

CPSC 304 Project Cover Page

Milestone #4

Date: 08-06-2023

Group Number: 32

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Jiawei Liu	55362669	b7j6x	1943743535@qq.com
Flora Deng	14085211	d1i2t	flora817@gmail.com
Tammie Liang	52445806	c1g1c	tammieliang@hotmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Project Description

Music is gradually becoming an integral part of people's lives. People with headphones can be seen everywhere. Various styles of music are played in shopping malls or stores, and every day there are different concerts or lives happening around the world. At the same time, artists and the record labels behind them are becoming more and more known and a topic of conversation. Some of the loved artists or songs are known all over the world.

Data plays a huge role in the dissemination of information about records, artists and the labels associated with them. Most music apps can see the current most searched artists, songs or albums, but their main functions are still focused on the listening function rather than songs or artists' information browsing. The music information contained is not complete, even some niche music information is completely blank.

Through this project, our team hopes to create a more concise and clear platform for searching music information through the establishment of a database.

Our Melodex is a comprehensive web platform tailored for music lovers. Built using cutting-edge technologies such as React (frontend design), SpringBoot (backend design) and MySQL (database). It stores music artists and discographies, as well as being able to find them easily. It is useful for, say, a music wikipedia staff that needs to store and be able to find information about all the music artists and discographies in the world. The platform aims to make it easy for users to access, manage, and find music-related information.

Our database includes navigation bars that can navigate users to artists, music, records, labels, and the home page. The platform's administrative features allow the user to add information into appropriate pages, or make editing changes, as well as delete invalid information.

In addition to this, our platform provides users with a number of additional functions for interacting with the data:

Selection operation —— find a discography that meets a certain condition.

Join operation —— find albums with the same name as the discography.

Projection operation —— find the name and release date of all albums.

Nested Aggregation —— find the artist with the lowest average number of songs per album among the average number of songs per album.

Division operation —— find listeners who have attended all live events.

The above queries take into account most of the lookup requirements and allow the user to observe the overall data comparison.

Final Schema

Our database has been focused on the acquisition of information content about songs, records, artists and music companies from the beginning of the project, and the platform's primary focus has stayed the same. However, during the implementation of the project, we removed some of the entities and attributes.

First of all, at the beginning of the design, we designed different account interfaces for artists and listeners. However, considering that the restriction that users need an account to use the platform would affect user experience and would not be conducive to quick and easy instant searches, we decided to remove this feature. Therefore, we removed 'email' and 'userPassword' from 'Artist_ContractedWith.'

In addition, our platform focuses on managing, collecting, and querying music information. The list of songs that users listen to changes many times a day, and the 'ListenTo' entity requires users to have a personal account and does not contribute to the collection of music information, as well as placing an unnecessary burden on the platform regarding information storage. Therefore, we removed the 'ListensTo' schema.

We also removed 'numOfLikes' and 'ranking' attributes from 'Song' and 'Album' entities because most users need an official source of the number of favorites when adding information, which they may not have, and the number changes too quickly to rely solely on user updates. The lack of favorites will lack a unified reference for ranking.

(NOTE: By “mainly implemented”, we mean that is where you will find the actual SQL query, but the backend logistics always begin from tracing the Controller function that uses it)

INSERTION QUERY

MAINLY implemented in Artist_ContractedWithRepository, insertNewArtist function (backend)

```
INSERT INTO Artist_ContractedWith (artistName, age, country, biography,
numOfMembers, labelID)
VALUES (:artistName, :age, :country, :biography, :numOfMembers, :labelID);
```

DELETE QUERY

MAINLY implemented in Artist_ContractedWithRepository, deleteArtist function (backend)

```
DELETE FROM Artist_ContractedWith
WHERE artistID = :artistID
```

UPDATE QUERY

MAINLY implemented in Artist_ContractedWithRepository, updateArtist function(backend)

```
UPDATE Artist
SET artistName = :artistName, age=:age, biography=:biography,
numOfMembers=:numOfMembers-- user should specify which attribute to update
WHERE artistID = :artistID;
```

SELECTION QUERY

MAINLY implemented in GeneralController, getDynamicSelection function (backend)

The user is able to select from either Artist_ContractedWith, or Discography_Main joined with Album table (because it is meaningless to get Discography_Main on its own–Discography_Main is either an Album or Song according to ER diagram concept).

They are then allowed to select **any number of attributes** regardless of table choice (so as long as the attributes exist for the respective table, of course)

- If the user chooses Artist_ContractedWith...
 - they can pick Field1 to be either “artistName” or “country”. Val1 can be anything
 - They can pick Field2 to be either “numOfMembers” or “age”. Val2 can be anything

For example,

```
SELECT artistID, age
FROM Artist_ContractedWith
WHERE country='South Korea'  where 'South Korea' is an example value for
:val1 AND numOfMembers>=1  where 1 is an example value for :val2
```

For example,

```
SELECT artistName, age, biography
```

```
FROM Artist_ContractedWith  
WHERE artistName='(G) I-DLE' AND age>=1
```

For example,

```
SELECT *  
FROM Artist_ContractedWith  
WHERE country='Canada' AND age>=20
```

- If the user chooses Discography_Main joined with Album...
 - They can pick Field1 to be either "genre" or "discoName". Val1 can be anything
 - They can pick Field2 to be either "dID" or "numOfSongs". Val2 can be anything

For example,

```
SELECT discoName, dID  
FROM Discography_Main, Album  
WHERE genre='Kpop' where 'Kpop' is an example value for {:val1} AND  
dID>=1 where 1 is an example value for {:val2}
```

For example,

```
SELECT discoName, dID, genre  
FROM Discography_Main, Album  
WHERE discoName='GITTU RAP' AND numOfSongs>=1
```

For example,

```
SELECT dID, genre, numOfSongs  
FROM Discography_Main, Album  
WHERE genre='Kpop' AND numOfSongs>=2
```

PROJECTION QUERY

MAINLY implemented in Artist_ContractedWithRepository, projection function (backend)

```
SELECT ATTRIBUTES_FROM_USER_INPUT  
FROM Artist_ContractedWith;
```

JOIN QUERY

MAINLY implemented in AlbumRepository, getAllAlbumsWithName function (backend)

```
SELECT *  
FROM Album A, Discography_Main D  
WHERE A.albumID = D.dID AND D.discoName = :discoName
```

AGGREGATION WITH GROUP BY QUERY

MAINLY implemented in Artist_ContractedWithRepository, aggregationGroupBy function (backend)

```
SELECT R.artistID, COUNT(dID) AS numOfDiscography
FROM Releases R
GROUP BY R.artistID;
```

AGGREGATION WITH HAVING QUERY

MAINLY implemented in Artist_ContractedWithRepository, aggregationHaving function (backend)

```
SELECT A.artistName, MIN(D.releaseDate) as earliestReleaseDate
FROM Artist_ContractedWith A, Discography_Main D, Releases R
GROUP BY A.artistName
HAVING COUNT(D.dID) > 1;
```

NESTED AGGREGATION WITH GROUP BY QUERY

MAINLY implemented in Artist_ContractedWithRepository, nestedAggregation function (backend)

```
SELECT artistID
FROM Artist_ContractedWith A
WHERE (
    SELECT AVG(songCount)
    FROM (
        SELECT albumID,
            (SELECT COUNT(*) FROM Song S WHERE S.songID = Album.albumID) AS songCount
        FROM Album
        WHERE A.artistID = Album.albumID
    ) AS inner_query
) =
(
    SELECT MIN(avgSongs)
    FROM (
        SELECT artistID,
            AVG(
                (SELECT COUNT(*) FROM Song S WHERE S.songID = Album.albumID)
            ) AS avgSongs
        FROM Album
        GROUP BY artistID
    ) AS overall_avg_query
);
```

DIVISION QUERY

MAINLY implemented in `Artist_ContractedWithRepository`, division function (backend)

```
SELECT artistID
FROM Artist_ContractedWith A
WHERE NOT EXISTS (
    SELECT DISTINCT genre
    FROM Discography_Main D
    WHERE D.genre NOT IN (
        SELECT DISTINCT Dm.genre
        FROM Discography_Main Dm, Song S, Releases R
        WHERE Dm.dID = R.dID AND S.songID = R.dID AND R.artistID = A.artistID
    )
);
```

Screenshots of the sample output of the queries using the GUI

INSERTION QUERY:

Before insertion:

Melodex							Artists	Song	Album	Labels	Home
						Delete	Update				
Artist Name		Age	Country	Bio	Number of Member	Delete	Update				
Lil	Gitu	30	Canada	Raps about databases	1	X					
		Artist Name	Age	Country	Bio	Number of Member	Delete	Update			
Taylor	Swift	33	United States	American singer-songwriter	1	X					
		Artist Name	Age	Country	Bio	Number of Member	Delete	Update			
Harry	Styles	29	United Kingdom	English singer and actor	1	X					

Melodex

Artists Song Album Labels Home

All the artists you like, right in our app.

Display All Artists *Aggregation with Group By prompt*
Aggregation with Having prompt
Nested Aggregation with Group By prompt Genre God (Division)

Didn't see the artist of your choice?
Add them yourself!

Add New Artist

Artist Name: Demo Artist

Age: 20

Country: Canada

Biography: this is a demo artist

Number of members: 1

Label ID: 1

Add

The screenshot shows a dark-themed mobile application interface for 'Melodex'. At the top, there's a navigation bar with tabs for 'Artists', 'Song', 'Album', 'Labels', and 'Home'. Below the navigation, a message says 'All the artists you like, right in our app.' followed by three code snippets related to database queries. Further down, there's a message about not finding an artist and an option to add them. On the right side, there's a form titled 'Add New Artist' with fields for name, age, country, biography, number of members, and label ID. The 'Artist Name' field contains 'Demo Artist', 'Age' is '20', 'Country' is 'Canada', 'Biography' is 'this is a demo artist', 'Number of members' is '1', and 'Label ID' is '1'. A large blue button labeled 'Add' is at the bottom of the form. The background features a subtle grid pattern of colored squares (blue, purple, pink).

Melodex

Artists Song Album Labels Home

All the artists you like, right in our app.

Display All Artists *Aggregation with Group By prompt*
Aggregation with Having prompt
Nested Aggregation with Group By prompt Genre God (Division)

Didn't see the artist of your choice?
Add them yourself!

Add New Artist

Artist Name: Demo Artist

Age: 20

Country: Canada

Biography: this is a demo artist

Number of members: 1

Label ID: 1

Added successfully!

Add

This screenshot is identical to the one above it, showing the 'Add New Artist' form. The difference is in the message below the form: instead of the standard 'Add' button, there is a message 'Added successfully!' indicating that the addition was successful. The rest of the interface, including the form fields and background, remains the same.

After insertion:

Melodex						
Artist Name	Age	Country	Bio	Number of Member	Delete	Update
Taylor Swift	33	United States	American singer-songwriter	1	X	
Artist Name	Age	Country	Bio	Number of Member	Delete	Update
Harry Styles	29	United Kingdom	English singer and actor	1	X	
Artist Name	Age	Country	Bio	Number of Member	Delete	Update
Demo Artist	20	Canada	this is a demo artist	1	X	

DELETION QUERY:

Before deletion:

Artist Details						
Artist Name	Age	Country	Bio	Number of Member	Delete	Update
1	asads	adas		1	X	Edit
Ryuichi Sakamoto	71	Japan		1	X	Edit
Lil Gittu	30	Canada	Raps about databases	1	X	Edit
American						

After deletion:

Artist Name

Age

Country

Bio

Number of Member

Delete

Update

Reiichi
Sakamoto

71

Japan

1

X

Edit

Artist Name

Age

Country

Bio

Number of Member

Delete

Update

Lil
Gittu

30

Canada

Raps
about
databases

1

X

Edit

Artist Name

Age

Country

Bio

Number of Member

Delete

Update

Taylor
Swift

33

United
StatesAmerican
singer-
songwriter

1

X

Edit

UPDATE QUERY

Before clicking “submit”:

All Artists

Artist Name	Age	Country	Bio	Number of Member	Delete	Update
HEIZE	21	South Korea	Bio about database	2	X	Edit
UPDATED HEIZE	311	South Korea	Biooo	2		
(G)I-DLE	5	South Korea	5 member subgroup producing Kpop girl group	5	X	Edit
Ryuchi Sakamoto	71	Japan		1	X	Edit
Lil' Gills	30	Canada	Bio about database	1	X	Edit

Artist Label ID: Submit

After clicking “submit”:

All Artists

Artist Name	Age	Country	Bio	Number of Member	Delete	Update
UPDATED HEIZE	311	South Korea	Biooo	2	X	Edit
UPDATED HEIZE	311	South Korea	Biooo	2		
(G)I-DLE	5	South Korea	5 member subgroup producing Kpop girl group	5	X	Edit
Ryuchi Sakamoto	71	Japan		1	X	Edit
Lil' Gills	30	Canada	Bios about databases	1	X	Edit

Artist Label ID: Submit

SELECTION QUERY:

Before clicking “search”:

The screenshot shows the Melodex homepage with a dark purple background. At the top, there's a navigation bar with links for Artists, Song, Album, Labels, and Home. Below the navigation, a large central area features the text "Welcome to Melodex" and a subtext "A music website for users to discover a whole world of artists and music, all in one place." A search interface is centered, with a dropdown menu set to "Table: Artists". Under "Attributes", several checkboxes are listed: Artist ID (unchecked), Artist Name (checked), Age (checked), Country (unchecked), Biography (unchecked), Number of Members (unchecked), and Record Label ID (unchecked). Below this, a "Filters" section includes dropdown menus for "Country" (set to "South Korea") and "Number of Members" (set to "is at least: 1"), followed by a "Search" button. To the right of the search interface is a decorative graphic of colored squares.

After clicking “search”:

This screenshot is identical to the previous one, showing the Melodex homepage before a search was performed. The search interface and filters remain the same, with no results displayed yet.

Before clicking search:

Melodex

Artists Song Album Labels Home

Welcome to Melodex

A music website for users to discover a whole world of artists and music, all in one place.

Try a simple search here:

Attributes: Album ID Album Name Genre Release Date Number of Songs Total Duration

Filters: Genre : Kpop Number of Songs : is at least: 5

Search

After clicking search:

Melodex

Artists Song Album Labels Home

Welcome to Melodex

A music website for users to discover a whole world of artists and music, all in one place.

Try a simple search here:

Attributes: Album ID Album Name Genre Release Date Number of Songs Total Duration

Filters: Genre : Kpop Number of Songs : is at least: 5

Search

discoName: K-GITTI-Pop
genre: Kpop
discoName: Round and Round
genre: Kpop
discoName: Merry Christmas Mr.Lawrence
genre: Kpop
discoName: I FEEL
genre: Kpop
discoName: Goblin OST
genre: Kpop

PROJECTION QUERY

Before clicking “project”:

The screenshot shows the Melodex application interface. At the top, there's a navigation bar with links for Artists, Song, Album, Labels, and Home. Below the navigation is a sidebar titled "Attributes" with checkboxes for Artist ID, Artist Name, Age, Country, Biography, Number of Members, and Record Label ID. The main content area is titled "All Artists" and displays a table with four rows of artist data. The columns are Artist Name, Age, Country, Bio, Number of Member, Delete, and Update. The first row is for HEIZE (Age 31, South Korea). The second row is for (G)I-DLE (Age 5, South Korea). The third row is for Ryuichi Sakamoto (Age 74, Japan). The fourth row is for Lil' Dete (Age 30, Canada). Each row has a delete button ('X') and an edit button ('Edit').

After clicking project:

The screenshot shows the same Melodex application interface after the "Project" button was clicked. A message at the top says "Projection success! Refresh to see all artists or to try a different projection." The main content area is now titled "Projected Artists". It contains five horizontal bars, each representing a different country and its corresponding artist. From left to right, the bars are: South Korea (HEIZE), South Korea ((G)I-DLE), Japan (Ryuichi Sakamoto), Canada (Lil' Dete), and United States (Taylor Swift). Each bar includes the country name, artist name, and age.

JOIN QUERY

Before clicking search:

The screenshot shows the Melodex application interface. On the left, there is a list of albums with columns for Album Name, Genre, Release Date, and Total Duration. The albums listed are:

- Goblin OST (Kpop, 2014-01-01, 15, 00:03:00)
- I FEEL (Kpop, 2022-03-15, 02:00, 00:04:15)
- Merry Christmas Mr Lawrence (Kpop, 2015-09-14, 12:00, 00:40:05)
- 1996 (Dance, 1996-04-01, 12, 00:09:27)

On the right, there is a detailed view of the "I FEEL" album with fields for Album Name, Genre, Release Date, Number of Songs, and Total Duration, along with an "Add" button.

After clicking search:

The screenshot shows the Melodex application interface after a search has been performed. The search term "I FEEL" is entered in the search bar. The results show the same list of albums, but the "I FEEL" album is now highlighted in the list.

On the right, the detailed view of the "I FEEL" album remains the same, showing the album name, genre, release date, number of songs, total duration, and an "Add" button.

For Division and all the Aggregations, all their buttons are on the same page so I will only show one “before” image.

Before:

Melodex

Artists Song Album Labels Home

Choose what you'd like to do!

Display All Artists

Fun

- *Aggregation with Group By prompt*
- *Aggregation with Having prompt*
- *Nested Aggregation with Group By prompt*
- Genre God (Division)

Didn't see the artist of your choice?
Add them yourself!

Add New Artist

Artist Name: Name

Age: Artist age or group age since debut

Country: Country

Biography: This artist is a cool singer

Number of members: 1 for solo artists

Label ID: Leave blank if none

Add

A screenshot of the Melodex web application. The top navigation bar includes links for Artists, Song, Album, Labels, and Home. Below the navigation, a message says "Choose what you'd like to do!" followed by a "Display All Artists" button. A "Fun" section contains four options: "Aggregation with Group By prompt*", "Aggregation with Having prompt*", "Nested Aggregation with Group By prompt*", and "Genre God (Division)". To the right, there's a form for adding a new artist with fields for Name, Age, Country, Biography, Number of members, and Label ID. At the bottom right is a blue "Add" button. The background features a dark theme with abstract geometric shapes.

AGGREGATION WITH HAVING

Melodex

Artists Song Album Labels Home

Choose what you'd like to do!

Display All Artists

Fun

- *Aggregation with Group By prompt*
- *Aggregation with Having prompt*
- *Nested Aggregation with Group By prompt*
- Genre God (Division)

Names of artists who released the earliest discographies:

Name: HEIZE Earliest Release Date: 1996-06-04 12:00
Name: (G)I-DLE Earliest Release Date: 1996-06-04 12:00
Name: Ryuchi Sakamoto Earliest Release Date: 1996-06-04 12:00
Name: Lil Gittu Earliest Release Date: 1996-06-04 12:00
Name: Taylor Swift Earliest Release Date: 1996-06-04 12:00
Name: Harry Styles Earliest Release Date: 1996-06-04 12:00

Didn't see the artist of your choice?
Add them yourself!

Add New Artist

Artist Name: Name

Age: Artist age or group age since debut

Country: Country

Biography: This artist is a cool singer

Number of members: 1 for solo artists

Label ID: Leave blank if none

Add

A screenshot of the Melodex web application, identical to the one above but with a specific section highlighted. A dashed box encloses a list of artists and their earliest release dates: HEIZE (1996-06-04), (G)I-DLE (1996-06-04), Ryuchi Sakamoto (1996-06-04), Lil Gittu (1996-06-04), Taylor Swift (1996-06-04), and Harry Styles (1996-06-04). The rest of the interface is the same, including the "Add New Artist" form and the overall layout.

AGGREGATION WITH GROUP BY

Melodex

Choose what you'd like to do!

Display All Artists

Fun

Aggregation with Group By prompt

Aggregation with Having prompt

Nested Aggregation with Group By prompt

Genre God (Division)

Number of Discographies Released by Each Artist:

ID: 1 Number of Releases: 2

ID: 2 Number of Releases: 2

ID: 3 Number of Releases: 2

ID: 4 Number of Releases: 6

ID: 5 Number of Releases: 2

NESTED AGGREGATION WITH GROUP BY PROMPT:

Melodex

Choose what you'd like to do!

Display All Artists

Fun

- *Aggregation with Group By prompt*
- *Aggregation with Having prompt*
- *Nested Aggregation with Group By prompt*
- Genre God (Division)

Artist IDs whose average number of songs released per album is the lowest among all artists:

Artist ID: 1
Artist ID: 2
Artist ID: 3
Artist ID: 5

DIVISION:**Melodex****Choose what you'd like to do!**[Display All Artists](#)**Fun**

Aggregation with Group By prompt

Aggregation with Having prompt

Nested Aggregation with Group By prompt

Genre God (Division)

Artist ID of artists who covered all genres:**Artist ID: 4**

Example snippet of how we use data from backend to frontend:

Here is an asynchronous function that makes a call to the API route that will fetch all the albums with the name we specified for us. We parse what we get back into JSON so we can use it. The implementation for the API routes are implemented in our Java backend.

```
const handleSearchClick = async (e) => {
  e.preventDefault();
  if (name === '') {
    return;
  }
  const response = await fetch('http://localhost:8080/api/albums/all/' + name, {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setAlbums(json);
    console.log("Fetched artists! ", json)
  } else {
    console.log("Could not fetch artists");
  }
}
```

Typically, a button will be clicked to activate a function like this.

```
<div className={styles.searchArea}>
  <h2>Albums</h2>
  <input type="text" placeholder="Search by Name" onChange={(e) => setName(e.target.value)}></input>
  <button onClick={handleSearchClick}>Search</button>
</div>
  <ul style={listStyleType none} >
    {albums.map(album => [
      <li key={album.id}>
```

Any requests that require a body, such as POST or UPDATE requests will have a JSON stringified body (see highlighted line). For example, in this function to add an artist, we create an artist object from whatever the user inputs from our frontend (its fields will be stored in states that “react” to user inputs). Then, again, we make the appropriate call to the according API route. Since we are not getting anything back from POST or UPDATE requests (it is set up like this in the backend), we do not do response.json() this time, i.e. we do not parse the response into json..

```
const handleAddClick = async (e) => {
  e.preventDefault();

  const artist = {
    artistName,
    age,
    country,
    biography: bio,
    numOfMembers,
    labelID,
  };

  const response = await fetch("http://localhost:8080/api/artists/add", {
    method: "POST",
    body: JSON.stringify(artist),
    headers: {
      "Content-Type": "application/json",
    },
  });

  if (response.ok) {
    console.log("New artist added!");
    setSuccessAdd(true);
    setError(false);
  } else {
    setError(true);
    setSuccessAdd(false);
    console.log("Error!");
  }
};
```

Once again, this function will be used by a button, like this:

```
        }
      <div className={styles.btn} onClick={handleAddClick}>
        Add
      </div>
```

Quick backend overview: For the API routes in the backend, they can all be found in the classes under the Controller folder.

For example, anything that is “<http://localhost:8080/artists/SOMETHING>” will be in Artist_ContractedWithController

```
1 package com.musicmanagementsystem.controller;
2
3 import ...
4
5 @RestController
6 @RequestMapping("/api/artists")
7 @CrossOrigin
8 public class Artist_ContractedWithController {
9   @Autowired
10  public Artist_ContractedWithService artist_contractedWithService;
11
12  @Autowired
13  public JdbcTemplate jdbcTemplate;
14
15  @PostMapping("/add")
16  public void addNewArtist(@RequestBody Artist_ContractedWithBody reqBody) {
17    artist_contractedWithService.insertNewArtist(reqBody.getArtistName(), reqBody.getAge(), reqBody.getCountry()
18      | reqBody.getBiography(), reqBody.getNumOfMembers(), reqBody.getLabelID());
19    System.out.println("Added new artist!");
20  }
21}
```

For the native queries (SQL) that the controllers actually use, they are MOSTLY found in the repositories. But, queries that don't return all attributes may be found in the entity classes (for DTO mapping), and dynamic queries may be found in the Controller (eg. selection query is in GeneralController, projection query is in Artist_ContractedWithController). The Service classes mainly add as a middle man for the Controller classes and Repositories.

Example of query implementation in Artist_ContractedRepository for the add a new artist function:

```

package com.musicmanagementsystem.repository;

import ...

@Repository
public interface Artist_ContractedWithRepository extends JpaRepository<Artist_ContractedWith, Integer> {

    String insertNewArtistQuery = "INSERT INTO Artist_ContractedWith (artistName, age, country, biography, numOfMembers, labelID) " +
        "VALUES (:artistName, :age, :country, :biography, :numOfMembers, :labelID)";
    String getAllArtistsQuery = "SELECT * FROM Artist_ContractedWith";
    String updateArtistQuery = "UPDATE Artist_ContractedWith " +
        "SET artistName= :artistName, age=:age, country=:country, biography=:biography, numOfMembers=:numOfMembers, labelID=:labelID " +
        "WHERE Artist_ContractedWith.artistID= :artistID";
    String dynamicSelection = "SELECT :attributes FROM :table ";
    String deleteArtistByIdQuery = " DELETE FROM Artist_ContractedWith WHERE artistID = :artistID";
    String nestAggregationQuery = "SELECT artistID\n" +
        "FROM Artist_ContractedWith A\n" +
        "WHERE (\n" +
        "    SELECT AVG(songCount)\n" +
        "    FROM (\n" +
        "        SELECT albumID,\n" +
        "            (SELECT COUNT(*) FROM Song S WHERE S.songID = Album.albumID) AS songCount\n" +
        "        FROM Album\n" +
        "        WHERE A.artistID = Album.albumID\n" +
        "    ) AS T
    ";

    @Modifying
    @Transactional
    @Query(value=insertNewArtistQuery, nativeQuery = true)
    public void insertNewArtist(@Param("artistName") String artistName, @Param("age") int age, @Param("country")String country,
                               @Param("biography") String biography, @Param("numOfMembers") int numOfMembers, @Param("labelID") Integer labelID);

    @Query(value=getAllArtistsQuery, nativeQuery = true)
    public List<Artist_ContractedWith> getAllArtists();

    @Modifying
    @Transactional
    @Query(value=updateArtistQuery, nativeQuery = true)
    public void updateArtist(@Param("artistID") int artistID, @Param("artistName") String artistName, @Param("age") int age, @Param("country")String country,
                           @Param("biography") String biography, @Param("numOfMembers") int numOfMembers, @Param("labelID") Integer labelID);

    // SELECTION OPERATION to submit
    @Query(value = dynamicSelection, nativeQuery = true)
    List<Object> getDynamicSelection(@Param("table") String table, @Param("attributes") String attributes);

    @Modifying
    @Transactional
    @Query(value = "INSERT INTO Releases (artistID, dID) VALUES (:artistID, :dID)", nativeQuery = true)
    public void releaseDiscography(@Param("artistID") Integer albumID, @Param("dID") Integer discoID);
}

```

