll_2022 (id, pay_amount) AS
SELECT performer.performer_id, performer.pay
FROM performer
WHERE (performer.ceremony_year = "2022");
```

Another application of views in my system is returning the amount each performer needs to be paid for a particular show (year). Running the query `SELECT * FROM academy_awards.performer_payroll_2022;` returns the table below, which is essentially the overall performers’ payroll for the year 2022. The “id” refers to the “performer_id”.

id	pay_amount
1	11111
2	10000
3	12000
4	9000

Figure 3: The amount each performer is to be paid for their performance at the 2022 Oscars.

A similar payroll view has been generated for the presenters of the 2022 Oscars which can be found in the full SQL code.

Populating a Table

```
INSERT INTO film (title, release_year, director)
VALUES
    ("Being the Ricardos", 2021, "Aaron Sorkin"),
    ("The Power of the Dog", 2021, "Jane Campion"),
    ("tick, tick… BOOM!", 2021, "Lin-Manuel Miranda"),
    ("The Lost Daughter", 2021, "Maggie Gyllenhaal"),
    ("Spencer", 2021, "Pablo Larrain"),
    ("West Side Story", 2021, "Steven Spielberg"),
    ("Belfast", 2021, "Kenneth Branagh"),

    ("The Father", 2020, "Fiorian Zeller"),
    ("Mank", 2020, "David Fincher"),
    ("Ma Rainey's Black Bottom", 2020, "George C. Wolfe"),
    ("Nomadland", 2020, "Chloe Zhao"),
    ("Promising Young Woman", 2020, "Emerald Fennell");
```

The INSERT INTO statement inserts a record into the specified table. The above code inserts the movies nominated for an award that were released in 2021 and 2020. The “film_id” is not inserted here because, as mentioned, that column is auto incremented and generated automatically. Being the Ricardos is automatically assigned a “film_id” of 1, The Power of the Dog is assigned a “film_id” of 2, and so on. The film entity in the ER model also had a genre attribute but as this is a multivalued attribute it is stored in another table. Any film that is nominated for an award (i.e is referenced in the “nomination” table) must be inserted into the “film” table first, as the “nomination” table contains “film_id” which is a foreign key of the “film_id” in the “film” table.

Retrieving Information from Database

There are endless combinations of data you can pull from the database. I've chosen to implement several SELECT statements that I believe are some of the more popular queries that people would be looking at in the run up to Oscar season.

```
SELECT nomination.category, nominee.nominee_name, film.title,
     nomination.winner
FROM nomination
INNER JOIN film
    ON nomination.film_id = film.film_id
INNER JOIN nominee
    ON nomination.nominee_id = nominee.nominee_id
WHERE nomination.nomination_year = "2021"
ORDER BY nomination.category;
```

This first SELECT statement will return all nominees for a particular year (here 2021), sorted by category. As soon as Oscar nominations are announced, this is most likely the one query most people are interested in. Because the above query is for a previous year, the "winner" column is populated with the data indicating whether a nominee won the award or not. Also note for simplicity I have only entered data for a few categories and nominees.

The INNER JOIN keyword allows you to query across more than one table by combining rows that have matching values. Here those matching values are "film_id" and "nominee_id". This allows you to get all data associated with a film from the "film" relation and all data associated with a nominee from the "nominee" relation using the "film_id" and "nominee_id" in the "nomination" table.

category	nominee_name	title	winner
Best Actor	Anthony Hopkins	The Father	1
Best Actor	Gary Oldman	Mank	0
Best Actor	Chadwick Boseman	Ma Rainey's Black Bottom	0
Best Actress	Frances McDormand	Nomadland	1
Best Actress	Viola David	Ma Rainey's Black Bottom	0
Best Actress	Carey Mulligan	Promising Young Woman	0
Best Director	Chloe Zhao	Nomadland	0
Best Director	Emerald Fennell	Promising Young Woman	1
Best Director	David Fincher	Mank	0
Best Supporting Actress	Olivia Colman	The Father	0

Figure 4: All nominations from 2021 sorted by category.

```
SELECT nomination.nomination_year, nomination.category, film.title,
     nomination.winner
FROM nomination
INNER JOIN film
    ON nomination.film_id = film.film_id
INNER JOIN nominee
    ON nomination.nominee_id = nominee.nominee_id
WHERE nominee.nominee_name = "Olivia Colman"
ORDER BY nomination.nomination_year DESC;
```

Another useful query is to retrieve all nominations for an individual, spanning all years. For example, Meryl Streep holds the record with 21 nominations, and if the database was filled with complete Oscar nomination data, the user could quickly see all films she has been nominated for, when she was nominated, and if she won.

nomination_ye...	category	title	winner
2022	Best Actress	The Lost Daughter	NULL
2021	Best Supporting Actress	The Father	0

Figure 5: All nominations for Olivia Colman in the database. Note that the 2022 ceremony has not yet taken place so the "winner" value is set to NULL.

A query similar to the one above can also be written to return all the nominations for a particular film. For example, the table shows all the nominations for Ma Rainey's Black Bottom (full SELECT statement is in the SQL file).

category	nominee_name	winner
Best Actor	Chadwick Boseman	0
Best Actress	Viola David	0

Figure 6: All nominations for Ma Rainey's Black Bottom in the database.

```
SELECT genre.genre, COUNT(*) AS count
FROM genre
GROUP BY genre
ORDER BY count DESC;
```

SQL also has built-in functions, such as COUNT, that can allow for analysis. For example, you can count how many nominations each genre of film has, and what genre the Oscars favour.

genre	count
Drama	4
Biographical	3
Musical	2
Thriller	2
Western	1
Romance	1
Historical	1

Figure 7: The genres of all the films in the database, sorted by how many nominations films of that genre have gotten.

```
SELECT voter.discipline,
    SUM(CASE WHEN voter.gender = "Male" THEN 1
      ELSE 0 END)/COUNT(*)*100 male_percentage,
    SUM(CASE WHEN voter.gender = "Female" THEN 1
      ELSE 0 END)/COUNT(*)*100 female_percentage
FROM voter
GROUP BY voter.discipline;
```

Finally, I have implemented a SELECT statement to give a breakdown of the demographic of voters in the Academy. The statement above counts the number of male and female voters and turns it into a percentage of the total number of voters to display to the user.

discipline	male_percentage	female_percentage
Production	66.6667	33.3333
Acting	75.0000	25.0000
Sound	50.0000	50.0000

Figure 8: A gender percentage breakdown of voters, sorted by the discipline (or branch) they belong to.

Another similar SELECT statement can be found in the full SQL file which gives a breakdown of voters by race.

Security Commands

```
-- Create roles
DROP ROLE IF EXISTS 'awards_admin', 'awards_read', 'awards_write',
      'public';
CREATE ROLE 'awards_admin', 'awards_read', 'awards_write', 'public';

-- Grant roles relevant privileges
GRANT ALL ON academy_awards.* TO 'awards_admin';
GRANT SELECT ON academy_awards.* TO 'awards_read';
GRANT INSERT, UPDATE, DELETE ON academy_awards.* TO 'awards_write';
GRANT SELECT ON academy_award_show TO 'public';
GRANT SELECT ON film TO 'public';
GRANT SELECT ON genre TO 'public';
GRANT SELECT ON nomination TO 'public';
GRANT SELECT ON nominee TO 'public';

-- Create sample users
DROP ROLE IF EXISTS 'jdoe'@'localhost', 'mconnor'@'localhost',
      'adoherty'@'localhost', 'dsmith'@'localhost';
CREATE USER 'jdoe'@'localhost' IDENTIFIED BY 'password123';
CREATE USER 'mconnor'@'localhost' IDENTIFIED BY 'Password!';
CREATE USER 'adoherty'@'localhost' IDENTIFIED BY 'my_password';
CREATE USER 'dsmith'@'localhost' IDENTIFIED BY 'pASSWORD';

-- Assign users roles/privileges
GRANT 'awards_admin' TO 'jdoe'@'localhost';
GRANT 'awards_read', 'awards_write' TO 'mconnor'@'localhost';
GRANT 'awards_read' TO 'adoherty'@'localhost';
GRANT 'public' TO 'dsmith'@'localhost';
```

SQL allows you to define roles and privileges for different types of users. I have chosen to create four roles, an admin, read-only, write-only, and public role. The first three roles are self-explanatory. Any users assigned the public role will only be able to view publicly available information such as nominated films and nominees. Information on performers and presenters (such as their salaries), as well as details on voters are not viewable to the public.

Sample users are created to illustrate its use case. For example, “jdoe” is a database administrator who needs full control over the database to do his job. The user “mconnor”, is an individual on the board of the Academy who has permission to read and write all data. The user “adoherty” is a manager who needs to be able to read all data about presenters, performers, and nominees, but does not have the authority to change any data. Finally, “dsmith” is not given access to any sensitive information and can only read data that is publicly available.