WolfMedia Streaming Service

CSC 540 - Project Report 3

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).
- Regardless of the value of royaltyPaidStatus, when the API that pays royalties is invoked, royalties are paid.

Transactions

Note: that autocommit is set to False by default.

Transaction 1: Update Podcast Ratings

```
# Gather user inputs
podcast = input('Which podcast would you like to update the ratings
for? ')
rating = input(f'Please enter the latest rating for {podcast}: ')

# Build queries
sql = (f'UPDATE Podcast SET rating = {rating} WHERE podcastName =
"{podcast}";')

view = (f'SELECT * FROM Podcast WHERE podcastName = "{podcast}";')

# conn, cur = dbconnect.connect_to_MariaDB()
conn, cur = db.Database().connect_to_MariaDB()

# Attempt update
try:
    # Execute update transaction
    cur.execute('START TRANSACTION;')
    cur.execute(sql)
    cur.execute('COMMIT;')

    # Confirmation
    print(f'{podcast} rating updated.')
    print(tabulate(pd.read_sql(view, conn), headers='keys',
    tablefmt='psql'))

    # Close connection
    conn.close()

except mariadb.Error as e:
    # Catch error, display, rollback attempted transaction, and close
    connection
    print(f'Error with update: {e}')
    cur.execute('ROLLBACK;')
    conn.close()
```

Transaction 2: payPodcastHosts

```
def payPodcastHosts(podcastEpisode=None, podcastName=None, month=None,
year=None):
    """Pay podcast host for an episode or process all payments.

    If None is passed for podcastEpisode and podcastName, payment
    generated for all podcastEpisodes.

    If the podcast data query fails the transaction is rolled back.

    Args:
        podcastEpisode: name of the podcast episode
        podcastName: name of the podcast show under which episodes
are
            recorded
        month: month to register the payment under
        year: year to register the payment under

    Returns:
        True if operation was successful and amount paid to each
party
        in a list.
        False if the operation fails and an empty list.
    """
    conn, cur = db.Database().connect_to_MariaDB()

    payment_date = date(year=year, month=month, day=1)

    try:
        if not all([podcastEpisode, podcastName]):
            cur.execute('START TRANSACTION;')

            cur.execute(f"SELECT creatorId, {_PODCAST_BASE_AMOUNT} +
{_PODCAST_BONUS} * advertisementCount FROM Hosts NATURAL JOIN Podcast
NATURAL JOIN PodcastEpisode;")
            result = cur.fetchall()

            for result_ in result:
                creator_id = result_[0]
                value = result_[1]

                payment_id = _get_new_payment_id(cur)

                cur.execute(f"INSERT Payment(paymentId, date, value)
values({payment_id}, '{payment_date}', {value});")
```

```
                cur.execute(f"INSERT into
PayToContentCreator(paymentId, creatorId) values({payment_id},
{creator_id});")

            main.print_debug(f"Payment processed for all creator
IDs...")

        else:

            cur.execute('START TRANSACTION;')
            cur.execute(f"SELECT creatorId, {_PODCAST_BASE_AMOUNT} +
{_PODCAST_BONUS} * advertisementCount FROM Hosts NATURAL JOIN Podcast
NATURAL JOIN PodcastEpisode WHERE podcastName = '{podcastName}' AND
title = '{podcastEpisode}';")

            result_ = cur.fetchall()[0]
            result = [result_]
            creator_id = result_[0]
            value = result_[1]

            payment_id = _get_new_payment_id(cur)

            cur.execute(f"INSERT Payment(paymentId, date, value)
values({payment_id}, '{payment_date}', {value});")
            cur.execute(f"INSERT into PayToContentCreator(paymentId,
creatorId) values({payment_id}, {creator_id});")

            main.print_debug(f"Payment processed for creator ID:
{creator_id}...")

    except IndexError:
        main.print_debug(f"No podcast data found...")
        cur.execute('ROLLBACK;')
        return False, []

    else:
        cur.execute('COMMIT;')
        return True, result

    finally:
        cur.close()
        conn.close()
```

Design Decisions

The WolfMedia application program, when booted up, presents the user to list the available commands via typing “help” or “?”. Once the user chooses among the actions of information processing, maintaining metadata and records, maintaining payments, and report generating. Typing in one of these commands will prompt users to more specific inputs that are necessary to perform the specific action. For example, when a user types in “enterRecordLabelInfo”, they are then prompted for the record label name about to be created. Once all inputs are received, the corresponding SQL query is executed and results may be displayed to further prove data manipulation in the form of tabulated results.

In terms of system design, main.py is the starting point of the program that is in a constant loop unless the user invokes quit. The file is broken down into methods that specifically invoke a given module’s command by calling the associated API function. For example, do_enterRecordLabelInfo will call the associated function in the recordLabel module where it’s defined. The modularizing of functionalities allowed concurrent development and fast iteration with minimal conflict and test result regression. Within the operations folder, the modules are broken up into informationProcessing (which contains multiple files), metadata.py, payments.py, and reports.py.

In terms of the database, there is a singleton class within operations/db.py that is defined that creates a connection pool for the instance of the mariadb database specified by user’s profile data. This allows all SQL query invocations to be passed through one channel, allowing for simpler debugging of potential issues and uniform connection behavior throughout the application and unit tests.

Functional Roles

Part 1:
Software Engineer: Zach (Prime), Andrew (Backup)
Database Designer/Administrator: Andrew (Prime), Corey (Backup)
Application Programmer: Ethan (Prime), Zach (Backup)
Test Plan Engineer: Corey (Prime), Ethan (Backup)

Part 2:
Software Engineer: Ethan (Prime), Zach (Backup)
Database Designer/Administrator: Corey (Prime), Ethan (Backup)
Application Programmer: Zach (Prime), Andrew (Backup)
Test Plan Engineer: Andrew (Prime), Corey (Backup)

Part 3:
Software Engineer: Andrew (Prime), Corey (Backup)
Database Designer/Administrator: Ethan (Prime), Zach (Backup)
Application Programmer: Corey (Prime), Ethan (Backup)
Test Plan Engineer: Zach (Prime), Andrew (Backup)

Revisions

Project 1

5. API Methods
Original report pages 4-9.
Summary of changes:

- The “uncalled for” operations were removed (User Info, Artist Status, Podcast Subscribers and Ratings) except for Podcast Subscribers and Ratings. **These two did become part of the narrative, though I cannot remember if they were included in the narrative when the first report was submitted. We believe this may have been a grading mistake relative to the current state of the project narrative.**
- We received a comment about the podcast host not being assigned to a podcast in report 1. **We have no updates for this comment, and believe it may have been a grading mistake.** The podcast host was assigned to podcast by way of the creatorId attribute in the following API method, which was submitted with Project 1:
    - assignToPodcast(creatorId(s), podcastEpisode(s), podcast)
      return confirmation
- *No mention of Podcast entity though it was in the ERD*
    - We realized that we missed these in the operations. The API methods we have included are below:

enterPodcastInfo(podcastName, language, country, rating)
    return confirmation

updatePodcastInfo(podcastName, language, country, rating)
    return confirmation
    * if NULL value provided for any input, those inputs will not be changed

deletePodcastInfo(podcastName)
    return confirmation, deletes whole tuple

8. *-5 Each Artist is related to exactly one Record Label according to the description, but the ERD showed differently.*

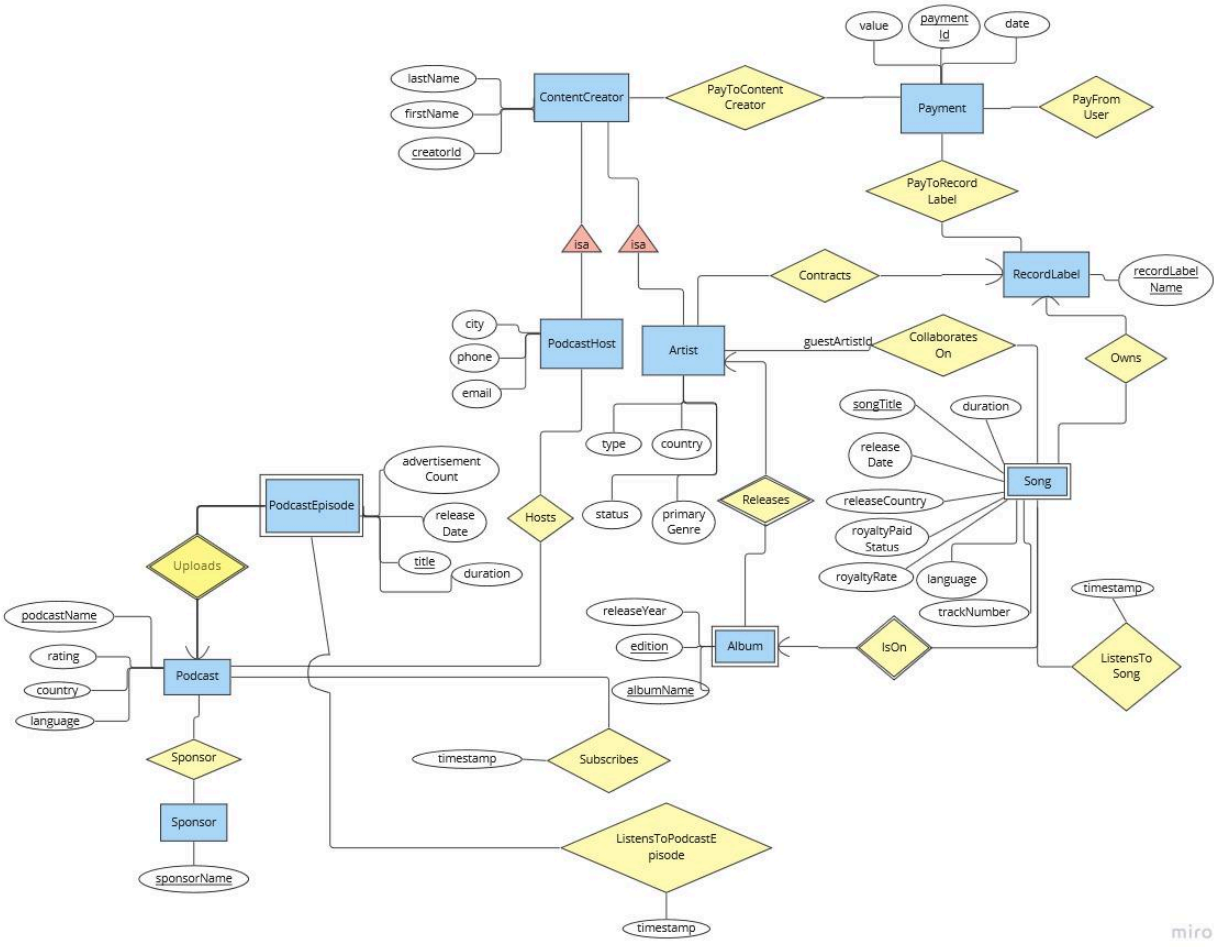
No updates for this comment, we believe it was a grading mistake.

Original Report page 14.

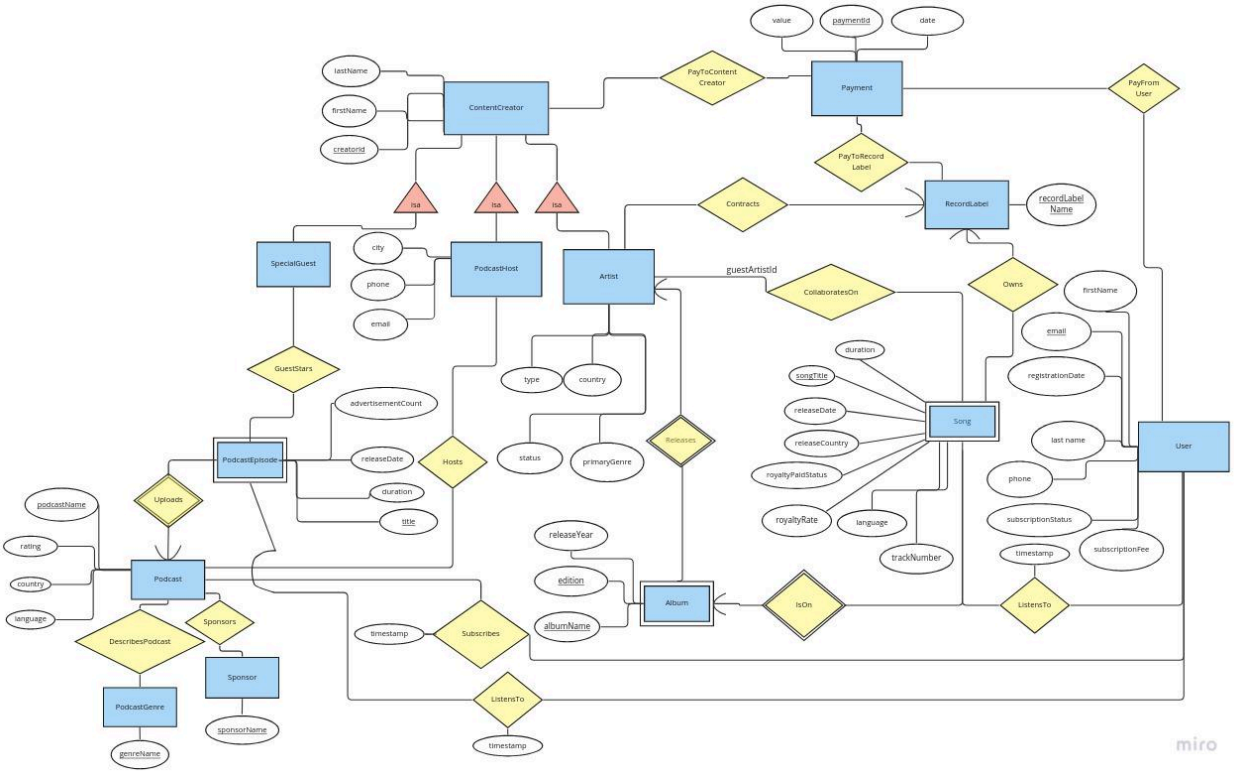
- The Releases supporting relationship associates Artists with the weak entity set Albums with the creatorId (key) from artist, which are associated with the weak entity set Song through the IsOn supporting relationship using the albumName and edition attributes (the key). Artists can work on songs in the role of guest artist in the collaboratesOn relationship.
- Every artist is contracted by exactly one record label, which owns the songs the artist produces while under that contract.
- When a content creator joins the platform, they are given a unique id to differentiate them to help avoid confusion if two creators happen to have very similar names.

See original ERDs below and in the original project 1 report.
Original Report pages 10-13

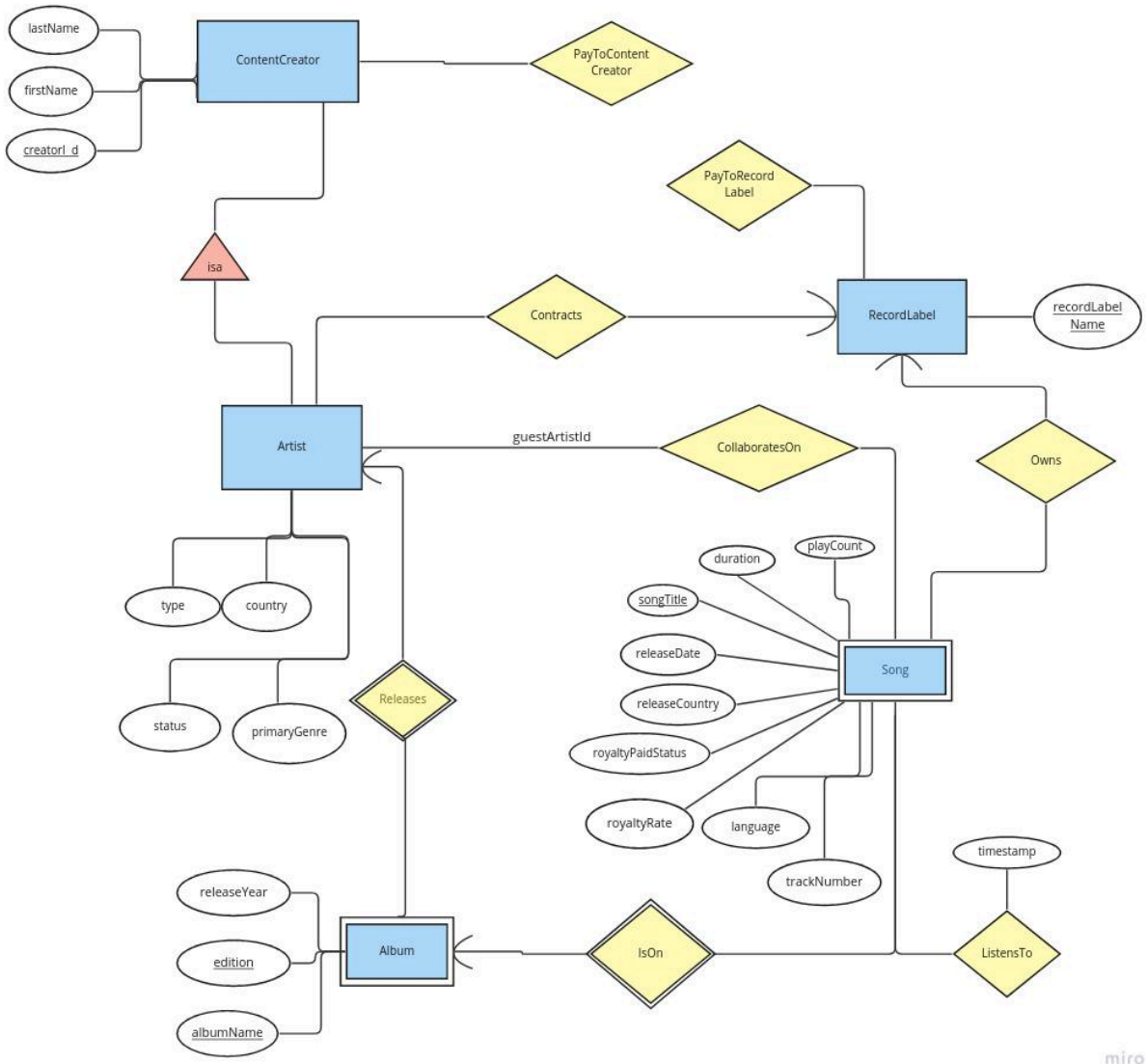
Management



Administrators



Artist



9. *-10 For Song schema, more than one primary keys were listed which is different from the ERD.*

Original Project Page 15

No updates for this comment, we believe it was a grading mistake.

The primary keys are from both the Song entity and the weak entities that it relies on such as Album, and ContentCreator. See ER diagrams above and in the original report 1.

PayFromUser(paymentId, email)

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

Project 2

1. *Unclear relation shown between tables Genre and Artist.*

Original Report Page 2

- Genre for artist is described by the primaryGenre attribute, not a separate relation for artist genres.

The above was added to the assumptions.

3. *'releaseYear' in table Album cannot be NULL according to your answer in Qn. 2.*

Original Report Page 14

```
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
);
```