WolfMedia Streaming Service

CSC 540 - Project Report 2
Team E

Zach Hadgraft (zhadgra), Corey Capooci (cvcapooc), Ethan Purnell (efpurne2), Andrew Shon (ashon)

March 12, 2023

Assumptions:

- Content creators include artists and podcasters. They interact with the database by uploading and removing their content.
- Every song belongs to an album.
- Every album belongs to a single primary artist.
- Every artist contracts with one record label.
    - Every artist is given a unique ID.
- The genre(s) of a song consists of the collection of primary genres for the artist(s) on the song.
- Every record label has a unique name to serve as the key
- Management tracks ratings externally and manually updates the attribute.
- Every podcast has a unique name which can serve as the key.
- Artist name is entered in the first name attribute of Content Creator.
- Additional operations were included beyond those explicitly listed in the project narrative to aid in testing of our database as it is built.
- Counted attributes such as subscribers and monthly listeners can be determined by aggregate values from relationship-based relations.
- If a record is an original release, we will assume the edition to take the value "Standard."
- Royalty rates will be incremented in whole cents (i.e., no royalty rate can take a value of half a cent or similar).
- If a re-release of an album is made, the edition will take the value "Remastered."
- "Duration" attributes will be reported as length in seconds.
- A user cannot "unlisten" to a song or podcast episode (i.e., there is no need to revise these numbers downwards).
- Timestamps are unique; if two operations try to occur at the same time the transaction will be held and inserted at a different time.
- Genre for artist is described by the primaryGenre attribute, not a separate relation for artist genres.

1. Global Relational Database Schema:

PodcastHost(<u>creatorId</u>, email, phone, city)

creatorId → email, phone, city holds because the creatorId is uniquely assigned to every content creator, identifying the individual's email, phone, city, and additional attributes in the **ContentCreator** relation.

email → creatorId, phone, city is a potential functional dependency (FD) which is valid because every email address must be unique, and would therefore be tied to a single **creatorId**.

phone → creatorId, email, city is another potential FD which is valid for the same reason as the email depency.

The left side of each of these FDs consists of a superkey; therefore, the relation is in BCNF, which implies 3NF. The only attribute remaining (**city**) could not lead to a functional dependency, as multiple content creators could be located in the same city.

Hosts(<u>podcastName</u>, <u>creatorId</u>)

podcastName, creatorId → podcastName, creatorId holds because the key consists of the entire set of attributes for this relation and is thus a superkey. Therefore, the relation is in BCNF, which implies 3NF.

Podcast(<u>podcastName</u>, language, country, rating)

podcastName → language, country, rating holds because of our assumption that no two podcasts will share the same **podcastName**. As a result, **podcastName** is a unique identifier that can determine the language, country, and rating of that podcast. One could imagine that it is possible for multiple podcasts to be in a given language, to be hosted from a particular country, or to have the same rating. Since the left side of the FD is a superkey, the relation is in BCNF, which implies 3NF.

Sponsors(<u>podcastName</u>, <u>sponsorName</u>)

podcastName, sponsorName → podcastName, sponsorName holds because the key consists of the entire set of attributes for this relation and is thus a superkey. Therefore, the relation is in BCNF, which implies 3NF.

Sponsor(<u>sponsorName</u>)

sponsorName → sponsorName holds because there is only a single attribute, which is also the key. Therefore, the relation is in BCNF, which implies 3NF.

PodcastGenre(<u>genreName</u>)

genreName → genreName holds because there is only a single attribute, which is also the key. Therefore, the relation is in BCNF, which implies 3NF.

DescribesPodcast(<u>podcastName</u>, <u>genreName</u>)

podcastName, genreName → podcastName, genreName holds because the key consists of the entire set of attributes for this relation and is thus a superkey. Therefore, the relation is in BCNF, which implies 3NF.

PodcastEpisode(<u>title</u>, <u>podcastName</u>, duration, releaseDate, advertisementCount)

title, podcastName → duration, releaseDate, advertisementCount is in 3NF due to our assumption that no two podcasts would share the same **podcastName**. One could imagine multiple podcasts having the same duration, release date, or number of advertisements. Since **PodcastEpisode** is a weak entity set, the combination of **title** (which could be duplicated between different podcasts) and **podcastName** can uniquely identify a given episode. Since the left side of this FD is a superkey, the relation is in BCNF, which implies 3NF.

GuestStars(<u>creatorId</u>, <u>title</u>, <u>podcastName</u>)

creatorId, title, podcastName → creatorId, title, podcastName holds because this relationship consists of 3 attributes which are all keys, thus leaving the left side as a superkey. As a result, this relation is in 3NF. No other set of attributes could definitively produce all 3 attributes.

SpecialGuest(<u>creatorId</u>)

creatorId → creatorId holds because there is only a single attribute, which is also the key. Therefore, the relation is in BCNF, which implies 3NF.

Artist(<u>creatorId</u>, status, type, country, primaryGenre)

creatorId → status, type, country, primaryGenre holds 3NF because the creatorId value uniquely identifies an artist. Therefore, creatorId is a superkey in this relation.

Contracts(<u>creatorId</u>, <u>recordLabelName</u>)

creatorId, recordLabelName → creatorId, recordLabelName holds 3NF because creatorId and recordLabelName are the entire set of attributes and is therefore a superkey. There are no other functional dependencies since creatorId and recordLabelName cannot derive from the other. There could be many creators per record label and a creator could have signed with multiple record labels throughout their career.

RecordLabel(<u>labelName</u>)

labelName → labelName holds 3NF because this relation only has one attribute labelName. Since labelName must be the key, labelName is also a superkey.

Owns(<u>creatorId</u>, <u>recordLabelName</u>, <u>songTitle</u>, <u>albumName</u>, <u>edition</u>)

creatorId, songTitle, albumName, edition → recordLabelName holds 3NF because the entire relation is provided in this functional dependency. Therefore, it shows that the LHS is a superkey that derives all attributes. This is true because the song is owned by one record label.

Other functional dependencies may seem to hold, for example **songTitle, creatorId → albumName, edition** . Based on our assumptions for this project, the song title may be used across many different albums and editions.

Also **songTitle → recordLabelName** does not hold since songTitle by itself does not uniquely identify the song.

Also **songTitle, albumName, edition → creatorId** is not valid since artists can cover other artists using the same song title, album name, and edition.

The FD **creatorId → recordLabelName** does not hold on this relation since the creator's record label may have changed and the current creator's record label may not reflect the record label for the song.

User(firstName, lastName, <u>email</u>, subscriptionFee, statusOfSubscription, phone, registrationDate)

email → firstName, lastName, subscriptionFee, statusOfSubscription, phone, registrationDate - email functionally determines all other attributes so the schema is in 3NF; the system is designed such that a single email points to a single account.

We consider other FDs with the following attributes on the LHS:

- firstName, lastName - consider that a user may lose track of their account and create a new one later; it is not unusual for someone to have more than one email address so the same firstName, lastName may point to a different email
- phone - again, a user may lose track of an account and create a new one with a new email with a number already in the system.

ListensToSong(<u>email</u>, <u>songTitle,</u> <u>creatorId</u>, <u>albumName</u>, <u>edition</u>, <u>timestamp</u>)

timestamp → email, songTitle, creatorId, albumName, edition

The LHS functionally determines all other attributes so the schema is in 3NF. We assume that someone can listen to a song at different times so any combination of the other five attributes will not point to a unique timestamp.

SubscribesToPodcast(<u>userEmail</u>, <u>podcastName</u>, <u>timestamp</u>)

timestamp → userEmail, podcastName

userEmail, podcastName → timestamp

The LHSes for all FDs functionally determine all attributes. Therefore, it is 3NF. A user subscribes once to a podcast so a tuple with the same userEmail and creatorId should point to a single timestamp; the reverse also holds.

ListensToPodcastEpisode(<u>userEmail</u>, <u>podcastEpisodeTitle</u>, <u>timestamp</u>)

timestamp → timestamp, userEmail, podcastEpisodeTitle

This LHS functionally determines all other attributes so the schema is in 3NF. We assume that someone can listen to a podcast episode at different times so any combination of the other two attributes will not point to a unique timestamp.

Payment(<u>paymentId</u>, date, value)

paymentId → date, value - payment IDs should uniquely identify a payment so payors can reference payments. This LHS functionally determines all other attributes so the schema is in 3NF.

PayToContentCreator(<u>creatorId</u>, <u>paymentId</u>) - in 3NF because it has two attributes.
PayToRecordLabel(<u>paymentId</u>, <u>recordLabelName</u>) - in 3NF because it has two attributes.

PayFromUser(paymentId, userEmail) - in 3NF because it has two attributes.
Pays(paymentId, userEmail) - in 3NF because it has two attributes.

Song(creatorId, albumName, edition, songTitle, trackNumber, duration, playCount, releaseDate, releaseCountry, language, royaltyPaidStatus, royaltyRate)

creatorId, albumName, edition, songTitle → creatorId, albumName, edition, songTitle, trackNumber, duration, playCount, releaseDate, releaseCountry, language, royaltyPaidStatus, royaltyRate satisfies 3NF because the current set of keys functionally determines all other attributes. The right hand side could have multiple tuples with the exact same value, but does not mean they come from the same creatorId, albumName, edition, or songTitle. The left-hand side is in BCNF, therefore, it is in 3NF.

creatorId, songTitle → albumName would not hold, as an album could be re-released or released as multiple editions, in which case this FD would not hold.

CollaboratesOn(creatorId, guestArtistId, songTitle, albumName, edition)

creatorId, guestArtistId, songTitle, albumName, edition → creatorId, guestArtistId, songTitle, albumName, edition satisfies 3NF because all of the attributes are within the key of the schema, so this makes it a superkey.

Album(creatorId, albumName, edition, releaseYear)

albumName, edition → releaseYear satisfies 3NF because each album and edition combination will functionally determine release year. The left-hand side is a superkey, therefore this relation is in BCNF, which also makes it 3NF.

ContentCreator(creatorId, firstName, lastName)

creatorId → firstName, lastName satisfies 3NF because creatorId functionally determines all the other attributes, since it's unique value
firstName, lastName → creatorId - we anticipate that multiple users might have the same first name and last name so we do not want to assert this FD.

2. Global Database Schema Design Decisions:

Design decision description:

We converted the entity sets from our diagram into relations with all of their respective attributes. We decided to manage some potential attributes (monthly listeners, subscribers, podcast listeners) from the Project Narrative implicitly. For example, there is a SubscribesTo relationship between User and Podcast – rather than having subscribers as an attribute of Podcast, the subscriptions are kept in their own relation and we developed a query to count the number of subscribers given a particular podcast. The entity sets that are subsets of ContentCreator were made into relations using the E/R approach to avoid redundancy as with the object-oriented approach and to reduce search complexity that would arise from the NULLS approach.

There are several many-one relationships or candidates for many-one relationships in our E/R diagram, though we were not able to compress them into attributes. In trying to faithfully recreate the real world with our design decisions, we avoided some simplifying assumptions (such as assuming that a podcast can only have one sponsor, allowing Sponsor to be collapsed to an attribute of Podcast). The other entities with many-one relationships were either weak entities with multiple dependencies or were involved in other many-many relationships (such as RecordLabel) and could not be collapsed to an attribute of any other entity.

Other relationships were turned into relations in our schema, except for the supporting relationships on weak entity sets. As the keys all overlap for the weak entity sets, no relationship relation was necessary to join those entity relations together. The remaining relations have keys of their connected entities as attributes, and in some cases have their own attributes (timestamp attributes were included on many "action" relationships such as ListensToSong and ListensToPodcastEpisode).

Relation explanations:

ContentCreator(creatorId, firstName, lastName)

creatorId is the primary key and cannot be NULL.
firstName cannot be NULL and lastName can be NULL; artist may use firstName only in some cases, e.g. a band name.

Song(creatorId, albumName, edition, songTitle, trackNumber, duration, playCount, releaseDate, releaseCountry, language, royaltyPaidStatus, royaltyRate)

Song requires that both the creator and album exist.
songTitle is a primary key.
creatorId is a foreign key which references the creator.
The albumName and edition are foreign keys referencing the album.
No values can be NULL.

CollaboratesOn(creatorId, guestArtistId, songTitle, albumName, edition)

Creators, album, and song are required to exist.
For the first entity creatorId and guestArtistId are foreign keys that reference creators.
albumName, edition, and songTitle are foreign keys that reference the song that is collaborated on.
No values can be NULL.

Album(creatorId, albumName, edition, releaseYear)

Creator is required to exist.
creatorId is a foreign key used to reference the creator the album is attributed to.
albumName and edition are primary keys and cannot be NULL.

releaseYear cannot be Null.

User(<u>email</u>, firstName, lastName, subscriptionFee, statusOfSubscription, phone, registrationDate)

email is the primary key, as every email address must be unique.
firstName, lastName, statusOfSubscription, and registrationDate are not allowed to be null.

subscriptionFee is not allowed to be null; if a user subscribes to a free plan where there is no fee the value defaults to zero.

phone is allowed to be null, as it is possible that a user may not have a phone and uses the service exclusively on his or her computer. In this case, null simply represents a non-existent phone number and implies that the user cannot be reached by phone.

ListensToSong(<u>email</u>, <u>songTitle</u>, <u>creatorId</u>, <u>albumName</u>, <u>edition</u>, <u>timestamp</u>)

Album, User, Song, and Creator associated with the keys must exist.

email, songTitle, id, albumName, and edition are all foreign keys that are required to track a user's listening activity. Timestamp is a primary key.

SubscribesToPodcast(<u>email</u>, <u>podcastName</u>, <u>timestamp</u>)

User and Podcast supplying the email and podcastName keys must exist.

email and podcastName are the two foreign keys required to make this relation, while timestamp is the primary key for the relation.

ListensToPodcastEpisode(<u>email</u>, <u>podcastEpisodeTitle</u>, <u>podcastName</u>, <u>timestamp</u>)

User, Podcast, and Podcast Episode associated with the keys must exist.

email, podcastEpisodeTitle, and podcastName are all foreign keys required to track the user's listening activity. Timestamp is a primary key.

Payment(<u>paymentId</u>, date, value)

paymentId is the primary key for this entity relation.

date and value are not allowed to be null. If value were null, there would be no payment, and if the date were null, it would not be possible to tell what month that payment covers.

PayToContentCreator(<u>creatorId</u>, <u>paymentId</u>)

ContentCreator and Payment entities supplying the keys must exist.

creatorId and paymentId are both foreign keys for this relationship relation.

PayToRecordLabel(<u>paymentId</u>, <u>recordLabelName</u>)

RecordLabel and Payment entities supplying the keys must exist.

paymentId and recordLabelName are both foreign keys for this relationship relation.

PayFromUser(<u>paymentId</u>, <u>email</u>)

User and Payment entities supplying the keys must exist.

paymentId and userEmail are both foreign keys for this relationship relation.

Artist(<u>creatorId</u>, status, type, country, primaryGenre)

Creator must exist
creatorId is the foreign key
creatorId, status, type, country, and primaryGenre are not allowed to be null

Contracts(<u>creatorId</u>, <u>recordLabelName</u>)

Creator and Record must exist
creatorId is one of the two foreign keys
recordLabelName is one of the two foreign keys

RecordLabel(<u>labelName</u>)

labelName is the primary key

Owns(<u>creatorId</u>, <u>recordLabelName</u>, <u>songTitle</u>, <u>albumName</u>, <u>edition</u>)

Creator, RecordLabel, Song, and Album must exist
creatorId is one of the five foreign keys
recordLabelName is the second of the five foreign keys
songTitle is the third of the five foreign keys
albumName is the fourth of the five foreign keys
edition is the fifth of the five foreign keys

PodcastHost(<u>creatorId</u>, email, phone, city)

Creator relation must exist.
creatorId is a foreign key.
email may be NULL if it is not provided. Therefore, NULL means empty.
phone may be NULL if it is not provided. Therefore, NULL means empty.
city may be NULL if it is not provided. Therefore, NULL means empty.

Hosts(<u>podcastName</u>, <u>creatorId</u>)

Creator relation must exist.
Podcast relation must exist.
podcastName is one of the 2 foreign keys.

creatorId is the second of the 2 foreign keys.

Podcast(<u>podcastName</u>, language, country, rating)
podcastName is the primary key.
language is not allowed to be NULL, since every listener would want to know the language.
country is not allowed to be NULL.
rating is not allowed to be NULL.

Sponsors(<u>podcastName</u>, <u>sponsorName</u>)
Podcast must exist.
Sponsor must exist.
podcastName is one of the 2 foreign keys.
sponsorName is the second of 2 foreign keys.

Sponsor(<u>sponsorName</u>)
sponsorName is the primary key.

PodcastGenre(<u>genreName</u>)
genreName is the primary key.

DescribesPodcast(<u>podcastName</u>, <u>genreName</u>)
Podcast must exist.
Genre must exist.
podcastName is one of the 2 foreign keys.
genreName is the second of 2 foreign keys.

PodcastEpisode(<u>title</u>, <u>podcastName</u>, duration, releaseDate, advertisementCount)
Podcast must exist.
title is a primary key.
podcastName is a foreign key.
duration must be provided.
releaseDate must be provided.
advertisementCount must be provided.

GuestStars(<u>creatorId</u>, <u>title</u>, <u>podcastName</u>)
Creator must exist.
PodcastEpisode must exist
Podcast must exist.
creatorId is the first of 3 foreign keys.
title is the second of 3 foreign keys.
podcastName is the third of 3 foreign keys.

SpecialGuest(<u>creatorId</u>)

Creator must exist.
creatorId is the foreign key.

3. Using the terminal for MariaDB for all your relation schemas create base relations

```
CREATE TABLE ContentCreator (
    creatorId INT,
    firstName VARCHAR(255) NOT NULL,
    lastName VARCHAR(255),
    PRIMARY KEY(creatorId)
);


CREATE TABLE User (
    email VARCHAR(255),
    firstName VARCHAR(255) NOT NULL,
    lastName VARCHAR(255) NOT NULL,
    subscriptionFee DECIMAL(9,2) NOT NULL,
    statusOfSubscription CHAR(1) NOT NULL
    CHECK (statusOfSubscription IN ('A', 'I')),
    phone VARCHAR(16),
    registrationDate DATE NOT NULL,
    PRIMARY KEY(email)
);


CREATE TABLE ListensToSong (
    email VARCHAR(255),
    songTitle VARCHAR(255),
    creatorId INT,
    albumName VARCHAR(255),
    edition VARCHAR(255),
    timestamp DATETIME,
      PRIMARY KEY(email, songTitle, creatorId, albumName, edition,
timestamp),
    FOREIGN KEY (email) REFERENCES User(email) ON UPDATE CASCADE,
    FOREIGN KEY (creatorId, albumName, edition, songTitle)
      REFERENCES Song(creatorId, albumName, edition, songTitle) ON
UPDATE CASCADE
);
```

```
CREATE TABLE SubscribesToPodcast (
    email VARCHAR(255),
    podcastName VARCHAR(255),
    timestamp DATETIME,
    PRIMARY KEY(email, podcastName, timestamp),
    FOREIGN KEY(email) REFERENCES User(email) ON UPDATE CASCADE,
       FOREIGN  KEY(podcastName)  REFERENCES  Podcast(podcastName)  ON
UPDATE CASCADE
);


CREATE TABLE ListensToPodcastEpisode (
    userEmail VARCHAR(255),
    podcastEpisodeTitle VARCHAR(255),
    podcastName VARCHAR(255),
    timestamp DATETIME,
         PRIMARY  KEY(userEmail,  podcastEpisodeTitle,  podcastName,
timestamp),
    FOREIGN KEY(userEmail) REFERENCES User(email) ON UPDATE CASCADE,
    FOREIGN KEY(podcastEpisodeTitle) REFERENCES PodcastEpisode(title)
ON UPDATE CASCADE,
       FOREIGN  KEY(podcastName)  REFERENCES  Podcast(podcastName)  ON
UPDATE CASCADE
);


CREATE TABLE Payment (
    paymentId INT,
    date DATE NOT NULL,
    value DECIMAL(9,2) NOT NULL,
    PRIMARY KEY(paymentId)
);


CREATE TABLE PayToContentCreator (
    creatorId INT,
    paymentId INT,
    PRIMARY KEY(creatorId, paymentId),
     FOREIGN KEY  (creatorId) REFERENCES ContentCreator(creatorId) ON
UPDATE CASCADE,
      FOREIGN  KEY(paymentId)  REFERENCES  Payment(paymentId)  ON  UPDATE
CASCADE
);
```

```
CREATE TABLE PayToRecordLabel(
    paymentId INT,
    recordLabelName VARCHAR(255),
    PRIMARY KEY(paymentId, recordLabelName),
      FOREIGN KEY(paymentId) REFERENCES Payment(paymentId) ON UPDATE
CASCADE,
     FOREIGN KEY (recordLabelName) REFERENCES RecordLabel(labelName)
ON UPDATE CASCADE
);


CREATE TABLE PayFromUser(
    paymentId INT,
    userEmail VARCHAR(255),
    PRIMARY KEY(paymentId, userEmail),
      FOREIGN KEY(paymentId) REFERENCES Payment(paymentId) ON UPDATE
CASCADE,
    FOREIGN KEY(userEmail) REFERENCES User(email) ON UPDATE CASCADE
);


CREATE TABLE Song (
    creatorId INT,
    albumName VARCHAR(255),
    edition VARCHAR(255),
    songTitle VARCHAR(255),
    trackNumber INT NOT NULL,
    duration INT NOT NULL,
    playCount INT NOT NULL,
    releaseDate DATE NOT NULL,
    releaseCountry VARCHAR(255) NOT NULL,
    language VARCHAR(25) NOT NULL,
    royaltyPaidStatus BOOLEAN,
    royaltyRate DECIMAL(2,2) NOT NULL,
    PRIMARY KEY(creatorId, albumName, edition, songTitle),
          FOREIGN KEY(creatorId, albumName, edition) REFERENCES
Album(creatorId, albumName, edition) ON UPDATE CASCADE
);
```

```
CREATE TABLE CollaboratesOn (
    creatorId INT,
    guestArtistId INT,
    songTitle VARCHAR(255),
    albumName VARCHAR(255),
    edition VARCHAR(255),
       PRIMARY  KEY(creatorId,  guestArtistId,  songTitle,  albumName,
edition),
     FOREIGN KEY (guestArtistId) REFERENCES ContentCreator(creatorId)
ON UPDATE CASCADE,
    FOREIGN KEY (creatorId, albumName, edition, songTitle)
      REFERENCES  Song(creatorId,  albumName,  edition,  songTitle)  ON
UPDATE CASCADE
);


CREATE TABLE Album (
    creatorId INT,
    albumName VARCHAR(255),
    edition VARCHAR(255),
    releaseYear INT(4) NOT NULL,
    PRIMARY KEY(creatorId, albumName, edition),
     FOREIGN  KEY  (creatorId)  REFERENCES  ContentCreator(creatorId)  ON
UPDATE CASCADE ON DELETE CASCADE
);


CREATE TABLE Artist (
    creatorId INT,
    status CHAR(1) NOT NULL
    CHECK (statusOfSubscription IN ('A', 'I')),
    type VARCHAR(25) NOT NULL,
    country VARCHAR(255) NOT NULL,
    primaryGenre VARCHAR(255) NOT NULL,
    PRIMARY KEY(creatorId),
      FOREIGN  KEY(creatorId)  REFERENCES  ContentCreator(creatorId)  ON
UPDATE CASCADE
);
```

```
CREATE TABLE Contracts (
    creatorId INT,
    recordLabelName VARCHAR(255),
    PRIMARY KEY(creatorId, recordLabelName),
      FOREIGN KEY(creatorId) REFERENCES ContentCreator(creatorId) ON
UPDATE CASCADE,
    FOREIGN KEY(recordLabelName) REFERENCES RecordLabel(labelName) ON
UPDATE CASCADE
);


CREATE TABLE RecordLabel (
    labelName VARCHAR(255),
    PRIMARY KEY (labelName)
);


CREATE TABLE Owns (
    creatorId INT,
    recordLabelName VARCHAR(255),
    songTitle VARCHAR(255),
    albumName VARCHAR(255),
    edition VARCHAR (255),
      PRIMARY KEY(creatorId, recordLabelName, songTitle, albumName,
edition),
    FOREIGN KEY(recordLabelName) REFERENCES RecordLabel(labelName) ON
UPDATE CASCADE,
     FOREIGN KEY(creatorId, albumName, edition, songTitle) REFERENCES
Song(creatorId, albumName, edition, songTitle) ON UPDATE CASCADE
);


CREATE TABLE PodcastHost (
    creatorId INT,
    email VARCHAR(255),
    phone VARCHAR(16),
    city VARCHAR(255),
    PRIMARY KEY(creatorId),
      FOREIGN KEY(creatorId) REFERENCES ContentCreator(creatorId) ON
UPDATE CASCADE
);
```

```
CREATE TABLE Hosts (
    podcastName VARCHAR(255) NOT NULL,
    creatorId INT,
    PRIMARY KEY(podcastName, creatorId),
       FOREIGN  KEY(podcastName)  REFERENCES  Podcast(podcastName)  ON
UPDATE CASCADE,
      FOREIGN  KEY(creatorId)  REFERENCES  ContentCreator(creatorId)  ON
UPDATE CASCADE
);


CREATE TABLE Podcast (
    podcastName VARCHAR(255),
    language VARCHAR(25) NOT NULL,
    country VARCHAR(255) NOT NULL,
    rating DECIMAL(2,1) NOT NULL,
    PRIMARY KEY(podcastName)
);


CREATE TABLE Sponsors (
    podcastName VARCHAR(255),
    sponsorName VARCHAR(255),
    PRIMARY KEY(podcastName, sponsorName),
       FOREIGN  KEY(podcastName)  REFERENCES  Podcast(podcastName)  ON
UPDATE CASCADE,
       FOREIGN  KEY(sponsorName)  REFERENCES  Sponsor(sponsorName)  ON
UPDATE CASCADE
);


CREATE TABLE Sponsor (
    sponsorName VARCHAR(255),
    PRIMARY KEY(sponsorName)
);


CREATE TABLE PodcastGenre (
    genreName VARCHAR(255),
    PRIMARY KEY(genreName)
);
```

```
CREATE TABLE DescribesPodcast (
    podcastName VARCHAR(255),
    genreName VARCHAR(255),
    PRIMARY KEY(podcastName, genreName),
       FOREIGN KEY(podcastName) REFERENCES Podcast(podcastName) ON
UPDATE CASCADE,
      FOREIGN KEY(genreName) REFERENCES PodcastGenre(genreName) ON
UPDATE CASCADE
);


CREATE TABLE PodcastEpisode (
    title VARCHAR(255),
    podcastName VARCHAR(255),
    duration INT NOT NULL,
    releaseDate DATE NOT NULL,
    advertisementCount INT NOT NULL,
    PRIMARY KEY(title, podcastName),
       FOREIGN KEY(podcastName) REFERENCES Podcast(podcastName) ON
UPDATE CASCADE
);


CREATE TABLE GuestStars (
    creatorId INT,
    title VARCHAR(255),
    podcastName VARCHAR(255),
    PRIMARY KEY(creatorId, title, podcastName),
      FOREIGN KEY(creatorId) REFERENCES ContentCreator(creatorId) ON
UPDATE CASCADE,
     FOREIGN KEY(title, podcastName) REFERENCES PodcastEpisode(title,
podcastName) ON UPDATE CASCADE
);
```

```
SELECT * FROM User;
+------------------------------------+-----------+------------+-----------------+----------------------+----------------+------------------+
| email                              | firstName | lastName   | subscriptionFee | statusOfSubscription | phone          | registrationDate |
+------------------------------------+-----------+------------+-----------------+----------------------+----------------+------------------+
| bigbadbureaucrat@planetexpress.com | Hermes    | Conrad     |            0.00 | I                    | NULL           | 3000-01-01       |
| curseWERNSTROM@planetexpress.com   | Hubert    | Farnsworth |            1.00 | A                    | NULL           | 2902-07-18       |
| cyclopsRULEZ@futuremail.net        | Turanga   | Leela      |           99.99 | A                    | NULL           | 3001-12-01       |
| girder_guy@planetexpress.com       | Bender    | Rodriguez  |            0.99 | A                    | (999) 635-6889 | 2994-04-21       |
| lonely_lobster@futuremail.net      | John      | Zoidberg   |           99.99 | A                    | (259) 444-8294 | 3001-12-02       |
| smallfry@aol.com                   | Philip    | Fry        |            9.99 | A                    | (555) 123-4567 | 1999-05-22       |
+------------------------------------+-----------+------------+-----------------+----------------------+----------------+------------------+
SELECT * FROM ContentCreator;
+-----------+-------------------+----------+
| creatorId | firstName         | lastName |
+-----------+-------------------+----------+
|         1 | The Mars Volta    | NULL     |
|         2 | Lex               | Fridman  |
|         3 | Joe               | Rogan    |
|         4 | The Fall of Troy  | NULL     |
|         5 | Mac Miller        | NULL     |
|         6 | TTNG              | NULL     |
|         7 | Tim               | Ferriss  |
|         8 | Leila             | Fadel    |
+-----------+-------------------+----------+

SELECT * FROM Payment;
+-----------+------------+-------+
| paymentId | date       | value |
+-----------+------------+-------+
|         1 | 3001-03-01 | 99.99 |
|         2 | 3001-04-01 | 99.99 |
|         3 | 3001-05-01 | 99.99 |
|         4 | 3001-06-01 | 99.99 |
|         5 | 3001-07-01 | 99.99 |
+-----------+------------+-------+


SELECT * FROM PayFromUser;
+-----------+--------------------------------+
| paymentId | userEmail                      |
+-----------+--------------------------------+
|         1 | lonely_lobster@futuremail.net |
|         2 | lonely_lobster@futuremail.net |
|         3 | cyclopsRULEZ@futuremail.net   |
|         4 | lonely_lobster@futuremail.net |
|         5 | cyclopsRULEZ@futuremail.net   |
+-----------+--------------------------------+


SELECT * FROM Album;
+-----------+------------------------------+----------+-------------+
| creatorId | albumName                    | edition  | releaseYear |
+-----------+------------------------------+----------+-------------+
|         1 | Amputechture                 | Standard |        2006 |
|         1 | De-Loused in the Comatorium | Standard |        2003 |
|         1 | The Bedlam in Goliath        | Standard |        2008 |
|         4 | Doppelganger                 | Standard |        2003 |
|         4 | Manipulator                  | Standard |        2007 |
|         4 | Phantom on the Horizon       | Standard |        2008 |
+-----------+------------------------------+----------+-------------+
SELECT * FROM Artist;
+-----------+--------+----------+---------------+-------------------+
| creatorId | status | type     | country       | primaryGenre      |
+-----------+--------+----------+---------------+-------------------+
|         1 | A      | Band     | United States | Progressive Rock |
|         4 | A      | Band     | United States | Mathcore          |
|         5 | I      | Musician | United States | Hip Hop           |
|         6 | A      | Band     | England       | Math rock         |
+-----------+--------+----------+---------------+-------------------+

SELECT * FROM RecordLabel;
+---------------+
| labelName     |
+---------------+
| Clouds Hill   |
| Equal Vision  |
| Rostrum       |
| Sargent House |
+---------------+

SELECT * FROM Contracts;
+-----------+-----------------+
| creatorId | recordLabelName |
+-----------+-----------------+
|         1 | Clouds Hill     |
|         4 | Equal Vision    |
|         5 | Rostrum         |
|         6 | Sargent House   |
+-----------+-----------------+

SELECT * FROM PodcastHost;
+-----------+---------------------+----------------+--------------+
| creatorId | email               | phone          | city         |
+-----------+---------------------+----------------+--------------+
|         2 | NULL                | (123) 456-7890 | Boston       |
|         3 | NULL                | NULL           | NULL         |
|         7 | timmy_f@4hours.com  | NULL           | East Hampton |
|         8 | leila_fadel@npr.org | (999) 999-9999 | NULL         |
+-----------+---------------------+----------------+--------------+

SELECT * FROM Podcast;
+---------------------------+----------+---------------+--------+
| podcastName               | language | country       | rating |
+---------------------------+----------+---------------+--------+
| Lex Fridman Podcast       | English  | United States |    4.9 |
| The Joe Rogan Experience | English  | United States |    4.8 |
| The Tim Ferriss Show      | English  | United States |    4.9 |
| Up First                  | English  | United States |    4.8 |
+---------------------------+----------+---------------+--------+
SELECT * FROM Hosts;
+--------------------------+-----------+
| podcastName              | creatorId |
+--------------------------+-----------+
| Lex Fridman Podcast      |         2 |
| The Joe Rogan Experience |         3 |
| The Tim Ferriss Show     |         7 |
| Up First                 |         8 |
+--------------------------+-----------+
```

```
SELECT * FROM PodcastEpisode;
+-------------------------------------------------------------------+--------------------------+----------+-------------+--------------------+
| title                                                             | podcastName              | duration | releaseDate | advertisementCount |
+-------------------------------------------------------------------+--------------------------+----------+-------------+--------------------+
| #1951 - Coffeezilla                                               | The Joe Rogan Experience |      185 | 2023-03-07  |                  3 |
| #1952 - Michael Malice                                            | The Joe Rogan Experience |      209 | 2023-03-08  |                  3 |
| #284 - Saifedean Ammous: Bitcoin, Anarchy, and Austrian Economics | Lex Fridman Podcast      |      238 | 2022-05-11  |                  4 |
| Florida Legislative Session, Powell Testimony, French Strikes     | Up First                 |       14 | 2023-03-07  |                  1 |
+-------------------------------------------------------------------+--------------------------+----------+-------------+--------------------+
```

```
SELECT * FROM ListensToPodcastEpisode;
+-----------------------------------+-------------------------------------------------------------------+--------------------------+-----------------
----+
| userEmail                         | podcastEpisodeTitle                                               | podcastName              | timestamp
|
+-----------------------------------+-------------------------------------------------------------------+--------------------------+-----------------
----+
| curseWERNSTROM@planetexpress.com  | #284 - Saifedean Ammous: Bitcoin, Anarchy, and Austrian Economics | Lex Fridman Podcast      | 2023-03-08
20:22:01 |
| cyclopsRULEZ@futuremail.net       | Florida Legislative Session, Powell Testimony, French Strikes     | Up First                 | 2023-03-08
20:21:41 |
| girder_guy@planetexpress.com      | #1952 - Michael Malice                                            | The Joe Rogan Experience | 2023-03-08
20:22:36 |
| lonely_lobster@futuremail.net     | #1951 - Coffeezilla                                               | The Joe Rogan Experience | 2023-03-08
20:21:34 |
+-----------------------------------+-------------------------------------------------------------------+--------------------------+-----------------
----+
```

```
SELECT * FROM SubscribesToPodcast;
+----------------------------------+--------------------------+---------------------+
| email                            | podcastName              | timestamp           |
+----------------------------------+--------------------------+---------------------+
| curseWERNSTROM@planetexpress.com | Lex Fridman Podcast      | 2023-03-08 20:27:55 |
| cyclopsRULEZ@futuremail.net      | Up First                 | 2023-03-08 20:27:18 |
| girder_guy@planetexpress.com     | The Joe Rogan Experience | 2023-03-08 20:26:20 |
| smallfry@aol.com                 | The Joe Rogan Experience | 2023-03-08 20:28:34 |
+----------------------------------+--------------------------+---------------------+
```

```
SELECT * FROM Song;
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
| creatorId | albumName    | edition  | songTitle | trackNumber | duration | playCount | releaseDate | releaseCountry | language | royaltyPaidStatus | royaltyRate |
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
|         1 | Amputechture | Standard | Title 1   |           1 |      439 |      1234 | 2023-01-12  | US             | English  |                 0 |        0.23 |
|         1 | Amputechture | Standard | Title 2   |           2 |      200 |    890029 | 2023-01-12  | US             | English  |                 0 |        0.25 |
|         1 | Amputechture | Standard | Title 3   |           3 |      250 |     50699 | 2023-01-12  | US             | English  |                 0 |        0.24 |
|         1 | Amputechture | Standard | Title 4   |           4 |      280 |      4600 | 2023-01-12  | US             | English  |                 0 |        0.10 |
|         4 | Doppelganger | Standard | Title a   |           1 |      310 |     60105 | 2023-05-07  | US             | English  |                 1 |        0.08 |
|         4 | Doppelganger | Standard | Title b   |           2 |      320 |    323105 | 2023-02-12  | US             | English  |                 1 |        0.27 |
|         4 | Doppelganger | Standard | Title c   |           3 |      270 |     69105 | 2023-05-07  | US             | English  |                 1 |        0.07 |
|         4 | Doppelganger | Standard | Title d   |           4 |      250 |    760105 | 2023-05-07  | US             | English  |                 1 |        0.10 |
|         4 | Doppelganger | Standard | Title e   |           5 |      255 |    160105 | 2023-05-07  | US             | English  |                 1 |        0.12 |
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
```

```
SELECT * FROM ListenToSong;
+----------------------------------+-----------+-----------+--------------+----------+---------------------+
| email                            | songTitle | creatorId | albumName    | edition  | timestamp           |
+----------------------------------+-----------+-----------+--------------+----------+---------------------+
| curseWERNSTROM@planetexpress.com | Title 1   |         1 | Amputechture | Standard | 2011-02-11 00:00:00 |
| curseWERNSTROM@planetexpress.com | Title 4   |         1 | Amputechture | Standard | 2011-02-11 00:00:00 |
| cyclopsRULEZ@futuremail.net      | Title a   |         4 | Doppelganger | Standard | 2011-02-11 00:00:00 |
| cyclopsRULEZ@futuremail.net      | Title a   |         4 | Doppelganger | Standard | 2011-02-11 00:03:00 |
| cyclopsRULEZ@futuremail.net      | Title d   |         4 | Doppelganger | Standard | 2011-02-11 00:00:00 |
| smallfry@aol.com                 | Title 1   |         1 | Amputechture | Standard | 2011-02-11 00:00:00 |
| smallfry@aol.com                 | Title 2   |         1 | Amputechture | Standard | 2011-02-11 00:03:00 |
| smallfry@aol.com                 | Title a   |         4 | Doppelganger | Standard | 2011-02-11 00:00:00 |
| smallfry@aol.com                 | Title c   |         4 | Doppelganger | Standard | 2011-02-11 00:00:00 |
+----------------------------------+-----------+-----------+--------------+----------+---------------------+
SELECT * FROM PayToContentCreator;
+-----------+-----------+
| creatorId | paymentId |
+-----------+-----------+
|         1 |         1 |
|         1 |         2 |
|         6 |         3 |
|         7 |         4 |
|         8 |         5 |
+-----------+-----------+

SELECT * FROM PayToRecordLabel;
+-----------+-----------------+
| paymentId | recordLabelName |
+-----------+-----------------+
|         1 | Clouds Hill     |
|         2 | Clouds Hill     |
|         3 | Equal Vision    |
|         4 | Rostrum         |
|         5 | Equal Vision    |
+-----------+-----------------+

SELECT * FROM CollaboratesOn;
+-----------+---------------+-----------+--------------+----------+
| creatorId | guestArtistId | songTitle | albumName    | edition  |
+-----------+---------------+-----------+--------------+----------+
|         1 |             5 | Title 1   | Amputechture | Standard |
|         1 |             6 | Title 1   | Amputechture | Standard |
|         1 |             6 | Title 2   | Amputechture | Standard |
|         4 |             1 | Title d   | Doppelganger | Standard |
|         4 |             6 | Title a   | Doppelganger | Standard |
+-----------+---------------+-----------+--------------+----------+

SELECT * FROM Owns;
+-----------+-----------------+-----------+--------------+----------+
| creatorId | recordLabelName | songTitle | albumName    | edition  |
+-----------+-----------------+-----------+--------------+----------+
|         1 | Clouds Hill     | Title 1   | Amputechture | Standard |
|         1 | Clouds Hill     | Title 2   | Amputechture | Standard |
|         1 | Equal Vision    | Title 3   | Amputechture | Standard |
|         4 | Sargent House   | Title a   | Doppelganger | Standard |
|         4 | Sargent House   | Title d   | Doppelganger | Standard |
+-----------+---------------+----------+------------+----------+



SELECT * FROM Sponsor;
+-----------------+
| sponsorName     |
+-----------------+
| Adidas          |
| Dewalt          |
| Ford            |
| General Motors  |
| Nike            |
+-----------------+

SELECT * FROM Sponsors;
+--------------------------+-----------------+
| podcastName              | sponsorName     |
+--------------------------+-----------------+
| Lex Fridman Podcast      | Adidas          |
| Lex Fridman Podcast      | Ford            |
| The Joe Rogan Experience | Dewalt          |
| The Joe Rogan Experience | Ford            |
| The Joe Rogan Experience | General Motors  |
+--------------------------+-----------------+

SELECT * FROM PodcastGenre;
+-------------------+
| genreName         |
+-------------------+
| Business          |
| Comedy            |
| Mystery           |
| News              |
| Self Improvement  |
| Sports            |
| Tech              |
+-------------------+

SELECT * FROM DescribesPodcast;
+--------------------------+-------------------+
| podcastName              | genreName         |
+--------------------------+-------------------+
| Lex Fridman Podcast      | Self Improvement  |
| The Joe Rogan Experience | Comedy            |
| The Tim Ferriss Show      | Business          |
| Up First                  | News              |
+---------------------------+-------------------+

SELECT * FROM SpecialGuest;
+-----------+
| creatorId |
+-----------+
|         2 |
|         3 |
|         5 |
|         7 |
|         8 |
+-----------+
```

```
SELECT * FROM GuestStars;
+-----------+---------------------------------------------------------------+--------------------------+
| creatorId | title                                                         | podcastName              |
+-----------+---------------------------------------------------------------+--------------------------+
|         2 | #1952 - Michael Malice                                        | The Joe Rogan Experience |
|         5 | #1951 - Coffeezilla                                           | The Joe Rogan Experience |
|         7 | Florida Legislative Session, Powell Testimony, French Strikes | Up First                 |
+-----------+---------------------------------------------------------------+--------------------------+
```

4. Write interactive SQL queries for each operation in the narrative

4.1 Operations

Information Processing:

Enter Song Info:

```
INSERT INTO Song Values(5, 'Ghost Stories', 'Standard', 'Magic', 1,
3, 10, '2014-1-1', 'USA', 'English', true, 0.99);

Query OK, 1 row affected (0.0044 sec)
```

Update Song Info:

```
UPDATE Song
SET playCount = 20
WHERE creatorId = 5 AND albumName = 'Ghost Stories' AND edition =
'Standard' AND songTitle = 'Magic';
```

```
Query OK, 1 row affected (0.0044 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete Song Info:

```
DELETE FROM Song WHERE creatorId = 5 AND albumName = 'Ghost Stories'
AND edition = 'Standard' AND songTitle = 'Magic';

Query OK, 1 row affected (0.0044 sec)
```

Enter Artist Info:

```
INSERT INTO Artist Values(5, 'A', 'Band', 'USA', 'Rock' );

Query OK, 1 row affected (0.0045 sec)
```

Update Artist Info:

```
UPDATE Artist
SET primaryGenre = 'Pop'
WHERE creatorId = 5;

Query OK, 1 row affected (0.0044 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete Artist Info:

```
DELETE FROM Artist WHERE creatorId=5;

Query OK, 1 row affected (0.0044 sec)
```

Enter Podcast Host Info:

```
INSERT INTO PodcastHost Values(6, 'mkbhd@gmail.com',
'4444444444444444', 'NJ');

Query OK, 1 row affected (0.0045 sec)
```

Update Podcast Host Info:

```
UPDATE PodcastHost
SET city = 'NYC'
WHERE creatorId = 6;

Query OK, 1 row affected (0.0053 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete Podcast Host Info:

```
DELETE FROM PodcastHost
WHERE creatorId=6;

Query OK, 1 row affected (0.0048 sec)
```

Enter Podcast Info:

```
INSERT INTO Podcast Values('Waveform Podcast', 'English', 'USA', 5);

Query OK, 1 row affected (0.0046 sec)
```

Edit Podcast Info:

```
UPDATE Podcast
SET language = 'Spanish'
WHERE podcastName = 'Waveform Podcast';

Query OK, 1 row affected (0.0045 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete Podcast Info:

```
DELETE FROM Podcast
WHERE podcastName = 'Waveform Podcast';

Query OK, 1 row affected (0.0052 sec)
```

Enter Podcast Episode Info:

```
INSERT INTO PodcastEpisode Values('Blind smartphone camera test
results', 'Waveform Podcast', 60, '2023-1-1', 100);

Query OK, 1 row affected (0.0046 sec)
```

Update Podcast Episode Info:

```
UPDATE PodcastEpisode
SET duration = 75
WHERE podcastName = 'Waveform Podcast';

Query OK, 1 row affected (0.0047 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete Podcast Episode Info:

```
DELETE FROM PodcastEpisode
WHERE podcastName = 'Waveform Podcast';

Query OK, 1 row affected (0.0049 sec)
```

Enter Album Info:
```
INSERT INTO Album Values(5, 'Ghost Stories', 'Standard', 2014);

Query OK, 1 row affected (0.0042 sec)
```

Update Album Info:
```
UPDATE Album
SET releaseYear = 2013
WHERE creatorId = 5 AND albumName = 'Ghost Stories'
AND edition = 'Standard';

Query OK, 1 row affected (0.0057 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete Album Info:
```
DELETE FROM Album WHERE creatorId = 5 AND albumName = 'Ghost Stories'
AND edition = 'Standard';

Query OK, 1 row affected (0.0046 sec)
```

Enter User Info:
```
INSERT INTO User
Values('mario@gmail.com', 'Mario', 'Brother', 2.30, 'I',
4444444444444444, '2023-1-1');
```

Query OK, 1 row affected (0.0040 sec)

Update User Info:
```
UPDATE User
SET subscriptionFee = 2.40
WHERE email = 'mario@gmail.com';

Query OK, 1 row affected (0.0045 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Delete User Info:
```
DELETE FROM User WHERE email = 'mario@gmail.com';

Query OK, 1 row affected (0.0044 sec)
```

Enter Record Label Info:

```
INSERT INTO RecordLabel Values('Red Rock Records');

Query OK, 1 row affected (0.0052 sec)
```

Delete Record Label Info:

```
DELETE FROM RecordLabel
WHERE labelName = 'Red Rock Records';

Query OK, 1 row affected (0.0208 sec)
```

Assign Record Label to Song:

```
INSERT INTO Owns Values(5, 'Red Rock Records', 'Magic',
'Ghost Stories', 'Standard);

Query OK, 1 row affected (0.0047 sec)
```

Assign Collaboration between Artists and Song:

```
INSERT INTO CollaboratesOn Values(5, 7, 'Magic', 'Ghost Stories',
'Standard');

Query OK, 1 row affected (0.0047 sec)
```

Maintaining metadata and records:

Enter/update song play count:

```
UPDATE Song SET playCount=1234 WHERE songTitle='Title 1' AND
albumName='Amputechture' AND edition='Standard' AND creatorId=1;

Query OK, 1 row affected (0.0032 sec)

Rows matched: 1  Changed: 1  Warnings: 0
```

Enter/update monthly listeners for artists:

Because we have elected to treat monthly listeners as an implicit/derived attribute, the monthly listeners would be updated by inserting a new instance into the ListensToSong relation, which would be counted by our monthlyListeners query. See demonstration below:

First, check monthly listeners for The Mars Volta (creatorId: 1):

```
SELECT creatorId, COUNT(*) AS monthlyListeners FROM (SELECT DISTINCT creatorId,
email FROM ListensToSong WHERE creatorId=1 AND timestamp>='2011-02-01 00:00:00'
AND timestamp<='2011-02-28 11:59:59') AS MonthlyListeners;
```

```
+-----------+------------------+
| creatorId | monthlyListeners |
+-----------+------------------+
|         1 |                2 |
+-----------+------------------+
1 row in set (0.0024 sec)
```

Then, a new “listen” event takes place:

```
INSERT INTO ListensToSong VALUES ('lonely_lobster@futuremail.net', 'Title 2',
1, 'Amputechture', 'Standard', '2011-02-14 10:35:44');

Query OK, 1 row affected (0.0034 sec)
```

And running the original query again shows that we have a new monthly listener:

```
SELECT creatorId, COUNT(*) AS monthlyListeners FROM (SELECT DISTINCT creatorId,
email FROM ListensToSong WHERE creatorId=1 AND timestamp>='2011-02-01 00:00:00'
AND timestamp<='2011-02-28 11:59:59') AS MonthlyListeners;

+-----------+------------------+
| creatorId | monthlyListeners |
+-----------+------------------+
|         1 |                3 |
+-----------+------------------+

1 row in set (0.0024 sec)
```

We did not account for there being a need to adjust the monthly listener count downward, as we have assumed that a ‘ cannot “unlisten” to anything.

Enter/update total count of subscribers and ratings for podcasts:

Update ratings:

```
UPDATE Podcast SET rating=4.6 WHERE podcastName='The Joe Rogan Experience';

Query OK, 1 row affected (0.0024 sec)
```

Update subscribers:

The subscriber count operations and updates are similar to those for monthly listeners. We derive the subscriber count for a chosen podcast with the following query:

```
SELECT podcastName, COUNT(*) AS subscribers FROM SubscribesToPodcast WHERE
podcastName='The Joe Rogan Experience';
```

```
+---------------------------+-------------+
| podcastName               | subscribers |
+---------------------------+-------------+
| The Joe Rogan Experience |           2 |
+---------------------------+-------------+

1 row in set (0.0030 sec)
```

Then, we have a new user subscribe to the podcast:

```
INSERT INTO SubscribesToPodcast VALUES('curseWERNSTROM@planetexpress.com', 'The
Joe Rogan Experience', '2023-03-12 17:32:00');

Query OK, 1 row affected (0.0045 sec)
```

And running the original query again shows that we have a new subscriber:

```
SELECT podcastName, COUNT(*) AS subscribers FROM SubscribesToPodcast WHERE
podcastName='The Joe Rogan Experience';

+---------------------------+-------------+
| podcastName               | subscribers |
+---------------------------+-------------+
| The Joe Rogan Experience |           3 |
+---------------------------+-------------+

1 row in set (0.0031 sec)
```

In the instance of subscribers, it seems reasonable that a user may decide to “unsubscribe.” If this were to happen, the tuple would be deleted from the table and the subscriber count reduced:

```
DELETE FROM SubscribesToPodcast WHERE email='curseWERNSTROM@planetexpress.com'
AND podcastName='The Joe Rogan Experience' AND timestamp='2023-03-12 17:32:00';

Query OK, 1 row affected (0.0037 sec)
```

Which takes us back to the original number of subscribers if we run the subscribers query again:

```
SELECT podcastName, COUNT(*) AS subscribers FROM SubscribesToPodcast WHERE
podcastName='The Joe Rogan Experience';

+---------------------------+-------------+
| podcastName               | subscribers |
+---------------------------+-------------+
| The Joe Rogan Experience |           2 |
+---------------------------+-------------+

1 row in set (0.0031 sec)
```

Enter/update listening count for podcast episodes:

Once again, the listening count is derived implicitly from the ListensToPodcastEpisode relation. The query used to derive listening count is as follows:

```
SELECT podcastEpisodeTitle, COUNT(*) AS listeners FROM ListensToPodcastEpisode
WHERE podcastEpisodeTitle='#1951 - Coffeezilla' AND podcastName='The Joe Rogan
Experience';

+---------------------+-----------+
| podcastEpisodeTitle | listeners |
+---------------------+-----------+
| #1951 - Coffeezilla |         1 |
+---------------------+-----------+

1 row in set (0.0035 sec)
```

And we can increase the listen count by having a user listen to the same podcast episode:

```
INSERT INTO ListensToPodcastEpisode VALUES('girder_guy@planetexpress.com',
'#1951 - Coffeezilla', 'The Joe Rogan Experience', '2023-03-12 12:00:04');

Query OK, 1 row affected (0.0038 sec)
```

Which results in the listener count being higher by one if we run the original query again:

```
SELECT podcastEpisodeTitle, COUNT(*) AS listeners FROM ListensToPodcastEpisode
WHERE podcastEpisodeTitle='#1951 - Coffeezilla' AND podcastName='The Joe Rogan
Experience';

+---------------------+-----------+
| podcastEpisodeTitle | listeners |
+---------------------+-----------+
| #1951 - Coffeezilla |         2 |
+---------------------+-----------+

1 row in set (0.0031 sec)
```

And, again, our assumption that a user cannot “unlisten” to something eliminates the need for a delete statement.

Find songs/podcast episodes:

Find songs given artist:

```
SELECT songTitle
FROM Song NATURAL JOIN Artist
```

```
WHERE creatorId = 4;

+-----------+
| songTitle |
+-----------+
| Title a   |
| Title b   |
| Title c   |
| Title d   |
| Title e   |
+-----------+
```

Find songs given album:

```
SELECT songTitle
FROM Song
WHERE Song.albumName = 'Doppelganger' AND Song.edition= 'Standard'
AND Song.creatorId = 4;

+-----------+
| songTitle |
+-----------+
| Title a   |
| Title b   |
| Title c   |
| Title d   |
| Title e   |
+-----------+
```

Find podcast episodes given a podcast

```
SELECT title
FROM PodcastEpisode
WHERE podcastName = 'Lex Fridman Podcast';

+-----------------------------------------------------------------+
| title                                                           |
+-----------------------------------------------------------------+
| #284 - Saifedean Ammous: Bitcoin, Anarchy, and Austrian Economics |
+-----------------------------------------------------------------+
```

Reports

Monthly Play Count per Song

```
SELECT COUNT(*)
FROM ListensToSong
WHERE songTitle = 'Title 1' AND creatorId = 1 AND albumName =
     'Amputechture' AND edition = 'Standard' AND timestamp >=
     '2011-02-01 00:00:00' AND timestamp <
     '2011-03-01 00:00:00';

+----------+
| COUNT(*) |
+----------+
|     2    |
+----------+
```

Monthly Play Count per Album

```
SELECT COUNT(*)
FROM ListensToSong
WHERE albumName = 'Doppelganger' AND edition = 'Standard' AND
     creatorId = 4 AND timestamp >= '2011-02-01 00:00:00' AND
     timestamp < '2011-03-01 00:00:00';

+----------+
| COUNT(*) |
+----------+
|     5    |
+----------+
```

Monthly Play Count per Artist

```
SELECT COUNT(*)
FROM ListensToSong
WHERE creatorId = 1 AND timestamp >=
     '2011-02-01 00:00:00' AND timestamp <
     '2011-03-01 00:00:00';

+----------+
| COUNT(*) |
+----------+
|     4    |
+----------+
```

Calculate total payment made to host per a given time period

```
SELECT SUM(value)
FROM Payment NATURAL JOIN PayToContentCreator NATURAL JOIN
     PodcastHost
```

```
WHERE creatorId = 8 AND date >= '3001-06-01' AND date
     <= '3001-08-01';

+------------+
| SUM(value) |
+------------+
|    99.99   |
+------------+
```

<u>Calculate total payment made to an artist per a given time period</u>

```
SELECT SUM(value)
FROM Payment NATURAL JOIN PayToContentCreator NATURAL JOIN
     Artist
WHERE creatorId = 1 AND date >= '3001-02-01' AND date
     <= '3001-08-01';
+------------+
| SUM(value) |
+------------+
|  199.98    |
+------------+
```

<u>Calculate total payment made to a record label per a given time period</u>

```
SELECT SUM(value)
FROM Payment NATURAL JOIN PayToRecordLabel, RecordLabel
WHERE RecordLabel.labelName = PayToRecordLabel.recordLabelName
     AND recordLabelName = 'Equal Vision' AND
     date >= '3001-03-01' AND date <= '3001-06-01';

+------------+
| SUM(value) |
+------------+
|    99.99   |
+------------+
```

<u>Total revenue of the streaming service per month.</u>

```
SELECT SUM(value)
FROM PayFromUser NATURAL JOIN Payment
WHERE date >= '3001-05-01' AND date < '3001-06-01';
```

```
+------------+
| SUM(value) |
+------------+
|  99.99     |
+------------+
```

Total revenue of the streaming service per year.

```
SELECT SUM(value) AS annualRevenue
FROM PayFromUser NATURAL JOIN Payment
WHERE date >= '3001-01-01' AND date <= '3001-12-31';
```

```
+---------------+
| annualRevenue |
+---------------+
|        499.95 |
+---------------+
```

Report all songs given an artist .

```
SELECT *
FROM Song
WHERE creatorId = 4;
```

```
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
| creatorId | albumName    | edition  | songTitle | trackNumber | duration | playCount | releaseDate | releaseCountry | language | royaltyPaidStatus | royaltyRate |
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
|         4 | Doppelganger | Standard | Title a   |           1 |      310 |     60105 | 2023-05-07  | US             | English  |                 1 |        0.08 |
|         4 | Doppelganger | Standard | Title b   |           2 |      320 |    323105 | 2023-02-12  | US             | English  |                 1 |        0.27 |
|         4 | Doppelganger | Standard | Title c   |           3 |      270 |     69105 | 2023-05-07  | US             | English  |                 1 |        0.07 |
|         4 | Doppelganger | Standard | Title d   |           4 |      250 |    760105 | 2023-05-07  | US             | English  |                 1 |        0.10 |
|         4 | Doppelganger | Standard | Title e   |           5 |      255 |    160105 | 2023-05-07  | US             | English  |                 1 |        0.12 |
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
```

Report all songs given an album / Report all songs given an album and artist.

```
SELECT *
FROM Song
WHERE  Song.albumName  =  'Doppelganger'  AND  Song.edition=  'Standard'
AND Song.creatorId = 4;

SELECT * FROM Song;
```

```
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
| creatorId | albumName    | edition  | songTitle | trackNumber | duration | playCount | releaseDate | releaseCountry | language | royaltyPaidStatus | royaltyRate |
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
|         4 | Doppelganger | Standard | Title a   |           1 |      310 |     60105 | 2023-05-07  | US             | English  |                 1 |        0.08 |
|         4 | Doppelganger | Standard | Title b   |           2 |      320 |    323105 | 2023-05-07  | US             | English  |                 1 |        0.27 |
|         4 | Doppelganger | Standard | Title c   |           3 |      270 |     69105 | 2023-05-07  | US             | English  |                 1 |        0.07 |
|         4 | Doppelganger | Standard | Title d   |           4 |      250 |    760105 | 2023-05-07  | US             | English  |                 1 |        0.10 |
|         4 | Doppelganger | Standard | Title e   |           5 |      255 |    160105 | 2023-05-07  | US             | English  |                 1 |        0.12 |
+-----------+--------------+----------+-----------+-------------+----------+-----------+-------------+----------------+----------+-------------------+-------------+
```

Report all podcast episodes given a podcast.

```
SELECT *
FROM PodcastEpisode
WHERE podcastName = 'Lex Fridman Podcast';
```

```
+-------------------------------------------------------------------+---------------------+----------+-------------+--------------------+
| title                                                             | podcastName         | duration | releaseDate | advertisementCount |
+-------------------------------------------------------------------+---------------------+----------+-------------+--------------------+
| #284 - Saifedean Ammous: Bitcoin, Anarchy, and Austrian Economics | Lex Fridman Podcast |      238 | 2022-05-11  |                  4 |
+-------------------------------------------------------------------+---------------------+----------+-------------+--------------------+
```

Payments

Receive Payment From Subscribers.

```
SET @email = 'smallfry@aol.com';
Query OK, 0 rows affected (0.00 sec)

SET @newPaymentId = (SELECT COUNT(*) from Payment);
Query OK, 0 rows affected (0.00 sec)

SELECT subscriptionFee FROM User WHERE email = @email AND
statusOfSubscription = 'A' INTO @value;
Query OK, 1 row affected (0.00 sec)

INSERT into Payment(paymentId, date, value) values(@newPaymentId,
CURRENT_DATE(), @value);
Query OK, 1 row affected (0.01 sec)

INSERT into PayFromUser (paymentId, userEmail) values(@newPaymentId,
@email);
Query OK, 1 row affected (0.00 sec)
```

Make payment to podcast hosts.

```
SET @baseAmount = 20.00;
Query OK, 0 rows affected (0.00 sec)

SET @bonus = 5.00;
Query OK, 0 rows affected (0.00 sec)

SET @creatorId = '3';
Query OK, 0 rows affected (0.00 sec)
```

```
SELECT SUM(@baseAmount + @bonus * advertisementCount) FROM Hosts
NATURAL JOIN Podcast NATURAL JOIN PodcastEpisode WHERE creatorId =
@creatorId INTO @value;
Query OK, 1 row affected (0.00 sec)

SET @newPaymentId = (SELECT COUNT(*) from Payment);
Query OK, 0 rows affected (0.00 sec)

INSERT Payment(paymentId, date, value) values(@newPaymentId,
CURRENT_DATE(), @value);
Query OK, 1 row affected (0.01 sec)

INSERT into PayToContentCreator(paymentId, creatorId)
values(@newPaymentId, @creatorId);
Query OK, 1 row affected (0.01 sec)
```

Make royalty payments for a given song.

```
SET @artistsShare = 0.70;
Query OK, 0 rows affected (0.00 sec)

SET @recordLabelShare = 0.30;
Query OK, 0 rows affected (0.00 sec)

SET @creatorId = 1;
Query OK, 0 rows affected (0.00 sec)

SET @albumName = 'Amputechture';
Query OK, 0 rows affected (0.00 sec)

SET @edition = 'Standard';
Query OK, 0 rows affected (0.00 sec)

SET @songTitle = 'Title 1';
Query OK, 0 rows affected (0.00 sec)

SELECT playCount, royaltyRate, recordLabelName  FROM Song natural
join Owns WHERE creatorId = @creatorId AND albumName = @albumName AND
edition = @edition AND songTitle = @songTitle INTO @playCount,
@royaltyRate, @recordLabelName;
Query OK, 1 row affected (0.00 sec)

CREATE TEMPORARY TABLE ArtistsToPay(SELECT guestArtistId FROM
CollaboratesOn WHERE creatorId = @creatorId AND albumName =
@albumName AND edition = @edition AND songTitle = @songTitle);
Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0

SET @totalArtists = (SELECT COUNT(*) FROM ArtistsToPay);
Query OK, 0 rows affected (0.00 sec)

SET @artistValue = @royaltyRate * @playCount * @artistsShare /
(@totalArtists + 1);
Query OK, 0 rows affected (0.00 sec)

SET @recordLabelValue = @royaltyRate * @playCount * @recordLabelShare
/ (@totalArtists + 1);
Query OK, 0 rows affected (0.00 sec)

SET @paymentId = (SELECT COUNT(*) from Payment);
Query OK, 0 rows affected (0.00 sec)

INSERT INTO Payment(paymentId, date, value) values(@paymentId,
CURRENT_DATE(), @recordLabelValue);
Query OK, 1 row affected, 1 warning (0.00 sec)

INSERT INTO PayToRecordLabel(recordLabelName, paymentId)
values(@recordLabelName, @paymentId);
Query OK, 1 row affected (0.00 sec)

SET @paymentId = (SELECT COUNT(*) from Payment);
Query OK, 0 rows affected (0.00 sec)

INSERT Payment(paymentId, date, value) values(@paymentId,
CURRENT_DATE(), @artistValue);
Query OK, 1 row affected, 1 warning (0.01 sec)

INSERT into PayToContentCreator(creatorId, paymentId)
values(@creatorId, @paymentId);
Query OK, 1 row affected (0.00 sec)

SET @paymentId = (SELECT COUNT(*) from Payment);
Query OK, 0 rows affected (0.00 sec)

INSERT Payment(paymentId, date, value) values(@paymentId,
CURRENT_DATE(), @artistValue);
Query OK, 1 row affected, 1 warning (0.00 sec)

INSERT into PayToContentCreator(creatorId, paymentId) values(5,
@paymentId);
Query OK, 1 row affected (0.00 sec)

SET @paymentId = (SELECT COUNT(*) from Payment);
Query OK, 0 rows affected (0.00 sec)

INSERT Payment(paymentId, date, value) values(@paymentId,
CURRENT_DATE(), @artistValue);
Query OK, 1 row affected, 1 warning (0.00 sec)

INSERT into PayToContentCreator(creatorId, paymentId) values(6,
@paymentId);
Query OK, 1 row affected (0.00 sec)
```

4.2 Explain

We found **only** two instances could benefit from an index.

```
4.2.1
```

This query shows that an index would be beneficial. However, the query does not use the index after it is created.

```
(1)
SELECT SUM(@baseAmount + @bonus * advertisementCount) FROM Hosts
NATURAL JOIN Podcast NATURAL JOIN PodcastEpisode WHERE creatorId =
@creatorId;
+-------------------------------------------------+
| SUM(@baseAmount + @bonus * advertisementCount) |
+-------------------------------------------------+
|               70.000000000000000000000000000000 |
+-------------------------------------------------+
1 row in set (0.00 sec)

(2)
EXPLAIN SELECT SUM(@baseAmount + @bonus * advertisementCount) FROM
Hosts NATURAL JOIN Podcast NATURAL JOIN PodcastEpisode WHERE
creatorId = @creatorId;
+------+-------------+----------------+--------+-------------------+-----------+---------+---------------------------+------+-------------------------------------------------+
| id   | select_type | table          | type   | possible_keys     | key       | key_len | ref                       | rows | Extra                                           |
+------+-------------+----------------+--------+-------------------+-----------+---------+---------------------------+------+-------------------------------------------------+
|    1 | SIMPLE      | Hosts          | ref    | PRIMARY,creatorId | creatorId | 4       | const                     |    1 | Using index                                     |
|    1 | SIMPLE      | Podcast        | eq_ref | PRIMARY           | PRIMARY   | 257     | zhadgra.Hosts.podcastName |    1 | Using index                                     |
|    1 | SIMPLE      | PodcastEpisode | ALL    | podcastName       | NULL      | NULL    | NULL                      |    4 | Using where; Using join buffer (flat, BNL join) |
+------+-------------+----------------+--------+-------------------+-----------+---------+---------------------------+------+-------------------------------------------------+
3 rows in set (0.00 sec)

(3)
```

```
CREATE INDEX PodcastEpisode_podcastName ON
PodcastEpisode(podcastName);
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

(4)

```
EXPLAIN SELECT SUM(@baseAmount + @bonus * advertisementCount) FROM
Hosts NATURAL JOIN Podcast NATURAL JOIN PodcastEpisode WHERE
creatorId = @creatorId;
+------+-------------+----------------+--------+----------------------------+-----------+---------+---------------------------+------+------------------------------------------------+
| id   | select_type | table          | type   | possible_keys              | key       | key_len | ref                       | rows | Extra                                          |
+------+-------------+----------------+--------+----------------------------+-----------+---------+---------------------------+------+------------------------------------------------+
|    1 | SIMPLE      | Hosts          | ref    | PRIMARY,creatorId          | creatorId | 4       | const                     |    1 | Using index                                    |
|    1 | SIMPLE      | Podcast        | eq_ref | PRIMARY                    | PRIMARY   | 257     | zhadgra.Hosts.podcastName |    1 | Using index                                    |
|    1 | SIMPLE      | PodcastEpisode | ALL    | PodcastEpisode_podcastName | NULL      | NULL    | NULL                      |    4 | Using where; Using join buffer (flat, BNL join) |
+------+-------------+----------------+--------+----------------------------+-----------+---------+---------------------------+------+------------------------------------------------+
3 rows in set (0.00 sec)
```

4.2.2

(1)

```
SELECT SUM(value)
FROM PayFromUser NATURAL JOIN Payment
WHERE date >= '3001-05-01' AND date < '3001-06-01';

+------------+
| SUM(value) |
+------------+
|    99.99   |
+------------+
1 row in set (0.0025 sec)
```

(2)

```
EXPLAIN(SELECT SUM(value) FROM PayFromUser NATURAL JOIN Payment WHERE date >=
'3001-05-01' AND date < '3001-06-01');
+----+-------------+-------------+-------+---------------+---------+---------+------+------+-------------------------------------------------+
| id | select_type | table       | type  | possible_keys | key     | key_len | ref  | rows | Extra                                           |
+----+-------------+-------------+-------+---------------+---------+---------+------+------+-------------------------------------------------+
|  1 | SIMPLE      | PayFromUser | index | PRIMARY       | PRIMARY | 261     | NULL |    5 | Using index                                     |
|  1 | SIMPLE      | Payment     | ALL   | PRIMARY       | NULL    | NULL    | NULL |    5 | Using where; Using join buffer (flat, BNL join) |
+----+-------------+-------------+-------+---------------+---------+---------+------+------+-------------------------------------------------+
2 rows in set (0.0029 sec)
```

(3)

```
CREATE INDEX payment_date_index ON Payment(date);
Query OK, 0 rows affected (0.0802 sec)

Records: 0  Duplicates: 0  Warnings: 0
```

(4)

```
EXPLAIN(SELECT SUM(value) FROM PayFromUser NATURAL JOIN Payment WHERE date >= '3001-05-01' AND
date < '3001-06-01');
+----+-------------+-------------+-------+---------------------------+--------------------+---------+---------------------------+------+----------
-------------+
| id | select_type | table       | type  | possible_keys             | key                | key_len | ref                       | rows | Extra
|
+----+-------------+-------------+-------+----------------------------+--------------------+---------+----------------------------+------+----------
-------------+
|  1 | SIMPLE      | Payment     | range | PRIMARY,payment_date_index | payment_date_index | 3       | NULL                       |    1 | Using
index condition |
|  1 | SIMPLE      | PayFromUser | ref   | PRIMARY                    | PRIMARY            | 4       | cvcapooc.Payment.paymentId |    2 | Using
index           |
+----+-------------+-------------+-------+----------------------------+--------------------+---------+----------------------------+------+----------
-------------+
2 rows in set (0.0030 sec)
```

4.3 Query Correctness

1) *Report all songs given an artist.*

```
SELECT songTitle
FROM Song NATURAL JOIN Artist
WHERE creatorId = 4;
```

$\pi_{\text{Song.songTitle}}(\text{Song} \bowtie (\sigma_{\text{creatorId = 4}}(\text{Artist})))$

Suppose p is any tuple in the Song relation and q is any tuple in the Artist relation, such that the value of p.creatorId and q.creatorId are identical. The combination from tuples p and q results in a tuple that contains both Song and Artist attributes with a shared creatorId. From the resulting tuples, we will be given a subset where the creatorId is of value, 4. Afterwards, we retrieve from the filtered tuples from the resulting relation, the corresponding songTitle that stems from each Song in the natural join.

2) *Total revenue of the streaming service in a given year.*

```
SELECT SUM(value) AS annualRevenue
FROM PayFromUser NATURAL JOIN Payment
WHERE date >= '3001-01-01' AND date <= '3001-12-31';
```

$\gamma_{\text{(SUM(value)->annualRevenue)}}(\sigma_{\text{date>='3001-01-01' AND date<='3001-12-31'}}(\text{PayFromUser} \bowtie \text{Payment}))$

Suppose m is a set of tuples in the PayFromUser relation and n is a set of tuples in the Payment relation, such that m.paymentId and n.paymentId have identical values. The combination of tuples m and n will return a relation that associates a paymentID with: 1) the user that made the payment, 2) the date the payment was made, and 3) the value of the payment. From this new relation, we filter out values for a given year as denoted in the select operator, where we take tuples with dates occurring between the first and last days of a given year, inclusive. The resulting relation contains all payments made by users in the specified year. From this relation, we aggregate the sum of the “value” column renamed as annualRevenue, which is the total of all payments received by WolfMedia in the selected year.