Team 1 Project Report

Team Members:

Brian Duffy

Ben Barasch

Ethan Fine

Sara Wenning

CSE 3241

Professor Madrid

Au2021

GitHub repo: github.com/bbarasch/CSE3421-DB-Project

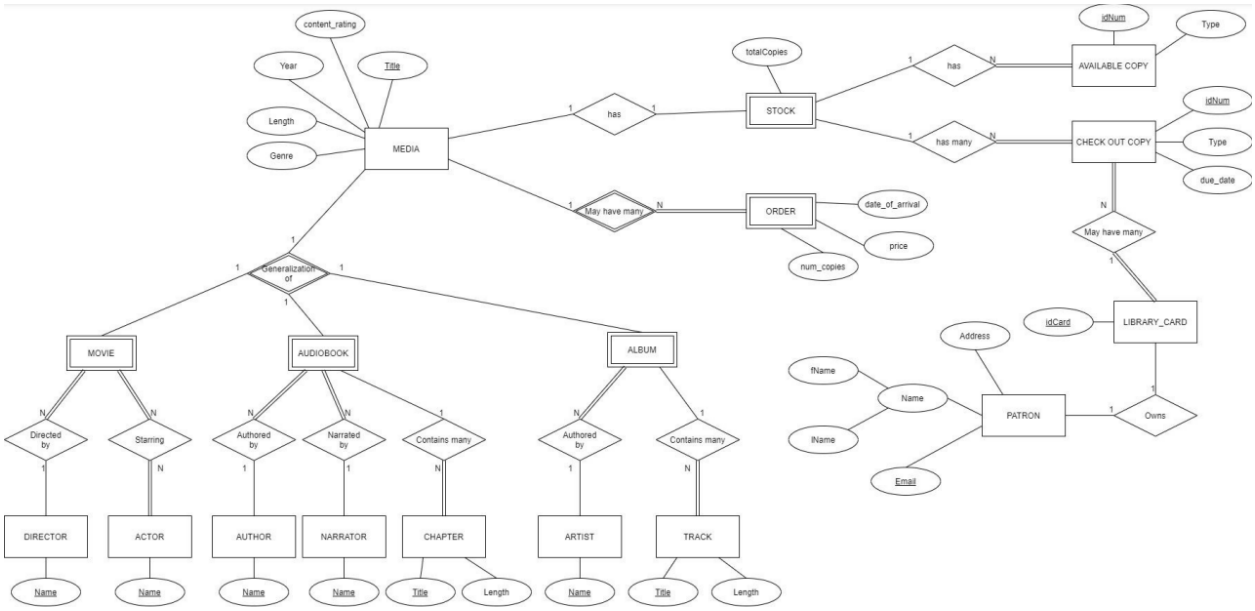
Table of Contents

Section 1. Database Description

ER Model

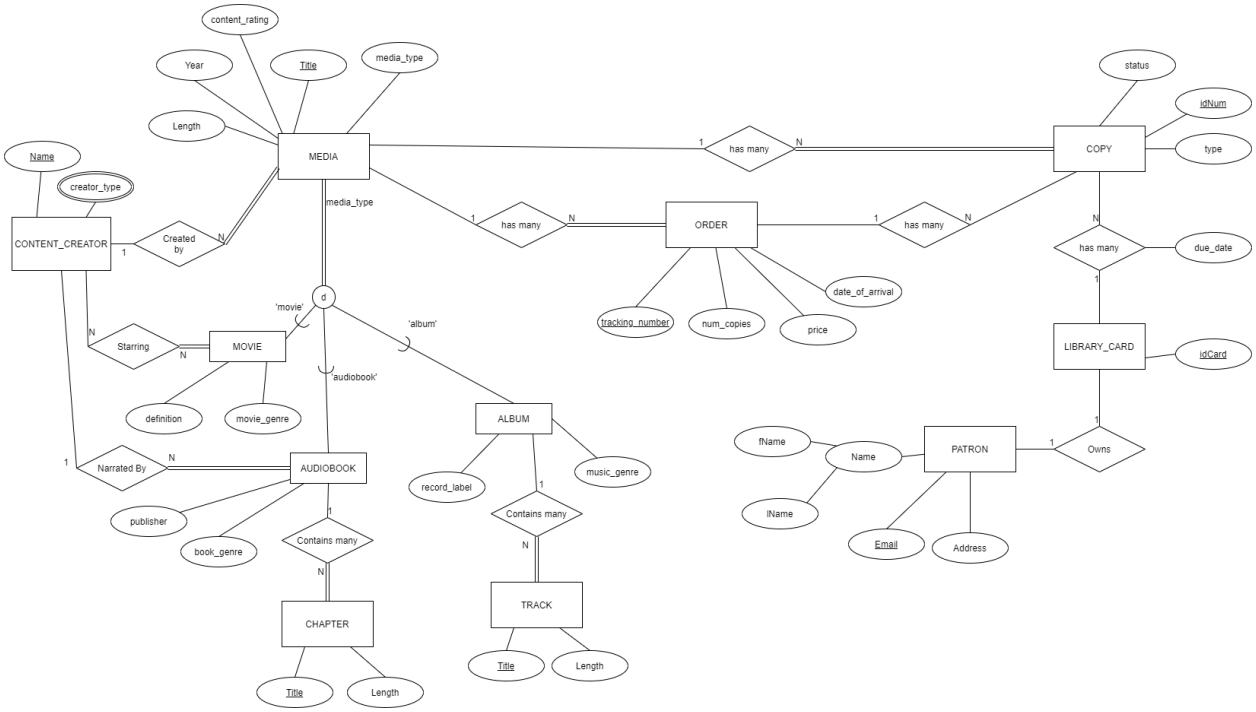
Checkpoint 1

The checkpoint 1 diagram was created before learning about disjoint subclasses and EER Design principles, as such we used a ternary relationship between MEDIA and the subclasses of MOVIE, AUDIOBOOK, and ALBUM. In addition, we had an object of stock that tracked the copies that existed for each piece of media, as well as dividing the copies into physical and digital copies as different relations.



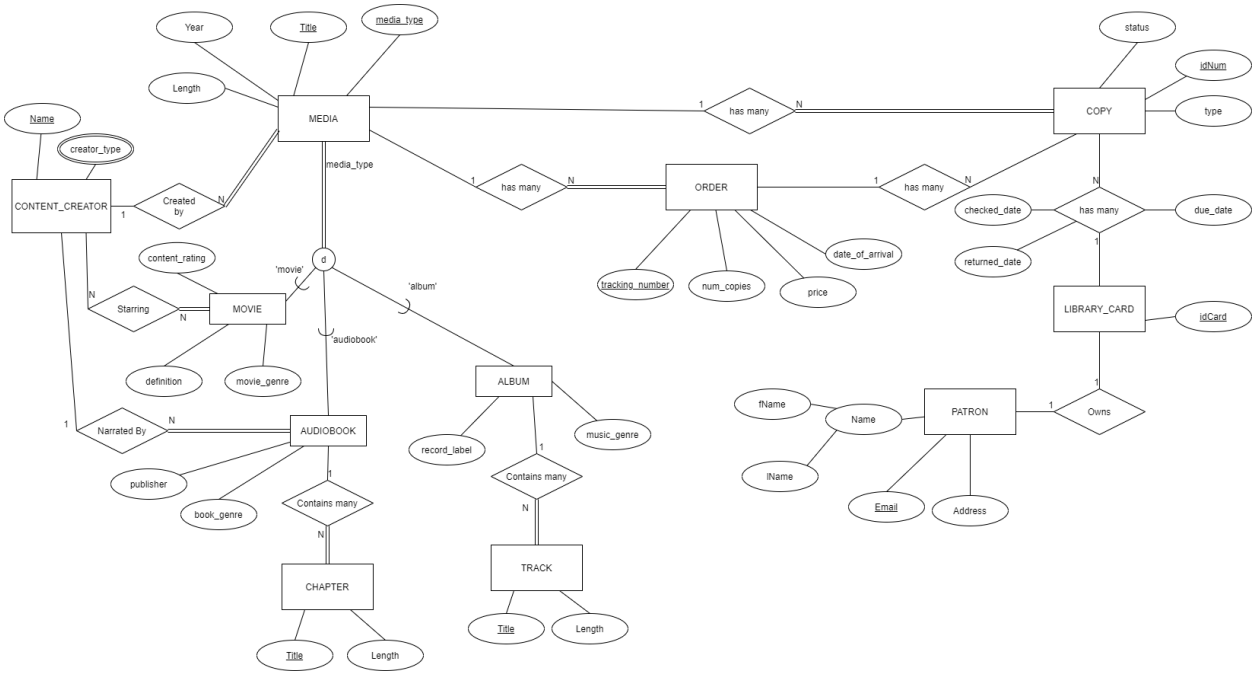
Checkpoint 2-4

The ER diagram did not change between checkpoint 2 and checkpoint 4. The major changes were going from a ternary relation to a disjoint relation for MEDIA and its subclasses. Instead of having 5 different relations for ARTIST, AUTHOR, NARRATOR, DIRECTOR, and ACTOR, a unifying relation was created named CONTENT_CREATOR, which has a creator_type attribute that allows artists to contribute to multiple different media types. In addition, STOCK was removed as it was no longer necessary, and the two types of copies were merged and an attribute of type (digital/physical) was added to replace that. Definition and movie_genre was added as an attribute to MOVIE, publisher and book_genre was added as an attribute to AUDIOBOOK, and record_label and music_genre were added as attributes to ALBUM.



Checkpoint 5

Major changes in checkpoint 5 were making media_type a part of the primary key of MEDIA, since media from different subclasses can share the same name (for example, Dune the movie and Dune the book). In addition, the return date is now tracked for copies, and content_rating is now an exclusive MOVIE attribute.



Relational Schema

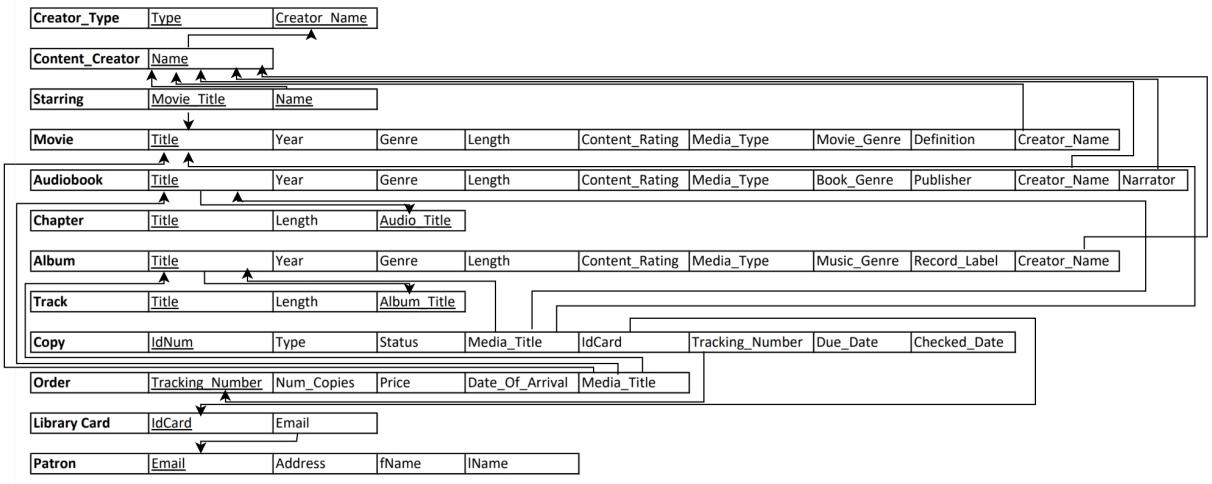


Figure 4: Relational Diagram for Checkpoints 3-4

For checkpoints 3 and 4 the relation diagram shown in Figure 4 was used as a reference to start building the database. Checkpoint 5 revealed some flaws in the design of the relational diagram, so the Checked_out entity was added as well as the Media entity for better data lookup and history tracking. The result led to the relational diagram shown below in Figure 5.

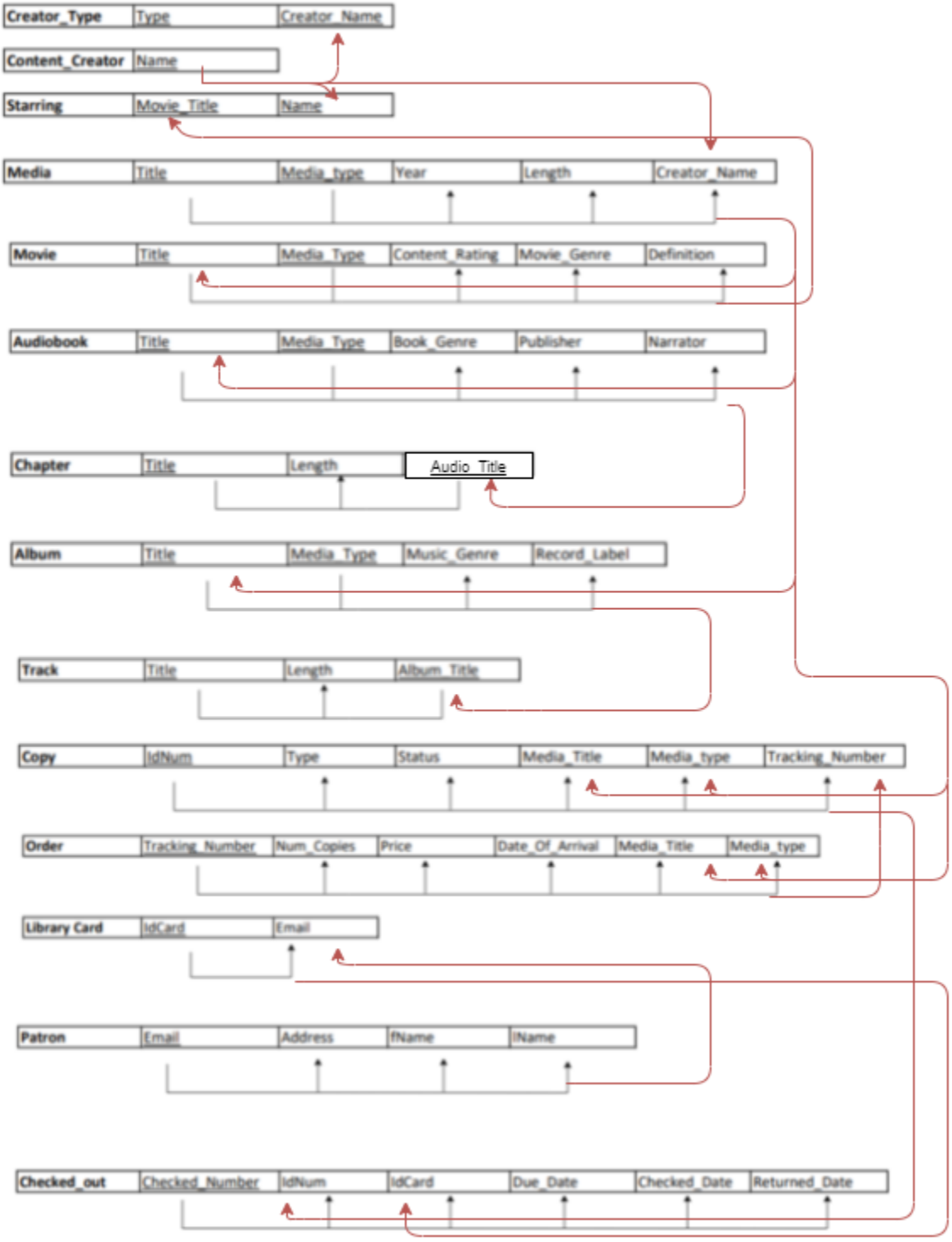


Figure 5: Relational Diagram for Checkpoint 5

Normalization

Album

Album does not have any multi-valued attributes or compound attributes. In the Album table, the music_genre and record_label are fully dependent on the two primary keys, title and media_type, and there are no other dependencies in the table, therefore the table is in BCNF.

Audiobook

Audiobook does not have any multi-valued attributes or compound attributes. In the Audiobook table, book_genre, publisher, and narrator are fully dependent on the primary keys, title and media_type, and there are no more dependencies in the table, therefore the table is in BCNF.

Library_Card

Library_Card does not have any multi-valued attributes or compound attributes. In the Libray_Card table, email is fully dependent on the primary key, Id_Card, and there are no other dependencies, therefore the table is in BCNF.

Chapter

Chapter does not have any multi-valued attributes or compound attributes. In the Chapter table, the length is fully dependent on the two primary keys, title and album_title, and there are no other dependencies, therefore the table is in BCNF.

Checked_Out

Checked_out does not have any multi-valued attributes or compound attributes. In the table, IdNum, due_date, checked_date, and returned_date are fully dependent on the primary key, checked_number, and there are no more dependencies, therefore the table is in BCNF.

Content_creator

Content_creator only has one attribute, Name. Name is not a compound or multi-valued attribute, therefore the table is in BCNF.

Copy

Copy does not have any multi-valued attributes or compound attributes. In the table, type, status, media_title, media_type, and tracking_number are all fully dependent on the primary key, IdNum, and there are no other dependencies, therefore the table is in BCNF.

Creator_Type

Creator_Type does not have any multi-valued attributes or compound attributes. It only consists of two primary keys, type and creator_name, with no dependencies, therefore it is in BCNF.

Media

Media does not have any multi-valued attributes or compound attributes. In the table, year, length, and creator_name are fully dependent on the two primary keys, title and media_type, and there are no other dependencies, therefore the table is in BCNF.

Movie

Movie does not have any multi-valued attributes or compound attributes. In the table, content_rating, movie_genre, and definition are fully dependent on the two primary keys, title and media_type, and there are no more dependencies, therefore the table is in BCNF.

Delivery_Order

Delivery_Order does not have any multi-valued attributes or compound attributes. In the table, media_type, mediea_title, date_of_arrival, price, and num_copies are fully dependent on the primary key, tracking_number, and there are no other dependencies, therefore the table is in BCNF.

Patron

Patron does not have any multi-valued attributes or compound attributes. In the table, lName, fName, and address are fully dependent on the primary key, email, and there are no other dependencies, therefore the table is in BCNF.

Starring

Starring does not have any multi-valued attributes or compound attributes. There are only two primary keys and no dependencies, therefore the table is in BCNF.

Track

Track does not have any multi-valued attributes or compound attributes. In the table, length is fully dependent on the primary keys, title and album_title, and there are no dependencies, therefore the table is in BCNF.

Indexes

Index 1: idx_media_title

This index is defined on the Title attribute of the MEDIA table.
Since all three types of media (Audiobook, Movie, Album) are all specializations of the Media type, then it is often searched in equality checks. As such, it was deemed appropriate to create an index for this attribute.

Index 2: idx_checked_out_card_id

This index is defined on the Card_Id attribute of the CHECKED_OUT table.
This table is the common group between patrons and the media/copies, since it tracks what copies have been checked out and when. As such, we have noticed that in our queries, we often require checking for equality between the ID of the library card with the IDs of the checked out media cards.

Index 3: idx_checked_out_copy_id

This index is defined on the Copy_Id attribute of the CHECKED_OUT table.
Similarly to the index above, this index serves to connect between the copies of media owned to their history of checkouts. Since we often check this history in regards to copies of media, we deemed it appropriate to create an index for said attribute.

Views

View 1: OverdueCounts

Description: Get the counts of items that are overdue for patrons
Relational Algebra:

$$T1 \leftarrow PATRON \bowtie_{Email=Email} LIBRARYCARD$$

$$T2 \leftarrow T1 \bowtie_{IdCard=CardId} CHECKEDOUT$$

$$T3 \leftarrow \sigma_{DueDate < CurrentDate}(T2)$$

$$RESULT \leftarrow \pi_{Fname,\ Lname,\ DueDateCount}({}_{Email}\mathcal{F}_{DueDate}(T3))$$

SQL Query:

```
CREATE VIEW OverdueCounts AS
SELECT P.Fname, P.Lname, COUNT(Q.Due_date) as Overdue
FROM CHECKED_OUT as Q, LIBRARY_CARD as L, PATRON as P
WHERE P.Email = L.Email AND L.Id_card = Q.Card_id AND Q.Due_date <
CURRENT_DATE
GROUP BY L.Email;
```

Sample Output:

	Fname	Lname	Overdue
1	Jane	Doe	2
2	John	Doe	2
3	John	Doe	2
4	John	Smith	3
5	Jane	Smith	2

View 2: ArtistAverage

Description: List the average length of a track by recording artist
Relational Algebra:

$$T1 \leftarrow MEDIA \bowtie_{Title=Title} ALBUM$$

$$T2 \leftarrow T1 \bowtie_{Title=AlbumTitle} TRACK$$

$$RESULT \leftarrow \pi_{CreatorName,\ AvgLength}({}_{CreatorName}\mathcal{F}_{AVERAGE(Length)}(T2))$$

SQL Query:

```
CREATE VIEW ArtistAverage AS
SELECT M.Creator_name, AVG(T.Length) as TrackAverage
FROM MEDIA as M, ALBUM as A, TRACK as T
WHERE M.Title = A.Title AND M.Title = T.Album_title
GROUP BY M.Creator_name;
```

Sample Output:

	Creator_name	TrackAverage
1	Billie Eilish	229.5
2	Bo Burnham	249
3	Clairo	189.3333333333334
4	Conan Gray	237
5	Harry Styles	220
6	Justin Bieber	205.5
7	Lin-Manuel Miranda	262
8	Olivia Rodrigo	196.75
9	Taylor Swift	195.4
10	Tim McGraw	246.5

Transactions

Transaction 1: Return_Items

Description: Returns all copies of any item checked out by a particular patron.
SQL code:

```
BEGIN TRANSACTION;

UPDATE COPY
SET Status = "Available"
WHERE Id_copy IN
(SELECT C.Id_copy
FROM COPY as C, CHECKED_OUT as O, LIBRARY_CARD as L
WHERE L.Email = "johndoe2@gmail.com" AND L.Id_card = O.Card_Id AND
O.Copy_Id = C.Id_copy);

UPDATE CHECKED_OUT
SET Returned_Date = "11/29/2021"
WHERE Copy_Id IN
(SELECT O.Copy_Id
FROM COPY as C, CHECKED_OUT as O, LIBRARY_CARD as L
WHERE L.Email = "johndoe2@gmail.com" AND L.Id_card = O.Card_Id AND
O.Copy_Id = C.Id_copy);

COMMIT;
```

Transaction 2: New_Card

Description: Gives a patron a new library card, and deletes the old one.
SQL code:

```
BEGIN TRANSACTION;
DELETE FROM LIBRARY_CARD
WHERE Email = "john.smith@gmail.com";

INSERT INTO LIBRARY_CARD
VALUES(201, "john.smith@gmail.com");

COMMIT;
```

Transaction 3: Change_Price

Description: Changes the price per copy of all orders due on 10/30 because of a previously unknown fee. The example for this assumes that each copy is now an additional $2 in price.
SQL code:

```
BEGIN TRANSACTION;
```

```sql
UPDATE DELIVERY_ORDER
SET Price = Price + 2 * Num_Copies
WHERE Date_of_Arrival = "10/30/2021";

COMMIT;
```

Section 2. User Manual

Table Explanations

Album

The Album table represents any albums the library may have in stock, will be receiving, or had in stock at one point. All keys are required to not be null. The primary key, title, is used to distinguish each album. It is a foreign key that references the media table. The title refers to the title of the album, and it is a varchar(30). Media_type is also a foreign key that references the media table. It is a varchar(15) that describes the type of media, in this case it would be “Album.” Music_genre is a key of type varchar(15) that describes the genre of the album. Record label is a key of type varchar(15) that describes the record label associated with the album.

Audiobook

The Audiobook table represents any audiobooks the library may have in stock, will be receiving, or had in stock at one point. All keys besides the narrator key are required to not be null. Narrator may be null since not all narrators are named on the recording. The primary key, title, is used to distinguish each audiobook. It is a foreign key that references the media table. The title refers to the title of the audiobook, and it is a varchar(30). Media_type is also a foreign key that references the media table. It is a varchar(15) that describes the type of media, in this case it would be “Audiobook.” Narrator is a foreign key of type varchar(30) that references the Content_Creator table which describes who narrates the audiobook. Book_genre is a key of type varchar(15) that describes the genre of the audiobook. Publisher is a key of type varchar(15) that describes what company published the audiobook.

Library_Card

The Libray_Card table represents any library_cards that have ever been activated at the library. Both keys are required to be not null. The primary key, Id_Card, is of type int, and it is a unique id number given to each card to find it in the system. The email key is a foreign key of type varchar(45) that references the patron table. It describes the email that is associated with a specific card.

Chapter

The Chapter table represents a chapter that is contained within an audiobook. All keys are required to not be null. There are two primary keys, title and audiobook_title, that are both of type varchar(30). Title represents the title of the chapter within the audiobook. Audiobook_title is a foreign key that references the Audiobook table which represents the title of the audiobook that contains the chapter. The length key is of type double and represents the length of the chapter in minutes.

Checked_Out

The Checked_Out table represents the history of media items that have been checked out at the library. All keys besides returned_date are required to not be null. Returned_date may be null if the media item has not been returned yet. The primary key, checked_number, is of type int, and it is a unique identifier for the check_out. Copy_Id is a foreign key of type int that references the Copy table. It represents the id number of the copy of the media item that was checked out. Card_Id is a foreign key of type int that references the Card table. It represents the id number of the patron that checked out the media item. Due_date is a key of type date that represents the date the media item was due. Checked_date is a key of type date that represents the date the media item was checked out. Returned_date is a key of type date that represents the date the media item was returned.

Content_creator

The Content_creator table represents any content creators within the library's system. It has one key, Name, that is of type varchar(30) which represents the name of the content creator. This key is required to not be null.

Copy

The Copy table represents all copies of media that have existed or will exist in the libraries system. All keys besides tracking_num are required to not be null. The library could have received the media items through means other than shipping, so a tracking number may not exist for all media copies. The primary key, Id_copy, is of type int and represents a unique identifier for the copy of the media. The key Type is of type varchar(10), and it represents if the specific media item is a physical or digital copy. The key Status is of type varchar(10), and it represents if the media copy is available, unavailable, or in transit. Media_title and Media_type are foreign keys of type varchar(15) that reference the Media table. Media_title represents the title of the copy of media, while Media_type represents the type of media of the copy. Tracking_num is a foreign key of type long that references the Delivery_Order table and represents the tracking number associated with the shipment of the copy.

Creator_Type

The Creator_Type table represents each creator along with what type of content they create. Since each creator may create more than one type of content, this had to be separate from the Content_Creator table. There are two primary keys, Type and Creator_name, that are both required to not be null. Type is of type varchar(15) and represents whether the creator is a director, actor, author, narrator, or musician. Creator_name is a foreign key of type varchar(30) that references the Content_Creator table and represents the name of the creator.

Media

The Media table represents the media items that are present within the library's system. All keys are required to not be null. There are two primary keys, title and media_type. Title is of

type varchar(30) and represents the title of the media item. Media_type is of type varchar(15) and represents whether the media item is a movie, album, or audiobook since the titles of movies, audiobooks, and albums may overlap. Creator_name is a foreign key that references the Content_Creator table and represents the name of the creator of the media item. Year is a key of type int that represents the year the media item was created. Length is a key of type double that represents the length of the media item in minutes.

Movie

The Movie table represents the movies that are present within the library's system. All keys are required to not be null except for content_rating since not all movies have content ratings. The primary key, title, is a foreign key of type varchar(30) that references the Media table and represents the title of the movie. Media_type is a foreign key of type varchar(15) that references the Media table and represents the type of media. Movie_genre is a key of type varchar(15) that represents the genre of the movie. Content_rating is a key of type varchar(15) that represents the content rating of the movie, such as PG or R. Definition is a key of type varchar(15) that represents the definition of the movie, such as HD or Blu-Ray.

Delivery_Order

The Delivery_Order table represents any orders of media items that were placed for delivery to the library. All keys are required to not be null. The primary key, Tracking_Number, is a key of type long that represents the tracking order of the shipment. Num_copies is a key of type int that represents how many copies the shipment contains. Price is a key of type double that represents the cost of the shipment. Date_of_arrival is a key of type date that represents the date the shipment is scheduled to or did arrive. Media_title is a foreign key of type varchar(15) that references the title of the media item in the shipment. Media_type is a foreign of type varchar(15) that references the Media table and represents the type of media in the shipment.

Patron

The Patron table represents any patrons within the library's system. All keys are required to not be null. The primary key, email, is of type varchar(45) and represents the email address of the patron associated with their account. The address key is of type varchar(45) and represents the address of the patron. The fName key is of type varchar(15) and represents the first name of the patron. The lName key is of type varchar(15) and represents the last name of the patron.

Starring

The Starring table represents the relationship between the Content_Creator table and the Movie table since actors can work on many movies and movies can have many actors. Movie_title and Actor_name are both primary keys of type varchar(30). Movie_title is a foreign key that references the Movie table and represents the title of the movie. Actor_name is a foreign key that references the Content_Creator table and represents the name of the actor starring the movie. Both keys are required to not be null.

Track

The track table represents tracks of the albums that the library has within its system. All keys are required to not be null. Title is a primary key of type varchar(30) that represents the title of the track. Album_title is a primary key and a foreign key of type varchar(30) that references the Album table and represents the title of the album associated with the track. Length is a key of type double that represents the length of the track in minutes.

Query Explanation

In this section we will discuss the queries created for checkpoint 4 and 5.

Checkpoint 4.3a

Description: Find the titles of all tracks by ARTIST released before YEAR, for this example we put placeholders of Billie Eilish as ARTIST and 2020 as YEAR.
Relational Algebra:

$$T1 \leftarrow MEDIA \bowtie_{Title=AlbumTitle} (TRACK)$$

$$T2 \leftarrow \sigma_{CreatorName=ARTIST\ AND\ MediaType='Album'\ AND\ Year<YEAR} (T1)$$

$$RESULT \leftarrow \pi_{Title} (TRACK)$$

SQL Query:

```
SELECT Title
FROM TRACK
WHERE Album_Title IN
(SELECT Title
 FROM MEDIA
 WHERE MEDIA.Creator_name = 'Billie Eilish' AND
      MEDIA.Year < '2020' AND MEDIA.Media_type = 'Album');
```

Checkpoint 4.3b

Description: Give all the movies and their date of their checkout from a single patron. Patrons are designated by the library card ID number.
Relational Algebra:

$$T1 \leftarrow COPY \bowtie_{IdCopy=CopyId} (CHECKED - OUT)$$

$$T2 \leftarrow \sigma_{MediaType='Movie'\ AND\ CardId=NUMBER} (T1)$$

$$RESULT \leftarrow \pi_{MediaTitle,\ CheckedDate} (T2)$$

SQL Query:

```
SELECT Media_Title, Checked_Date
FROM COPY, CHECKED_OUT
WHERE COPY.Media_type = 'Movie' AND
```

```
    CHECKED_OUT.Card_Id = '100' AND
    COPY.Id_copy = CHECKED_OUT.Copy_Id;
```

Checkpoint 4.3c

Description: List all the albums and their unique identifiers with less than 2 copies held by the library.

Relational Algebra:

$$T1 \leftarrow \rho_{(MediaTitle,\ NoCopies)}(MediaTitle\ \mathcal{F}_{COUNT(MediaTitle)}(COPY))$$

$$T2 \leftarrow \sigma_{NoCopies < 2}(T1)$$

$$RESULT \leftarrow \pi_{MediaTitle}(T2)$$

SQL Query:

```
SELECT Media_title
FROM COPY
WHERE COPY.Media_type = 'Album'
GROUP BY COPY.Media_title
HAVING COUNT(Media_Title) < 2;
```

Checkpoint 4.3d

Description: Give all the patrons who checked out a movie by ACTOR and the movies they checked out. Zendaya serves as a placeholder here for ACTOR.

Relational Algebra:

$$T1 \leftarrow \sigma_{ActorName\ =\ ACTOR}(STARRING)$$

$$T2 \leftarrow LIBRARYCARD \bowtie_{IdCard=CardId} (CHECKEDOUT)$$

$$T3 \leftarrow T2 \bowtie_{CopyId=IdCopy} (COPY)$$

$$T4 \leftarrow T3 \bowtie_{Email=Email} (PATRON)$$

$$T5 \leftarrow T1 \bowtie_{MovieTitle=MediaTitle} (T4)$$

$$RESULT \leftarrow \pi_{Fname,\ Lname,\ Email,\ MovieTitle}(T5)$$

SQL Query:

```
SELECT PATRON.Fname, PATRON.Lname, PATRON.Email, STARRING.Movie_title
FROM PATRON, STARRING, LIBRARY_CARD, CHECKED_OUT, COPY
WHERE PATRON.Email = LIBRARY_CARD.Email AND
    STARRING.Actor_name = "Zendaya" AND
    LIBRARY_CARD.Id_card = CHECKED_OUT.Card_Id AND
    CHECKED_OUT.Copy_Id = COPY.Id_copy AND
    STARRING.Movie_title = COPY.Media_title;
```

Checkpoint 4.3e

Description: Find the total number of albums checked out by a single patron. Patrons are designated by the library card ID number.
Relational Algebra:

$$T1 \leftarrow CHECKEDOUT \bowtie_{CopyId=IdCopy} (COPY)$$

$$T2 \leftarrow \sigma_{MediaType='Album' \text{ AND } CardId = NUMBER} (T1)$$

$$RESULT \leftarrow \mathcal{F}_{COUNT(MediaTitle)} (T2)$$

SQL Query:

```
SELECT count(*) as 'Album Count'
FROM CHECKED_OUT, COPY
WHERE CHECKED_OUT.Copy_Id = COPY.Id_copy AND
    COPY.Media_type = 'Album' AND
    CHECKED_OUT.Card_Id = '101';
```

Checkpoint 4.3f

Description: Find the patron who has checked out the most videos and the total number of videos they have checked out.
Relational Algebra:

$$T1 \leftarrow LIBRARYCARD \bowtie_{IdCard=CardId} (CHECKEDOUT)$$

$$T2 \leftarrow T1 \bowtie_{CopyId=IdCopy} (COPY)$$

$$T3 \leftarrow T2 \bowtie_{Email=Email} (PATRON)$$

$$T4 \leftarrow \sigma_{MediaType='Movie'} (T3)$$

$$RESULT \leftarrow \pi(_{Email}\mathcal{F}_{MAX(COUNT(Email))} (T4)$$

SQL Query:

```
SELECT PATRON.Fname, PATRON.Lname, PATRON.Email, COUNT(*) AS MoviesChecked
FROM PATRON, LIBRARY_CARD, CHECKED_OUT, COPY
WHERE PATRON.Email = LIBRARY_CARD.Email AND
    LIBRARY_CARD.Id_card = CHECKED_OUT.Card_Id AND
    CHECKED_OUT.Copy_Id = COPY.Id_copy AND
    COPY.Media_type = 'Movie'
GROUP BY PATRON.Email
ORDER BY MoviesChecked DESC
LIMIT 1;
```

Checkpoint 4.4a

Description: Find the number of copies and date of arrival of all orders for a single movie
Relational Algebra:

$T1 \leftarrow \sigma_{MediaType='Movie' \text{ AND } MediaTitle=TITLE}(DELIVERYORDER)$

$RESULT \leftarrow \pi_{NumCopies,\ DateOfArrival}(T1)$

SQL Query:

```
SELECT DELIVERY_ORDER.Num_Copies, DELIVERY_ORDER.Date_of_Arrival
FROM DELIVERY_ORDER
WHERE DELIVERY_ORDER.Media_type = "Movie" AND DELIVERY_ORDER.Media_Title =
"Dune";
```

Checkpoint 4.4b

Description: Find all the available copies of audiobooks narrated by a single narrator
Relational Algebra:

$T1 \leftarrow COPY \bowtie_{MediaTitle=Title}(AUDIOBOOK)$

$RESULT \leftarrow \sigma_{Narrator=NAME \text{ AND } Status="available"}(T1)$

SQL Query:

```
SELECT COPY.Media_title, COPY.Id_copy
FROM  COPY, AUDIO_BOOK
WHERE COPY.Media_title = AUDIO_BOOK.Title AND AUDIO_BOOK.Narrator = "Ilyana
Kadushin" AND COPY.Status = "Available";
```

Checkpoint 4.4c

Description: Find the total number of copies of each item checked in to the library.
Relational Algebra:

$T1 \leftarrow \sigma_{Status="Available"}(COPY)$

$RESULT \leftarrow \pi_{MediaTitle,\ MediaType,\ CopyCount}({}_{MediaTitle,\ MediaType}\mathcal{F}_{COUNT(IdCopy)}(T1))$

SQL Query:

```
SELECT COPY.Media_Title, COPY.Media_type, count(*) as "Available copies"
FROM  COPY
WHERE Status = "Available"
GROUP BY COPY.Media_title, COPY.Media_type;
```

Checkpoint 5.4a

Description: Provides a list of patron names, along with the total combined running time of all the movies they have checked out.
Relational Algebra:

$T1 \leftarrow PATRON \bowtie_{Email=Email} LIBRARYCARD$

$T2 \leftarrow T1 \bowtie_{IdCard=CardId} CHECKEDOUT$

$T3 \leftarrow T2 \bowtie_{CopyId=IdCopy} COPY$

$T4 \leftarrow T3 \bowtie_{MediaTitle=Title} MEDIA$

$T5 \leftarrow \sigma_{MediaType="Movie"}(T4)$

$RESULT \leftarrow \pi_{Fname, Lname, SumLength}({}_{Email}\mathcal{F}_{SUM(Length)}(T5))$

SQL Query:

```
SELECT PATRON.Fname, PATRON.Lname, SUM(MEDIA.Length)
FROM PATRON, MEDIA, LIBRARY_CARD, COPY, CHECKED_OUT
WHERE PATRON.Email = LIBRARY_CARD.Email AND LIBRARY_CARD.Id_card =
CHECKED_OUT.Card_Id AND MEDIA.Media_type = "Movie" AND COPY.Media_title =
MEDIA.Title AND CHECKED_OUT.Copy_Id = COPY.Id_copy
GROUP BY PATRON.Email
ORDER BY PATRON.Lname;
```

Checkpoint 5.4b

Description: Provides a list of patron names and email addresses for patrons who have checked out more albums than the average patron.

Relational Algebra:

$T1 \leftarrow PATRON \bowtie_{Email=Email} LIBRARYCARD$

$T2 \leftarrow T1 \bowtie_{IdCard=CardId} CHECKEDOUT$

$T3 \leftarrow T2 \bowtie_{CopyId=IdCopy} COPY$

$T4 \leftarrow \sigma_{MediaType="Album"} T3$

$avgCheckedOut \leftarrow \pi_{avgCheckedOut}({}_{Email}\mathcal{F}_{AVERAGE(COUNT(CardId))}(T4))$

$T5 \leftarrow \mathcal{F}_{COUNT(CardId)}(T4)$

$RESULT \leftarrow \sigma_{T5.CountCheckedOut > avgCheckedOut}(T5)$

SQL Query:

```
SELECT PATRON.Fname, PATRON.Lname, PATRON.Email, COUNT(CHECKED_OUT.Card_Id)
FROM PATRON, (LIBRARY_CARD JOIN CHECKED_OUT ON LIBRARY_CARD.Id_card =
CHECKED_OUT.Card_Id JOIN COPY ON CHECKED_OUT.Copy_Id = COPY.Id_copy AND
COPY.Media_type = "Album")
WHERE PATRON.Email = LIBRARY_CARD.Email
GROUP BY PATRON.Email
HAVING COUNT(CHECKED_OUT.Card_Id) >
    (SELECT AVG(c.cCount) FROM
        (SELECT COUNT(CHECKED_OUT.Card_Id) as cCount
        FROM PATRON, (LIBRARY_CARD JOIN CHECKED_OUT ON LIBRARY_CARD.Id_card
= CHECKED_OUT.Card_Id JOIN COPY ON CHECKED_OUT.Copy_Id = COPY.Id_copy AND
COPY.Media_type = "Album")
```

```
WHERE PATRON.Email = LIBRARY_CARD.Email
GROUP BY PATRON.Email) c);
```

Checkpoint 5.4c

Description: Provide a list of the movies in the database and associated total copies lent to patrons, sorted from the movie that has been lent the most to the movies that has been lent the least.

Relational Algebra:

$$T1 \leftarrow COPY \rtimes_{IdCopy=CopyId}(CHECKEDOUT)$$

$$T2 \leftarrow \sigma_{MediaType="Movie"}(T1)$$

$$RESULT \leftarrow \pi_{MediaTitle,\ CopyCount}({}_{MediaTitle}\mathcal{F}_{COUNT(CopyId)}(T2))$$

SQL Query:

```
SELECT COPY.Media_title, COUNT(CHECKED_OUT.COPY_Id) as Lent
FROM (COPY LEFT OUTER JOIN CHECKED_OUT ON CHECKED_OUT.Copy_Id =
COPY.Id_copy)
WHERE COPY.Media_type = "Movie"
GROUP BY COPY.Media_title
ORDER BY COUNT(CHECKED_OUT.COPY_Id) DESC;
```

Checkpoint 5.4d

Description: Provides a list of the albums in the database and associated totals for copies checked out to customers, sorted from the ones that have been checked out the highest amount to the ones checked out the lowest.

Relational Algebra:

$$T1 \leftarrow CHECKEDOUT \rtimes_{CopyId=IdCopy}(COPY)$$

$$T2 \leftarrow \sigma_{MediaType="Album"\ AND\ ReturnedDate=NULL}(T1)$$

$$RESULT \leftarrow \pi_{MediaTitle,\ CardCount}({}_{MediaTitle}\mathcal{F}_{COUNT(CardId)}(T2))$$

SQL Query:

```
SELECT COPY.Media_title, COUNT(CHECKED_OUT.Card_Id) as "Checked Out"
FROM COPY, CHECKED_OUT
WHERE CHECKED_OUT.Copy_Id = COPY.Id_copy AND COPY.Media_type = "Album" AND
CHECKED_OUT.Returned_Date IS NULL
GROUP BY COPY.Media_title
ORDER BY COUNT(CHECKED_OUT.Card_Id) DESC;
```

Checkpoint 5.4e

Description: Find the most popular actor in the database (i.e. the one who has had the most lent movies).
Relational Algebra:

$$T1 \leftarrow STARRING \bowtie_{MovieTitle=MediaTitle} (COPY)$$

$$T2 \leftarrow T1 \bowtie_{IdCopy=CopyId} (CHECKEDOUT)$$

$$RESULT \leftarrow \pi_{ActorName,\, MaxMovieCount} ({}_{ActorName}\mathcal{F}_{MAX(COUNT(MovieTitle))} (T2))$$

SQL Query:

```
SELECT actors.Actor_name, MAX(max_movie_count)
FROM STARRING, (SELECT STARRING.Actor_name, COUNT(CHECKED_OUT.COPY_Id) as
max_movie_count
     FROM STARRING, COPY, CHECKED_OUT
     WHERE STARRING.Movie_title = COPY.Media_title AND COPY.Id_copy =
CHECKED_OUT.Copy_Id
     GROUP BY STARRING.Actor_name) as actors;
```

Checkpoint 5.4f

Description: Find the most listened to artist in the database (use the running time of the album and number of times the album has been lent out to calculate).
Relational Algebra:

$$T1 \leftarrow MEDIA \bowtie_{Title=MediaTitle} COPY$$

$$T2 \leftarrow T1 \bowtie_{IdCopy=CopyId} CHECKEDOUT$$

$$T3 \leftarrow \sigma_{MediaType="Album"} (T2)$$

$$RESULT \leftarrow \pi_{CreatorName,\, MaxCountListening} ({}_{CreatorName}\mathcal{F}_{MAX(COUNT(COUNT(CopyId)*Length))} (T3))$$

SQL Query:

```
SELECT creators.Creator_name, MAX(max_listening) as "Time (Minutes)"
FROM (SELECT MEDIA.Creator_name, (COUNT(CHECKED_OUT.Copy_Id) *
MEDIA.Length) as max_listening
     FROM MEDIA, COPY, CHECKED_OUT
     WHERE MEDIA.Title = COPY.Media_title AND COPY.Id_copy =
CHECKED_OUT.Copy_Id AND MEDIA.Media_type = "Album"
     GROUP BY MEDIA.Creator_name) as creators;
```

Checkpoint 5.4g

Description: Provides a list of customer information for patrons who have checked out anything by the most watched actor in the database.
Relational Algebra:

$T1 \leftarrow {}_{ActorName}\mathcal{F}_{COUNT(MovieTitle)}(STARRING)$

$T2 \leftarrow {}_{ActorName}\mathcal{F}_{MAX(MovieTitle)}(STARRING)$

$T3 \leftarrow T1 \bowtie_{ActorName=ActorName} T2$

$T4 \leftarrow \sigma_{MAX(MovieTitle)=COUNT(MovieTitle)} T3$

$T5 \leftarrow PATRON \bowtie_{Email=Email} LIBRARYCARD$

$T6 \leftarrow T5 \bowtie_{IdCard=CardId} CHECKEDOUT$

$T7 \leftarrow T6 \bowtie_{CopyId=IdCopy} COPY$

$T8 \leftarrow T7 \bowtie_{MediaTitle=MovieTitle} T4$

$RESULT \leftarrow \pi_{Lname,\ Fname,\ Email,\ MediaTitle}(T8)$

SQL Query:

```
SELECT PATRON.Lname, PATRON.Fname, PATRON.Email, COPY.Media_title as "Movie
Title"
FROM PATRON, LIBRARY_CARD, COPY, CHECKED_OUT, (SELECT title as movie_title,
MAX(max_movie_count) as count
            FROM (SELECT STARRING.Movie_title as title,
COUNT(CHECKED_OUT.Copy_Id) as max_movie_count
                FROM STARRING, COPY, CHECKED_OUT
                WHERE STARRING.Movie_title = COPY.Media_title AND
COPY.Id_copy = CHECKED_OUT.Copy_Id
                GROUP BY STARRING.Actor_name)) as MAXM
WHERE PATRON.Email = LIBRARY_CARD.Email AND LIBRARY_CARD.Id_card =
CHECKED_OUT.Card_Id AND CHECKED_OUT.Copy_Id = COPY.Id_copy AND
COPY.Media_title = MAXM.movie_title
GROUP BY PATRON.Email
ORDER BY PATRON.Lname;
```

Checkpoint 5.4h

Description: Provides a list of artists who authored the albums checked out by customers who have checked out more albums than the average customer.

Relational Algebra:

$T1 \leftarrow PATRON \bowtie_{Email=Email} LIBRARYCARD$

$T2 \leftarrow T1 \bowtie_{IdCard=CardId} CHECKEDOUT$

$T3 \leftarrow T2 \bowtie_{CopyId=IdCopy} COPY$

$T4 \leftarrow \sigma_{MediaType="Album"} T3$

$avgCheckedOut \leftarrow \pi_{avgCheckedOut}({}_{Email}\mathcal{F}_{AVERAGE(COUNT(CardId))}(T4))$

$T5 \leftarrow \mathcal{F}_{COUNT(CardId)}(T4)$

$T6 \leftarrow \sigma_{T5.CountCheckedOut > avgCheckedOut}(T5)$

$T7 \leftarrow T6 \bowtie_{MediaTitle=Title} MEDIA$

$RESULT \leftarrow \pi_{CreatorName}(T7)$

SQL Query:

```
SELECT MEDIA.Creator_name
FROM MEDIA, COPY, CHECKED_OUT, (SELECT LIBRARY_CARD.Id_card as cardId,
COUNT(CHECKED_OUT.Card_Id) as album_count
            FROM (LIBRARY_CARD LEFT OUTER JOIN (CHECKED_OUT JOIN COPY ON
CHECKED_OUT.Copy_Id = COPY.Id_copy AND COPY.Media_type = "Album") ON
LIBRARY_CARD.Id_card = CHECKED_OUT.Card_Id)
            GROUP BY LIBRARY_CARD.Id_card),
            (SELECT AVG(album_count) as avg_count
            FROM (SELECT COUNT(CHECKED_OUT.Card_Id) as album_count
            FROM (LIBRARY_CARD LEFT OUTER JOIN (CHECKED_OUT JOIN COPY ON
CHECKED_OUT.Copy_Id = COPY.Id_copy AND COPY.Media_type = "Album") ON
LIBRARY_CARD.Id_card = CHECKED_OUT.Card_Id)
            GROUP BY LIBRARY_CARD.Id_card))
WHERE cardId = CHECKED_OUT.Card_Id AND CHECKED_OUT.Copy_Id = COPY.Id_copy
AND COPY.Media_type = "Album" AND COPY.Media_title = MEDIA.Title
GROUP BY MEDIA.Creator_name
HAVING album_count > avg_count;
```

Insert Syntax

Track

In order to insert a track into the database, the album that the track is on must be added first. This insert statement is assuming that the album has already been added. The song “Yellow” by Coldplay is the example used for this.

```
INSERT INTO TRACK
VALUES (‘Yellow’, 266, ‘Parachutes’);
```

Album

In order to insert an album into the database, it must first be added to the Media table. The artist must be added to the Content_Creator table first as well. An album that needs to be added to the database must not already exist in the Media table, so the insert statement for the Media table will be included in this. The artist can possibly exist, if other albums by this artist have already been added to the database, so this insert statement will assume that the artist already exists. The album “A Head Full of Dreams” by Coldplay is the example used for this.

```
INSERT INTO MEDIA
VALUES ('A Head Full of Dreams', 2015, 45, 'Album', 'Coldplay');

INSERT INTO ALBUM
VALUES ('A Head Full of Dreams', 'Alternative', 'Parlophone', 'Album');
```

Movie

Much like an album, a movie must first be added to the Media table. The director of the movie must be added to the Content_Creator table as well, in addition to any actors that need to be included in the Starring table. A movie that needs to be added to the database must not already exist in the Media table, so the insert statement for the Media table will be included in this. The director can possibly exist, if other movies directed by them have already been added to the database. Additionally, actors starring in this movie can possibly exist, if they have starred in other movies already added to the database. The Starring table cannot already include anything with this movie, so these relations must be added as well. So, the insert statements for the director and the actors will not be included in this, but the insert statements adding the movie and actors to the Starring table will be included. The movie “The Hunger Games: Mockingjay Part 2” directed by Francis Lawrence and starring Jennifer Lawrence, Josh Hutcherson, and Liam Hemsworth, will be the example used for this insert statement.

```
INSERT INTO MEDIA
VALUES ('The Hunger Games: Mockingjay Part 2', 2015, 137, 'Movie', 'Francis
Lawrence');

INSERT INTO MOVIE
VALUES ('The Hunger Games: Mockingjay Part 2', 'Sci-Fi', 'PG-13',
'Blu-Ray', 'Movie');

INSERT INTO STARRING
VALUES ('The Hunger Games: Mockingjay Part 2', 'Jennifer Lawrence');

INSERT INTO STARRING
VALUES ('The Hunger Games: Mockingjay Part 2', 'Josh Hutcherson');

INSERT INTO STARRING
VALUES ('The Hunger Games: Mockingjay Part 2', 'Liam Hemsworth');
```

Audiobook

Inserting an audiobook has the same basic setup as a movie or an album. First, it must be added to the Media table. The author of the book and the narrator of the audiobook must both be added to the Creator table. An audiobook that needs to be added to the database must not already exist in the Media table, so the insert statement to do this will be included. The author

can possibly exist, if other books written by them have already been added to the database. The narrator may also already exist, if the database includes other audiobooks narrated by them. As a result, this example will assume that both the author and the narrator already exist within the database. This example for this insert statement is “Children of Dune” by Frank Herbert, narrated by Scott Brick.

```
INSERT INTO MEDIA
VALUES ('Children of Dune', 1976, 1011, 'Audiobook', 'Frank Herbert');

INSERT INTO AUDIO_BOOK
VALUES ('Children of Dune', 'Sci-Fi', 'Macmillan Audio', 'Scott Brick',
'Audiobook');
```

Content Creator

The insert statement for a content creator is simple. First, they must be added to the Content_Creator table. Then, they must be added to the Creator_Type. Both of these are necessary for a content creator of any type to be added. The example for this insert statement is Katy Perry, who is a musician.

```
INSERT INTO CONTENT_CREATOR
VALUES ('Katy Perry');

INSERT INTO CREATOR_TYPE
VALUES ('Musician', 'Katy Perry');
```

Patron

Adding a patron to the database is straightforward. First, they must be added to the Patron table. If they are being added to the database, they must not already have a library card, so this insert statement will include adding a library card to the database as well.

```
INSERT INTO PATRON
VALUES ('john.smith@gmail.com', '101 N High Street', 'John', 'Smith');

INSERT INTO LIBRARY_CARD
VALUES (200, 'john.smith@gmail.com');
```

Delete Syntax

Due to cascading deletion, all of the foreign key dependencies are either removed or nullified automatically by just deleting the entity.

Track

Tracks have no dependent relations so they can be easily removed without removing any entries in other tables. The primary key for tracks is both the track title and the album it's associated with. The track called “TrackTitle” from the album “AlbumTitle” is removed.

```
DELETE FROM TRACK WHERE Title = "TrackTitle" AND Album_title =
"AlbumTitle";
```

Album

To remove an album we must remove both the album and media entries manually. Here the album called “AlbumTitle” is removed from the media and album tables along with the associated tracks, copies, delivery orders, and checked out entries.

```
DELETE FROM MEDIA WHERE Title = "AlbumTitle" AND Media_type = "Album";
```

Movie

The same procedure as used with the album table; we remove the main media entry and all references cascade. The movie with the title “MovieTitle” is removed.

```
DELETE FROM MEDIA WHERE Title = "MovieTitle" AND Media_type = "Movie";
```

Audiobook

Same as the movie and audiobook tables; the delete cascades through the various references in the database. Here the audio book with the title “BookTitle" is removed.

```
DELETE FROM MEDIA WHERE Title = "BookTitle" AND Media_type = "Audiobook";
```

Content Creator

Content creators are referenced in a number of places as a foreign key, each of those references are set to cascade on delete. Removing a content creator removes many entries across the database (Including in Starring, Copy, Checked Out, Media, and all its subtypes). . Here we remove the content creator named “CreatorName”.

```
DELETE FROM CONTENT_CREATOR WHERE Name = "CreatorName";
```

Patron

Patrons are referenced in library_card, and library_card is mentioned in checked_out. Due to cascading delete, deleting the patron is enough to delete foreign key references.

```
DELETE FROM PATRON WHERE Email = "PatronEmail";
```