Media Manager Database System

CSE 3241 Leon Madrid TuTh 3:55

Team 04
Christian Barrett, Andrew Kramer, Avijit Kumar, Zachary Mason

April 14, 2022

Table of Contents

Section 1 - Data Description

Project Introduction

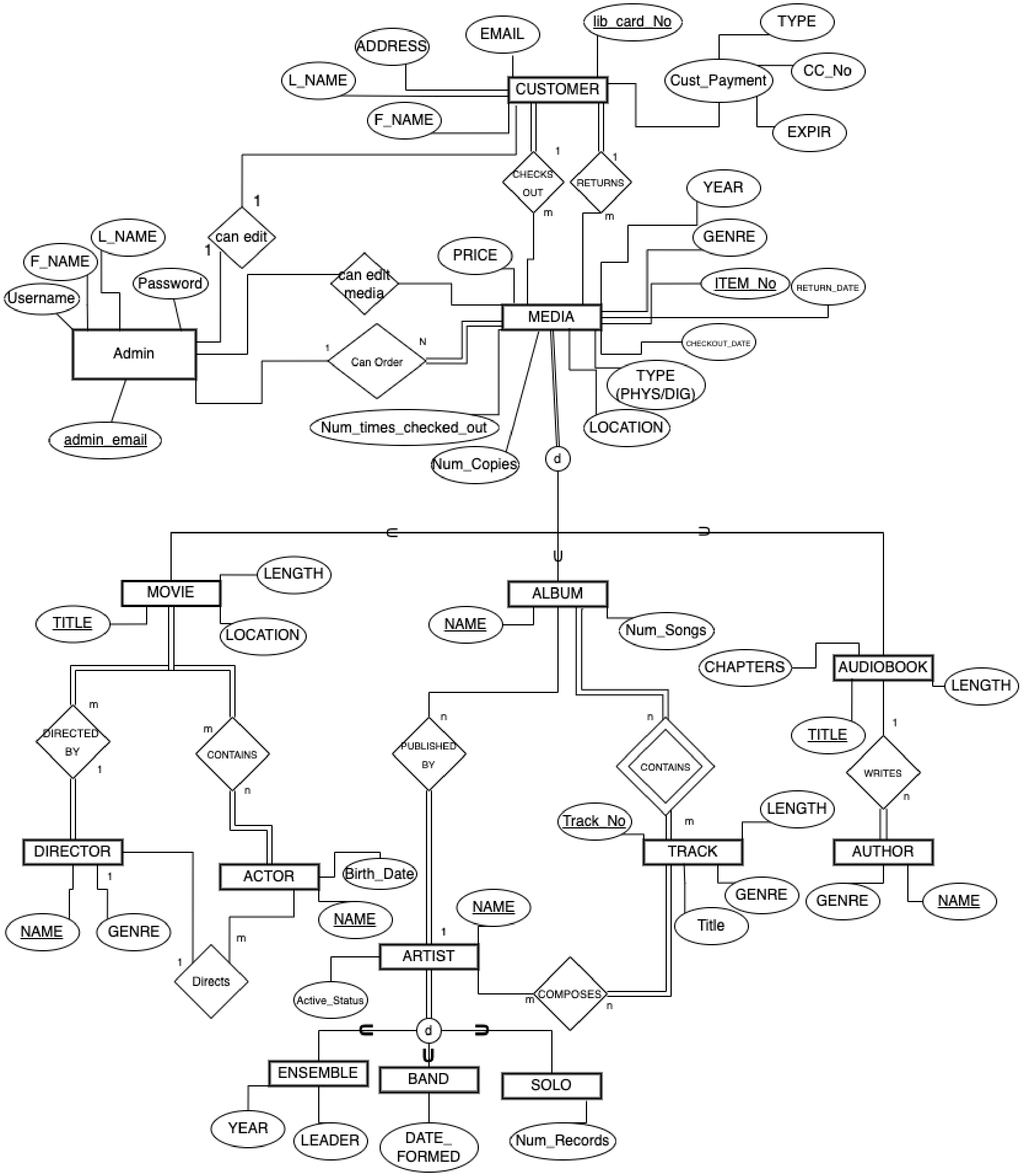
A local library has requested assistance in building an application to manage their music and video collection and needs a simple database management system to support their inventory and circulation operations.

Our team has been tasked with designing the database as well as developing a Java application to integrate with the database.

Over the past 10 weeks, we have been hard at work developing a fully functional application to not only organize, store, and track media, but also integrate a user-friendly program to create custom queries that real library patrons could utilize to access the database.

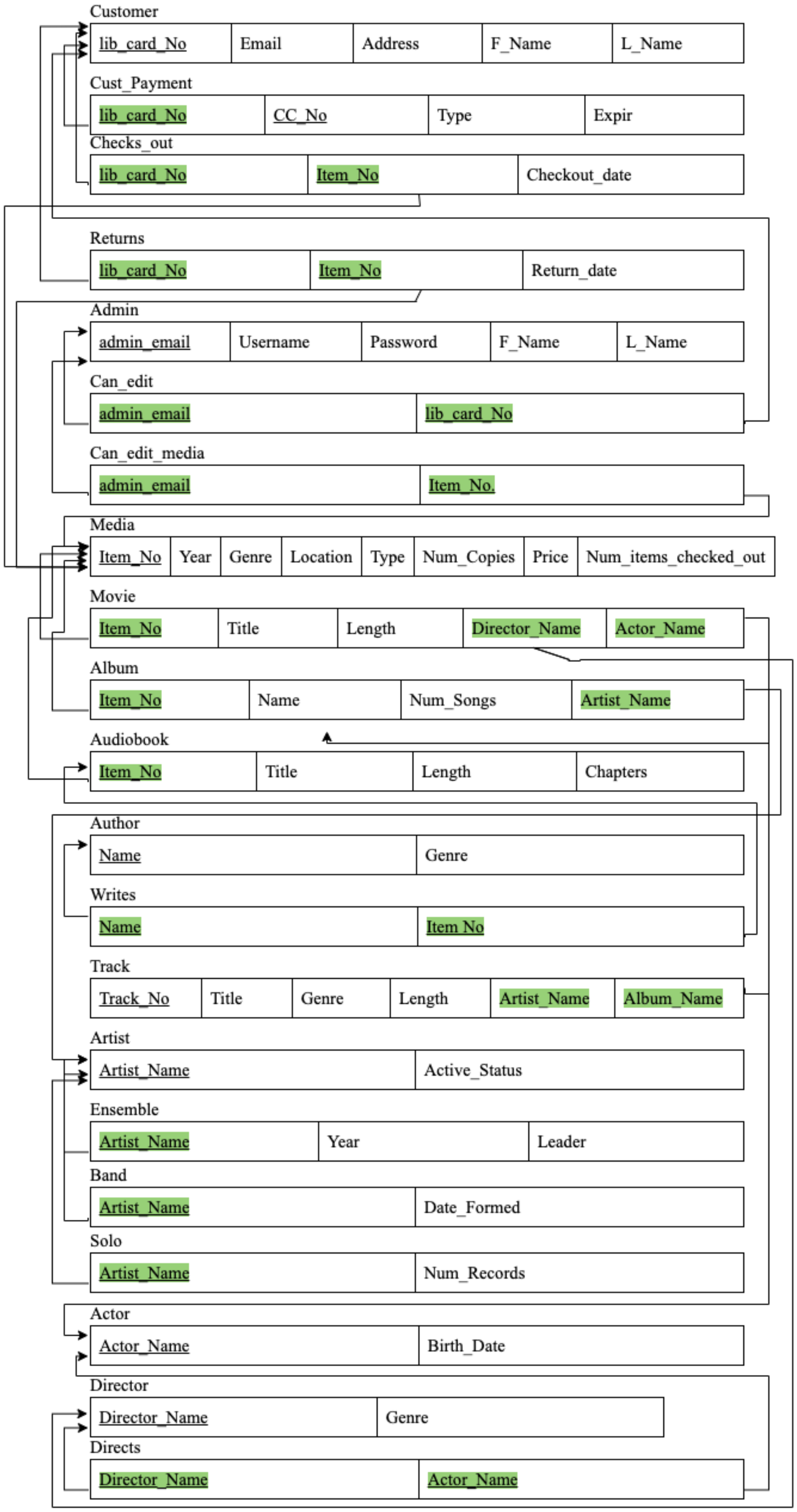
From designing every relationship to building the database management system, our team has excelled in all aspects of project design and we believe the work presented in this and all attached documents to be adequate for public library use.

Entity Relationship Diagram



Relational Schema

Underline = Primary Key Green Highlight = Foreign Key



Customer

lib_card_No	Email	Address	F_Name	L_Name

Cust_Payment

lib_card_No	CC_No	Type	Expir

Checks_out

lib_card_No	Item_No	Checkout_date

Returns

lib_card_No	Item_No	Return_date

Admin

admin_email	Username	Password	F_Name	L_Name

Can_edit

admin_email	lib_card_No

Can_edit_media

admin_email	Item_No.

Media

Item_No	Year	Genre	Location	Type	Num_Copies	Price	Num_items_checked_out

Movie

Item_No	Title	Length	Director_Name	Actor_Name

Album

Item_No	Name	Num_Songs	Artist_Name

Audiobook

Item_No	Title	Length	Chapters

Author

Name	Genre

Writes

Name	Item No

Track

Track_No	Title	Genre	Length	Artist_Name	Album_Name

Artist

Artist_Name	Active_Status

Ensemble

Artist_Name	Year	Leader

Band

Artist_Name	Date_Formed

Solo

Artist_Name	Num_Records

Actor

Actor_Name	Birth_Date

Director

Director_Name	Genre

Directs

Director_Name	Actor_Name

Normalization

The following is the description of each entity's normalization level. Unless otherwise noted, each entity in the relational schema is in Boyce Codd Normal Form (BCNF).

CUSTOMER

{lib_card_No, Email, Address, Fname, Lname}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Lib_Card_No} → Email, Address, Fname, Lname

CUST_PAYMENT

{Lib_Card_No, CC_No, Type, Expir}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {lib_card_No} → CC_No, Type, Expir

CHECKS_OUT

{Lib_Card_No, Item_No, Checkout_date}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Item_No, lib_card_No} → Checkout_date

RETURNS

{Lib_Card_No, Item_No, Return_date}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Item_No, lib_card_No} → Return_date

ADMIN

{Admin_Email, Username, Password, Fname, Lname}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {admin_email} → Username, Password, F_Name, L_Name

CAN_EDIT

{Admin_Email, Lib_Card_No}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:
   There are no functional dependencies, because all attributes are primary keys.

CAN_EDIT_MEDIA

{Admin_Email, Item_No}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:
   There are no functional dependencies, because all attributes are primary keys.

MEDIA

{Item_No, Year, Genre, Location, Type, Num_Copies, Price, Status}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:
   {Item_No} → Year, Genre, Location, Type, Num_Copies, Price, Status

MOVIE

{Item_No, Title, Length, Director_Name, Actor_Name}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:
   {Item_No} → Title, Length, Director_Name, Actor_Name

ALBUM

{Item_No, Name, Num_Songs, Artist_Name}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:
   {Item_No} → Name, Num_Songs, Artist_Name

AUDIOBOOK

{Item_No, Title, Length, Chapters}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Item_No} → Title, Length, Chapters

AUTHOR

{Name, Genre}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Name} → Genre

WRITES

{Name, Item_No}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies

   There are no functional dependencies, because all attributes are primary keys.

TRACK

{Track_no, Title, Genre, Length, Artist_Name, Album_Name}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Track_No} → Title, Genre, Length, Artist_Name, Album_Name

ARTIST

{Artist_Name, Active_Status}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Artist_Name} → Active_Status

ENSEMBLE

{Artists_Name, Year, Leader}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Artist_Name} → Year, Leader

BAND

{Artists_Name, Date_Formed}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Artists_Name} → Date_Formed

SOLO

{Artists_Name, Num_Records}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Artist_Name} → Num_Records

ACTOR

{Actor_Name, Birth_Date}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Actor_Name} → Birth_Date

DIRECTOR

{Director_Name, Genre}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:

   {Director_Name} → Genre

DIRECTS

{Director_Name, Actor_Name}

1. This entity is in BCNF because each attribute is fully, non-transitively dependent on the whole key, and no key attribute is functionally dependent on a non-key attribute.
2. Functional Dependencies:
   There are no functional dependencies, because all attributes are primary keys.

Indexes

*SQLite only supports B-Tree indices so the implementations are of B-Trees but the types described are assuming we had other options

1. We analyzed the usage of the database from the library patron's perspective and **search by artist** was a popular feature. Because of this we decided to implement a hash index* on the Artist table for Artist_Name. This will improve equality searches of artist names.

   ```
   CREATE INDEX Artist_Hash_Index
   ON Artist (Artist_Name);
   ```

2. We analyzed the usage of the database from the library admin's perspective and getting **ranges of values from the num_copies** column on the media table was a popular feature. Because of this we decided to implement a B-Tree index on the media table by num_copies. This will improve range searches.

   ```
   CREATE INDEX Num_Copies_Index
   ON Media (Num_Copies);
   ```

3. We analyzed the usage of the database from the library admin's perspective and **search by patron's email** was a popular feature. Because of this we decided to implement a hash index* on the customer table for email. This will improve equality searches of emails.

   ```
   CREATE INDEX Patron_Email_Index
   ON Customer (Email);
   ```

Views

View # 1 - Customer_Info

I. This view produces all the customer information that is present in the database (Library Card Number, Full Name, Email, Credit Card Number, Type, Expiration) as well as the current total number of items they have checked out. This view is extremely helpful because the admin could instantly pull up info on any customer to make edits or to see how many items they have checked out. They could decide whether or not to allow them to check out more or see if they have items they may have forgotten to return.

II. RELATIONAL ALGEBRA:

$\pi_{\text{customer . lib\_card\_no, customer . email, cust\_payment . cc\_no, cust\_payment . type, cust\_payment . expir} \rightarrow \text{expiration, COUNT (item\_no)} \rightarrow \text{num\_items\_checked\_out}}$

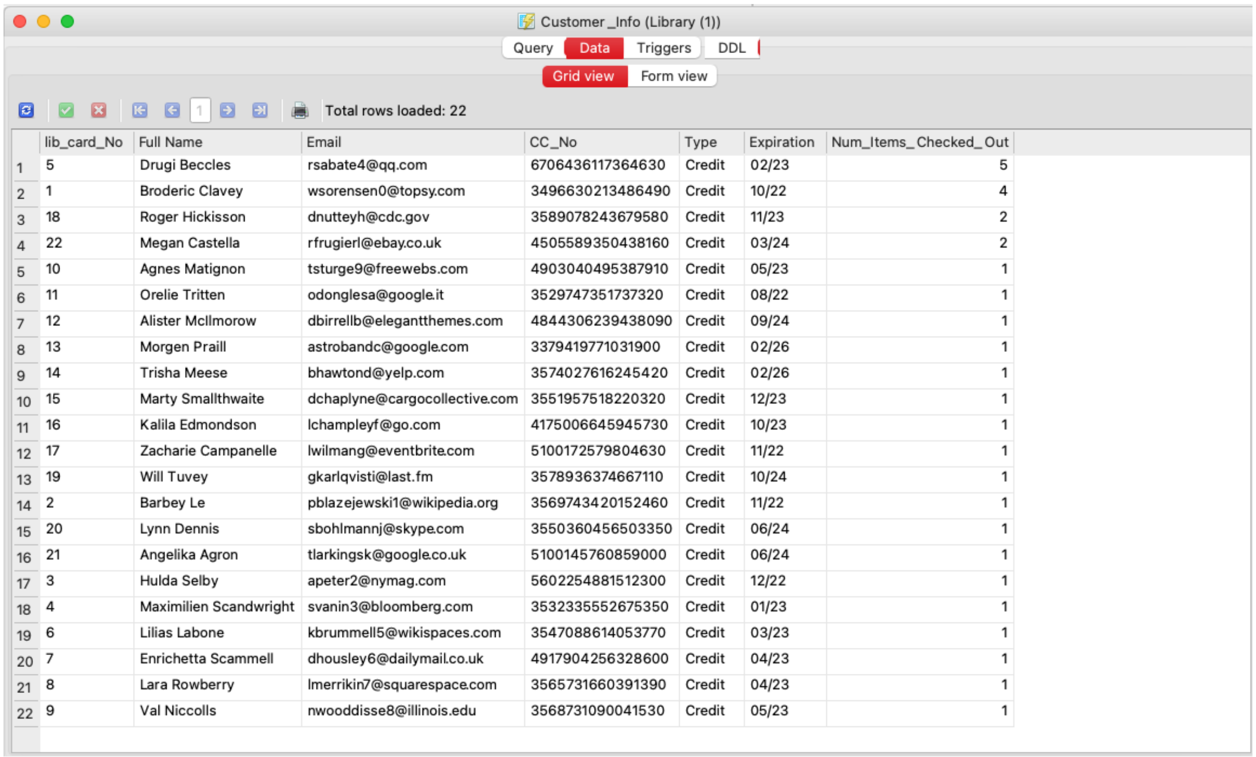
$\Im_{\text{lib\_card\_no, COUNT (item\_no)}}$

$\sigma_{\text{customer . lib\_card\_no = cust\_payment . lib\_card\_no AND customer . lib\_card\_no = checks\_out . lib\_card\_no}}(\text{customer} \times \text{cust\_payment} \times \text{checks\_out})$

III. SQL Statement to Produce the view:

```
CREATE VIEW Customer_Info AS
  SELECT Customer.lib_card_No,
       Customer.F_Name || ' ' || Customer.L_Name AS [Full Name],
       Customer.Email,
       Cust_Payment.CC_No,
       Cust_Payment.Type,
       Cust_Payment.Expir AS Expiration,
       COUNT(Checks_out.Item_No) AS Num_Items_Checked_Out
   FROM Customer,
       Cust_Payment,
       Checks_out
  WHERE Customer.lib_card_No = Cust_Payment.lib_card_No AND
       Customer.lib_card_No = Checks_out.lib_card_No
  GROUP BY Checks_out.lib_card_No
  ORDER BY 7 DESC;
```

IV. Sample Output:



Total rows loaded: 22

	lib_card_No	Full Name	Email	CC_No	Type	Expiration	Num_Items_Checked_Out
1	5	Drugi Beccles	rsabate4@qq.com	6706436117364630	Credit	02/23	5
2	1	Broderic Clavey	wsorensen0@topsy.com	3496630213486490	Credit	10/22	4
3	18	Roger Hickisson	dnutteyh@cdc.gov	3589078243679580	Credit	11/23	2
4	22	Megan Castella	rfrugierl@ebay.co.uk	4505589350438160	Credit	03/24	2
5	10	Agnes Matignon	tsturge9@freewebs.com	4903040495387910	Credit	05/23	1
6	11	Orelie Tritten	odonglesa@google.it	3529747351737320	Credit	08/22	1
7	12	Alister Mcllmorow	dbirrellb@elegantthemes.com	4844306239438090	Credit	09/24	1
8	13	Morgen Praill	astrobandc@google.com	3379419771031900	Credit	02/26	1
9	14	Trisha Meese	bhawtond@yelp.com	3574027616245420	Credit	02/26	1
10	15	Marty Smallthwaite	dchaplyne@cargocollective.com	3551957518220320	Credit	12/23	1
11	16	Kalila Edmondson	lchampleyf@go.com	4175006645945730	Credit	10/23	1
12	17	Zacharie Campanelle	lwilmang@eventbrite.com	5100172579804630	Credit	11/22	1
13	19	Will Tuvey	gkarlqvisti@last.fm	3578936374667110	Credit	10/24	1
14	2	Barbey Le	pblazejewski1@wikipedia.org	3569743420152460	Credit	11/22	1
15	20	Lynn Dennis	sbohlmannj@skype.com	3550360456503350	Credit	06/24	1
16	21	Angelika Agron	tlarkingsk@google.co.uk	5100145760859000	Credit	06/24	1
17	3	Hulda Selby	apeter2@nymag.com	5602254881512300	Credit	12/22	1
18	4	Maximilien Scandwright	svanin3@bloomberg.com	3532335552675350	Credit	01/23	1
19	6	Lilias Labone	kbrummell5@wikispaces.com	3547088614053770	Credit	03/23	1
20	7	Enrichetta Scammell	dhousley6@dailymail.co.uk	4917904256328600	Credit	04/23	1
21	8	Lara Rowberry	lmerrikin7@squarespace.com	3565731660391390	Credit	04/23	1
22	9	Val Niccolls	nwooddisse8@illinois.edu	3568731090041530	Credit	05/23	1

View # 2 - Remaining_Copies

I. This view produces the items, in this case movies, that are currently checked out, the title, and the remaining number of copies in the library. This view is very helpful as the admin, or whomever, could instantly open the view to see how many copies of a movie are in the library. With some simple manipulation, this could be done for all media in the database such as audiobooks or albums.

II. RELATIONAL ALGEBRA

$$\pi_{\text{checks\_out . item\_no, movie . title, media . num\_copies - COUNT (lib\_card\_no)} \rightarrow \text{remaining\_copies}}$$

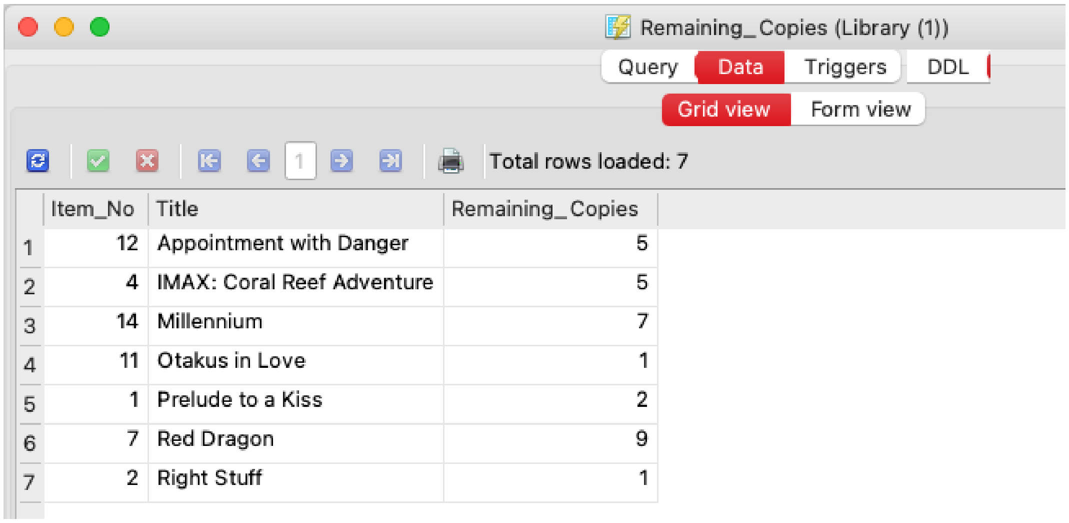
$$\sigma_{\text{movie . item\_no = checks\_out . item\_no AND media . item\_no = checks\_out . item\_no}} (\text{movie} \times \text{checks\_out} \times \text{media})$$

III. SQL Statement to Produce the view:

```
CREATE VIEW Remaining_Copies AS
  SELECT Checks_out.Item_No,
        Movie.Title,
        (Media.Num_Copies - COUNT(Checks_out.lib_card_No) )
Remaining_Copies
```

```
 FROM Movie,
     Checks_out,
     Media
WHERE Movie.Item_No = Checks_out.Item_No AND
     Media.Item_No = Checks_out.Item_No
GROUP BY Movie.Title;
```

IV. Sample Output:



Total rows loaded: 7

	Item_No	Title	Remaining_Copies
1	12	Appointment with Danger	5
2	4	IMAX: Coral Reef Adventure	5
3	14	Millennium	7
4	11	Otakus in Love	1
5	1	Prelude to a Kiss	2
6	7	Red Dragon	9
7	2	Right Stuff	1

View # 3 - No_Movies_Checked

I. This view produces the library card numbers and the total number of movies each has checked out of the database. While this is a somewhat specific view, it could be easily altered to track other media items that are checked out such as albums or audiobooks.

II. RELATIONAL ALGEBRA

$\pi_{\text{Checks\_out.lib\_card\_No, COUNT (Checks\_out.Item\_No)} \rightarrow \text{No\_Movies\_Checked\_Out}}$

$\sigma_{\text{Checks\_out.Item\_No = Movie.Item\_No AND Customer.lib\_card\_No = Checks\_out.lib\_card\_No}}$

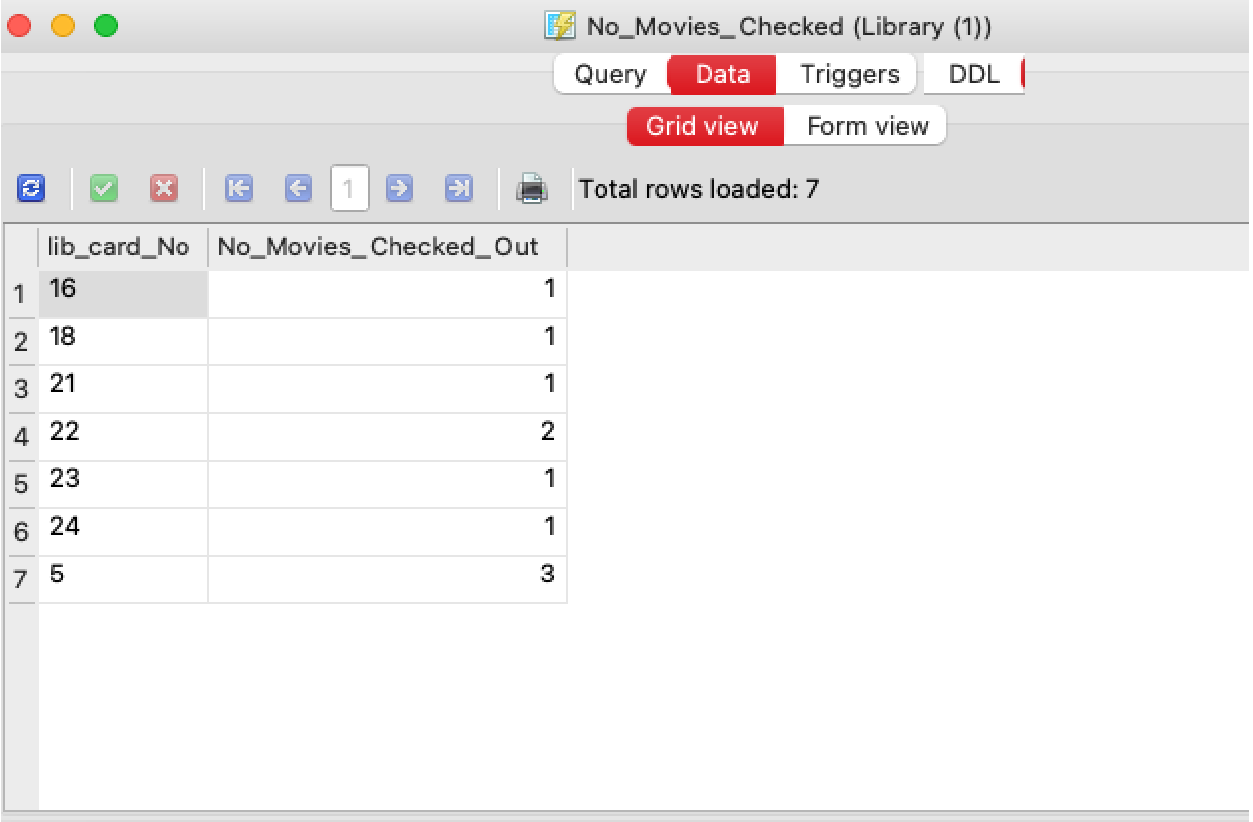
(Movie × Checks_out)

III. SQL Statement to Produce the view:

```
CREATE VIEW No_Movies_Checked AS
  SELECT Checks_out.lib_card_No,
       COUNT(Checks_out.Item_No) AS No_Movies_Checked_Out
```

```
 FROM Checks_out,
      Movie,
      Customer
WHERE Checks_out.Item_No = Movie.Item_No AND
      Customer.lib_card_No = Checks_out.lib_card_No
GROUP BY Customer.lib_card_No;
```

IV. Sample Output:



Total rows loaded: 7

	lib_card_No	No_Movies_Checked_Out
1	16	1
2	18	1
3	21	1
4	22	2
5	23	1
6	24	1
7	5	3

Transactions

Adding a band requires adding values to the artist table and the band table.

```
BEGIN TRANSACTION ADD_BAND
        INSERT INTO Artist VALUES (‘The Wiggles’, True);
        IF error THEN GO TO UNDO; END IF;
        INSERT INTO Band VALUES (‘The Wiggles’, 01/01/2000);
        IF error THEN GO TO UNDO; END IF;
        COMMIT;
        GO TO FINISH;
        UNDO:
            ROLLBACK;
        FINISH:
    END TRANSACTION
```

Ordering an item requires writing a new value and then updating that value later.

```
    BEGIN TRANSACTION ORDER_ITEM
        --Set location as “Ordered” and Num_Copies as 0 when first ordered
        INSERT INTO Media VALUES ((SELECT Max(Item_No) FROM Media, True),
            “1982”, “Punk”, “Ordered”, “Digital”, 0, 15, 0)
        IF error THEN GO TO UNDO; END IF;
        --Upon arrival update the entry to correct location and Num_Copies
        UPDATE Media
        SET Location = “Lithuania”, Num_Copies = 42
        WHERE Item_No = 89 --previously determined item#
        IF error THEN GO TO UNDO; END IF;
        COMMIT;
        GO TO FINISH;
        UNDO:
            ROLLBACK;
        FINISH:
    END TRANSACTION
```

Adding a new Director

```
    BEGIN TRANSACTION ADD_DIRECTOR
        INSERT INTO Director VALUES (“John Krasinski”, “Horror”)
        IF error THEN GO TO UNDO; END IF;
        INSERT INTO Directs VALUES (“John Krasinski”, “Emily Blunt”)
        IF error THEN GO TO UNDO; END IF;
        COMMIT;
        GO TO FINISH;
        UNDO:
            ROLLBACK;
        FINISH:
    END TRANSACTION
```

Section 2 - User Manual

Entity Descriptions

Underline = Primary Key (FK) = Foreign Key

MEDIA - This is a superclass that represents all forms of media in the inventory. The forms of media include movies, albums, and audiobooks.

- **Item_No** - An INTEGER assigned to each item in the database that corresponds to each form of media under the superclass.
- **Year** - A VARCHAR(4) that details the year each item was released.
- **Genre** - A VARCHAR(15) that details which genre type the item falls under.
- **Location** - A VARCHAR(25) that details where the media type originated from. Marked as "unavailable" if the item has been ordered but not arrived.
- **Type** - A VARCHAR(10) that states whether the media is in digital or physical format.
- **Num_Copies** - An INTEGER detailing the number of copies in the library database.
- **Price** - A DECIMAL that details how much the item cost to purchase.
- **Num_times_checked_out** - An INTEGER that shows how many times each item has been checked out of the database system.

MOVIE - This is a subclass of MEDIA, which represents all movies in the media inventory.

- **Item_No** (FK) - An INTEGER assigned to each movie in the database that is unique to each movie. *Movie.Item_No REFERENCES Media.Item_No.*
- **Title** - A VARCHAR(25) that details the title of each movie.
- **Length** - An INTEGER that details how long the runtime of each movie is in minutes.
- **Director_Name** (FK) - A VARCHAR(40) that details the director of each movie. *Movie.Director_Name REFERENCES Director.Name.*
- **Actor_Name** (FK) - A VARCHAR(40) that details the leading actor of each movie. *Movie.Actor_Name REFERENCES Actor.Name.*

ALBUM - This is a subclass of MEDIA, which represents all music albums in the media inventory.

- **Item_No** (FK) - An INTEGER assigned to each album in the database that is unique to each album. *Album.Item_No REFERENCES Media.Item_No.*
- **Name** - A VARCHAR(25) that details the name of each album.
- **Num_Songs** - An INTEGER that details the number of songs on each album.
- **Artist_Name** (FK) - A VARCHAR(40) that details the artist's name who created each album. *Album.Artist_name REFERENCES Artist.Name*
- **Album_Length** - An INTEGER that details the length of each album in minutes.

AUDIOBOOK - This is a subclass of MEDIA, which represents all audiobooks in the media inventory.

- **Item_No** (FK) - An INTEGER assigned to each album in the database that is unique to each audiobook. *Audiobook.Item_No REFERENCES Media.Item_No.*
- **Title** - A VARCHAR(50) that details the title of each audiobook.

- **Length** - An INTEGER that details how long the runtime of each audiobook is in minutes.
- **Chapters** - An INTEGER that details the number of chapters in an audiobook.

Checks_out - This is a relation representing a customer checking out some form of media, which include movies, albums, and audiobooks.

- **Item_No** (FK) - An INTEGER assigned to each item in the database that corresponds to each form of media under the superclass. *Checks_out.Item_No REFERENCES Media.Item_No.*
- **lib_card_No** (FK) - A VARCHAR(9) assigned to each customer in the database that is unique to each customer/patron. *Checks_out.lib_card_No REFERENCES Customer.lib_card_No.*
- **Checkout_date** - A DATE logged when an item is checked out to a customer.

Returns - This is a relation representing a customer returning some form of media, which include movies, albums, and audiobooks.

- **Item_No** (FK) - An INTEGER assigned to each item in the database that corresponds to each form of media under the superclass. *Returns.Item_No REFERENCES Media.Item_No.*
- **lib_card_No** (FK) - A CHAR(9) assigned to each customer in the database that is unique to each customer/patron. *Returns.lib_card_No REFERENCES Customer.lib_card_No.*
- **Return_date** - A DATE logged when an item is returned to the library.

CUSTOMER - This entity represents the customer and all information that describes a unique customer with payment information.

- **lib_card_No** - A CHAR(9) assigned to each customer in the database that is unique to each customer/patron. UNIQUE and NOT NULL.
- **Email** - A VARCHAR(40) that details the email address of the customer/patron. NOT NULL.
- **Address** - A VARCHAR(40) that details the address of the customer/patron.
- **F_name** - A VARCHAR(25) that details the first name of the customer/patron.
- **L_name** - A VARCHAR(30) that details the last name of the customer/patron.

Cust_Payment - This entity represents the customer payment information such as credit card information.

- **lib_card_No** (FK) - A CHAR(9) assigned to each customer in the database that is unique to each customer/patron. *Cust_Payment.lib_card_No REFERENCES Customer.lib_card_No.*
- **CC_No** - A VARCHAR(16) that holds the credit card number of the customer using 16 digits.
- **Type** - A STRING(8) that details the payment type/method of the customer.
- **Expir** - A VARCHAR(4) that holds the expiration date of the credit card of the customer using 4 digits.

Can_edit - This is a relation representing an admin being able to edit the unique customer information.

- **admin_email** (FK) - A VARCHAR(50) that details the email address of the administrator of the library. *Can_edit.admin_email REFERENCES admin.admin_email.*
- **lib_card_No** (FK) - A VARCHAR(9) assigned to each customer in the database that is unique to each customer/patron. *Can_edit.lib_card_No REFERENCES Customer.lib_card_No.*

Can_edit_media - This is a relation representing an admin editing information about some form of media.

- **admin_email** (FK) - A VARCHAR(50) that details the email address of the administrator of the library. *Can_edit_media.admin_email REFERENCES Admin.admin_email.*
- **Item_No** (FK) - An INTEGER assigned to each item in the database that corresponds to each form of media under the superclass. *Can_edit_media.Item_No REFERENCES Media.Item_No.*

Admin - This entity represents the admin and all information that describes a unique admin and account information.

- **admin_email** - A VARCHAR(50) that details the email address of the administrator of the library.
- **Username** - A VARCHAR(25) that details the username of the customer/patron.
- **Password** - A VARCHAR(20) that details the password of the customer/patron.
- **F_Name** - A VARCHAR(20) that details the first name of the customer/patron.
- **L_Name** - A VARCHAR(20) that details the last name of the customer/patron.

Directs - This is a relation describing the relationship between the director and the actor. The director who is leading the actor's within the movie.

- **Director_Name** (FK) - A VARCHAR(40) that details the director's name of each movie. *Directs.Director_Name REFERENCES Director.Director_Name.*
- **Actor_Name** (FK) - A VARCHAR(40) that details the actor's name of each movie. *Directs.Actor_Name REFERENCES Actor.Actor_Name.*

DIRECTOR - This represents the director and all information relevant to describing the director.

- **Director_Name** - A VARCHAR(40) that details the director's name for each movie.
- **Genre** - A VARCHAR(15) that details the genre that each director directs.

ACTOR - This represents the actor and all information relevant to describing the actor.

- **Actor_Name** - A VARCHAR(40) that details the actor's name for each movie.
- **Birth_Date** - A DATE that holds the birth date of the actor who acted in the movie.

ARTIST - This is a superclass that represents all forms of artists in the artist list. The forms of artists include ensemble, band, and solo.

- **Artist_Name** - A VARCHAR(40) that details the artist's name of each track or album.
- **Active_status** - A BOOLEAN that details whether an artist or type of artist is active or inactive.

ENSEMBLE - This is a subclass of ARTIST, which represents all ensembles in the list of artists.

- **Artist_Name** (FK) - A VARCHAR(40) that details the ensemble's name of each track and album. *Ensemble.Artist_name REFERENCES Artist.Name*
- **Year** - A INT(4) that details the year the ensemble was formed.
- **Leader** - A VARCHAR(40) that details the lead's name of each track and album.

BAND - This is a subclass of ARTIST, which represents all bands in the list of artists.

- **Artist_Name** (FK) - A VARCHAR(40) that details the band's name of each track and album. *Band.Artist_name REFERENCES Artist.Name*
- **Date_formed** - A DATE that details the year the band was formed.

SOLO - This is a subclass of ARTIST, which represents all solo artists in the list of artists.

- **Artist_Name** (FK) - A VARCHAR(40) that details the solo artist's name of each track and album. *Solo.Artist_name REFERENCES Artist.Name*
- **Num_Records** - An INTEGER that details how many tracks, albums, and records the artist has released.

TRACK - This represents each track in an album and describes each unique track in each album within the Media inventory.

- **Track_No** - An INTEGER assigned to each track in the database that is unique to each track.
- **Title** - A VARCHAR(35) that details the title of each track.
- **Genre** - A VARCHAR(15) that details which genre type the track falls under.
- **Length** - An INTEGER that details how long the runtime of each track is in minutes.
- **Artist_Name** (FK) - A VARCHAR(40) that details the band's name of each track and album. *Track.Artist_name REFERENCES Artist.Name*
- **Album_Name** (FK) - A VARCHAR(30) that details the name of each album. *Track.Album_Name REFERENCES Album.Name*

AUTHOR - This represents each author of an audiobook and describes each unique author of each audiobook within the Media inventory.

- **Name** - A VARCHAR(40) that details the author's name of each audiobook.
- **Genre** - A VARCHAR(15) that details which genre type the audiobook falls under.

WRITES - This represents a relationship between author and audiobook to connect which authors wrote which audiobooks.

- **Name** (FK) - A VARCHAR(40) that details the author's name of each audiobook. *Writes.Name REFERENCES Author.Name.*
- **Item_No** (FK) - An INTEGER assigned to each item in the database that corresponds to each form of media under the superclass. *Writes.Item_No REFERENCES Audiobook.Item_No.*

Sample SQL Queries

Final Queries:

Checkpoint 4:
3a)
The following query lists the titles of all tracks by ARTIST (AC/DC) released before YEAR (2000).

Relational Algebra:

$$\pi_{\text{title}}(\sigma_{\text{media.item\_no = album.item\_no AND track.album\_name = album.name AND media.year < 2000 AND album.artist\_name = "AC/DC"}}(\text{album} \times \text{track} \times \text{media}))$$

SQL Query:

```
SELECT Title
FROM Album, Track, Media
WHERE Media.Item_No = Album.Item_No
        AND Track.Album_Name = Album.Name
        AND Media.Year < 2000
        AND Album.Artist_Name = "AC/DC";
```

3b)

The following query gives all the movies and their date of their checkout from a single patron (we chose to designate by library card number “5”).

Relational Algebra:

$$\pi_{\text{title, checkout\_date}}(\sigma_{\text{movie.item\_no = checks\_out.item\_no AND checks\_out.lib\_card\_no = customer.lib\_card\_no AND customer.lib\_card\_no = "5"}}(\text{movie} \times \text{customer} \times \text{checks\_out}))$$

SQL Query:

```
SELECT Title, Checkout_Date
FROM Movie, Customer, Checks_out
WHERE Movie.Item_No = Checks_out.Item_No
        AND Checks_out.lib_card_No = Customer.lib_card_No
        AND Customer.lib_card_No = "5";
```

3c)

The following query lists all the albums and their unique identifiers with less than 2 copies held by the library.

Relational Algebra:

$$\pi_{\text{Name, Album.Item\_No}}(\sigma_{\text{Media.Num\_Copies} < 2}(\text{Album} \bowtie_{\text{Album.Item\_No = Media.Item\_No}} \text{Media}))$$

SQL Query:

```
SELECT Name, Album.Item_No
```

```
FROM Album, Media
WHERE Album.Item_No = Media.Item_No
          AND Media.Num_Copies < 2;
```

3d)

The following query gives all patrons who checked out a movie by ACTOR (“Demetra Points”) and the movies they checked out.

Relational Algebra:

$\pi_{\text{Customer.F\_Name, Movie.Title}}(\sigma_{\text{Movie.Actor\_Name = “Demetra Points”}}(\text{Customer} \bowtie_{\text{Customer.lib\_card\_No = Checks\_out.lib\_card\_No)}} \text{Checks\_out AND Checks\_out} \bowtie_{\text{Checks\_out.Item\_No = Movie.Item\_No}} \text{Movie}))$

SQL Query:

```
SELECT Customer.F_Name, Movie.Title
FROM Checks_out, Movie, Customer
WHERE Checks_out.Item_No = Movie.Item_No
      AND Customer.lib_card_No = Checks_out.lib_card_No
          AND Movie.Actor_Name = "Demetra Points";
```

3e)

The following query finds the total number of albums checked out by a single patron (we chose to designate the patron by library card number “1”).

Relational Algebra:

$\pi_{\text{COUNT (item\_no)} \rightarrow \text{num\_of\_checked\_out\_albums}}$

$\mathfrak{I}_{\text{COUNT (item\_no)}}$

$\sigma_{\text{album.item\_no = checks\_out.item\_no AND customer.lib\_card\_no = "1" AND customer.lib\_card\_no = checks\_out.lib\_card\_no}}(\text{customer} \times \text{album} \times \text{checks\_out})$

SQL Query:

```
SELECT count(Checks_out.Item_No) AS num_of_checked_out_albums
FROM Customer, Album, Checks_out
WHERE Album.Item_No = Checks_out.Item_No
          AND Customer.lib_card_No = "1"
          AND Customer.lib_card_No = Checks_out.lib_card_No;
```

3f)

The following query finds the patron who has checked out the most videos and the total number of videos they have checked out.

Relational Algebra:

$\pi_{\text{checks\_out.lib\_card\_no, MAX (no\_movies\_checked\_out)}}(\pi_{\text{checks\_out.lib\_card\_no, COUNT (item\_no)} \rightarrow \text{no\_movies\_checked\_out}}$

$\Im_{\text{lib\_card\_no, COUNT (item\_no)}}$

$\sigma_{\text{checks\_out.item\_no = movie.item\_no AND customer.lib\_card\_no = checks\_out.lib\_card\_no}}(\text{checks\_out} \times \text{movie} \times \text{customer}))$

$\Im_{\text{MAX (no\_movies\_checked\_out)}}(\text{checks\_out} \times \text{movie} \times \text{customer})$

SQL Query:

```
SELECT Checks_out.lib_card_No, MAX(No_Movies_Checked_Out)
FROM
  (SELECT Checks_out.lib_card_No,
       COUNT(Checks_out.Item_No) AS No_Movies_Checked_Out
   FROM Checks_out,
       Movie,
       Customer
   WHERE Checks_out.Item_No = Movie.Item_No AND
       Customer.lib_card_No = Checks_out.lib_card_No
   GROUP BY Customer.lib_card_No);
```

4a)

The following query finds the title of the movie with the specified Actor (“Sidonia Cadamy”) and Director (“Andy Ackerman”) names.

Relational Algebra:

$\pi_{\text{movie.title, actor\_name, director\_name}}$

$\sigma_{\text{actor\_name = "Sidonia Cadamy" AND director\_name = "Andy Ackerman"}}\ \text{movie}$

SQL Query:

```
SELECT Movie.Title, Actor_Name, Director_Name
FROM Movie
WHERE Actor_Name = "Sidonia Cadamy" AND Director_Name = "Andy Ackerman";
```

4b)
The following query lists each Movie title, the location it was made, and the number of copies in the library.

Relational Algebra:

$\pi_{\text{movie.title, media.location, media.num\_copies}}$
$\sigma_{\text{movie.item\_no = media.item\_no}}$ (movie × media)

SQL Query:

```
SELECT Movie.Title, Media.Location, Media.Num_Copies
FROM Movie, Media
WHERE Movie.Item_No = Media.Item_No;
```

4c)
The following query lists all the Authors and Titles of Audiobooks in the library that belong to the specified Genre ("Fiction");

Relational Algebra:

$\pi_{\text{author.name, audiobook.title, author.genre}}$
$\sigma_{\text{audiobook.item\_no = writes.item\_no AND author.name = writes.name AND author.genre = "Fiction"}}$ (author × audiobook × writes)

SQL Query:

```
SELECT Author.Name, Audiobook.Title, Author.Genre
FROM Author, Audiobook, Writes
WHERE Audiobook.Item_No = Writes.Item_No
  AND Author.Name = Writes.Name
  AND Author.Genre = "Fiction";
```

From Checkpoint 5:

4a)
The following query provides a list of patron names, along with the total combined running time of all the movies they have checked out.

Relational Algebra:

$\pi_{\text{customer.f\_name, customer.l\_name, SUM (length)} \rightarrow \text{total\_combined\_running\_time\_of\_checked\_out\_movies}}$

$\Im_{\text{email, SUM (length)}}$

$\sigma_{\text{movie.item\_no = checks\_out.item\_no AND customer.lib\_card\_no = checks\_out.lib\_card\_no}} (\text{movie} \times \text{customer} \times \text{checks\_out})$

SQL Query:

```
SELECT Customer.F_Name, Customer.L_Name, Sum(Length) AS "Total Combined Running
Time of Checked out Movies"
        FROM Movie, Customer, Checks_out
        WHERE Movie.Item_No = Checks_out.Item_No
        AND Customer.lib_card_No = Checks_out.lib_card_No
        GROUP BY Customer.Email;
```

4b)

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

Relational Algebra:

$\pi_{\text{customer.lib\_card\_no, customer.f\_name, customer.l\_name, customer.email}}$

$\sigma_{\text{COUNT (*) >}}\pi_{\text{customer.email, COUNT (*)} \rightarrow \text{cnt}}$

$\Im_{\text{email, COUNT (*)}} (\text{customer} \bowtie_{\text{checks\_out.lib\_card\_no = customer.lib\_card\_no}} \text{checks\_out})$

$\Im_{\text{f\_name, l\_name, email, COUNT (*)}} (\text{customer} \bowtie_{\text{customer.lib\_card\_no = checks\_out.lib\_card\_no}} \text{checks\_out})$

SQL Query:

```
SELECT Customer.lib_card_No, Customer.F_Name, Customer.L_Name, Customer.Email
FROM Customer JOIN Checks_out ON Customer.lib_card_No = Checks_out.lib_card_No
GROUP BY F_Name, L_Name, Email
HAVING COUNT(*) > (
SELECT AVG(COUNT)
FROM (
SELECT Customer.Email, COUNT(*) COUNT
FROM Customer JOIN Checks_out ON Checks_out.lib_card_No = Customer.lib_card_No
GROUP BY Email
)
);
```

4c)

The following query provides 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:

$\tau_{\text{media.num\_times\_checked\_out} \downarrow}$

$\pi_{\text{title, media.num\_times\_checked\_out}}$

$\Im_{\text{title,}}$

$\sigma_{\text{movie.item\_no = media.item\_no}} (\text{movie} \times \text{media})$

SQL Query:

```
SELECT Title, Media.Num_times_checked_out
        FROM Movie, Media
        WHERE Movie.Item_No = Media.Item_No
        GROUP BY Movie.Title
        ORDER BY Media.Num_times_checked_out DESC;
```

4d)

The following query provides a list of the albums in the database and associated totals for copies checked out to customers, sorted from ones that have been checked out the highest amount to the ones checked out the lowest.

Relational Algebra:

$\tau_{\text{COUNT (*)} \downarrow}$

$\Im_{\text{name, COUNT (*)}} (\text{album} \bowtie_{\text{album.item\_no = media.item\_no}} \text{media} \bowtie_{\text{media.item\_no = checks\_out.item\_no}} \text{checks\_out})$

SQL Query:

```
SELECT Album.Name, COUNT(*)
FROM ALbum
  JOIN Media ON Album.Item_No = Media.Item_No
  JOIN Checks_out ON Media.Item_No = Checks_out.Item_No
GROUP BY Album.Name
ORDER BY COUNT(*) DESC;
```

4e)

The following query finds the most popular actor in the database (the one who has had the most lent movies).

Relational Algebra:

$$\tau_{\text{media.num\_times\_checked\_out} \downarrow}$$

$$\pi_{\text{actor\_name, media.num\_times\_checked\_out}}$$

$$\mathfrak{I}_{\text{actor\_name,}}$$

$$\sigma_{\text{movie.item\_no = media.item\_no}}\,(\text{movie} \times \text{media})$$

SQL Query:

```
SELECT Actor_Name, Media.Num_times_checked_out
FROM Movie, Media
WHERE Movie.Item_No = Media.Item_No
GROUP BY Actor_Name
ORDER BY Media.Num_times_checked_out DESC
LIMIT 1;
```

4f)

The following query finds the most listened to artist in the database by using the running time of the album and the number of times the album has been lent out.

Relational Algebra:

$$\tau_{\text{total\_listening\_time\_sec} \downarrow}$$

$$\pi_{\text{artist\_name, SUM (album\_length) * media.num\_times\_checked\_out} \rightarrow \text{total\_listening\_time\_sec}}$$

$$\mathfrak{I}_{\text{artist\_name,}}$$

$$\sigma_{\text{album.item\_no = media.item\_no}}\,(\text{album} \times \text{media})$$

SQL Query:

```
SELECT Artist_Name, SUM(Album_Length) * Media.Num_times_checked_out AS
"Total_Listening_Time_sec"
        FROM Album, Media
        WHERE Album.Item_No = Media.Item_No
        GROUP BY Artist_Name
ORDER BY Total_Listening_Time_sec DESC
LIMIT 1;
```

4g)

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

<u>Relational Algebra:</u>

$\text{Inner} \longleftarrow \tau_{\text{MAX (num\_times\_checked\_out)} \downarrow}$

$\mathfrak{I}_{\text{actor\_name,}} (\text{movie} \bowtie_{\text{movie.item\_no = media.item\_no}} \text{media} \bowtie_{\text{media.item\_no = checks\_out.item\_no}} \text{checks\_out} \bowtie_{\text{movie.actor\_name = actor.actor\_name}} \text{actor})$

$\pi_{\text{Customer.lib\_card\_No, Customer.F\_Name, Customer.L\_Name, Customer.Email}} (\text{Customer} \bowtie_{\text{Customer.lib\_card\_No = Checks\_out.lib\_card\_No}} \text{Checks\_out} \bowtie_{\text{Checks\_out.Item\_No = Media.Item\_No}} \text{Media} \bowtie_{\text{Movie.Item\_No = Media.Item\_No}} \text{Movie} \bowtie_{\text{Actor.Actor\_Name = Movie.Actor\_Name}} \text{Actor})$

$\sigma_{(\text{Inner = *Movie.Actor\_Name})}$

<u>SQL Query:</u>

```
SELECT Customer.lib_card_No, Customer.F_Name, Customer.L_Name, Customer.Email
FROM Customer
  JOIN Checks_out ON Customer.lib_card_No = Checks_out.lib_card_No
  JOIN Media ON Checks_out.Item_No = Media.Item_No
  JOIN Movie ON Movie.Item_No = Media.Item_No
  JOIN Actor ON Actor.Actor_Name = Movie.Actor_Name
WHERE (
  SELECT Movie.Actor_Name
     FROM Movie
        JOIN Media ON Movie.Item_No = Media.Item_No
        JOIN Checks_out ON Media.Item_No = Checks_out.Item_No
        JOIN Actor ON Movie.Actor_Name = Actor.Actor_Name
     GROUP BY Movie.Actor_Name
     ORDER BY MAX(Num_times_checked_out) DESC
     LIMIT 1
  ) LIKE Movie.Actor_Name;
```

4h)

The following query provides a list of artists who authored the albums checked out by customers who have checked out more albums than the average customer.

<u>Relational Algebra:</u>

INNERTWO ←

$\pi_{\text{Customer.Email, COUNT (*)} \rightarrow \text{cnt}}$

$\Im_{\text{Email, COUNT (*)}} (\text{Customer} \bowtie_{\text{Checks\_out.lib\_card\_No = Customer.lib\_card\_No}} \text{Checks\_out})$

INNER ←

$\pi_{\text{Customer.lib\_card\_No}}$

$\sigma_{\text{COUNT (*)} > \text{INNERTWO}}$

$\Im_{\text{F\_Name, L\_Name, Email, COUNT (*)}} (\text{Customer} \bowtie_{\text{Checks\_out.lib\_card\_No = Customer.lib\_card\_No}} \text{Checks\_out})$

$\pi_{\text{Name}} (\text{Checks\_out} \bowtie_{\text{Media.Item\_No = Checks\_out.Item\_No}} \text{Media} \bowtie_{\text{Album.Item\_No = Media.Item\_No}} \text{Album} \bowtie_{\text{Artist.Artist\_Name = Album.Artist\_Name}} \text{Artist})$

$\sigma_{(\text{Checks\_out.lib\_card\_No = INNER})}$

SQL Query:

```
SELECT DISTINCT NAME
FROM Checks_out
  JOIN Media ON Media.Item_No = Checks_out.Item_No
  JOIN Album ON Album.Item_No = Media.Item_No
  JOIN Artist ON Artist.Artist_Name = Album.Artist_Name
WHERE Checks_out.lib_card_No IN (
  SELECT Customer.lib_card_No
  FROM Customer
     JOIN Checks_out ON Checks_out.lib_card_No = Customer.lib_card_No
  GROUP BY Customer.F_Name, Customer.L_Name, Customer.Email
  HAVING COUNT(*) > (
     SELECT AVG(count)
        FROM (
           SELECT Customer.Email, COUNT(*) AS count
              FROM Customer
                 JOIN Checks_out ON Checks_out.lib_card_No = Customer.lib_card_No
              GROUP BY Customer.Email
        )
  )
);
```

INSERT / DELETE Statement Samples

c. Insert Statements:

INSERT into ACTOR(Actor_Name, Birth_Date)
Values("Tom Hanks", "07/09/1959");

INSERT into ADMIN(admin_email, Username, Password, F_Name, L_Name)
Values("name@library.org", "username", "password", "John", "Smith");

INSERT into ALBUM(Item_No, Name, Num_Songs, Artist_Name, Album_Length)
Values(101, "Still Bill", 10, "Bill Withers", 2174);

Artist_Name must be the primary key of a value in Artist. When adding to Album, you must also insert a value into Media with the same Item_No. This can be done in either order, but must be sequential.

INSERT into ARTIST(Artist_Name, Active_Status)
Values("Bill Withers", False);

INSERT into AUDIOBOOK(Item_No, Title, Length, Chapters)
Values(104, "Winesburg, Ohio", 200, 12);

When adding to Audiobook, you must also insert a value into Media with the same Item_No. This can be done in either order, but must be sequential.

INSERT into AUTHOR(Name, Genre)
Values("Sherwood Anderson", "Fiction");

INSERT into BAND(Artist_Name, Date_Formed)
Values("The Chords", "01/01/1951");

- *Name* must be the primary key of a value in Artist.

INSERT into CAN_EDIT(admin_email, lib_card_No)
Values(“name@library.org”, 101010101);

- *lib_card_no* must be a primary key of a relation in Customer. *email* must be a primary key of a relation in Admin.

INSERT into CAN_EDIT_Media(admin_email, Item_No)
Values(“name@library.org”, 104);
Item_No must be a primary key of a relation in Media. admin_email must be a primary key of a relation in Admin.

INSERT into CHECKS_OUT(lib_card_No, Item_No, Checkout_Date)
Values(101010101, 104, “04/10/2022”);

Item_no must be the primary key of a relation in Media. lib_card_No must be a primary key of a relation in Customer

INSERT into CUST_PAYMENT(lib_card_No, CC_No, Type, Expir)
Values(101010101, “5555444433332222”, “MasterCard”, “11/25”);
lib_card_No must be a primary key of a relation in Customer.

INSERT into CUSTOMER(lib_card_No, Email, Address, F_Name, L_Name)
Values(101010101, “name@organization.org”, “123 W Main Street, Columbus, OH”, “Peter”, “Jones”);

INSERT into DIRECTOR(Director_Name, Genre)
Values(“Coppola”, “Drama”);

INSERT into DIRECTS(Director_Name, Actor_Name)
Values(“Coppola”, “Pacino”);

Director_Name and Actor_Name must be primary keys of Director and Actor relations, respectively.

```
INSERT into ENSEMBLE(Artist_Name, Year, Leader)
Values(“Group 04 Ensemble”, 2022, “Jim Smith”);
```

Artist_Name must be the primary key of an Artist relation.

```
INSERT into MEDIA(Item_No, Year, Genre, Location, Type, Num_Copies, Price, Num_times_checked_out)
Values(101, 1972, “Soul”, “Chicago”, “Digital”, 5, 3, 0);
```

You must insert a value with an identical Item_No into Album, Movie, or Audiobook at the same time. Can be done in either order, but must be sequential.

```
INSERT into MOVIE(Item_No, Title, Length, Director_Name, Actor_Name)
Values(105, “The Godfather”, 150, “Coppola”, “Pacino”);
```

Director_Name and Actor_Name must be primary keys of Director and Actor relations, respectively. When adding to Album, you must also insert a value into Media with the same Item_No. This can be done in either order, but must be sequential.

```
INSERT into RETURNS(lib_card_No, Item_No, Return_date)
Values(101010101, 104, 04/11/2022);
```

lib_card_No and Item_No must be primary keys of relations in Customer and Media, respectively.

```
INSERT into SOLO(Artist_Name, Num_Records)
Values(“Bill Withers”, 9);
```

Artist_Name must be the primary key of a relation in Artist.

```
INSERT into TRACK(Track_No, Title, Genre, Length, Artist_Name, Album_Name)
Values(5, “Lean On Me”, “Soul”, 258, “Bill Withers”, “Still Bill”);
```

Artist_Name must be the primary key of a relation in Artist. Album_Name must be defined in Album.

```
INSERT into WRITES(Name, Item_No)
Values(“Sherwood Anderson”, 104);
```

Name must be the primary key of a relation in Author. Item_No must be the primary key of a relation in Audiobook.

d. Delete Statements:

```
DELETE FROM ACTOR
WHERE Actor_Name=‘Lorri Pechold’;
```

- Actor_Name from Movie is dependent on ACTOR.Actor_Name
- Actor_Name from Directs is dependent on ACTOR.Actor_Name

```
DELETE FROM ADMIN
WHERE F_name=‘Ava’ AND L_name=‘Poole’;
```

- There are no dependencies for F_name and L_name

```
DELETE FROM ALBUM
WHERE Artist_Name=‘Deep Purple’;
```

- Artist_Name is dependent on ARTIST.Artist_Name. If deleted from ALBUM, should be deleted from Artist

```
DELETE FROM ARTIST
WHERE Artist_Name LIKE ‘c%’;
```

- Artist_Name from Album is dependent on ARTIST.Artist_Name

- Artist_Name from Ensemble is dependent on ARTIST.Artist_Name
- Artist_Name from Band is dependent on ARTIST.Artist_Name
- Artist_Name from Solo is dependent on ARTIST.Artist_Name

DELETE FROM AUDIOBOOK
WHERE Title LIKE ‘Harry Potter%’ AND Length=500;

- There are no dependencies for AUDIOBOOK.Title and AUDIOBOOK.Length

---

DELETE FROM AUTHOR
WHERE Genre=‘Nuclear’;

- There are no dependencies for AUTHOR.Genre

DELETE FROM BAND
WHERE Date_Formed < 1980;

- There are no dependencies for BAND.Date_Formed

DELETE FROM CAN_EDIT
WHERE admin_email=‘a_hawkins@library.com’;

- There are no dependencies for CAN_EDIT.admin_email

DELETE CAN_EDIT_Media
WHERE Item_No > 15;

- There are no dependencies for CAN_EDIT_Media.Item_No

DELETE FROM CHECKS_OUT
WHERE Checkout_date=‘03/02/2022’;

- There are no dependencies for CHECKS_OUT.Checkout_date

DELETE FROM CUST_PAYMENT
WHERE CC_No='3%';

- There are no dependencies for CUST_PAYMENT.CC_No

DELETE FROM CUSTOMER
WHERE Email='wsorensen0@topsy.com' AND Email='svanin3@bloomberg.com' AND Email='dnutteyh@cdc.gov';

- There are no dependencies for CUSTOMER.Email

DELETE FROM DIRECTOR
WHERE Genre='Fiction';

- There are no dependencies for DIRECTOR.Genre

DELETE FROM DIRECTS
WHERE Director_Name LIKE '%Abbott' AND Actor_Name='Sidonia Cadamy';

- There are no dependencies for DIRECTS.Director_Name and DIRECTS.Actor_Name

DELETE FROM ENSEMBLE
WHERE Year > 1970;

- There are no dependencies for ENSEMBLE.Year

DELETE FROM MEDIA
WHERE Price > 99 AND Type='Physical' AND Item_No < 3;

- Item_No from Movie is dependent on MEDIA.Item_No
- There are no dependencies for MEDIA.Price and MEDIA.Type

DELETE FROM MOVIE
WHERE Length < 210;

- There are no dependencies for MOVIE.Length

DELETE FROM RETURNS
WHERE Item_No < 20;

- There are no dependencies for RETURNS.Item_No

DELETE FROM SOLO
WHERE Num_Records < 4 AND Num_Records > 8;

- There are no dependencies for SOLO.Num_Records

DELETE FROM TRACK
WHERE Artist_Name= ‘AC/DC’ AND Title= ‘C.O.D.’ AND Length < 206;

- There are no dependencies for TRACK.Artist_Name, TRACK.Title, and TRACK.Length

---

DELETE FROM WRITES
WHERE Item_No = 56;

- There are no dependencies for WRITES.Item_No
