WolfMedia Database Management System

Team S

Ameya Girish Vaichalkar (agvaicha)

Kartik Hemendra Rawool (khrawool)

Dhrumil Jignesh Shah (dshah6)

Subodh Suryakant Gujar (sgujar)

Assumptions

1. Artist can have a contract with exactly one record label at a time.
2. Podcast is deleted then all the episodes within that podcast will also be deleted.
3. Album is deleted then all the album tracks within that album will also be deleted.
4. All artists, record labels, and podcast hosts are paid on first day of the month.
5. Users pays subscription fees on the first day of the month.
6. Monthly active listeners include only the users who have subscribed to the artist or the podcast host.
7. Phone numbers entered by users are unique.
8. Email-ID entered by users are unique.
9. Song will always be part of at most one album.
10. Song will be sung in only one language.
11. Each track number is unique within an album. For eg., multiple albums can have same track number.
12. Each episode number is unique within a podcast. For eg., multiple podcast episode can have the same episode number.
13. Different tables are maintained for keeping the records of song history, artist history, and podcast history.
14. Data will be added in History tables only at the last day of the month.
15. Each song has exactly one primary artist.
16. Royalties will be paid for the songs to the record label of the primary artist.
17. Service account represents all the transactions done by and to WolfMedia Streaming service.
18. Podcast ratings will be in the range of 0-5.
19. There are multiple plans for subscription to WolfMedia. Each plan has different subscription fees.
20. All key attributes are NON NULL.
21. All referential integrity will have ON UPDATE CASCADE and ON DELETE CASCADE constraint.
22. Songs table will have the playcount of the current month.

1 Revised changes

Report 1
Feedback: -20 for all views: "trackNumber" missing supporting key, supporting relationship "consist of" is redundant, should not be converted.

Location: Report 1 page number 19, 20, 21, 22 in admin, artist, podcast host, user and record label views
trackNumber(number) should be trackNumber(albumId, number)
consistOf(podcastId, number) is reductant as consistOf relations is a weak entity and thus its schema should not be created

Report 2
Feedback: -10 for distributesroyalties, songId should belong to referential integrity
Location: Report 2 page number 17

Previous Schema
distributesroyalties(artistId, recordId, songId, date, amount)

- Keys
    1. artistId, recordId, date
- NON-NULL
    1. artistId
    2. recordId
    3. date
    4. amount (default 0)
    5. songId
- Referential Integrity
    1. artistId
    2. recordId

Corrected Schema
distributesroyalties(artistId, recordId, songId, date, amount)

- Keys
    1. artistId, recordId, date, songId
- NON-NULL
    1. artistId
    2. recordId
    3. date

4. amount (default 0)
5. songId

- Referential Integrity

1. artistId
2. recordId
3. songId

songId should be a referential integrity of distributesroyalties schema.

Updated pages showcasing changes attached at the end of the report

2 Transactions:

Transaction 1: Make Royalty Payment of a Song for a given month

The code for the method can be found at src/main/java/edu/ncsu/dbms/wolfmedia/services /PaymentsService.java

```
public String makeRoyaltyPayment(int songId, int month, int year) {
        Connection connection = genericDAO.createConnection();
        try {
            connection.setAutoCommit(false);
            Statement statement = connection.createStatement();
            ResultSet songHistoryMonth = statement.executeQuery("
            SELECT * FROM songHistory WHERE month = "+month+"
            AND songId = "+songId+";");
            if(songHistoryMonth.next()) {
                return "Royalty payment for this song was already paid
                in the given month";
            }
            ResultSet royaltiesResultSet = statement.executeQuery(
            "SELECT playCount * royaltyRate AS amount FROM songs");
            statement.executeQuery("INSERT INTO
            songHistory (songId, month, year, playCount)
            VALUES ("+songId+", "+month+", "+year+",
            (SELECT playCount FROM songs WHERE songId = "+songId+"))");
            statement.execute("UPDATE songs SET playCount = 0
            WHERE songId = "+songId);
            // get the royalty generated for the mentioned song in the
            month specified
            royaltiesResultSet.next();
            double royaltiesGenerated =
            Double.parseDouble(new DecimalFormat("#.##").format(
            royaltiesResultSet.getDouble(1)));
            //find the record label of the primary artist of the given song
            ResultSet recordLabelResultSet = statement.executeQuery(
            "SELECT recordId FROM artists WHERE artistId = (
            SELECT primaryArtist FROM songs WHERE songId = " + songId + ")");
            recordLabelResultSet.next();
            int recordLabelId = recordLabelResultSet.getInt(1);
            //count the collaborators of the song
            ResultSet countArtistsResultSet = statement.executeQuery(
            "select count(*) from creates where
            songId = " + songId + ";");
            countArtistsResultSet.next();
            int artistCount = countArtistsResultSet.getInt(1) + 1;
            //make payment to record label
            statement.executeQuery("INSERT INTO
            serviceAccount (date, amount, type) VALUES (curdate(), "+
```

```
            royaltiesGenerated + ", 'Debit');");
            ResultSet lastInsertIdResultSet = statement.executeQuery(
            "SELECT LAST_INSERT_ID();");
            lastInsertIdResultSet.next();
            int transactionId = lastInsertIdResultSet.getInt(1);
            //keep a track of the Record Label payment
            statement.executeQuery("INSERT INTO receives (transactionId, recordId)
            VALUES (" + transactionId + ", " + recordLabelId + ");");
            //get artists associated to the song
            ResultSet artistsResultSet =
            statement.executeQuery("SELECT artistId FROM creates WHERE
            songId = " + songId + " UNION " +
                    " SELECT primaryArtist FROM songs WHERE songId = " + songId);
            List<Integer> artists = new ArrayList<>();
            while (artistsResultSet.next()) {
                artists.add(artistsResultSet.getInt(1));
            }
            double artistPayment = (royaltiesGenerated * 0.7) / artistCount;
            System.out.println(artists);
            //making payments to each artist
            for (int i = 0; i < artistCount; i++) {
                statement.executeQuery("INSERT INTO distributesRoyalties
                (artistId, recordId, songId, date, amount) " +
                    " VALUES (" + artists.get(i) + ", " + recordLabelId + ", " +
                    songId + ", curdate(), " + artistPayment + ")");
            }
            //update song royalty paid status to yes
            statement.executeQuery(
            "UPDATE songs SET royaltyStatus = 'yes' WHERE songId = "+songId);
            connection.commit();
            return "Payment Successful";
        } catch (SQLException e) {
            e.printStackTrace();
            if(connection != null){
                try{
                    System.err.print("Transaction is being rolled back");
                    //rollback if the above insert operations fail
                    connection.rollback();
                } catch (SQLException excep){
                    excep.printStackTrace();
                }
            }
            return "Payment Failed. Transaction is rolled back";
        }finally {
            closeConnection(connection);
        }
    }
```

Documentation:

The method makeRoyaltyPayment() is responsible for making royalty payments to the record label and the artists that are associated with the given song. If an exception is thrown while inserting a tuple in the serviceAccount table, the whole transaction will get affected as there will be a possibility of incorrect data getting stored. Hence rollback method is called in the catch block that terminates the transaction and will keep the previous values.

Transaction 2:

The code for the method can be found at src/main/java/edu/ncsu/dbms/wolfmedia/services /PaymentsService.java

```
public String  makePaymentToPodcastHost(int month) {
    Connection connection = genericDAO.createConnection();
    try {
        connection.setAutoCommit(false);
        Statement statement = connection.createStatement();
        ResultSet bonusResultSet = statement.executeQuery(
        "SELECT createdBy.hostId, SUM(episodes.AdvertisementCount * 10)
        as bonus\n" +
                " FROM podcasts JOIN createdBy ON podcasts.podcastId =
                createdBy.podcastId " +
                " JOIN episodes ON episodes.podcastId = podcasts.podcastId " +
                " WHERE MONTH(episodes.releaseDate) = "+month +
                " GROUP BY createdBy.hostId;");
        Map<Integer, Double> hostPayments = new HashMap<>();
        while(bonusResultSet.next()) {
            hostPayments.put(bonusResultSet.getInt(1),
            Double.parseDouble(new DecimalFormat("#.##").format(
            bonusResultSet.getDouble(2))));
        }
        ResultSet flatFeeResultSet = statement.executeQuery("SELECT\n" +
                "createdBy.hostId, podcasts.flatFee * COUNT(episodes.number)
                AS FlatFee\n" +
                "FROM\n" +
                "createdBy\n" +
                "JOIN podcasts ON createdBy.podcastId = podcasts.podcastId\n" +
                "JOIN episodes ON podcasts.podcastId = episodes.podcastId\n" +
                "WHERE MONTH(episodes.releaseDate) = "+month+
                "GROUP BY createdBy.hostId;");
        while(flatFeeResultSet.next()) {
            int hostId = flatFeeResultSet.getInt(1);
            double payment = hostPayments.get(hostId) +
            Double.parseDouble(new DecimalFormat("#.##").format(
            flatFeeResultSet.getDouble(2)));
            hostPayments.put(hostId, payment);
        }
        for (Map.Entry<Integer, Double> entry : hostPayments.entrySet()) {
            ResultSet insertIntoServiceAccount = statement.executeQuery(
            "INSERT INTO serviceAccount (date, amount, type)\n" +
                    "VALUES (curdate(), "+entry.getValue()+", 'Debit');");
            ResultSet lastInsertIdResultSet = statement.executeQuery(
            "SELECT LAST_INSERT_ID();");
            lastInsertIdResultSet.next();
```

```
                int transactionId = lastInsertIdResultSet.getInt(1);
                ResultSet insertIntoGivesPaymentTo = statement.executeQuery(
                "INSERT INTO givesPaymentTo (transactionId, hostId)\n" +
                        "VALUES ("+transactionId+", "+entry.getKey()+");");
                connection.commit();
            }
            return "Payment to Podcast Host is Successful";
        } catch (SQLException e) {
            e.printStackTrace();
            if (connection != null){
                try{
                    System.err.print("Transaction is being rolled back");
                    //rollback transaction if the above insert operations fail
                    connection.rollback();
                }catch(SQLException excep){
                    excep.printStackTrace();
                }
            }
            return "Payment to Podcast Host has Failed. Transaction is rolled back";
        }finally {
            closeConnection(connection);
        }
    }
```

Documentation:

The method makePaymentToPodcastHost() is responsible for making payments to the podcast host. The bonus is calculated as 10*advertisement count for each episode. The flat fee is calculated from the podcast and for each episode under the podcast. If an exception is thrown while inserting a tuple in the serviceAccount table, the whole transaction will get affected as there will be a possibility of incorrect data getting stored. The rollback method is called in the catch block that terminates the transaction and will keep the previous values.

3 Documentation

We have used Javadoc to make documentation for all the operations. The documentation can be found at docs/index.html. Please open index.html in the browser to view the documentation for every task and operation.

4 Design Decisions:

We have used Java based spring-boot framework for creating REST(Representational State Transfer) APIs for easy access. There are sepearate APIs written to perform each operation. APIs can take inputs as a query parameters, path parameter or JSON object. Response for these APIs will be in JSON format, even the errors thrown by the code. We have created models for every table schema in the database which is used to retrieve data and display data through REST APIs.

There are 2 files for each operations namely controller and service and one common file GenericDAO to handle query execution. REST APIs are written in the controller files with GET for getting data from the database, POST for inserting data into the database, PUT for updating data in the database, and DELETE for deleting data from the database. Service files contains application logic through JDBC connectivity for each API. GenricDAO java file handles the actual implementation of JDBC connectivity and execution of SQL queries.

5 Functional Roles:

Part 1:

- Software Engineer: Ameya (Prime), Dhrumil (Backup)
- Database Designer/Administrator: Dhrumil (Prime), Subodh (Backup)
- Application Programmer: Subodh (Prime), Kartik(Backup)
- Test Plan Engineer: Kartik(Prime), Ameya (Backup)

Part 2:

- Software Engineer: Subodh (Prime), Kartik(Backup)
- Database Designer/Administrator: Kartik(Prime), Dhrumil(Backup)
- Application Programmer: Dhrumil (Prime), Ameya(Backup)
- Test Plan Engineer: Ameya(Prime), Subodh(Backup)

Part 3:

- Software Engineer: Dhrumil (Prime), Subodh(Backup)
- Database Designer/Administrator: Ameya(Prime), Dhrumil(Backup)
- Application Programmer: Kartik(Prime), Ameya(Backup)
- Test Plan Engineer: Subodh (Prime), Kartik(Backup)

6 Corrected Changes

Below pages are updated pages pf previous reports according to suggestions given in feedback of Report 1 and Report 2.

9 Local Relational Schema

Admin:

1. songs(songId, royaltyRate, title, royaltyStatus, playCount, country, language, duration, primaryArtist, albumId)
2. album(albumId, name, releaseYear, edition)
3. trackNumber(albumId,number)
4. genre(genreId, name)
5. SongHas(songId,genreId)
6. artistHas(artistId, albumId)
7. creates(artistId, songId, type)
8. Podcast(podcastId, name, country, language, rating, episodeCount, totalSubscribers)
9. episode(podcastId, number, title, duration, releaseDate, ListeningCount, AdvertisementCount)
10. episodeFeaturesGuest(podcastId, number, guestId)
11. ~~consistOf(podcastId, number)~~
12. podcastHas(podcastId, genreId)
13. podcastHistory(podcastId, month, year, subscribers, rating)
14. specialGuests(guestId, name)
15. createdBy(podcastId, hostId)
16. sponsors(sponsorId, name, amount)
17. sponseredBy(sponsorId, podcastId)
18. artist(artistId, name, status, country, primaryGenre, monthlyListeners, recordId)
19. belongsTo(id, artistId)
20. artistType(id, type)
21. collaboratedWith(artistId1, artistId2)
22. recordLabel(recordId, name)
23. receives(transactionId, recordId)
24. user(id, email, firstName, lastName, email, subscriptionStatus, subscriptionFee, phone, registrationDate)

25. serviceAccount(transactionId, date, amount, type)
26. songHistory(songId, month, year, playCount)
27. pays(userId, transactionId)
28. podcastHost(podcastId, Contact, firstName, lastName, contact, email, flatFee, city)
29. follow(artistId, userId)
30. subscribes(podcastId, userId)
31. listensTo(userId, podcastId)
32. givesPaymentTo(transactionId, HostId)
33. artistHistory(artistId, Month, Year, Subscribers)
34. distributesroyalties(artistId, recordId, songId, date, amount)

User(Listener):

1. songs(songId, royaltyRate, title, royaltyStatus, playCount, country, language, duration, primaryArtist, albumId)
2. album(albumId, name, releaseYear, edition)
3. trackNumber(albumId,number)
4. genre(genreId, name)
5. SongHas(songId, genreId)
6. artistHas(artistId, albumId)
7. creates(artistId, songId, type)
8. podcast(podcastId, name, country, language, rating, episodeCount)
9. episode(podcastId,number, title, duration, releaseDate, ListeningCount)
10. episodeFeaturesGuest(podcastId, number, guestId)
11. ~~consistOf(podcastId, number)~~
12. podcastHas(podcastId, genreId)
13. specialGuests(guestId, name)
14. createdBy(podcastId, HostId)
15. sponsors(sponsorId, name)
16. sponseredBy(sponsorId, podcastId)

17. artist(artistId, name, status, country, primaryGenre, monthlyListeners)
18. belongsTo(id, artistId)
19. artistType(id, type)
20. user(id, email, firstName, lastName, email, subscriptionStatus, subscriptionFee, phone, registrationDate)
21. serviceAccount(transactionId, date, amount, type)
22. pays(userId, transactionId)
23. podcastHost(podcastId, Contact, firstName, lastName, contact, email, flatFee, city)
24. subscribes(artistId, userId)
25. listensTo(userId, podcastId)

Artist:

1. songs(songId, royaltyRate, title, royaltyStatus, playCount, country, language, duration, primaryArtist, albumId)
2. album(albumId, name, releaseYear, edition)
3. trackNumber(albumId,number)
4. genre(genreId, name)
5. SongHas(songId, genreId)
6. artistAlbum(artistId, albumId)
7. creates(artistId, songId, type)
8. artist(artistId, name, status, country, primaryGenre, monthlyListeners, recordId)
9. belongsTo(id, artistId)
10. artistType(id, type)
11. collaboratedWith(artistId1, artistId2)
12. recordLabel(recordId, name)
13. receives(transactionId, recordId)
14. user(id, email, firstName, lastName, email, subscriptionStatus, subscriptionFee, phone, registrationDate)
15. serviceAccount(transactionId, date, amount, type)
16. songHistory(songId, month, year, playCount)

17. distributesroyalties(artistId, recordId, songId, date, amount)

Podcast Host:

1. Podcast(podcastId, name, country, language, rating, episodeCount, totalSubscribers)
2. episode(podcastId, number, title, duration, releaseDate, ListeningCount, AdvertisementCount)
3. episodeFeaturesGuest(podcastId, number, guestId)
4. ~~consistOf(podcastId, number)~~
5. podcastHas(podcastId, genreId)
6. specialGuest(guestId, name)
7. createdBy(podcastId, podcastHostId)
8. sponsors(sponsorId, name, amount)
9. sponseredBy(sponsorId, podcastId)
10. receives(transactionId, recordId)
11. user(id, email, firstName, lastName, email, subscriptionStatus, subscriptionFee, phone, registrationDate)
12. serviceAccount(transactionId, date, amount, type)
13. pays(userId, transactionId)
14. podcastHost(podcastId, Contact, firstName, lastName, contact, email, flatFee, city)
15. listensTo(userId, podcastId)
16. givesPaymentTo(transactionId, podcastHostId)

Record Label:

1. songs(songId, royaltyRate, title, royaltyStatus, playCount, country, language, duration, primaryArtist, albumId)
2. album(albumId, name, releaseYear, edition)
3. trackNumber(albumId,number)
4. genre(genreId, name)
5. SongHas(songId, genreId)
6. artistHas(artistId, albumId)

7. creates(artistId, songId, type)
8. sponsors(sponsorId, name, amount)
9. sponseredBy(sponsorId, podcastId)
10. artist(artistId, name, status, country, primaryGenre, monthlyListeners, recordId)
11. belongsTo(id, artistId)
12. artistType(id, type)
13. collaboratedWith(artistId1, artistId2)
14. recordLabel(recordId, name)
15. receives(transactionId, recordId)
16. serviceAccount(transactionId, date, amount, type)
17. songHistory(songId, month, year, playCount)
18. subscribes(artistId, userId)
19. artistHistory(artistId, month, year, subscribers)
20. distributesroyalties(artistId, recordId, songId, date, amount)

   (a) transactionId, HostId
- Referential Integrity
   (a) transactionId
   (b) HostId

31. **artistHistory(<u>artistId</u>, <u>Month</u>, <u>Year</u>, Subscribers)**
- Keys
   (a) artistId, Month, Year
- NON-NULL
   (a) artistId
   (b) Month
   (c) Year
   (d) Subscribers (default 0)
- Referential Integrity
   (a) artistId

32. **distributesroyalties(<u>artistId</u>, <u>recordId</u>, <u>songId</u>, <u>date</u>, amount)**
- Keys
   (a) artistId, recordId, date
- NON-NULL
   (a) artistId
   (b) recordId
   (c) date
   (d) amount (default 0)
   (e) songId
- Referential Integrity
   (a) artistId
   (b) recordId
   (c) songId