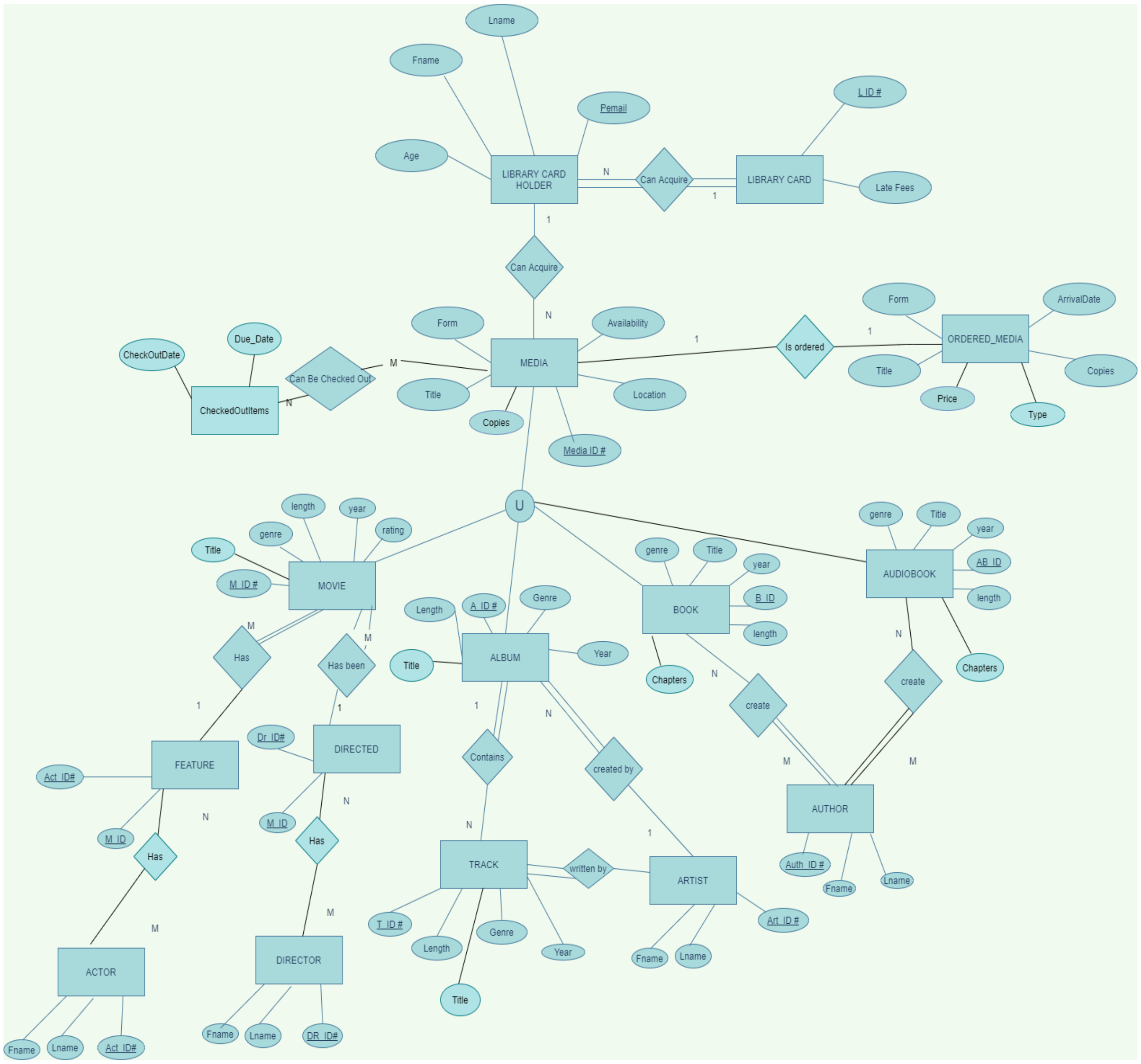
Final Project Report

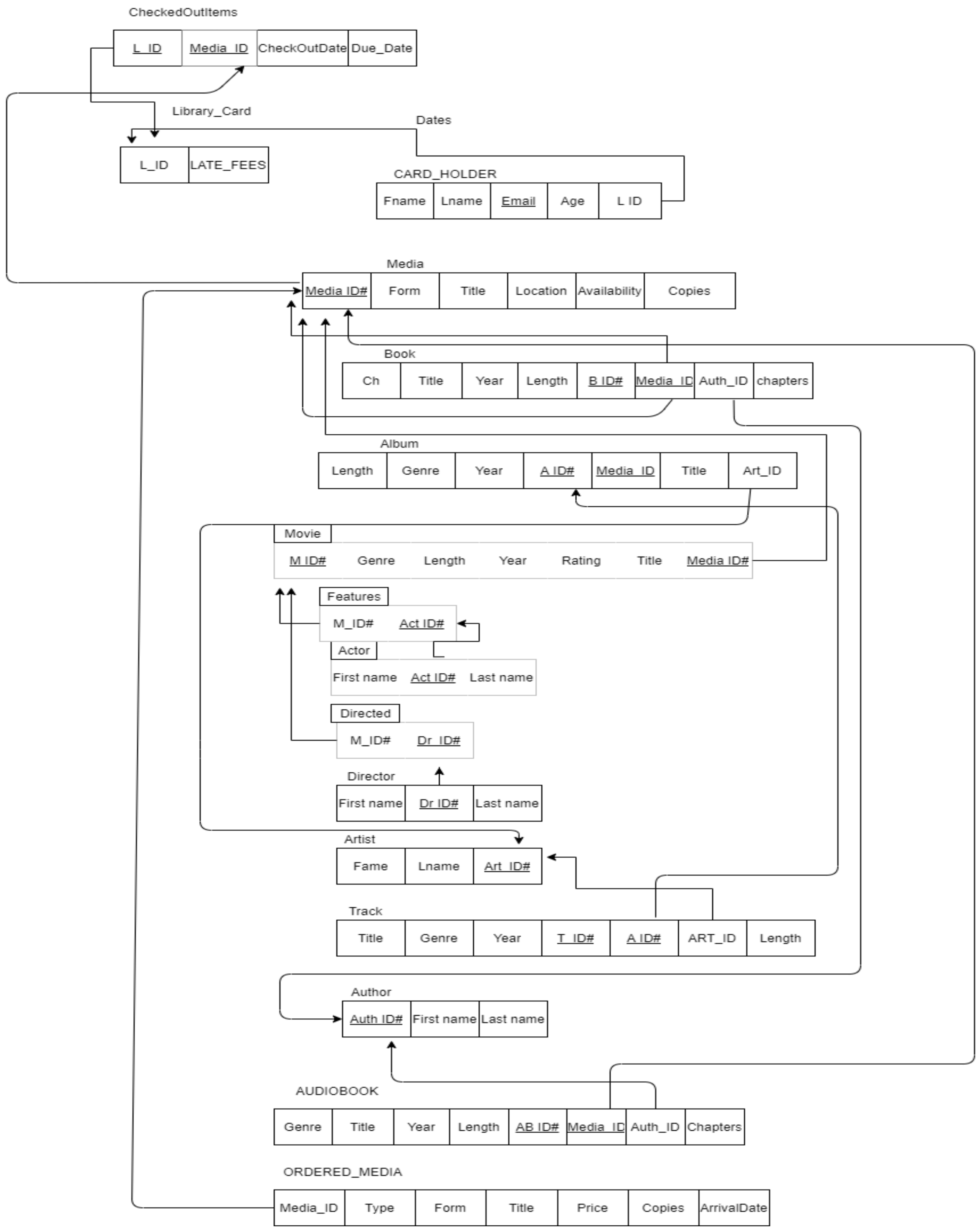
Sudi Yussuf | Harold Henderson | Nikhil Singh | Robby Berling

Section 1: Database Description

A. Current and Final ER Diagram of Our Database



B. Current Relational Diagram of Our Database



C. Normalization of each Table in Database

- Checked Out Items
  - Boyce Codd Normal Form

  Checked Out Items is in BCNF because there is only one functional dependency, and the Canidate key determines the nonprime attributes.

- Library Card
  - Boyce Codd Normal Form

  Library Card is in BCNF because all its attributes are only dependent on the primary key which is the library ID.

- Library Card Holder
  - Boyce Codd Normal Form

  Library Card Holder is in BCNF because all the attributes are only dependent on the primary key which is the email of the card holder.

- Media
  - Boyce Codd Normal Form

  Media is in BCNF because the attributes are only dependent on the primary key which is Media_ID.

- Book
  - Boyce Codd Normal Form

  Book is in BCNF because the attributes are only dependent on the primary key which is the Book_ID.

- Album
  - Boyce Codd Normal Form

  Album is in BCNF because the attributes are only dependent on the primary key which is the Album_ID.

- Movie
  - Boyce Codd Normal Form

  Movie is in BCNF because the attributes are only dependent on the primary key which is the Movie_ID.

- Features
  - Boyce Codd Normal Form

  Features is in BCNF because the attributes are all foreign keys

- Actor
  - Boyce Codd Normal Form

  Actor is in BCNF because the attributes are only dependent on the primary key which is the Actor_ID.

- Director
    - Boyce Codd Normal Form

  Director is in BCNF because the attributes are only dependent on the primary key which is the Director_ID.

- Directed
    - Boyce Codd Normal Form

  Directed is in BCNF because the attributes are all foreign keys

- Artist
    - Boyce Codd Normal Form

  Artist is in BCNF because the attributes are only dependent on the primary key which is the Artist_ID.

- Track
    - Boyce Codd Normal Form

  Track is in BCNF because the attributes are only dependent on the primary key which is the Track_ID.

- Author
    - Boyce Codd Normal Form

  Author is in BCNF because the attributes are only dependent on the primary key which is the Author_ID.

- Audiobook
    - Boyce Codd Normal Form

  Audiobook is in BCNF because the attributes are only dependent on the primary key which is the Audiobook_ID.

- Ordered Media
    - Boyce Codd Normal Form

  Ordered Media is in BCNF because the attributes are only dependent on the primary key which is the Title and Media_ID.

D.

1) CREATE INDEX CopyIndex ON MEDIA (Copies);

We chose to create an index on the total copies available of media items in the database. This index would allow for queries to access the copies of a media item quicker for deciding whether an item is available or unavailable.

2) CREATE INDEX AvailableIndex ON MEDIA (Availability);

Next, we chose to create an index on the availability of media items in the database. This index will allow for quicker access to items that are available to be checked out or for viewing all the items that are unavailable.

3) CREATE INDEX DueDateIndex ON CheckedOutItems (Due_Date);

Then, we chose to create an index on the due date of media items in the database. This index will allow for quicker access to items with a certain due date or items with their due date in a certain range. This is a popular query for checking items past due or with their due date coming up.

4) CREATE INDEX AgeIndex ON CARD_HOLDER (Age);

Finally, we chose to create an index on the age of the card holder. This index would allow for easier access to people within a certain age range for useful reports. Additionally, we believe the size of the card holder table will increase rapidly over time, so this index will increase efficiency with the high-volume table.

E.

PastDue View: This view produces the names of each card holder in the database that have a checked-out item past due and how many they have. This is useful in order to quickly see the people who need to be charged late fees.

Relational algebra expression to produce this view:

PastDue:

Join1←CheckedOutItems⋈L_ID=L_ID(Library_Card)

Join2←L_Join1 ⋈L_ID=L_ID(Card_Holder)

Π Fname, Lname, Count(Media_ID)(σ 2021-04-20 > Due_Date(Join2))

SQL statements to produce the view:

CREATE VIEW PastDue

AS SELECT Fname, Lname, count(distinct CheckedOutItems.Media_ID)

FROM CheckedOutItems, Library_Card, CARD_HOLDER

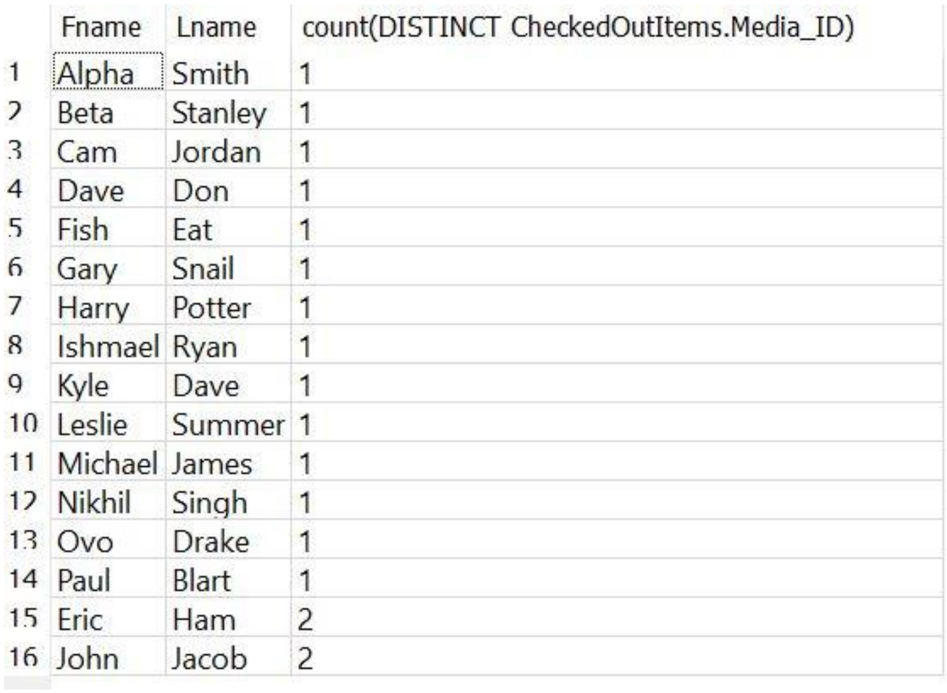
WHERE Library_Card.L_ID = CheckedOutItems.L_ID AND Library_Card.L_ID = CARD_HOLDER.L_ID AND '2021-04-20' > Due_Date

GROUP BY Fname

ORDER BY count(distinct CheckedOutItems.Media_ID);

Sample output from the view:

Past Due



	Fname	Lname	count(DISTINCT CheckedOutItems.Media_ID)
1	Alpha	Smith	1
2	Beta	Stanley	1
3	Cam	Jordan	1
4	Dave	Don	1
5	Fish	Eat	1
6	Gary	Snail	1
7	Harry	Potter	1
8	Ishmael	Ryan	1
9	Kyle	Dave	1
10	Leslie	Summer	1
11	Michael	James	1
12	Nikhil	Singh	1
13	Ovo	Drake	1
14	Paul	Blart	1
15	Eric	Ham	2
16	John	Jacob	2

CheckedOut View: This view produces the names of each card holder in the database that have checked out items and how many they have. This view is useful because it allows to quickly see the people with items checked out and who likes to check out the most items.

Relational algebra expression to produce this view:

CheckedOut:

Join1←CheckedOutItems⋈L_ID=L_ID(Library_Card)

Join2←L_Join1 ⋈L_ID=L_ID(Card_Holder)

Π Fname, Lname, Count(Media_ID)(Join2)

SQL statements to produce the view:

CREATE VIEW CheckedOut

AS SELECT Fname, Lname, count(distinct CheckedOutItems.Media_ID)

FROM CheckedOutItems, Library_Card, CARD_HOLDER

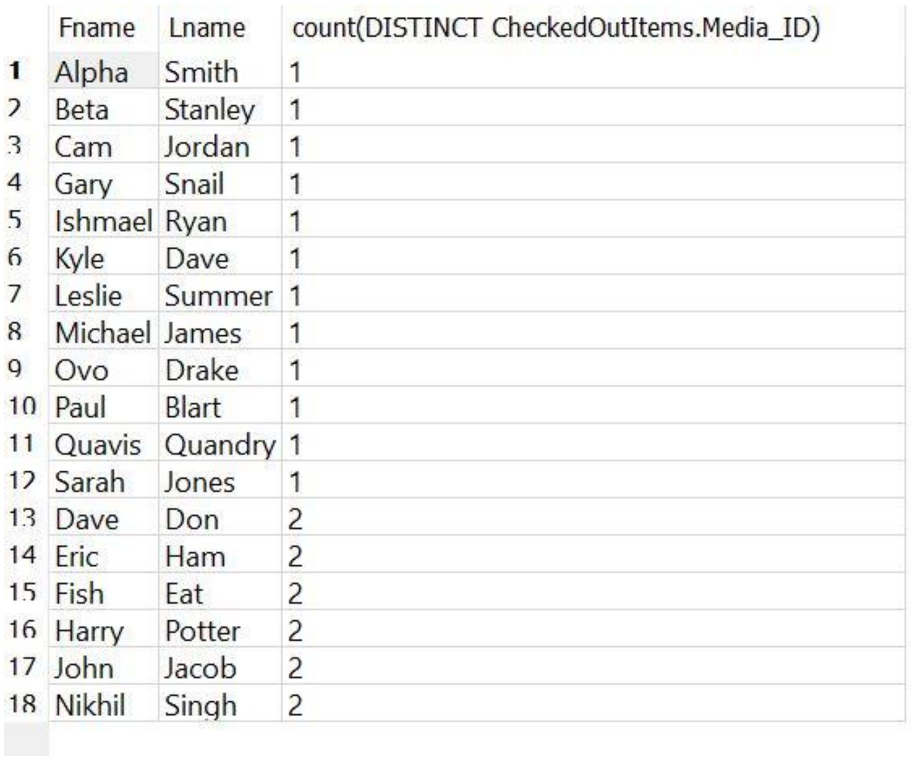
WHERE Library_Card.L_ID = CheckedOutItems.L_ID AND Library_Card.L_ID = CARD_HOLDER.L_ID

GROUP BY Fname

ORDER BY count(distinct CheckedOutItems.Media_ID);

Sample output from the view:

Checked Out



	Fname	Lname	count(DISTINCT CheckedOutItems.Media_ID)
1	Alpha	Smith	1
2	Beta	Stanley	1
3	Cam	Jordan	1
4	Gary	Snail	1
5	Ishmael	Ryan	1
6	Kyle	Dave	1
7	Leslie	Summer	1
8	Michael	James	1
9	Ovo	Drake	1
10	Paul	Blart	1
11	Quavis	Quandry	1
12	Sarah	Jones	1
13	Dave	Don	2
14	Eric	Ham	2
15	Fish	Eat	2
16	Harry	Potter	2
17	John	Jacob	2
18	Nikhil	Singh	2

F.

1) When the database receives a book order the database must delete the book from Ordered_Media, and update the availability and copies of the book in Media. The unit of work in this case would be receiving an ordered book that already exists in the database.

SQL Code:

```
BEGIN TRANSACTION RecieveExisting;

DELETE FROM Ordered_Media WHERE Title = 'A Long Walk';

UPDATE OR ROLLBACK Media SET Availability = 'Available' WHERE Title = 'A Long Walk';

UPDATE OR ROLLBACK Media SET Copies = Copies + 1 WHERE Title = 'A Long Walk';

COMMIT;
```

2) When the database orders a media item, the database must insert it into media with its availability set to ordered and insert the item into Ordered_Media. The unit of work in this case would be ordering a new media item.

```
BEGIN TRANSACTION OrderNew;

INSERT OR ROLLBACK INTO Media VALUES ('000000000000081', 'Movie', 'StarStruck', 'Columbus', 'Ordered', '3');

INSERT OR ROLLBACK INTO Ordered_Media VALUES('000000000000081', 'Movie', 'Digital', 'StarStruck', 12, 3, '2021-04-21');

COMMIT;
```

3) When the database needs to edit an ordered item, the database must update the features in Ordered_Media which in this case are the copies and title and update the item in Media. The unit of work in this case would be editing an ordered item.

```
BEGIN TRANSACTION edit;

UPDATE OR ROLLBACK Ordered_Media SET Title = 'StarStruck' AND ArrivalDate = '2021-04-28' WHERE Title = 'A Long Walk';

UPDATE OR ROLLBACK  Media SET Title = 'StarStruck' WHERE Title = 'A Long Walk';

COMMIT;
```

Section 2 : User Manual

A. A Brief Description of All of the Relations in Our Database.

- LIBRARY_CARD: This table represents a library card.
    - L_ID : a ten-digit library card number represented by a char of 15 characters.
        - This is the primary key.
    - LATE_FEES: a dollar amount of late fees for the library card represented by an integer.

- CARD_HOLDER: This table represents a card holder account.
    - Fname: the first name of the patron represented by a string of 50 characters or less.
    - Lname: the last name of the patron represented by a string of 50 characters or less.
    - Email: a string that represents the email of the library card owner.
        - This is the primary key.
    - Age: the age of the patron represented by an integer.
    - L_ID : a ten-digit library card number represented by a char of 15 characters.
        - This is a foreign key from the LIBRARY_CARD table.

- CheckedOutItems: this table represents all the checked-out items in the library.
    - Media_ID: this is a 15-char attribute that represents the Media ID number.
        - This is a foreign key from the MEDIA table.
        - This is also part of the primary key.
    - CheckOutDate: the date that an item was checked out represented by a date.
    - Due_Date: the date that an item is due to the library represented by a date.
    - L_ID : a ten-digit library card number represented by a char of 15 characters.
        - This is a foreign key from the LIBRARY_CARD table.
        - This is also part of the primary key.

- MEDIA: this table represents a media item in a library. This relation is a superclass that encompasses the BOOK, AUDIBOOK, ALBUM, and MOVIE relations.
    - Media_ID: this is a 15-char attribute that represents the Media ID number.
        - This is the primary key.
    - Form: attribute that tells you whether an item is physical or digital.
    - Title: the title of a media item represented by a string of at most 500 characters.
    - Location: this attribute tells you where a library item is located represented by a string of at most 50 characters.
    - Availability: tells you whether or not an item is available represented by a 50-character string.
    - Copies: the number of copies available of an item represented by an integer.

- BOOK: this table represents a book.

- Media_ID: this is a 15-char attribute that represents the Media ID number.
  - This is a foreign key from the MEDIA table.
- B_ID: the identifier for a specific book represented by a 10-character char.
  - This is the primary key for the BOOK table.
- Genre: the genre of a book represented by a string of 50-characters or less
- Title: the title of a book represented by a string of 500-characters or less.
- Year: the year a book was written represented by an integer.
- Length: the length of a book in pages represented by an integer.
- Chapters: the number of chapters in a book represented by a integer.
- Auth_ID: the identifier for the author that wrote a book represented by a 10-character char.
  - This is a foreign key from the AUTHOR table.

- AUDIOBOOK: this table represents an audiobook.
  - Media_ID: this is a 15-char attribute that represents the Media ID number.
    - This is a foreign key from the MEDIA table.
  - AB_ID: the identifier for a specific audiobook represented by a 10-character char.
    - This is the primary key for the AUDIOBOOK table.
  - Genre: the genre of an audio book represented by a string of 50-characters or less
  - Title: the title of an audiobook represented by a string of 500-characters or less.
  - Year: the year an audiobook was written represented by an integer.
  - Length: the length of an audiobook in pages represented by an integer.
  - Chapters: the number of chapters in an audiobook represented by an integer.
  - Auth_ID: the identifier for the author that wrote an audiobook represented by a 10-character char.
    - This is a foreign key from the AUTHOR table.

- ALBUM: this table represents an album.
  - Media_ID: this is a 15-char attribute that represents the Media ID number.
    - This is a foreign key from the MEDIA table.
  - A_ID: the identifier for a specific album represented by a 10-character char.
    - This is the primary key for the ALBUM table.
  - Genre: the genre of an album represented by a string of 50-characters or less
  - Title: the title of an album represented by a string of 500-characters or less.
  - Year: the year an album was written represented by an integer.
  - Length: the length of an album in tracks represented by a string of 10 charachters or less.
  - Art_ID: the identifier for a specific artist represented by a 10-character char.
    - This is a foreign key from the ARTIST table.

- TRACK: this table represents a song.
  - T_ID: the identifier for a specific track represented by a 10-character char.
    - This is the primary key for the TRACK table.

  - A_ID: the identifier for a specific album represented by a 10-character char.
    - This is a foreign key from the ALBUM table.
  - Genre: the genre of a track represented by a string of 50-characters or less
  - Title: the title of a track represented by a string of 500-characters or less.
  - Year: the year a track was written represented by an integer.
  - Length: the length of a track represented by time.
  - Art_ID: the identifier for a specific artist represented by a 10-character char.
    - This is a foreign key from the ARTIST table.

- MOVIE: this table represents a movie.
  - Media_ID: this is a 15-char attribute that represents the Media ID number.
    - This is a foreign key from the MEDIA table.
  - M_ID: the identifier for a specific movie represented by a 10-character char.
    - This is the primary key for the MOVIE table.
  - Genre: the genre of a movie represented by a string of 50-characters or less
  - Title: the title of a movie represented by a string of 500-characters or less.
  - Year: the year a movie was made represented by an integer.
  - Length: the length of a movie represented by time.
  - Rating: the rating of the movie represented by a string of 10 characters or less.

- ACTOR: this table represents an actor.
  - Act_ID: the identifier for a specific actor represented by a 10-character char.
    - This is the primary key for the ACTOR table.
  - Fname: the first name of a specific actor represented by a string of 50 characters or less.
  - Lname: the last name of a specific actor represented by a string of 50 characters or less.

- DIRECTOR: this table represents a director
  - Dr_ID: the identifier for a specific director represented by a 10-character char.
    - This is the primary key for the DIRECTOR table.
  - Fname: the first name of a specific director represented by a string of 50 characters or less.
  - Lname: the last name of a specific director represented by a string of 50 characters or less.

- ARTIST: this table represents an artist.
  - Art_ID: the identifier for a specific artist represented by a 10-character char.
    - This is the primary key for the ARTIST table.
  - Fname: the first name of a specific artist represented by a string of 50 characters or less.
  - Lname: the last name of a specific artist represented by a string of 50 characters or less.

- AUTHOR: this table represents an author.
  - Auth_ID: the identifier for a specific author represented by a 10-character char.
    - This is the primary key for the AUTHOR table.
  - Fname: the first name of a specific author represented by a string of 50 characters or less.
  - Lname: the last name of a specific author represented by a string of 50 characters or less.

- FEATURES: this table represents the main actor featured in each movie.
  - M_ID: the identifier for a specific movie represented by a 10-character char.
    - This is a foreign key from the MOVIE table.
    - This attribute is part of the primary key.

  - Act_ID: the identifier for a specific actor represented by a 10-character char.
    - This is a foreign key from the ACTOR table.
    - This attribute is part of the primary key.

- DIRECTED: This table represents the director that directed each movie.
  - M_ID: the identifier for a specific movie represented by a 10-character char.
    - This is a foreign key from the MOVIE table.
    - This attribute is part of the primary key.
  - Dr_ID: the identifier for a specific director represented by a 10-character char.
    - This is a foreign key for the DIRECTOR table.
    - This attribute is part of the primary key.

- ORDERED_MEDIA: this table represents items ordered but not yet checked out by patrons.
  - Media_ID: this is a 15-char attribute that represents the Media ID number.
    - This is a foreign key from the MEDIA table.
    - This is also part of the primary key for ORDERED_MEDIA.
  - Type: attribute that tells you what type of media an item is represented by a string of 50 characters or less.
  - Form: attribute that tells you whether an item is physical or digital is represented by a string of 50 characters or less.
  - Title: the title of a media item represented by a string of at most 500 characters.
    - This is part of the primary key for ORDERED_MEDIA.
  - Price: the price of a media item in integers.
  - ArrivalDate: the date that an ordered item will be available in circulation represented by a date.
  - Copies: the number of copies available of an item represented by an integer.

B. Sample Queries for Our SQL Database

Queries:
Find the titles of all tracks by ARTIST released before YEAR

SQL
Select Title
From ARTIST, TRACK
WHERE ARTIST.Art_ID = TRACK.Art_ID
AND YEAR < '2017-12-12';

Relational Algebra:
TrackW←ARTIST⋈Art_ID=Art_ID(TRACK)
Π (б← (date<2017-12-12) (TrackW))

Give all the movies and their date of their checkout from a single patron (you choose how to designate the patron)

SQL:
SELECT Title, CheckOutDate
FROM CheckedOutItems, MOVIE
WHERE CheckedOutItems.Media_ID = MOVIE.Media_ID
AND L_ID = '0000000001';

Relational Algebra:
Checked_Movies←CheckedOutItems⋈Media_ID=Media_ID(MOVIE)
Π (б← (L_ID = 0000000001)(Checked_Movies))

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

SQL:
SELECT Title
FROM ALBUM, MEDIA
WHERE ALBUM.Media_ID = MEDIA.Media_ID
AND MEDIA.Copies < 2;

Relational Algebra:
AM←ALBUM⋈Media_ID=Media_ID(MEDIA)

Π (б← (copies< 2)(AM))

Give all the patrons who checked out a movie by ACTOR and the movies they checked out.

SQL:

```
SELECT CARD_HOLDER.Fname, CARD_HOLDER.Lname
FROM CheckedOutItems, MOVIE, ACTOR, CARD_HOLDER, FEATURES
WHERE MOVIE.M_ID = FEATURES.M_ID AND ACTOR.Act_ID = FEATURES.Act_ID
AND CheckedOutItems.Media_ID = MOVIE.Media_ID
AND CheckedOutItems.L_ID = CARD_HOLDER.L_ID;
```

Relational Algebra:

F_Actors←ACTORS⋈Act_ID=Act_ID(FEATURES)
L_Actors←F_Actors⋈M_ID=M_ID(MOVIE)
Actors_checked←L_Actors ⋈Media_ID=Media_ID(CheckedOutItems)
Patrons_movies←L_Actors ⋈L_ID=L_ID(CARD_HOLDER)
Π Fname, Lname(Patrons_Movies)

e.Find the total number of albums checked out by a single patron (you choose how to designate the patron)

SQL:

```
SELECT COUNT(CheckedOutItems.Media_ID)
FROM CARD_HOLDER, CheckedOutItems, ALBUM
WHERE CheckedOutItems.Media_ID = ALBUM.Media_ID
AND CheckedOutItems.L_ID = CARD_HOLDER.L_ID
AND CARD_HOLDER.Fname = 'A';
```

Relational Algebra:

Albums_Checked←CheckedOutItems⋈Media_ID=Media_ID(ALBUM)
Patron_Albums← Albums_Checked ⋈L_ID=L_ID(CARD_HOLDER)
Π (б← (Fname = A)(Patron_Albums))

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

SQL:

```
SELECT Fname, Lname
FROM CARD_HOLDER, CheckedOutItems, MOVIE
WHERE CARD_HOLDER.L_ID = CheckedOutItems.L_ID
```

```
AND MOVIE.Media_ID = CheckedOutItems.Media_ID
GROUP BY CARD_HOLDER.L_ID
HAVING MAX(MOVIE.Media_ID)
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Relational Algebra:
Checked_Patrons←CheckedOutItems⋈L_ID=L_ID(CARD_HOLDER)
Checked_Movies←Checked_Patrons⋈Media_ID=Media_ID(MOVIE)
Max_length← $\mathscr{F}$MAX Media_ID(Checked_Movies)
Π (Fname, Lname, Max_Length))

Find the titles of all movies by a director released before a certain year.

SQL:
```
SELECT     Title
FROM   MOVIE M, DIRECTOR  D, DIRECTED DD
WHERE   M.M_ID = DD.M_ID  AND  D.Dr_ID = DD.Dr_ID
HAVING    M.YEAR < 2012;
```

Relational Algerba:
$Mov\_Dir \leftarrow Director \bowtie DIRECTOR = MovID(Movie)$
$\pi title(\sigma\ Year > 2012\ (Mov\_Dir))$

Find the availability of a Media item with a certain media ID.

SQL:
```
SELECT   Availability
FROM   MEDIA M
WHERE   M.Media_ID = '000000000000001';
```

Relational Algebra:
π availability (σ(media_ID= 000000000000001) (Media))

Advanced Queries:

Provide a list of patron names, along with the total combined running time of all the movies they have checked out.

SQL:
```
SELECT Fname, Lname, COUNT(Title) * Length AS Runtime
FROM MOVIE AS MO, CheckedOutItems AS CO, CARD_HOLDER AS CH
WHERE CO.Media_ID = MO.Media_ID AND CO.L_ID = CH.L_ID
GROUP BY Fname
ORDER BY COUNT(*) DESC;
```

Relational Algebra:
```
Checked_Movies←CheckedOutItems⋈Media_ID=Media_ID(MOVIE)
Movie_holder←Checked_Movies⋈l_ID=L_ID(CARD_HOLDER)
NUM← ℱCOUNT title(Movie_holder) * length(movie_holder) (movie_holder)
Π Fname, Lname, NUM(Movie_holder)
```

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

SQL:
```
SELECT Fname, Lname, EMAIL
FROM CARD_HOLDER AS CH, ALBUM AS A, CheckedOutItems AS CO
WHERE CH.L_ID = CO.L_ID AND A.Media_ID = CO.Media_id AND
(SELECT NUM > AVG(NUM) AS Average
FROM (SELECT COUNT(*) AS NUM
FROM ALBUM AS A, CheckedOutItems AS CO
WHERE CO.Media_ID = A.Media_ID
GROUP BY Title
ORDER BY COUNT(*) DESC))
GROUP BY Fname;
```

Relational Agebra:
```
Checked_Albums←CheckedOutItems⋈Media_ID=Media_ID(ALBUM)
NUM← ℱCOUNT title(Checked_Albums)
GAVG ← ℱNUM > ℱAVG(NUM)
Checked_Patrons←CheckedOutItems⋈L_ID=L_ID(CARD_HOLDER)
Albums_Checked←Checked_Patrons⋈Media_ID=Media_ID(ALBUM)
Average_Albums←Albums_Checked⋈Media_ID=Media_ID(GAVG)
Π Fname,Lname (Average_Albums)
```

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.

SQL:
```
SELECT Title, COUNT(*) AS NUM
FROM MOVIE AS MO, CheckedOutItems AS CO
```

```
WHERE CO.Media_ID = MO.Media_ID
GROUP BY Title
ORDER BY COUNT(*) DESC;
```

Relational Algebra:
Checked_Movies←CheckedOutItems⋈Media_ID=Media_ID(MOVIE)
NUM← ℱCOUNT Media_ID(Checked_Movies)
Π Title,NUM (Checked_Movies)

Provide 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.

SQL:

```
SELECT Title, COUNT(*) AS NUM
FROM ALBUM AS A, CheckedOutItems AS CO
WHERE CO.Media_ID = A.Media_ID
GROUP BY Title
ORDER BY COUNT(*) DESC;
```

Relational Algebra:
Checked_Albums←CheckedOutItems⋈Media_ID=Media_ID(ALBUM)
NUM← ℱCOUNT Media_ID(Checked_Albums)
Π Title, NUM (Checked_Albums)

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

SQL:

```
Fname, Lname
FROM ACTOR AS A, FEATURES AS F, CheckedOutItems AS C, MOVIE AS M
WHERE A.Act_ID = F.Act_ID AND F.M_ID = M.M_ID AND M.Media_ID = C.Media_ID
GROUP BY Fname
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Relational Algebra:
F_Actors←ACTORS⋈Act_ID=Act_ID(FEATURES)
L_Actors←F_Actors⋈M_ID=M_ID(MOVIE)
Actors_checked←L_Actors ⋈Media_ID=Media_ID(CheckedOutItems)
Π Fname, Lname (Actors_checked)

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)

SQL:
```
SELECT AR.Fname, AR.Lname
FROM ARTIST AS AR, CheckedOutItems AS C, ALBUM AS A, TRACK AS T
WHERE A.Art_ID = AR.Art_ID AND C.Media_ID = A.Media_ID AND T.Art_ID = A.Art_ID
GROUP BY AR.Fname
HAVING MAX(T.Length)
ORDER BY COUNT(*) DESC
LIMIT 1;
```

Relational Algebra:
Album_writer←ALBUM⋈Art_ID=Art_ID(ARTIST)
Checked_Albums←Album_writer⋈Media_ID=Media_ID(CheckedOutItems)
Album_Tracks←Checked_Albums⋈ Art_ID=Art_ID(ALBUM)
Max_length← 𝓕MAX length(Album_Tracks)
Π Fname, Lname (Max_length)

Provide a list of customer information for patrons who have checked out anything by the most watched actors in the database.

SQL:
```
SELECT Fname, Lname, Email, Age, L_ID
FROM CARD_HOLDER AS CH, ACTOR AS A, FEATURES AS F, CheckedOutItems AS C,
MOVIE AS M
WHERE CH.L_ID = C.L_ID AND A.Act_ID = F.Act_ID AND F.M_ID = M.M_ID AND
M.Media_ID = C.Media_ID AND A.Fname =
(SELECT Fname
FROM ACTOR AS A, FEATURES AS F, CheckedOutItems AS C, MOVIE AS M
WHERE A.Act_ID = F.Act_ID AND F.M_ID = M.M_ID AND M.Media_ID = C.Media_ID
GROUP BY Fname
ORDER BY COUNT(*) DESC
LIMIT 1);
```

Relational Algebra:
F_Actors←ACTORS⋈Act_ID=Act_ID(FEATURES)
L_Actors←F_Actors⋈M_ID=M_ID(MOVIE)
Actors_checked←L_Actors ⋈Media_ID=Media_ID(CheckedOutItems)
Patrons_movies←L_Actors ⋈L_ID=L_ID(CARD_HOLDER)
Π Fname, Lname, Email, age, L_ID (Patrons_movies)

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

SQL:

```
SELECT AR.Fname, AR.Lname
FROM ARTIST AS AR, ALBUM AS A, CARD_HOLDER AS CH
WHERE AR.Art_ID = A.Art_ID AND CH.L_ID IN
(SELECT CH.L_ID
FROM CARD_HOLDER AS CH, ALBUM AS A, CheckedOutItems AS CO
WHERE CH.L_ID = CO.L_ID AND A.Media_ID = CO.Media_id AND
(SELECT NUM > AVG(NUM) AS Average
FROM (SELECT COUNT(*) AS NUM
FROM ALBUM AS A, CheckedOutItems AS CO
WHERE CO.Media_ID = A.Media_ID
GROUP BY Title
ORDER BY COUNT(*) DESC))
GROUP BY Fname)
GROUP BY AR.Fname;
```

Relational Algebra

Checked_Albums←CheckedOutItems⋈Media_ID=Media_ID(ALBUM)
NUM← 𝓕COUNT title(Checked_Albums)
GAVG ← 𝓕NUM > 𝓕AVG(NUM)
Checked_Patrons←CheckedOutItems⋈L_ID=L_ID(CARD_HOLDER)
Albums_Checked←Checked_Patrons⋈Media_ID=Media_ID(ALBUM)
Average_Albums←Albums_Checked⋈Media_ID=Media_ID(GAVG)
Artist_Checked←Albums_Checked⋈Art_ID=Art_ID(ARTIST)
Π Fname,Lname (Artist_Checked)

C. INSERT statement syntax for adding new tracks, albums, movies/videos, audiobooks, artists and patrons to your system. If there are dependencies in your system that require multiple records to be added to tables in a specific order to add one of these items, make sure you clearly indicate what those restrictions are.

General Note About Writing Insert Statements

Remember when inserting any values into a table follow this form:

```
    INSERT INTO (TABLE_NAME)
    VALUES('val_1','val_2',…,'val_n'); <------- Remember to end Queries with a Semicolon.
```

Albums and Tracks

The Track Table contains two foreign keys A_ID and Art_ID, these are the identifiers for the artist of the song and the album which it belongs to. Therefore, to insert a Track into the database it must include values for A_ID and Art_ID that already exist. The Album Table similarly contains foreign keys for Art_ID and Media_ID. The Tables that contain these values are Artist and Media, meaning you must have existing values in these two tables to make a valid Album insert. Due to neither table supporting any foreign keys you may create either one first. However, we recommend creating the artist first as you cannot have a song or album without an artist to make it.

When Inserting Albums and Tracks into the database, first begin by Inserting a new entry into the Artist Table, create a unique 10-digit Art_ID, followed by the Artist First and Last Name for the next two columns. EX) (‘1234567891’, ‘John’, ‘Doe’). Next create the Media insert, it should be entered in the following order, unique 15-character number for the Media_ID, The Album’s form (Physical / Digital), Album Title, Location, Availability, Copies. EX) (‘12345678912345’, Digital, Cooking Beans II, Columbus, Not Available, 1). Once corresponding Media and Artist exist you may create and an Album Entry. Form: Media_ID, A_ID, Genre, Title, Year, Length, Art_ID. EX) (‘12345678912345’, ‘0101010101’, ‘Soul’, ‘Cooking Beans II’, ‘2021’, ‘7’, ‘1234567891’) and Media Entry have been created you can add tracks to the Album. To Insert the Track, follow this form. T_ID, A_ID, Art_ID, Genre, Track Title, Year of Release, Length(Min:Sec). EX (‘1010101010’, ‘0101010101’, ‘1234567891’, ‘Soul’, ‘Beans in the Pot’, ‘2020’,’2:34’).

```
    INSERT INTO ARTIST
    VALUES('Art_ID’, 'First Name', 'Last Name’);
    INSERT INTO MEDIA
```

```
VALUES('Media_ID’, 'Form’, ‘Title’, 'Location’, ‘Availability’, ‘Copies’);
INSERT INTO ALBUM
VALUES('Media_ID’, 'A_ID', 'Genre’, ‘Album Title’, 'Year’, '# of Tracks’, 'Art_ID’);
INSERT INTO TRACK
VALUES('T_ID’, 'A_ID’, 'Art_ID’, 'Genre’, ‘Track Title’, 'Year’,’Length’);
```

Movies

For Inserting Movies into the Database there is a bit of freedom with the order in which you insert all required Items. To be able to insert a Movie into the Table it requires an existing Media_ID. If a Media_ID in the Media Table matches the value entered for Media_ID in your Movie insert it will execute properly. However, you logically can’t have a Movie without a Director, so it is recommended to create an entry in the Director Table first, then create a Media Entry, next the Movie Entry and finally add the Director and Movie to the DIRECTED Table.

```
INSERT INTO DIRECTOR
VALUES ('Dr_ID’, 'First Name’, 'Last Name’);
INSERT INTO MEDIA
VALUES ('Media_ID’, 'Form’, 'Title’, ‘Location’, 'Availability’, '1');
INSERT INTO MOVIE
VALUES (‘Media_ID', 'M_ID’, ‘Genre’, 'Title’, 'Year’, 'Length',
'PG13');
INSERT INTO DIRECTED
VALUES (‘M_ID’, ‘Dr_ID’);
```

Audiobooks

The Audiobook Table contains two foreign keys and one primary key, (Media_ID, Auth_ID) and (AB_ID). Thus, create entries in the Author and Media Tables relating to the book you want to add to the database. First insert the Author of the Audiobook, then the Media Entry and finally the Audiobook itself.

```
INSERT INTO AUTHOR
VALUES(Auth_ID, First Name, Last Name);
INSERT INTO MEDIA
```

```
VALUES(Media_ID, Form, Title, Location, Availability, Copies);
INSERT INTO AUDIOBOOK
VALUES(Media_ID, AB_ID, Genre, Title, Year, Length, Chapters, Auth_ID);
```

Artist

Entering Artist Entries into the database is quite simple. All you need to enter is a 10-digit unique Art_ID for the artist as its primary key, and the Artist's first and last name.

```
INSERT INTO ARTIST
VALUES('Art_ID', 'First Name', 'Last Name');
```

Patrons

To add a Patron to the Database, First enter an insert for the Library Card Entry, followed by inserting in data for the Card_Holder Table. Once you have two corresponding entries linked by a common L_ID value a new Patron has successfully been added to the database.

```
INSERT INTO LIBRARY_CARD
VALUES('L_ID', 'Late Fees');
INSERT INTO CARD_HOLDER
VALUES('Fname', 'Lname', 'Email', 'Age', 'L_ID' );
```

d. DELETE statement syntax for removing tracks, albums, movies/videos, audiobooks, artists and patrons from your system. Again, indicate any dependencies that exist on the order that the steps in your DELETE must take. In addition, provide an example set of DELETE statements for each entity in your database.

General DELETE Syntax

DELETE FROM *table_name* WHERE *condition*;

Actor

To remove an Actor

```
DELETE FROM ACTOR
WHERE Act_ID = ‘1112223334’;
```

Album

To remove an Album you will need the A_ID and Media_ID of the Album. First remove all Tracks contained in the Album, next the Album itself, and then its media entry.

```
DELETE FROM TRACK
WHERE TRACK.ART_ID = ‘10-digit Number’;

DELETE FROM Album
WHERE A_ID = ‘10-digit Number’;
DELETE FROM  MEDIA
WHERE Media_ID = ‘15-Digit Number’;
```

Artist

To remove an artist from the database, first remove all related media connected to said artist. Remove Track(s), Album(s), and then the Artist itself.

```
DELETE FROM TRACK
WHERE TRACK.ART_ID = ‘10-digit Number’;

DELETE FROM ALBUM

WHERE ALBUM.ART_ID = '10-digit Number’;

DELETE FROM MEDIA
WHERE Media_ID = '15-digit Number’;
DELETE FROM ARTIST
WHERE Art_ID = ‘10-digit Number’;
```

Audiobook

To Delete an Audiobook you will need the audiobook’s AB_ID and its Media ID.

Delete the Audiobook and then delete it from the Media Table.

```
DELETE FROM AUDIOBOOK
WHERE AB_ID = ‘10-digit Number’;
DELETE FROM  MEDIA
WHERE Media_ID = ‘15-Digit Number’;
```

Author

To remove an artist from the database you will need the B_ID from books and/or AB_ID from audiobooks written by the author, the Author’s ID, and the Media_ID of the books and audiobooks. First remove all related media connected to said author, then Delete Book(s) and/or Audiobook(s) related to the Author, and their corresponding media entry or entries. Then finally remove the Author.

```
DELETE FROM BOOK
WHERE B_ID = ‘10-Digit Number’;
DELETE FROM  MEDIA
WHERE Media_ID = ‘15-Digit Number’;
DELETE FROM AUTHOR
WHERE Auth_ID = '10-Digit Number’;
```

Book

To remove a book entry you will need the B_ID and Media_ID of the book you want to remove. Delete the Book and then remove its entry in the Media Table.

```
DELETE FROM BOOK
WHERE B_ID = '10-Digit Number’;
DELETE FROM  MEDIA
WHERE Media_ID = ‘15-Digit Number’;
```

Card_Holder

```
DELETE FROM CARD_HOLDER
WHERE L_ID = '10-Digit Number';
```

CheckedOutItems

To Remove a checked-out item you must supply two values as the Primary Keys are Media_ID and L_ID

```
DELETE FROM CheckedOutItems
WHERE Media_ID = ‘15-Digit Number’ AND L_ID =’10-Digit Number’;
```

Directed

To remove an entry in the directed table supply both the Director ID and Movie ID

```
DELETE FROM DIRECTED
WHERE Dr_ID = '10-Digit Number ' AND M_ID = ‘10-Digit Number’;
```

Director

```
DELETE FROM DIRECTOR
WHERE Dr_ID = ‘10-Digit Number’;
```

Features

To delete an actor’s feature appearance, supply the Movie ID and Actor ID.

```
DELETE FROM FEATURES
WHERE Dr_ID = ‘10-Digit Number’ AND M_ID = ‘10-Digit Number’;
```

Library_Card

To remove a library card from the database you will need the card’s L_ID. First remove the card holder associated with the card, then remove the card itself.

```
DELETE FROM CARD_HOLDER
WHERE L_ID = '10-Digit Number';
```

```
DELETE FROM LIBRARY_CARD
WHERE L_ID = '10-Digit Number ';
```

Media

```
DELETE FROM  MEDIA
WHERE Media_ID = ‘15-Digit Number’;
```

Movie

To Remove a Movie from the database you will need the Movie’s M_ID, Media_ID, the Director’s Dr_ID and the Act_ID of any Actors featured in the film. First delete the entry in the Directed Table related to the movie and any entries in the Features table related to the film, next delete the movie from the Movie Table, and then remove its entry in Media.

```
DELETE FROM DIRECTED
WHERE Dr_ID = '10-Digit Number ' AND M_ID = ‘10-Digit Number’;
DELETE FROM FEATURES
WHERE M_ID = '10-Digit Number ‘AND Act_ID = '10-Digit Number ';
DELETE FROM MOVIE
WHERE M_ID = ‘10-Digit Number’;
DELETE FROM MEDIA
WHERE Media_ID = ‘15-Digit Number’;
```

Ordered_Media

To delete an entry from Ordered Media you will need the Title and the Media_ID.

```
DELETE FROM ORDERED_MEDIA
WHERE Title = ‘Title of Item’ And Media_ID = ‘15-digit Number’;
```

Track

To remove a Track from the database you will need the T_ID of the Track.

```
DELETE FROM Track
WHERE T_ID = ‘10-digit Number’;
```

Syntax to Clear All Tables, input in this order.

```
DELETE FROM DIRECTED;
DELETE FROM FEATURES;
DELETE FROM BOOK;
DELETE FROM TRACK;
DELETE FROM ALBUM;
DELETE FROM AUDIOBOOK;
DELETE FROM MOVIE;
DELETE FROM CARD_HOLDER;
DELETE FROM CheckedOutItems;
DELETE FROM LIBRARY_CARD;
DELETE FROM ORDERED_MEDIA;
DELETE FROM ACTOR;
DELETE FROM ARTIST;
DELETE FROM AUTHOR;
DELETE FROM DIRECTOR;
DELETE FROM MEDIA;
```

Section 3 – Program (software code)

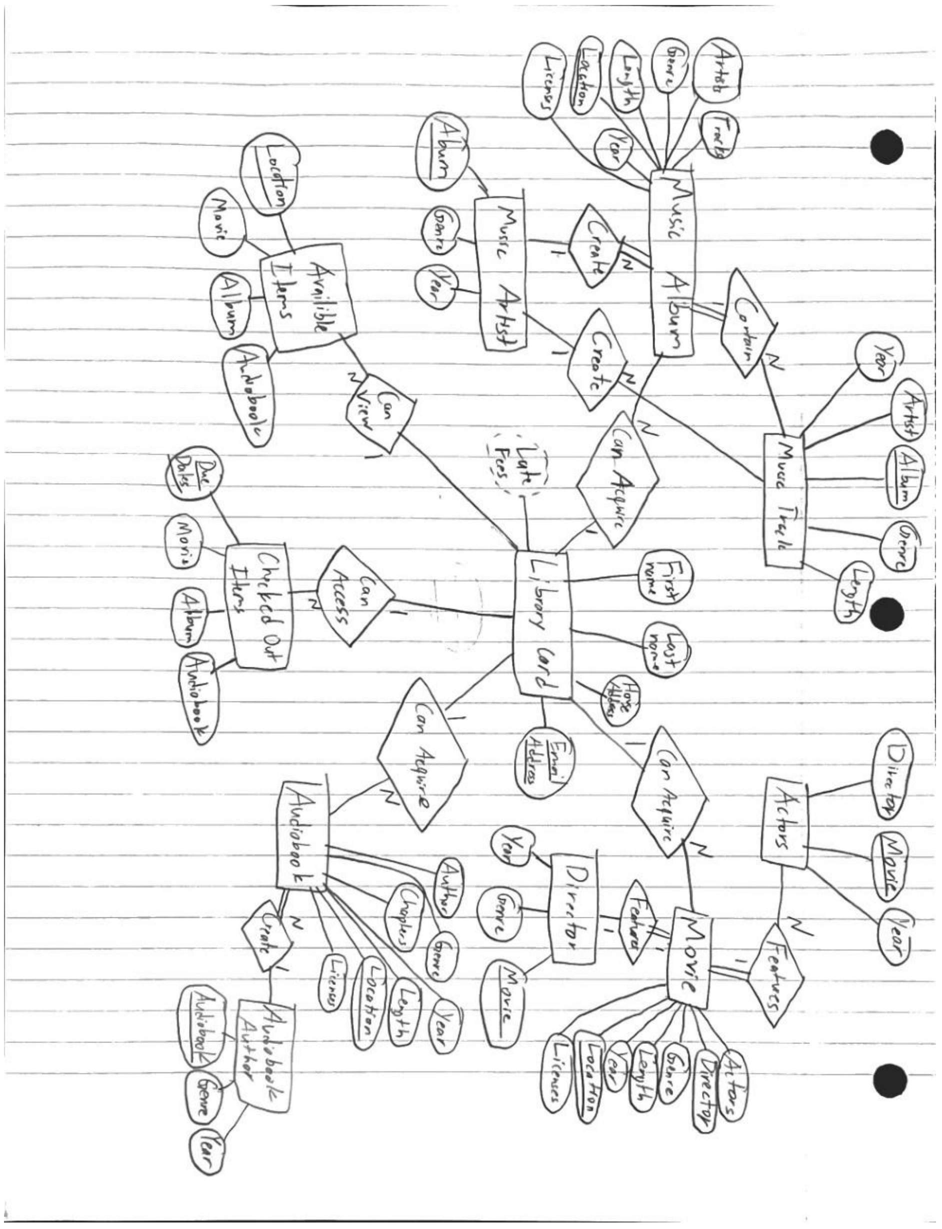
See attached file: DatabaseProject.java

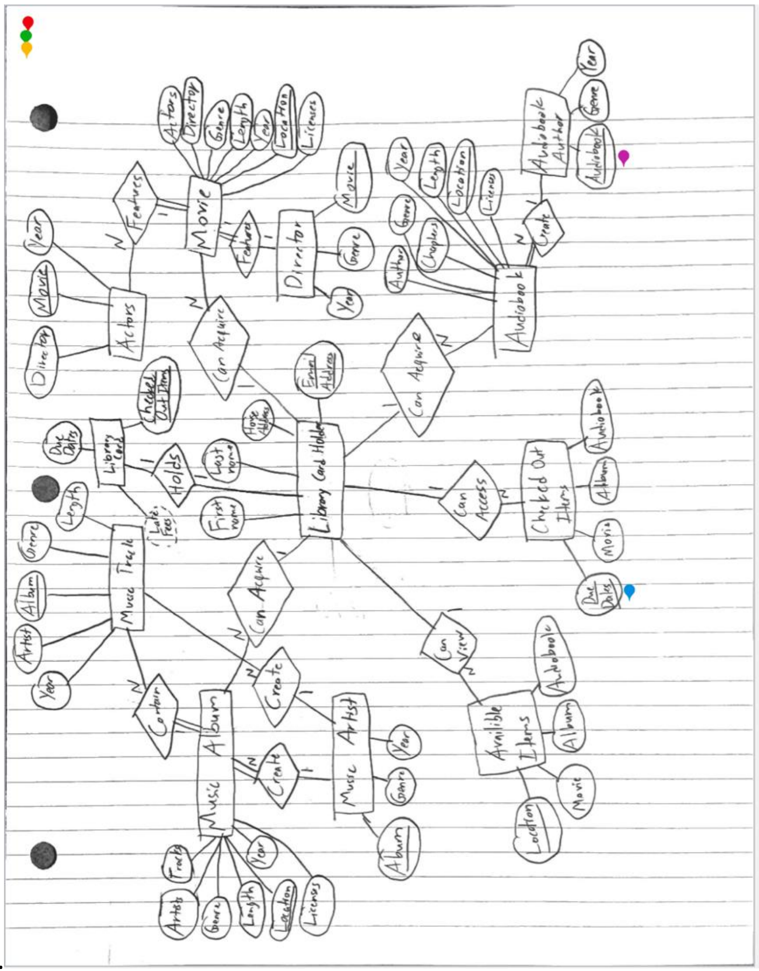
Appendix – Graded Checkpoint Documents

Graded Checkpoints

Checkpoint 1

In the first checkpoint the group just gave a rough draft of the ER diagram. It started off with the library card holder in the middle, who had the ability to acquire audiobooks, movies and albums. The card holder also held a library card which holds information on his checkouts, and the card holder can also access to see which items are checked out and which items are available.





Matthew Moriarty (moriarty.73)

Please type this next time using Draw IO, Powerpoint, Word, or something else. This is a little hard to read.

Matthew Moriarty (moriarty.73)

This image is also sideways and I can't rotate it, so please submit an upright one next time.

Matthew Moriarty (moriarty.73)

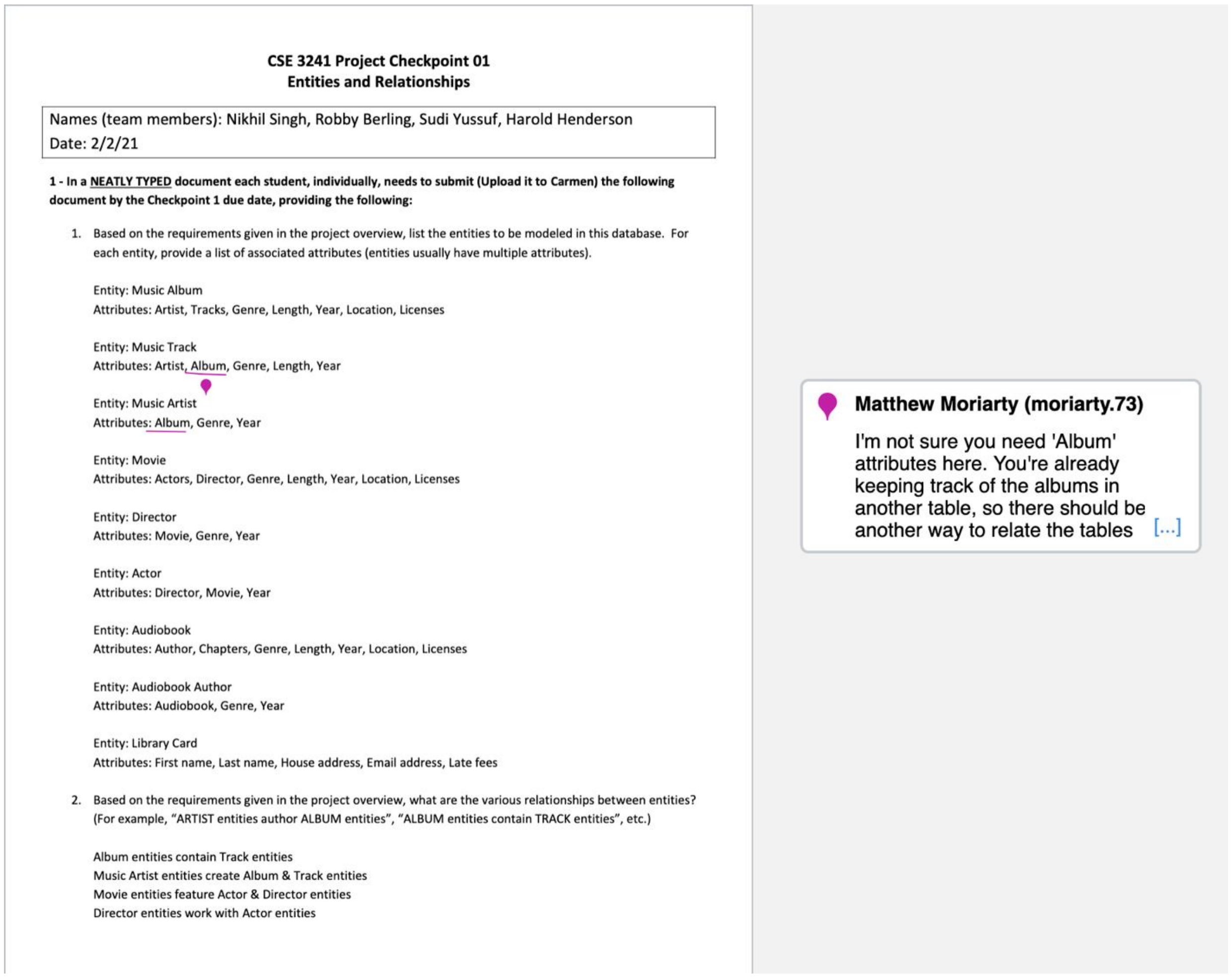
I think you need to look a little bit more into the attributes for each entity. It will be hard to add to certain entities in here without adding to other ones. For instance, see the [...]

Matthew Moriarty (moriarty.73)

What if you want to add an author without adding any audiobooks? What will this primary key value be?

Matthew Moriarty (moriarty.73)

Is 'due dates' unique here? I don't think it would be able to be. This would mean that there can only be one item due on each day, but th
will be several items due each da [...]



CSE 3241 Project Checkpoint 01
Entities and Relationships

Names (team members): Nikhil Singh, Robby Berling, Sudi Yussuf, Harold Henderson
Date: 2/2/21

1 - In a NEATLY TYPED document each student, individually, needs to submit (Upload it to Carmen) the following document by the Checkpoint 1 due date, providing the following:

1. Based on the requirements given in the project overview, list the entities to be modeled in this database. For each entity, provide a list of associated attributes (entities usually have multiple attributes).

   Entity: Music Album
   Attributes: Artist, Tracks, Genre, Length, Year, Location, Licenses

   Entity: Music Track
   Attributes: Artist, Album, Genre, Length, Year

   Entity: Music Artist
   Attributes: Album, Genre, Year

   Entity: Movie
   Attributes: Actors, Director, Genre, Length, Year, Location, Licenses

   Entity: Director
   Attributes: Movie, Genre, Year

   Entity: Actor
   Attributes: Director, Movie, Year

   Entity: Audiobook
   Attributes: Author, Chapters, Genre, Length, Year, Location, Licenses

   Entity: Audiobook Author
   Attributes: Audiobook, Genre, Year

   Entity: Library Card
   Attributes: First name, Last name, House address, Email address, Late fees

2. Based on the requirements given in the project overview, what are the various relationships between entities? (For example, "ARTIST entities author ALBUM entities", "ALBUM entities contain TRACK entities", etc.)

   Album entities contain Track entities
   Music Artist entities create Album & Track entities
   Movie entities feature Actor & Director entities
   Director entities work with Actor entities

Matthew Moriarty (moriarty.73)

I'm not sure you need 'Album' attributes here. You're already keeping track of the albums in another table, so there should be another way to relate the tables [...]

Audiobook Author entities create Audiobook Entities
Library Card Holder entities can acquire Album, Movie, & Audiobook entities

3. Propose at least two additional entities that it would be useful for this database to model beyond the scope of the project requirements. Provide a list of possible attributes for the additional entities and possible relationships they may have with each other and the rest of the entities in the database. Give a brief, one sentence rationale for why adding these entities would be interesting/useful to the stakeholders for this database project.

   Checked out items, Available items, & Library Card.
   Library Card Holders can view Available items.
   Library Card Holders can access Checked Out Items.
   Library Card Holders hold Library Cards.
   These are useful because the make the information accessing part very organized in terms of what information you're looking for.

4. Give at least four examples of some informal queries/reports that it might be useful for this database to generate. Include one example for each of the additional entities you proposed in question 3 above.
   1) A report on all of the checked out items and due dates of a card holder
   2) Report on all of the available items in the library
   3) Query on all of the albums a musical artist has created that are in the library
   4) All of the movies in the library organized by director alphabetically

5. Suppose we want to add a new artist or a new actor to the database. How would we do that given the entities and relationships you've outlined above? Given your above description, is it possible to add a new artist or actor to your database without knowing the title of any albums or movie they have released? If not, revise your model to allow for artists or actors to be added as separate entities.
   If we want to add an artist or actor into our database we could, because we have those objects as entities. These entities all contain multiple attributes.

6. Determine at least three other informal update operations and describe what entities would need to have attributes altered and how they would need to be changed given your above descriptions. Include one example for each of the additional entities you proposed in question 3 above.
   1) We can add directors without having a movie associated with it
   2) Also add audiobook authors without audiobook written
   3) Music tracks not contained in albums can also be made, like a single for example

7. Provide an ER diagram for your database. Make sure you include ALL entities and relationships in question 1, 2, above ***INCLUDING the entities for question 3 also***, and remember that ***EVERY*** entity in your model needs to connect to another entity in the model via some kind of relationship (No entity should be isolated). Consider the direction of each relationship, do not forget about specifying the type of relationship (1 to 1, 1 to N, N to 1, or N to N).

2 – Individual & Team submission: Before the team submission, each student must submit his/her own individual ER to the Carmen dropbox (see ER Selection due date in the calendar); after this, all the members of the team need to meet to discuss which one of the ER models proposed the team will choose one to be submitted as the team ER diagram for the project (the team may do some adjustments/corrections to the final one).

Matthew Moriarty (moriarty.73)
Good!

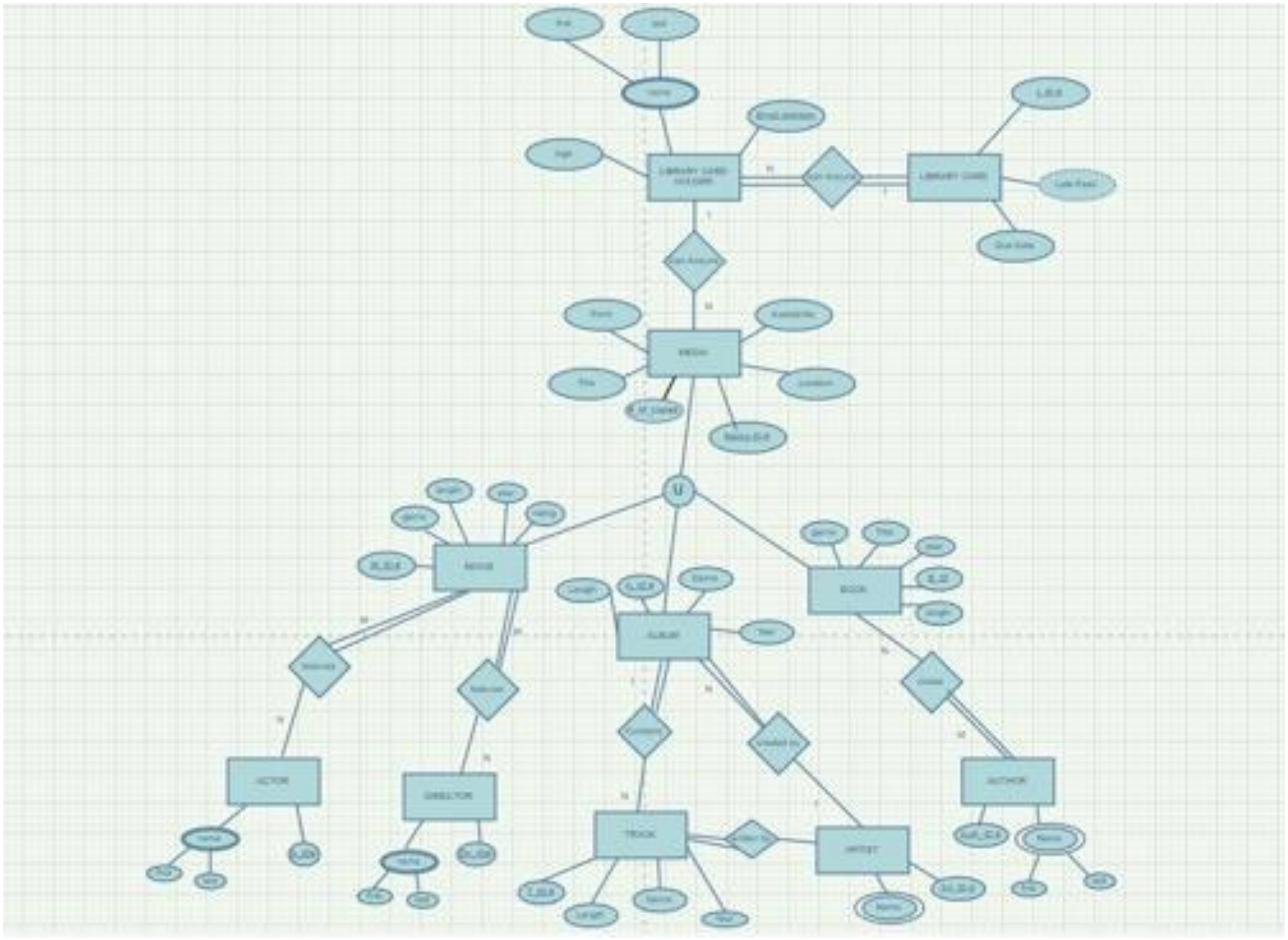
Matthew Moriarty (moriarty.73)
Your 'Music Artist' entity contains a primary key 'Album', though, so you won't actually be able to add an artist without including the primary key, which is the name of an Album.

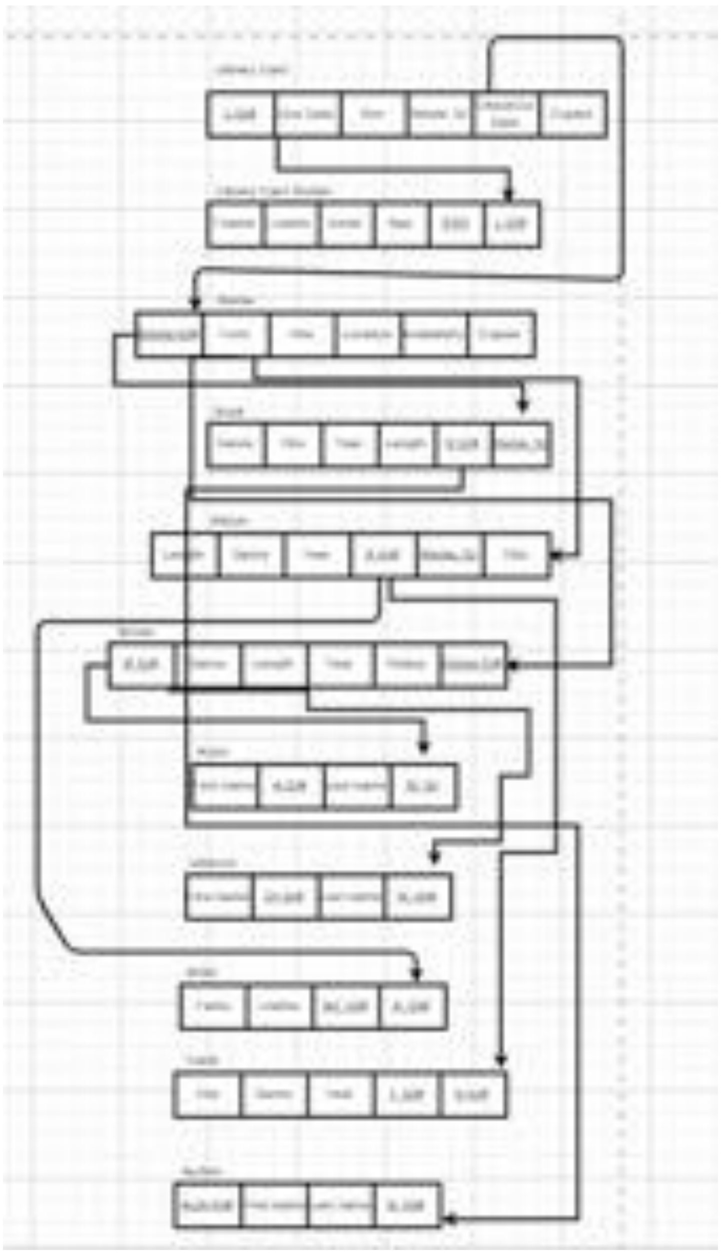
Matthew Moriarty (moriarty.73)
The 'Audiobook Author' entity has a primary key 'Audiobook', so how could you add the author without adding the audiobook?

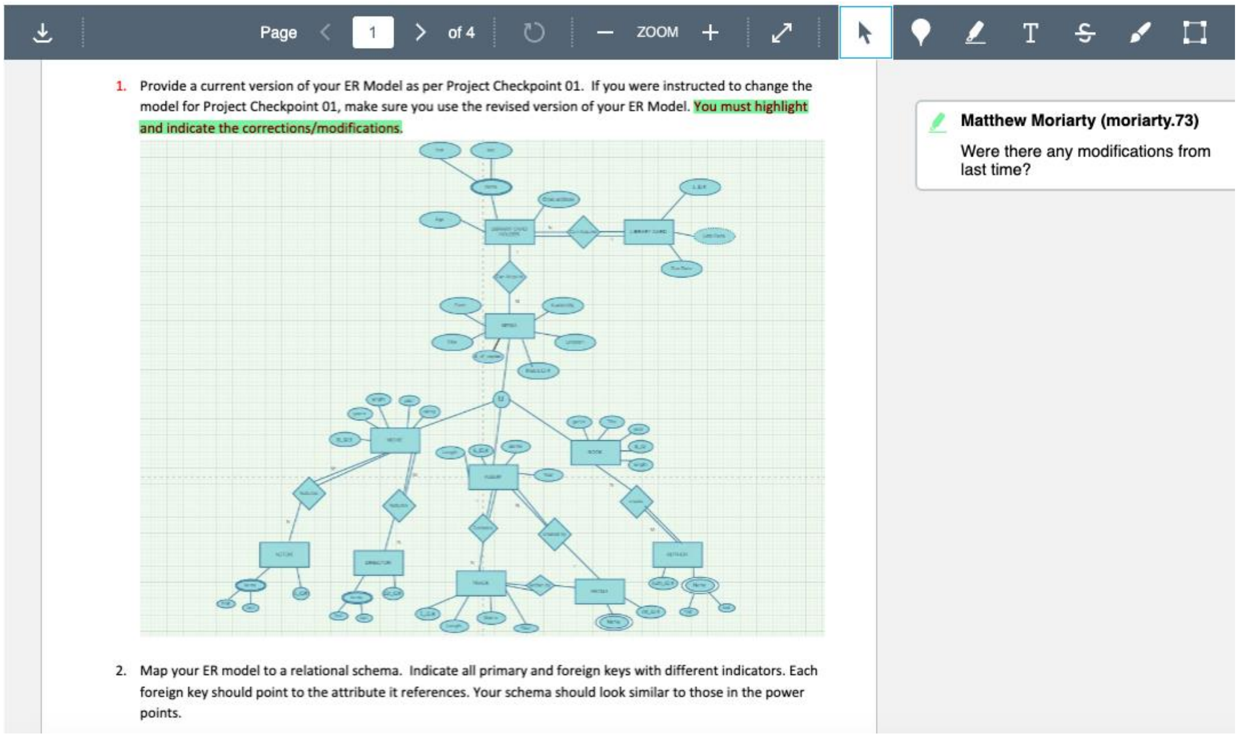
Matthew Moriarty (moriarty.73)
Same thing here. 'Music Track' has primary key 'Album', so how will you add a single that doesn't belong to an album?

Checkpoint 2

Since the last checkpoint, the major addition was the Media entity in the ER diagram, this created a web for the library card to be able to access the movies, albums, and books. For the three-table group of album, track, and artist, we made a triangle to link all of them to each other. The Library Card Holder and Library Card table still have the same philosophy of Media being connected to Library Card Holder and Library Card Holder being connected to Library Card. This is also the first checkpoint where we created the relational model, using the ER diagram.





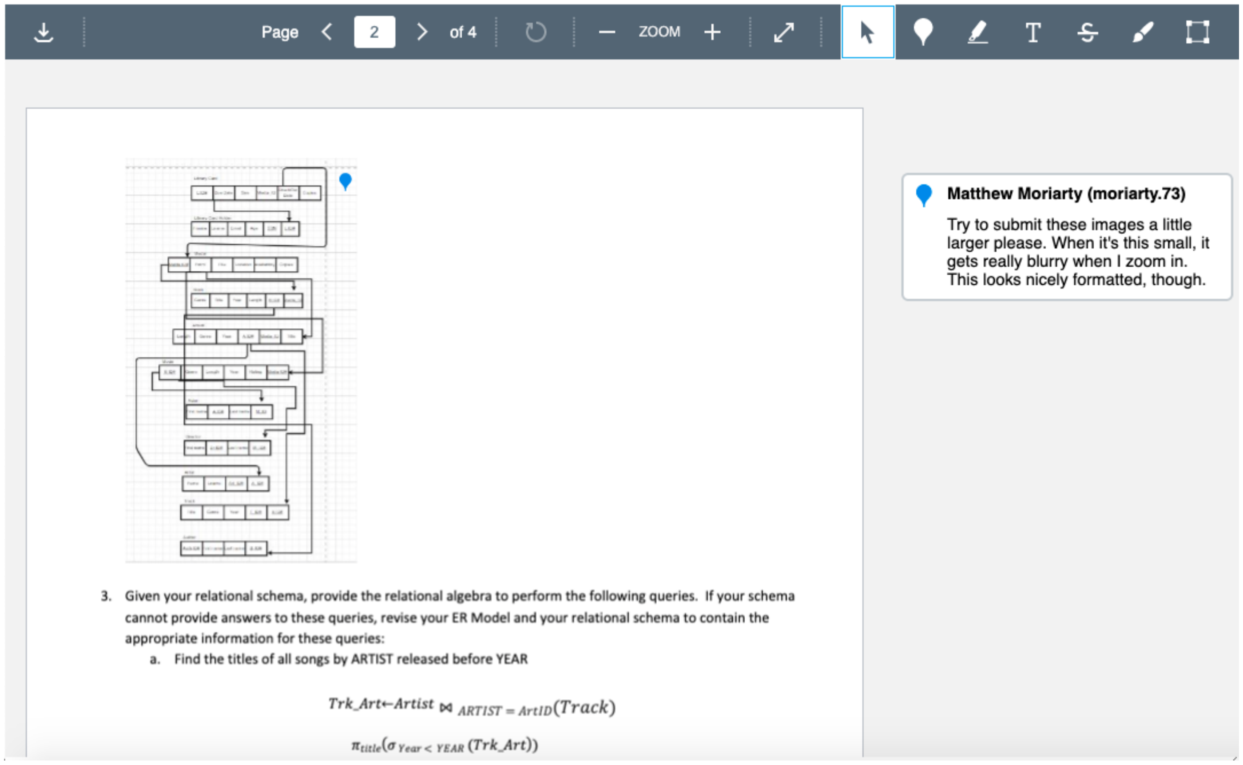


1. Provide a current version of your ER Model as per Project Checkpoint 01. If you were instructed to change the model for Project Checkpoint 01, make sure you use the revised version of your ER Model. You must highlight and indicate the corrections/modifications.

2. Map your ER model to a relational schema. Indicate all primary and foreign keys with different indicators. Each foreign key should point to the attribute it references. Your schema should look similar to those in the power points.

Matthew Moriarty (moriarty.73)

Were there any modifications from last time?



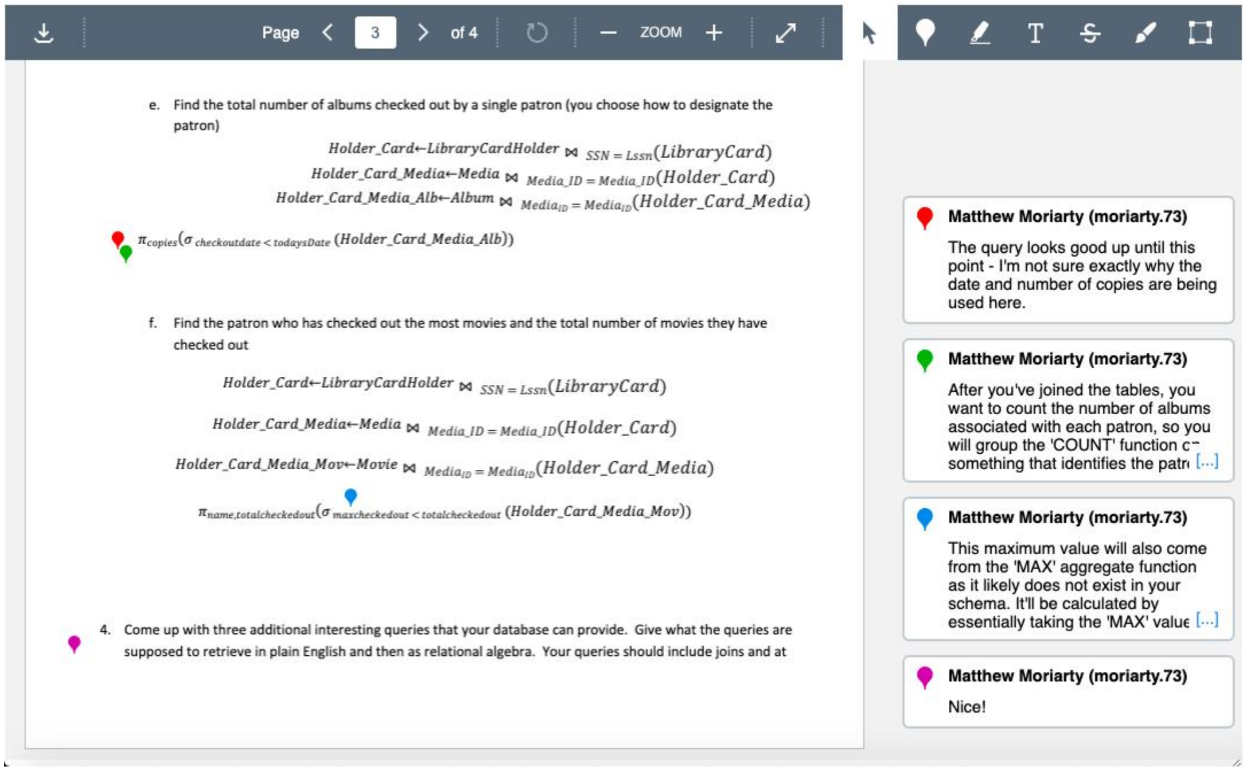
Matthew Moriarty (moriarty.73)

Try to submit these images a little larger please. When it's this small, it gets really blurry when I zoom in. This looks nicely formatted, though.

3. Given your relational schema, provide the relational algebra to perform the following queries. If your schema cannot provide answers to these queries, revise your ER Model and your relational schema to contain the appropriate information for these queries:
   a. Find the titles of all songs by ARTIST released before YEAR

$$Trk\_Art \leftarrow Artist \bowtie_{ARTIST = ArtID}(Track)$$

$$\pi_{title}(\sigma_{Year < YEAR}(Trk\_Art))$$



e. Find the total number of albums checked out by a single patron (you choose how to designate the patron)

$$Holder\_Card \leftarrow LibraryCardHolder \bowtie_{SSN = Lssn}(LibraryCard)$$
$$Holder\_Card\_Media \leftarrow Media \bowtie_{Media\_ID = Media\_ID}(Holder\_Card)$$
$$Holder\_Card\_Media\_Alb \leftarrow Album \bowtie_{Media_{ID} = Media_{ID}}(Holder\_Card\_Media)$$

$$\pi_{copies}(\sigma_{checkoutdate < todaysDate}(Holder\_Card\_Media\_Alb))$$

f. Find the patron who has checked out the most movies and the total number of movies they have checked out

$$Holder\_Card \leftarrow LibraryCardHolder \bowtie_{SSN = Lssn}(LibraryCard)$$
$$Holder\_Card\_Media \leftarrow Media \bowtie_{Media\_ID = Media\_ID}(Holder\_Card)$$
$$Holder\_Card\_Media\_Mov \leftarrow Movie \bowtie_{Media_{ID} = Media_{ID}}(Holder\_Card\_Media)$$
$$\pi_{name,totalcheckedout}(\sigma_{maxcheckedout < totalcheckedout}(Holder\_Card\_Media\_Mov))$$

4. Come up with three additional interesting queries that your database can provide. Give what the queries are supposed to retrieve in plain English and then as relational algebra. Your queries should include joins and at

Matthew Moriarty (moriarty.73)

The query looks good up until this point - I'm not sure exactly why the date and number of copies are being used here.

Matthew Moriarty (moriarty.73)

After you've joined the tables, you want to count the number of albums associated with each patron, so you will group the 'COUNT' function on something that identifies the patr [...]

Matthew Moriarty (moriarty.73)

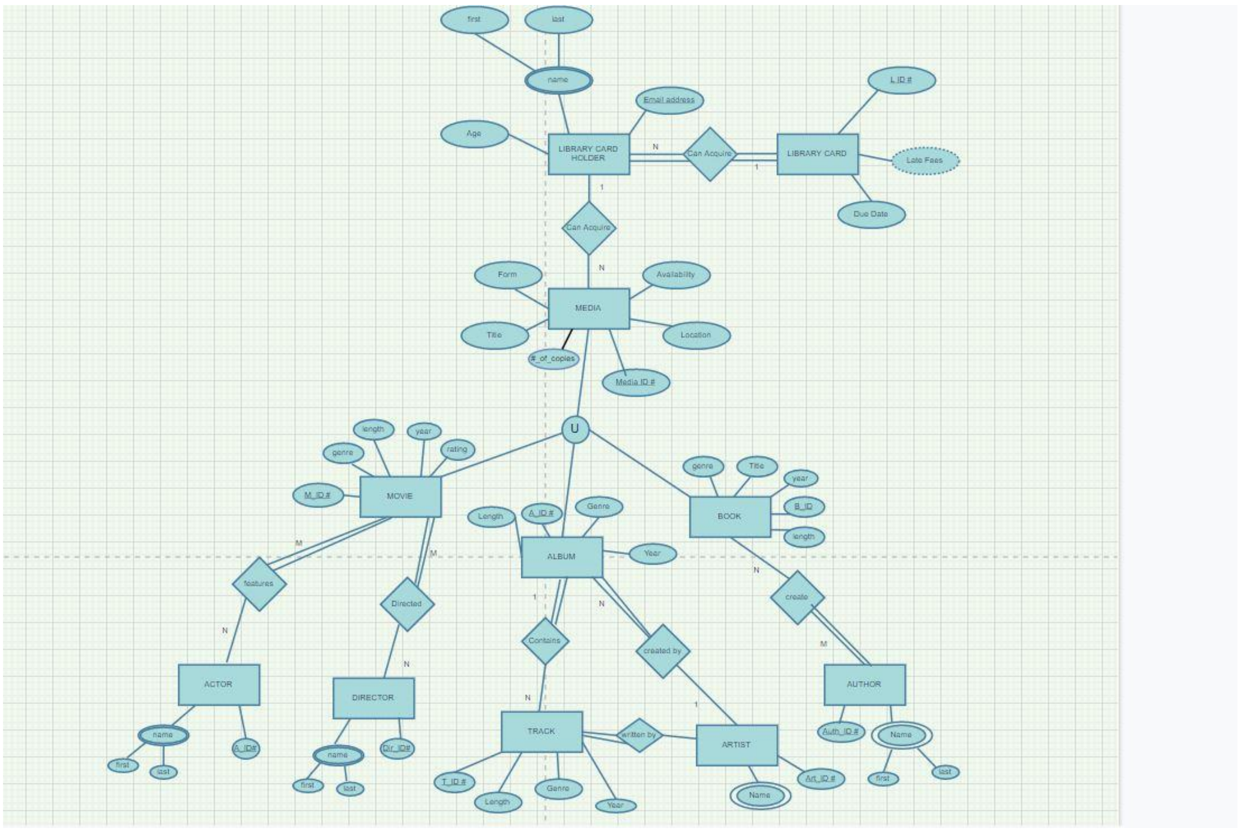
This maximum value will also come from the 'MAX' aggregate function as it likely does not exist in your schema. It'll be calculated by essentially taking the 'MAX' value [...]

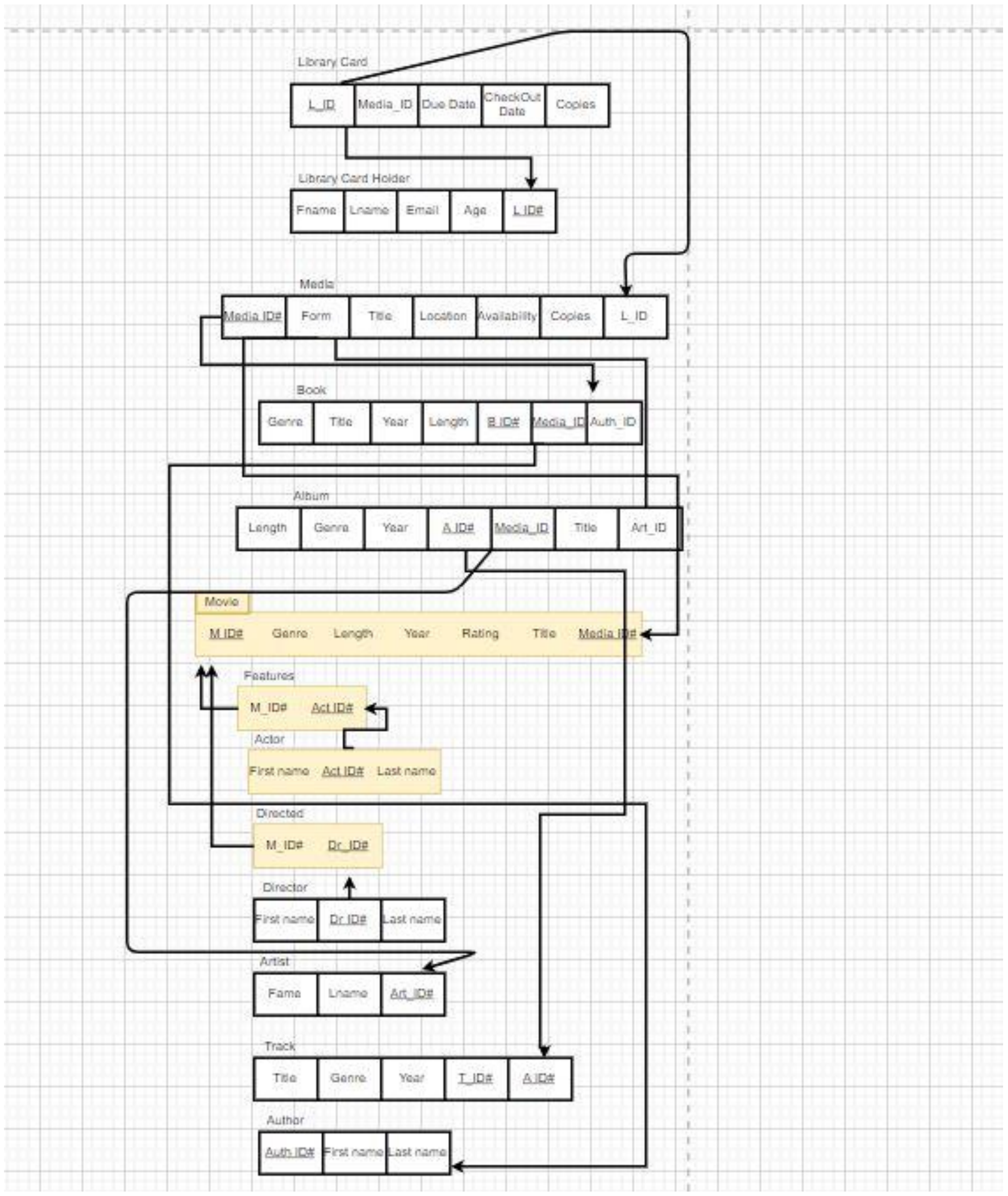
Matthew Moriarty (moriarty.73)

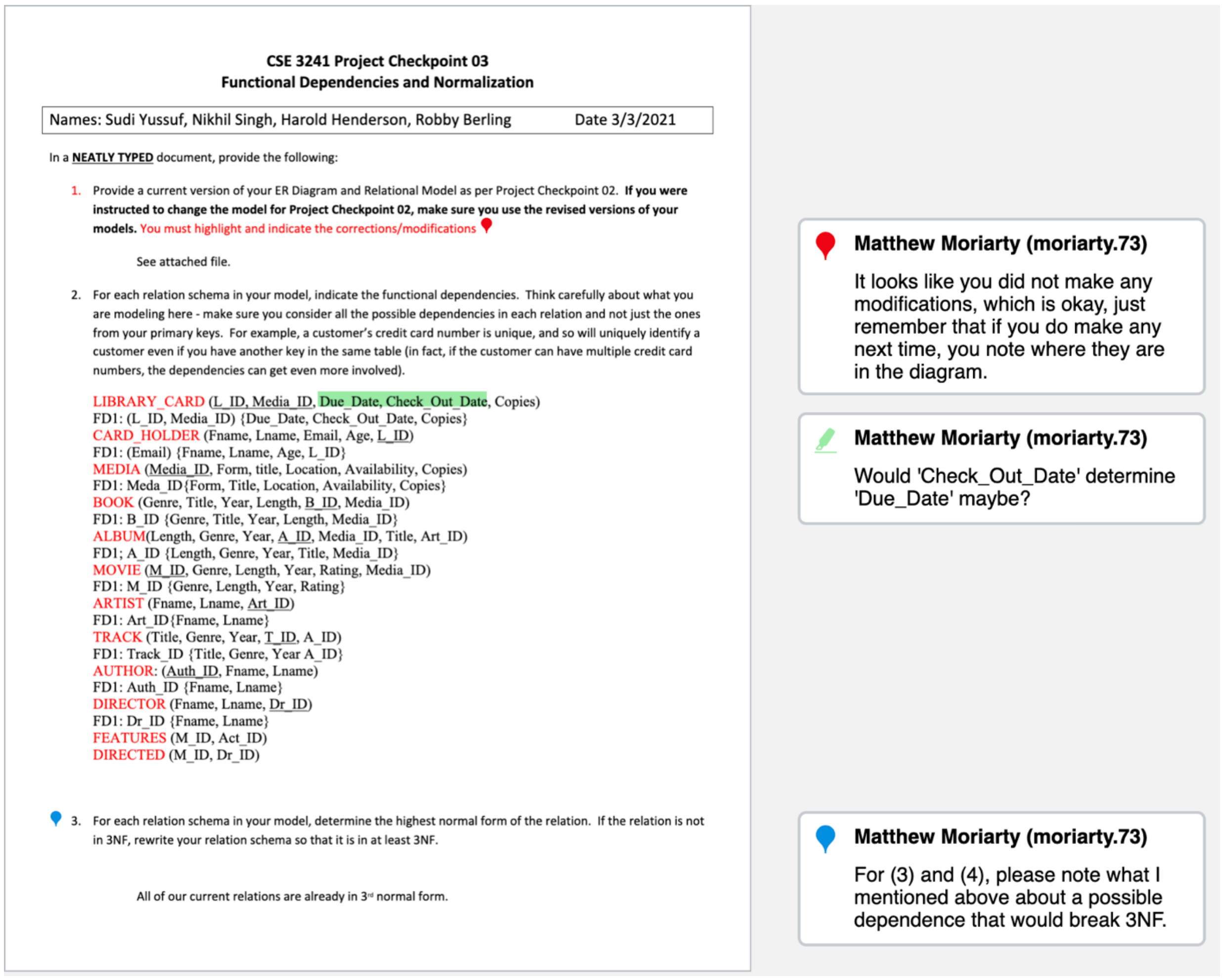
Nice!

Checkpoint 3

From Checkpoint 2 to Checkpoint 3 there wasn't any structural changes to the ER diagram or the relational model, besides altering the album section a little to help each provided ID number to be better referenced and to improve the Album, Track, and Artist triangle.




Library Card
L_ID | Media_ID | Due Date | CheckOut Date | Copies
Library Card Holder
Fname | Lname | Email | Age | L ID#
Media
Media ID# | Form | Title | Location | Availability | Copies | L_ID
Book
Genre | Title | Year | Length | B ID# | Media_ID | Auth_ID
Album
Length | Genre | Year | A ID# | Media_ID | Title | Art_ID
Movie
M ID# | Genre | Length | Year | Rating | Title | Media ID#
Features
M_ID# | Act ID#
Actor
First name | Act ID# | Last name
Directed
M_ID# | Dr_ID#
Director
First name | Dr ID# | Last name
Artist
Fame | Lname | Art_ID#
Track
Title | Genre | Year | T_ID# | A ID#
Author
Auth ID# | First name | Last name



CSE 3241 Project Checkpoint 03
Functional Dependencies and Normalization

Names: Sudi Yussuf, Nikhil Singh, Harold Henderson, Robby Berling	Date 3/3/2021

In a **NEATLY TYPED** document, provide the following:

1. Provide a current version of your ER Diagram and Relational Model as per Project Checkpoint 02. **If you were instructed to change the model for Project Checkpoint 02, make sure you use the revised versions of your models.** You must highlight and indicate the corrections/modifications

   See attached file.

2. For each relation schema in your model, indicate the functional dependencies. Think carefully about what you are modeling here - make sure you consider all the possible dependencies in each relation and not just the ones from your primary keys. For example, a customer's credit card number is unique, and so will uniquely identify a customer even if you have another key in the same table (in fact, if the customer can have multiple credit card numbers, the dependencies can get even more involved).

   LIBRARY_CARD (L_ID, Media_ID, Due_Date, Check_Out_Date, Copies)
   FD1: (L_ID, Media_ID) {Due_Date, Check_Out_Date, Copies}
   CARD_HOLDER (Fname, Lname, Email, Age, L_ID)
   FD1: (Email) {Fname, Lname, Age, L_ID}
   MEDIA (Media_ID, Form, title, Location, Availability, Copies)
   FD1: Meda_ID{Form, Title, Location, Availability, Copies}
   BOOK (Genre, Title, Year, Length, B_ID, Media_ID)
   FD1: B_ID {Genre, Title, Year, Length, Media_ID}
   ALBUM(Length, Genre, Year, A_ID, Media_ID, Title, Art_ID)
   FD1; A_ID {Length, Genre, Year, Title, Media_ID}
   MOVIE (M_ID, Genre, Length, Year, Rating, Media_ID)
   FD1: M_ID {Genre, Length, Year, Rating}
   ARTIST (Fname, Lname, Art_ID)
   FD1: Art_ID{Fname, Lname}
   TRACK (Title, Genre, Year, T_ID, A_ID)
   FD1: Track_ID {Title, Genre, Year A_ID}
   AUTHOR: (Auth_ID, Fname, Lname)
   FD1: Auth_ID {Fname, Lname}
   DIRECTOR (Fname, Lname, Dr_ID)
   FD1: Dr_ID {Fname, Lname}
   FEATURES (M_ID, Act_ID)
   DIRECTED (M_ID, Dr_ID)

3. For each relation schema in your model, determine the highest normal form of the relation. If the relation is not in 3NF, rewrite your relation schema so that it is in at least 3NF.

   All of our current relations are already in 3rd normal form.

Matthew Moriarty (moriarty.73)
It looks like you did not make any modifications, which is okay, just remember that if you do make any next time, you note where they are in the diagram.

Matthew Moriarty (moriarty.73)
Would 'Check_Out_Date' determine 'Due_Date' maybe?

Matthew Moriarty (moriarty.73)
For (3) and (4), please note what I mentioned above about a possible dependence that would break 3NF.

4. For each relation schema in your model that is in 3NF but not in BCNF, either rewrite the relation schema to BCNF or provide a short justification for why this relation should be an exception to the rule of putting relations into BCNF.

   All of our current relations are already in BCNF normal form.

5. Write a Java program (no database are needed at this point). The program will present the library user with a list of options provided below to choose from, and then each option will ask the user for the input needed. You will need to store that data into any Java data structure that you consider more appropriate (ArrayList, Sets, ..) and then be able to retrieve it as an output when other option requires it:

   a. Entering data for a new artist (use the data attributes that you defined for that entity)
   b. Entering data for a new track-song (use the data attributes that you defined for that entity)
   c. Entering data for new media items ordered: type of media (albums, videos and audiobooks), with number of copies purchase, price and an estimated date of arrival.
   d. Retrieve information about one specific artist (provided by the user)
   e. Retrieve information about a track (provided by the user)
   f. Retrieve information about the media items that are ordered by the library (provided by the user)
   g. Editing existing entries for an artist (artist selected by the user)
   h. Deleting an existing track (track selected by the user)

   Notes:

   - All the options must be implemented and tested.
   - The program must compile and execute.
   - The test output for each option must be included
   - The program must run on eclipse so we can test it
   - You don't need user graphical interface. Input and output could be from the console
   - DO NOT create databases at this time, this will be done on another checkpoint.

6. Each team member, individually, needs to fill out the Peer-evaluation form provided and submit it to Carmen.

Please DO NOT zip the report file when you submit so that the grader can give you detailed feedback in Carmen.

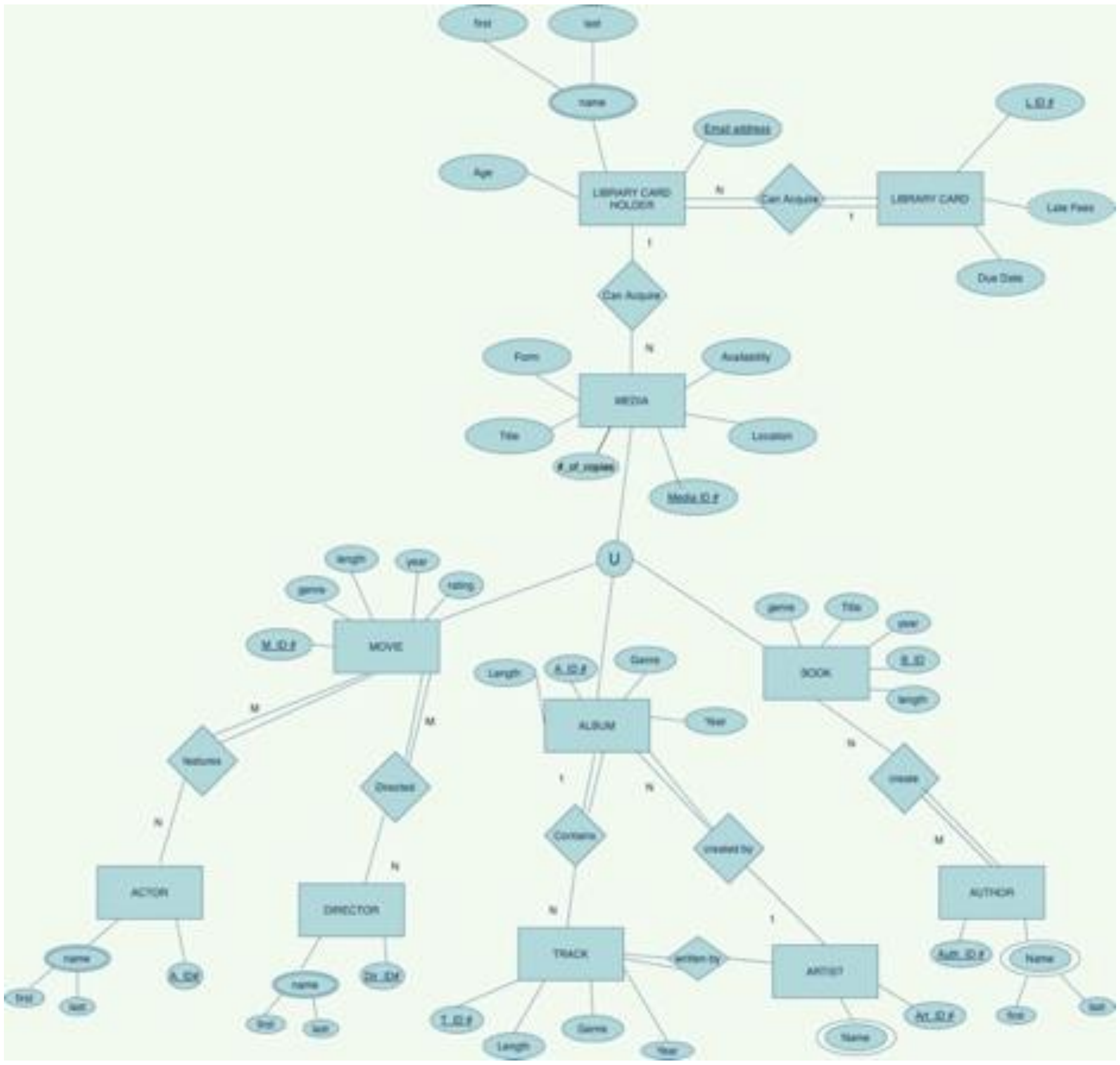
Note: Your submission should include your ER-diagram, the relation schema before this checkpoint (before normalizing), and the relational schema after this checkpoint (after normalizing). Give them clear file name that grader can distinguish 2 versions. Please DO NOT zip your files when submitting, so that the grader can give detailed feedback on Carmen.

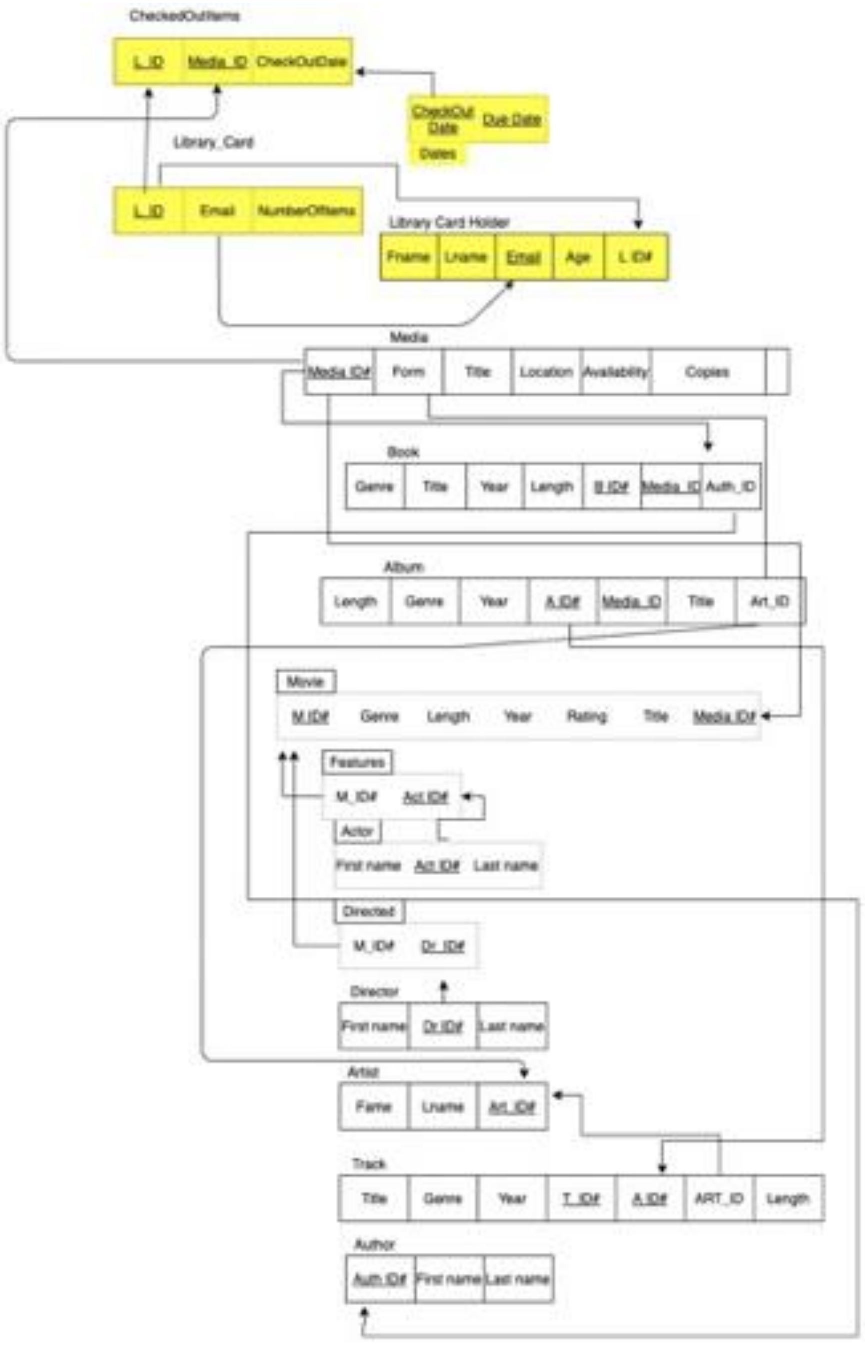
Matthew Moriarty (moriarty.73)

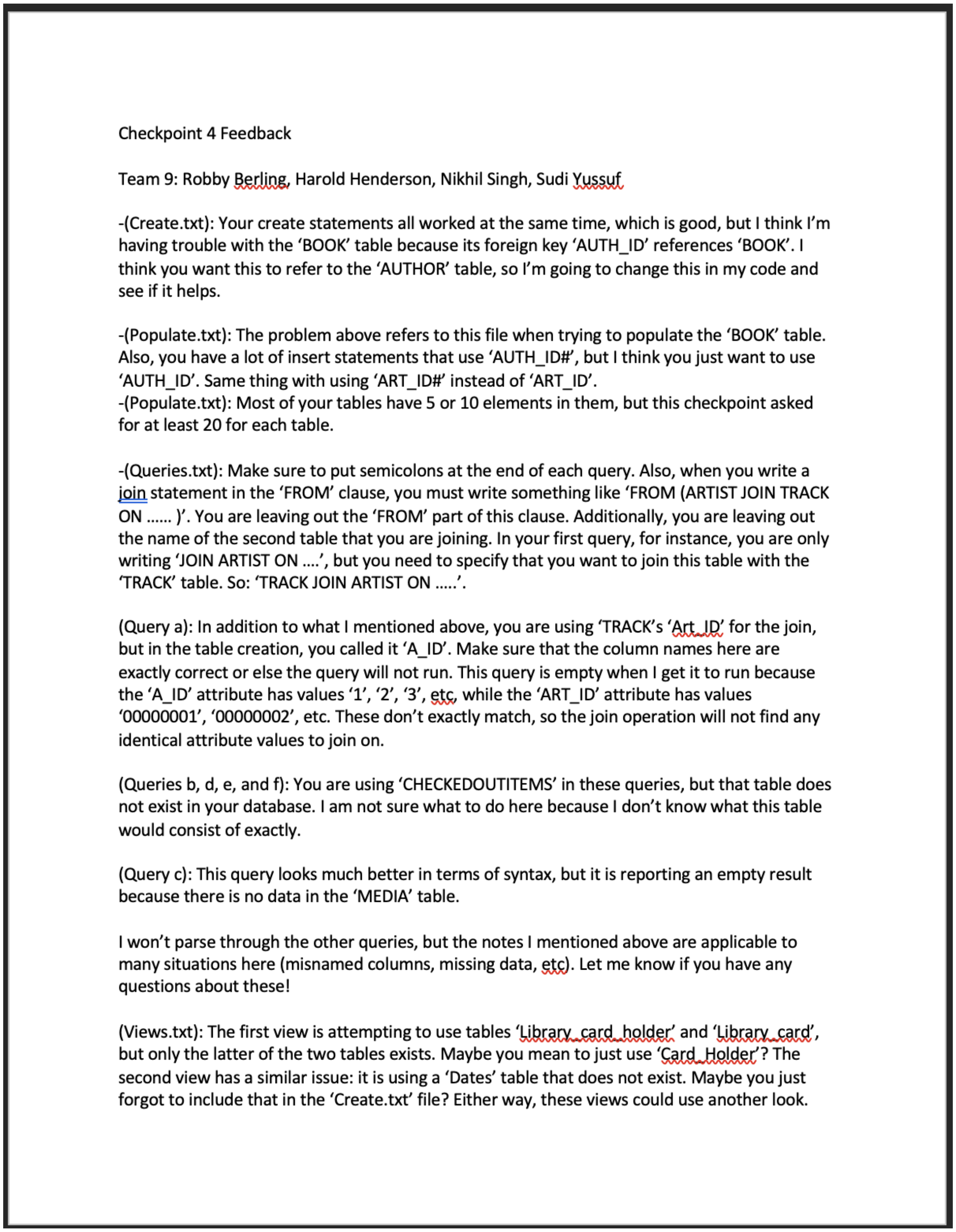
Your Java program looks nice, but next time, please zip it up and submit the entire Java project so that I can run it in Eclipse myself.

Checkpoint 4

The highlighted portion in the relational model is the changes we made from checkpoint 3 to checkpoint 4. The attributes of Library Card Holder and Library Card were changed a little bit to store the proper information needed. CheckedOutItems table was added to the database to be able to access what each Library Card Holder possessed. Also a Dates table was created to be referenced by the CheckedOutItems for determining when something was checked out or due.




CheckedOutItems
L_ID
Media_ID
CheckOutDate
CheckOut Date
Due Date
Dates
Library_Card
L_ID
Email
NumberOfItems
Library Card Holder
Fname
Lname
Email
Age
L ID#
Media
Media ID#
Form
Title
Location
Availability
Copies
Book
Genre
Title
Year
Length
B ID#
Media_ID
Auth_ID
Album
Length
Genre
Year
A ID#
Media_ID
Title
Art_ID
Movie
M ID#
Genre
Length
Year
Rating
Title
Media ID#
Features
M_ID#
Act ID#
Actor
First name
Act ID#
Last name
Directed
M_ID#
Dr ID#
Director
First name
Dr ID#
Last name
Artist
Fame
Lname
Art_ID#
Track
Title
Genre
Year
T_ID#
A ID#
ART_ID
Length
Author
Auth ID#
First name
Last name



Checkpoint 4 Feedback

Team 9: Robby Berling, Harold Henderson, Nikhil Singh, Sudi Yussuf

-(Create.txt): Your create statements all worked at the same time, which is good, but I think I'm having trouble with the 'BOOK' table because its foreign key 'AUTH_ID' references 'BOOK'. I think you want this to refer to the 'AUTHOR' table, so I'm going to change this in my code and see if it helps.

-(Populate.txt): The problem above refers to this file when trying to populate the 'BOOK' table. Also, you have a lot of insert statements that use 'AUTH_ID#', but I think you just want to use 'AUTH_ID'. Same thing with using 'ART_ID#' instead of 'ART_ID'.
-(Populate.txt): Most of your tables have 5 or 10 elements in them, but this checkpoint asked for at least 20 for each table.

-(Queries.txt): Make sure to put semicolons at the end of each query. Also, when you write a join statement in the 'FROM' clause, you must write something like 'FROM (ARTIST JOIN TRACK ON ……)'. You are leaving out the 'FROM' part of this clause. Additionally, you are leaving out the name of the second table that you are joining. In your first query, for instance, you are only writing 'JOIN ARTIST ON ….', but you need to specify that you want to join this table with the 'TRACK' table. So: 'TRACK JOIN ARTIST ON …..'.

(Query a): In addition to what I mentioned above, you are using 'TRACK's 'Art_ID' for the join, but in the table creation, you called it 'A_ID'. Make sure that the column names here are exactly correct or else the query will not run. This query is empty when I get it to run because the 'A_ID' attribute has values '1', '2', '3', etc, while the 'ART_ID' attribute has values '00000001', '00000002', etc. These don't exactly match, so the join operation will not find any identical attribute values to join on.

(Queries b, d, e, and f): You are using 'CHECKEDOUTITEMS' in these queries, but that table does not exist in your database. I am not sure what to do here because I don't know what this table would consist of exactly.

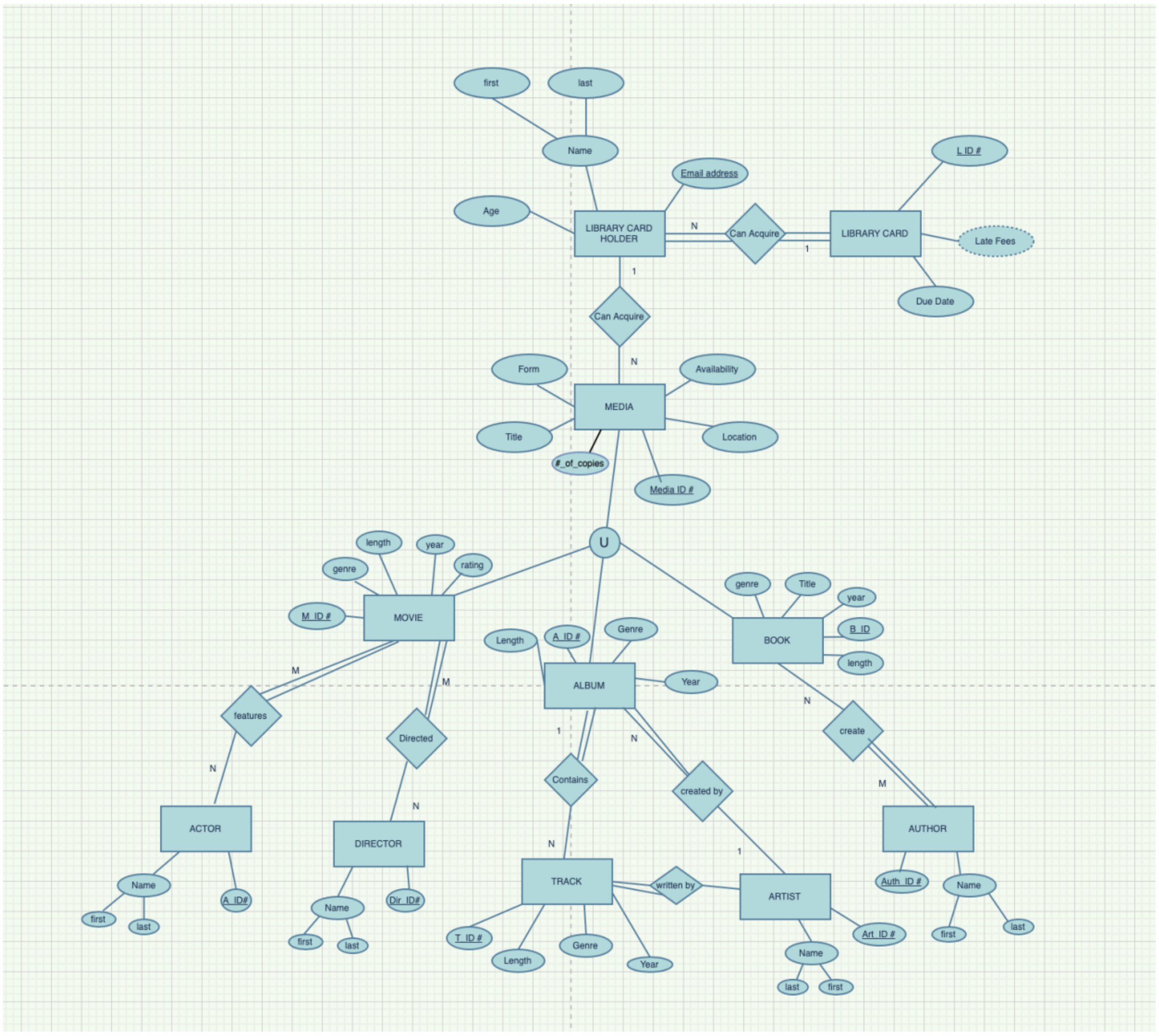
(Query c): This query looks much better in terms of syntax, but it is reporting an empty result because there is no data in the 'MEDIA' table.

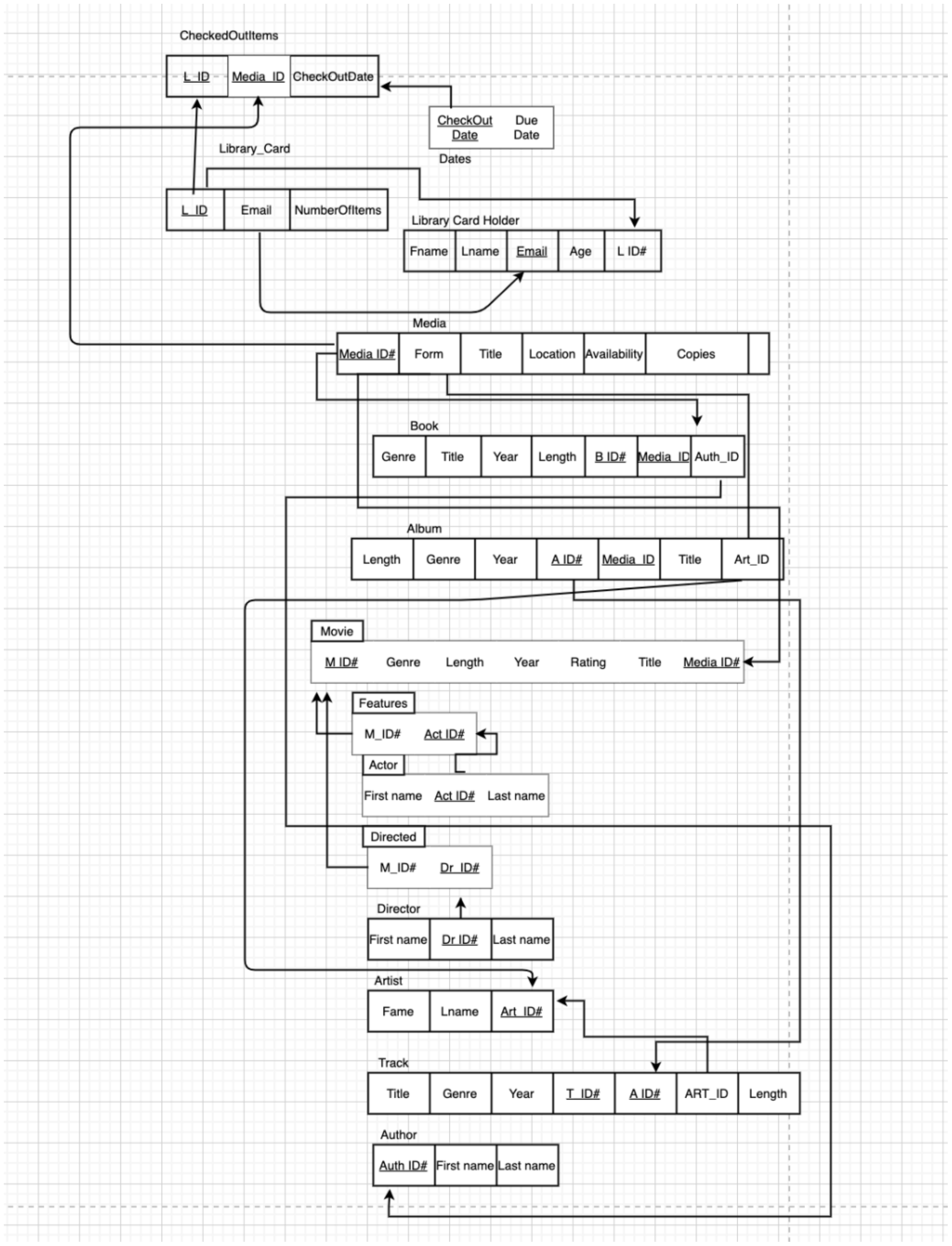
I won't parse through the other queries, but the notes I mentioned above are applicable to many situations here (misnamed columns, missing data, etc). Let me know if you have any questions about these!

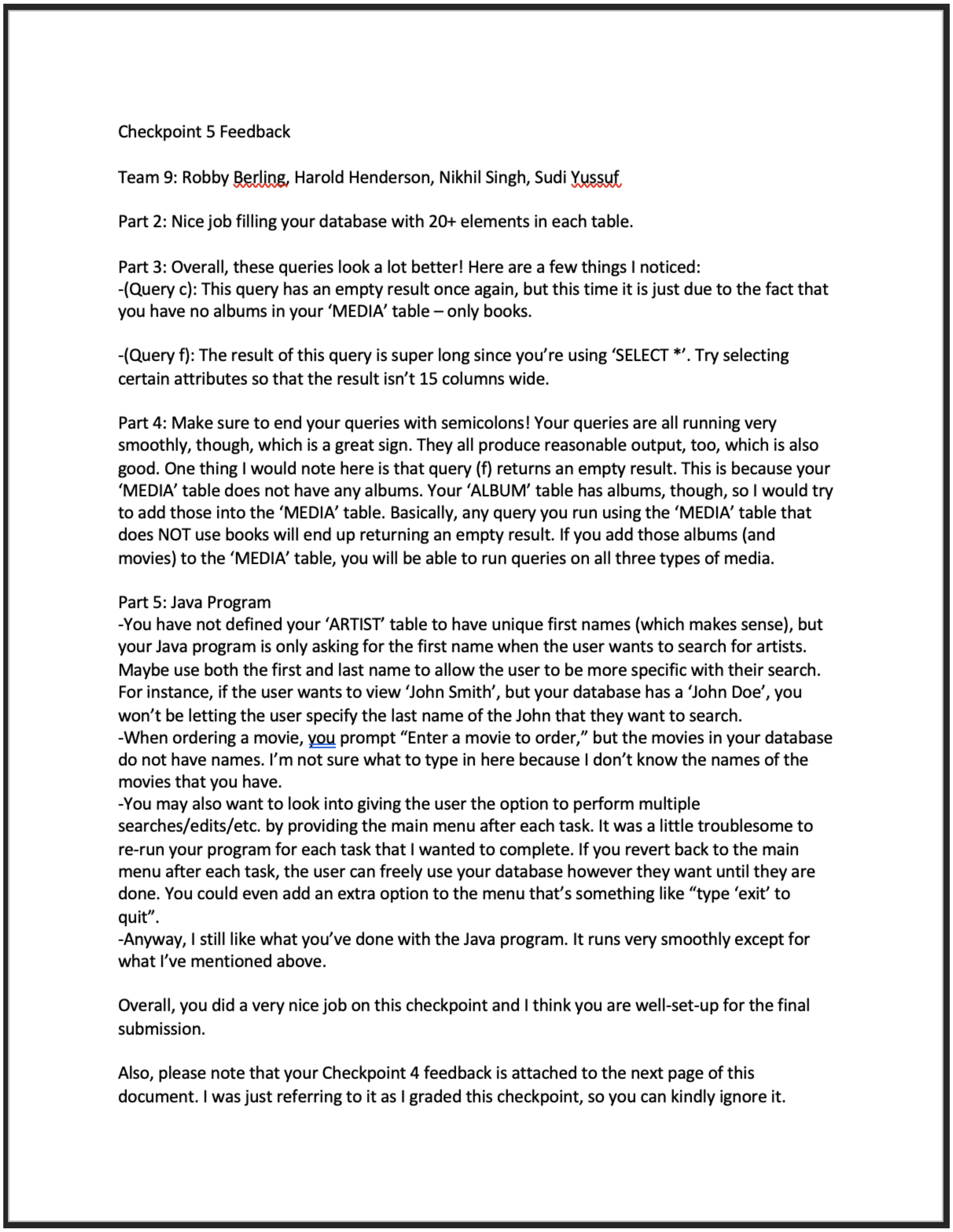
(Views.txt): The first view is attempting to use tables 'Library_card_holder' and 'Library_card', but only the latter of the two tables exists. Maybe you mean to just use 'Card_Holder'? The second view has a similar issue: it is using a 'Dates' table that does not exist. Maybe you just forgot to include that in the 'Create.txt' file? Either way, these views could use another look.

Checkpoint 5

From Checkpoint 4 to Checkpoint 5, the changes made were to an attribute called date for Album, Book, Movie, and Track. The format for the date was changed from YEAR-MONTH-DAY to just YEAR.




CheckedOutItems
L_ID
Media_ID
CheckOutDate
CheckOut Date
Due Date
Dates
Library_Card
L_ID
Email
NumberOfItems
Library Card Holder
Fname
Lname
Email
Age
L ID#
Media
Media ID#
Form
Title
Location
Availability
Copies
Book
Genre
Title
Year
Length
B ID#
Media_ID
Auth_ID
Album
Length
Genre
Year
A ID#
Media_ID
Title
Art_ID
Movie
M ID#
Genre
Length
Year
Rating
Title
Media ID#
Features
M_ID#
Act ID#
Actor
First name
Act ID#
Last name
Directed
M_ID#
Dr_ID#
Director
First name
Dr ID#
Last name
Artist
Fame
Lname
Art_ID#
Track
Title
Genre
Year
T_ID#
A ID#
ART_ID
Length
Author
Auth ID#
First name
Last name



Checkpoint 5 Feedback

Team 9: Robby Berling, Harold Henderson, Nikhil Singh, Sudi Yussuf

Part 2: Nice job filling your database with 20+ elements in each table.

Part 3: Overall, these queries look a lot better! Here are a few things I noticed:
-(Query c): This query has an empty result once again, but this time it is just due to the fact that you have no albums in your 'MEDIA' table – only books.

-(Query f): The result of this query is super long since you're using 'SELECT *'. Try selecting certain attributes so that the result isn't 15 columns wide.

Part 4: Make sure to end your queries with semicolons! Your queries are all running very smoothly, though, which is a great sign. They all produce reasonable output, too, which is also good. One thing I would note here is that query (f) returns an empty result. This is because your 'MEDIA' table does not have any albums. Your 'ALBUM' table has albums, though, so I would try to add those into the 'MEDIA' table. Basically, any query you run using the 'MEDIA' table that does NOT use books will end up returning an empty result. If you add those albums (and movies) to the 'MEDIA' table, you will be able to run queries on all three types of media.

Part 5: Java Program
-You have not defined your 'ARTIST' table to have unique first names (which makes sense), but your Java program is only asking for the first name when the user wants to search for artists. Maybe use both the first and last name to allow the user to be more specific with their search. For instance, if the user wants to view 'John Smith', but your database has a 'John Doe', you won't be letting the user specify the last name of the John that they want to search.
-When ordering a movie, you prompt "Enter a movie to order," but the movies in your database do not have names. I'm not sure what to type in here because I don't know the names of the movies that you have.
-You may also want to look into giving the user the option to perform multiple searches/edits/etc. by providing the main menu after each task. It was a little troublesome to re-run your program for each task that I wanted to complete. If you revert back to the main menu after each task, the user can freely use your database however they want until they are done. You could even add an extra option to the menu that's something like "type 'exit' to quit".
-Anyway, I still like what you've done with the Java program. It runs very smoothly except for what I've mentioned above.

Overall, you did a very nice job on this checkpoint and I think you are well-set-up for the final submission.

Also, please note that your Checkpoint 4 feedback is attached to the next page of this document. I was just referring to it as I graded this checkpoint, so you can kindly ignore it.