Information System Management Report

Léandre BROSSIER, Mathys DECKER, Benoît HUA, Kylie WU - ESILV A4 CDOF3

Find the \DATABASE\WMLMS_Step_3_Database_Setup.sql in the zip file to get all our sql queries displayed in this document !

Chapter 1 : Preparing the relational schema

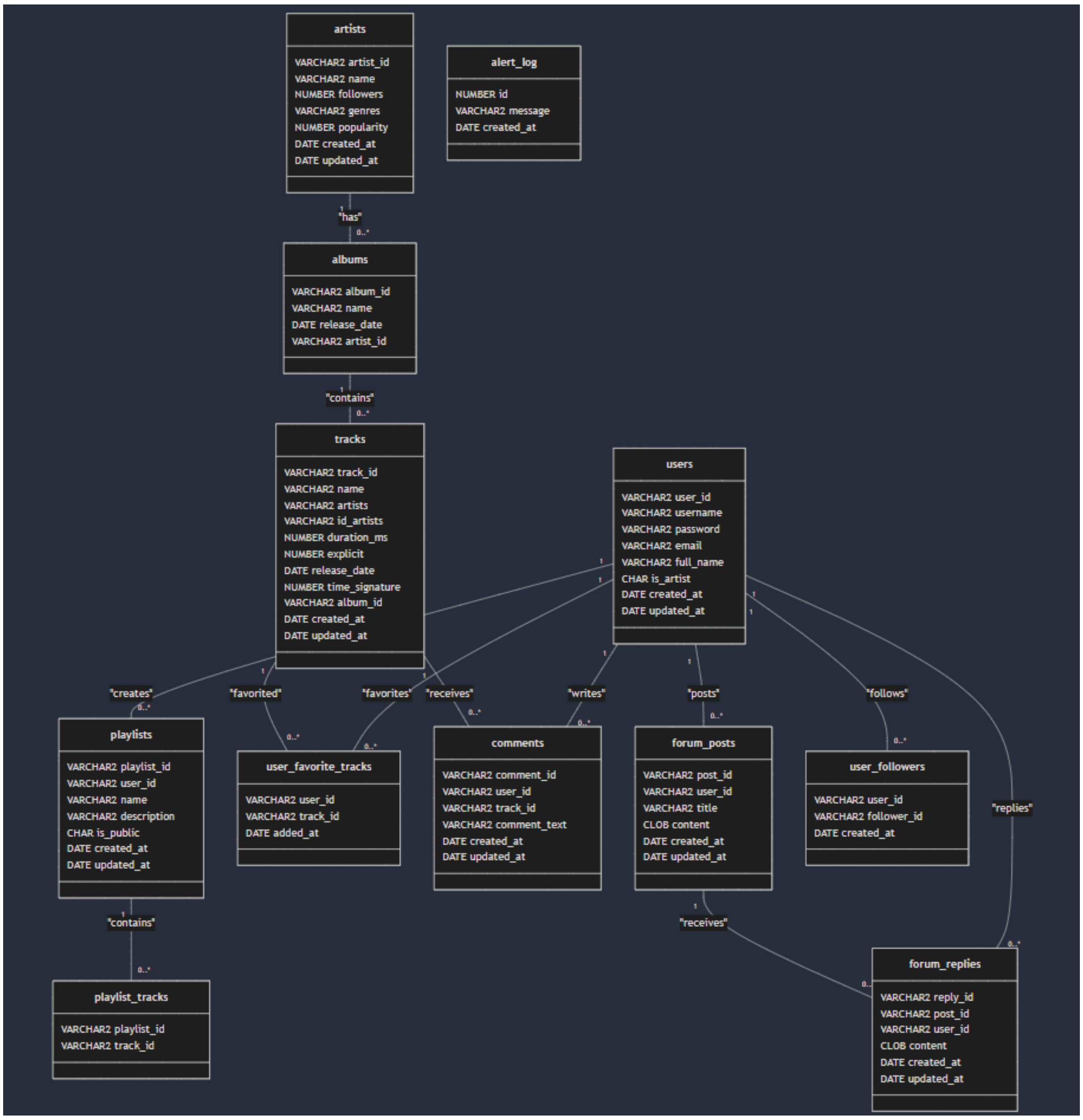
1. Data search

The dataset used for this project is the Spotify Dataset from Kaggle : https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks, with “artists.csv” and “tracks.csv” (more than 500 000 lines in each one). We chose to split “tracks.csv” into “tracks.csv” and “tracks_audio_features.csv” thanks to a python script (available in the Github repository), with the columns ['id', 'name','artists', 'id_artists','duration_ms', 'explicit','release_date','time_signature'] for tracks.csv and ['id', 'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo'] for “tracks_audio_features.csv”. We also created and added additional tables : albums, playlists, users, user_favorite_tracks (and some more throughout the project, see the board below).

2. Design the relational schema

Table name	Primary key	Foreign keys
artists	artist_id VARCHAR2(36)	-
albums	album_id VARCHAR2(36)	artist_id
tracks	track_id VARCHAR2(36)	album_id
tracks_audio_features	track_id VARCHAR2(36)	track_id
users	user_id VARCHAR2(36)	-
playlists	playlist_id VARCHAR2(36)	user_id
user_favorite_tracks	user_id VARCHAR2(36) track_id VARCHAR2(36)	user_id, track_id
alert_log	id NUMBER	-
playlist_tracks	playlist_id VARCHAR2(36) track_id VARCHAR2(36)	playlist_id, track_id
comments	comment_id VARCHAR2(36)	user_id, track_id

forum_posts	post_id VARCHAR2(36)	user_id
forum_replies	reply_id VARCHAR2(36)	post_id, user_id
user_followers	user_id VARCHAR2(36) follower_id VARCHAR2(36)	user_id, follower_id
logs	log_id VARCHAR2(36)	user_id



Relationships :

- The artists table is independent and serves as a reference for albums
- Each album is associated with one artist through the artist_id foreign key
- Tracks are linked to albums via the album_id foreign key
- Tracks_audio_features contains additional information for each track, linked by track_id

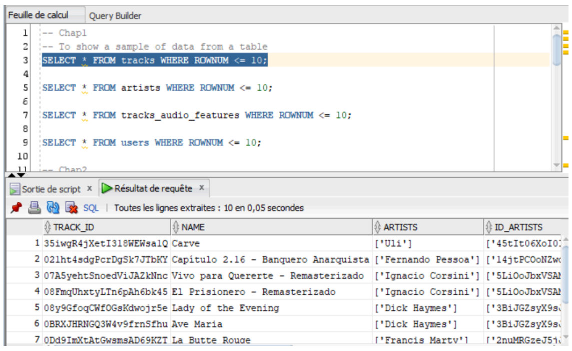
- Users table stores user information independently
- Playlists are created by users, hence the user_id foreign key
- User_favorite_tracks is a junction table linking users to their favorite tracks
- Playlist_tracks links playlists to tracks, forming a many-to-many relationship
- Comments are associated with both users and tracks
- Forum_posts are created by users
- Forum_replies are linked to forum_posts and created by users
- User_followers establishes a self-referential relationship within the users table
- Logs track user activities and are linked to the users table

Integrity constraints include :

- Primary keys are unique and not null
- Username and email in the users table are unique
- Generate_uuid function for unique identifiers (in the IDs)
- Date fields have default values of SYSDATE or SYSTIMESTAMP
- Boolean fields use CHAR(1) with 'Y' or 'N' values
- ON DELETE CASCADE is used for some foreign keys to maintain data consistency
- Some tables use composite primary keys (i.e. user_followers, playlist_tracks)
- Text fields like comment_text and event_description have specified maximum lengths
- CLOB data type is used for longer text content in forum posts and replies

3. **Populating the tables** (lines 170-325 in the WMLMS_tables_creation.sql)

We imported the .csv data into our database according to the defined schema, and populated the tables needed.



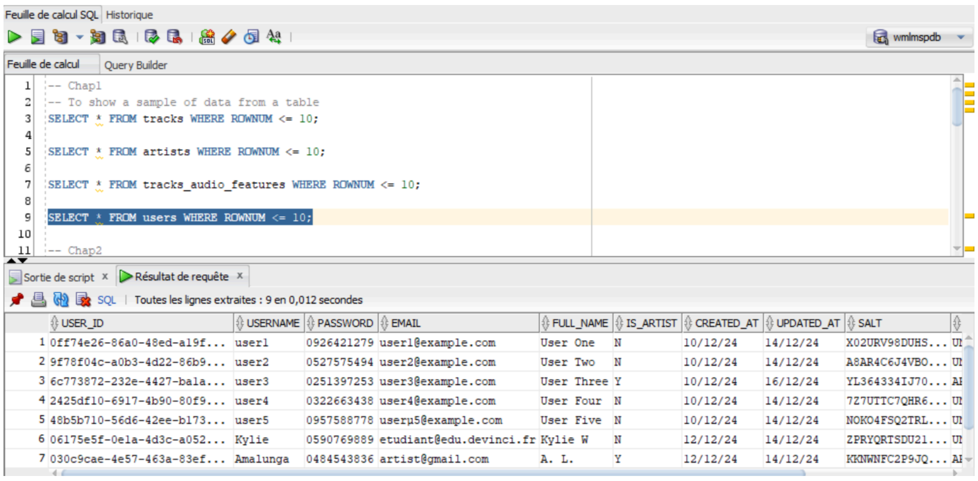
Feuille de calcul | Query Builder

```sql
-- Chap1
-- To show a sample of data from a table
SELECT * FROM tracks WHERE ROWNUM <= 10;

SELECT * FROM artists WHERE ROWNUM <= 10;

SELECT * FROM tracks_audio_features WHERE ROWNUM <= 10;

SELECT * FROM users WHERE ROWNUM <= 10;

-- Chap2
```

Sortie de script | Résultat de requête

SQL | Toutes les lignes extraites : 10 en 0,05 secondes

	TRACK_ID	NAME	ARTISTS	ID_ARTISTS
1	35iwgR4jXetI318WEWsa1Q	Carve	['Uli']	['45tIt06XoI0
2	021ht4sdgPcrDgSk7JTbKY	Capítulo 2.16 - Banquero Anarquista	['Fernando Pessoa']	['14jtPCOoNZw
3	07A5yehtSnoedViJAZkNnc	Vivo para Quererte - Remasterizado	['Ignacio Corsini']	['5LiOoJbxVSA
4	08FmqUhxtyLTn6pAh6bk45	El Prisionero - Remasterizado	['Ignacio Corsini']	['5LiOoJbxVSA
5	08y9GfoqCWfOGsKdwojr5e	Lady of the Evening	['Dick Haymes']	['3BiJGZsyX9s
6	0BRXJHRNGQ3W4v9frnSfhu	Ave Maria	['Dick Haymes']	['3BiJGZsyX9s
7	0Dd9ImXtAtGwsmsAD69KZT	La Butte Rouge	['Francis Marty']	['2nuMRGzeJ5j



Feuille de calcul SQL | Historique

wmlmspdb

Feuille de calcul | Query Builder

```sql
-- Chap1
-- To show a sample of data from a table
SELECT * FROM tracks WHERE ROWNUM <= 10;

SELECT * FROM artists WHERE ROWNUM <= 10;

SELECT * FROM tracks_audio_features WHERE ROWNUM <= 10;

SELECT * FROM users WHERE ROWNUM <= 10;

-- Chap2
```

Sortie de script | Résultat de requête

SQL | Toutes les lignes extraites : 9 en 0,012 secondes

	USER_ID	USERNAME	PASSWORD	EMAIL	FULL_NAME	IS_ARTIST	CREATED_AT	UPDATED_AT	SALT	
1	0ff74e26-86a0-48ed-a19f...	user1	0926421279	user1@example.com	User One	N	10/12/24	14/12/24	X02URV98DUHS...	UI
2	9f78f04c-a0b3-4d22-86b9...	user2	0527575494	user2@example.com	User Two	N	10/12/24	14/12/24	A8AR4C6J4VBO...	UI
3	6c773872-232e-4427-ba1a...	user3	0251397253	user3@example.com	User Three	Y	10/12/24	16/12/24	YL364334IJ70...	AI
4	2425df10-6917-4b90-80f9...	user4	0322663438	user4@example.com	User Four	N	10/12/24	14/12/24	7Z7UTTC7QHR6...	UI
5	48b5b710-56d6-42ee-b173...	user5	0957588778	userµ5@example.com	User Five	N	10/12/24	14/12/24	NOKO4FSQ2TRL...	UI
6	06175e5f-0e1a-4d3c-a052...	Kylie	0590769889	etudiant@edu.devinci.fr	Kylie W	N	12/12/24	14/12/24	ZPRYQRTSDU21...	UI
7	030c9cae-4e57-463a-83ef...	Amalunga	0484543836	artist@gmail.com	A. L.	Y	12/12/24	14/12/24	KKNWNFC2P9JQ...	AI

Chapter 2 : Security and user management

1. Creating users

Application user (app) : manages application operations, application user with general privileges

Administrator (admin) : administrator with full access, for database administration tasks

Regular user (regular_user) : standard user with limited read-only access to specific views

2. Assigning privileges

Privileges are granted specifically to each type of user :

For the app user :
- Has CONNECT, RESOURCE and ALL privileges
- Can select from app_view_tracks, app_view_playlists, and app_view_user_playlists

For the admin user :
- Has CONNECT and DBA privileges
- Has UNLIMITED TABLESPACE privilege
- Full access to all tables (SELECT, INSERT, UPDATE, DELETE)
- Can SELECT from all admin views (admin_view_users, admin_view_artists, admin_view_tracks) and app views (app_view_tracks, app_view_playlists, app_view_user_playlists)

For the regular_user :
- Has CONNECT privilege
- Read-only (SELECT) access to app_view_tracks, app_view_playlists, and app_view_user_playlists

3. Securing the system

a) There is a generate_uuid function for unique identifiers, and a trigger-based UUID assignment, which ensures UUID generation when inserting new users.

b) The use of views (app_view_tracks, app_view_playlists...) allows limiting access to sensitive data and precisely controlling which information is accessible to each type of user.

c) Passwords are hashed and salted before storage :
- We modified the users table to store hashed passwords (ALTER TABLE users MODIFY password VARCHAR2(256))
- A salt field was added to enhance hashing security (ALTER TABLE users ADD (salt VARCHAR2(64));)
- custom hash function (custom_hash) to generate a unique hash for each password
- Function to hash passwords with salt (hash_password), combining the password and salt for increased security
- The insertion trigger for users (trg_users_before_insert) was modified to automatically generate a salt, hash the password, and store both in the users table
- A similar trigger (trg_users_before_update) was created to handle password updates, ensuring that passwords are always properly hashed and salted when changed

Chapter 3 : Queries and optimization

1. **Designing SQL queries** (lines 436 to 499 in WMLMS_tables_creation.sql)

JOIN between tables to show the tracks names, artists, albums and playlists for the tracks that are more than 5 minutes long.

```
EXPLAIN PLAN FOR
SELECT t.name AS track_name, a.name AS artist_name, al.name AS album_name,
p.name AS playlist_name
FROM tracks t
JOIN artists a ON INSTR(t.id_artists, a.artist_id) > 0
LEFT JOIN albums al ON t.album_id = al.album_id
LEFT JOIN user_favorite_tracks uft ON t.track_id = uft.track_id
LEFT JOIN users u ON uft.user_id = u.user_id
LEFT JOIN playlists p ON u.user_id = p.user_id
WHERE t.duration_ms > 300000;
```

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY); -- 0,313s
```

Analytical function AVG() to compute the average energy of the tracks per year or release

```
EXPLAIN PLAN FOR
SELECT EXTRACT(YEAR FROM t.release_date) AS release_year, AVG(taf.energy) AS
avg_energy
FROM tracks t
JOIN tracks_audio_features taf ON t.track_id = taf.track_id
GROUP BY EXTRACT(YEAR FROM t.release_date)
ORDER BY release_year DESC;
```

```
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY); -- 0,101s
```

Analytical function COUNT() to count the number of tracks per artist and how only those who have more than 5 tracks

```
SELECT a.name AS artist_name, COUNT(t.track_id) AS track_count
FROM artists a
JOIN tracks t ON INSTR(t.id_artists, a.artist_id) > 0
GROUP BY a.name
HAVING COUNT(t.track_id) > 5
ORDER BY track_count DESC;
```

Analytical function SUM() to have the total number of followers for each genre of artist

```
SELECT a.genres, SUM(a.followers) AS total_followers
FROM artists a
GROUP BY a.genres
```

```sql
ORDER BY total_followers DESC;
```

Nested query to retrieve artists whose popularity is above the average of all artists

```sql
SELECT name, popularity FROM artists
WHERE popularity > (SELECT AVG(popularity) FROM artists);
```

Subquery to identify popular artists compared to the average number of followers in the database

```sql
SELECT name, followers FROM artists
WHERE followers > (SELECT AVG(followers) FROM artists);
```

Window function to rank artists by number of followers

```sql
SELECT name, followers, RANK() OVER (ORDER BY followers DESC) AS follower_rank
FROM artists;
```

Window function to rank artists with more than 1 million followers

```sql
SELECT name, followers, RANK() OVER (ORDER BY followers DESC) AS follower_rank
FROM artists
WHERE followers > 1000000;
```

Window function to rank artists based on the total number of tracks they have, and display the number of tracks for each artist

```sql
SELECT a.name AS artist_name, COUNT(t.track_id) AS track_count, RANK() OVER
(ORDER BY COUNT(t.track_id) DESC) AS track_rank
FROM artists a
JOIN tracks t ON INSTR(t.id_artists, a.artist_id) > 0
GROUP BY a.name;
```

2. **Query optimization** (lines 502 to 540)

With the creation of indexes on frequently used columns in joins and WHERE conditions, and materialized views to speed up searches, then rewriting certain queries for greater efficiency :

```sql
CREATE INDEX idx_user_favorite_tracks_user_id ON user_favorite_tracks(user_id);
CREATE INDEX idx_tracks_album_id ON tracks(album_id);
CREATE INDEX idx_tracks_id_artists ON tracks(id_artists);
CREATE INDEX idx_playlists_user_id ON playlists(user_id);
```

Rewriting of some queries to enhance their performances

Show the tracks names, artists, albums and related playlists for the tracks that are more than 5 minutes long

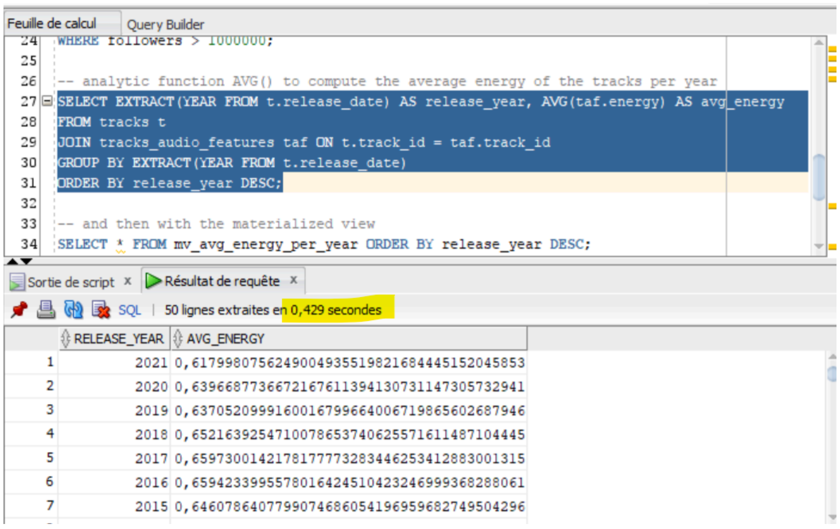
```sql
EXPLAIN PLAN FOR
SELECT
    t.track_id,
    t.name AS track_name,
    a.name AS artist_name,
    al.name AS album_name,
    u.username AS user_name,
    p.name AS playlist_name
FROM tracks t
JOIN artists a ON INSTR(t.id_artists, a.artist_id) > 0
LEFT JOIN albums al ON t.album_id = al.album_id
LEFT JOIN user_favorite_tracks uft ON t.track_id = uft.track_id
LEFT JOIN users u ON uft.user_id = u.user_id
LEFT JOIN playlists p ON u.user_id = p.user_id
WHERE t.duration_ms > 300000;
```

```sql
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY); -- 0,034s
```

Materialized view :

Get the average energy per year :

```sql
CREATE MATERIALIZED VIEW mv_avg_energy_per_year AS
SELECT EXTRACT(YEAR FROM t.release_date) AS release_year, AVG(taf.energy) AS
avg_energy
FROM tracks t
JOIN tracks_audio_features taf ON t.track_id = taf.track_id
GROUP BY EXTRACT(YEAR FROM t.release_date);
```



```sql
WHERE followers > 1000000;

-- analytic function AVG() to compute the average energy of the tracks per year
SELECT EXTRACT(YEAR FROM t.release_date) AS release_year, AVG(taf.energy) AS avg_energy
FROM tracks t
JOIN tracks_audio_features taf ON t.track_id = taf.track_id
GROUP BY EXTRACT(YEAR FROM t.release_date)
ORDER BY release_year DESC;

-- and then with the materialized view
SELECT * FROM mv_avg_energy_per_year ORDER BY release_year DESC;
```

50 lignes extraites en 0,429 secondes

	RELEASE_YEAR	AVG_ENERGY
1	2021	0,6179980756249004935519821684445152045853
2	2020	0,6396687736672167611394130731147305732941
3	2019	0,6370520999160016799664006719865602687946
4	2018	0,6521639254710078653740625571611487104445
5	2017	0,6597300142178177773283446253412883001315
6	2016	0,6594233995578016424510423246999368288061
7	2015	0,6460786407799074686054196959682749504296

Use the materialized view mv_avg_energy_per_year in a query for instance :

```sql
SELECT * FROM mv_avg_energy_per_year ORDER BY release_year DESC; -- 0,009s
```

Chapter 4 : Automation of the Information System

1. Triggers

a) Triggers for UUID generation :

```
CREATE OR REPLACE TRIGGER trg_artists_id
BEFORE
INSERT ON artists
FOR EACH ROW
BEGIN
    IF :NEW.artist_id IS NULL THEN
        :NEW.artist_id := generate_uuid();
    END IF;
END;
/
```

```
CREATE OR REPLACE TRIGGER trg_albums_id
BEFORE INSERT ON albums
FOR EACH ROW
BEGIN
    IF :NEW.album_id IS NULL THEN
        :NEW.album_id := generate_uuid();
    END IF;
END;
/
```

We also have similar scripts for trg_tracks_id, trg_users_id, trg_playlists_id, trg_comments_id, trg_forum_posts_id, trg_forum_replies_id or trg_logs_id.

b) Triggers for table updates (lines 547 to 578) :

```
CREATE OR REPLACE TRIGGER trg_update_artists_updated_at
BEFORE UPDATE ON artists
FOR EACH ROW
BEGIN
    :NEW.updated_at := SYSDATE;
END;
/
```

```
CREATE OR REPLACE TRIGGER trg_update_tracks_updated_at
BEFORE UPDATE ON tracks
FOR EACH ROW
BEGIN
    :NEW.updated_at := SYSDATE;
END;
/
```

```
CREATE OR REPLACE TRIGGER trg_update_users_updated_at
BEFORE UPDATE ON users
FOR EACH ROW
BEGIN
    :NEW.updated_at := SYSDATE;
END;
/
```

```
CREATE OR REPLACE TRIGGER trg_update_playlists_updated_at
BEFORE UPDATE ON playlists
FOR EACH ROW
BEGIN
    :NEW.updated_at := SYSDATE;
END;
/
```

To add an user in the artists table when their artist status is updated :

```
CREATE OR REPLACE TRIGGER trg_insert_artist
AFTER UPDATE OF is_artist ON users
FOR EACH ROW
BEGIN
    IF :NEW.is_artist = 'Y' AND :OLD.is_artist = 'N' THEN
        INSERT INTO artists (artist_id, name, followers, genres, popularity,
created_at, updated_at)
        VALUES (generate_uuid(), :NEW.full_name, 0, NULL, 0, SYSDATE, SYSDATE);
    END IF;
END;
/
```

c) Triggers for passwords hashing :

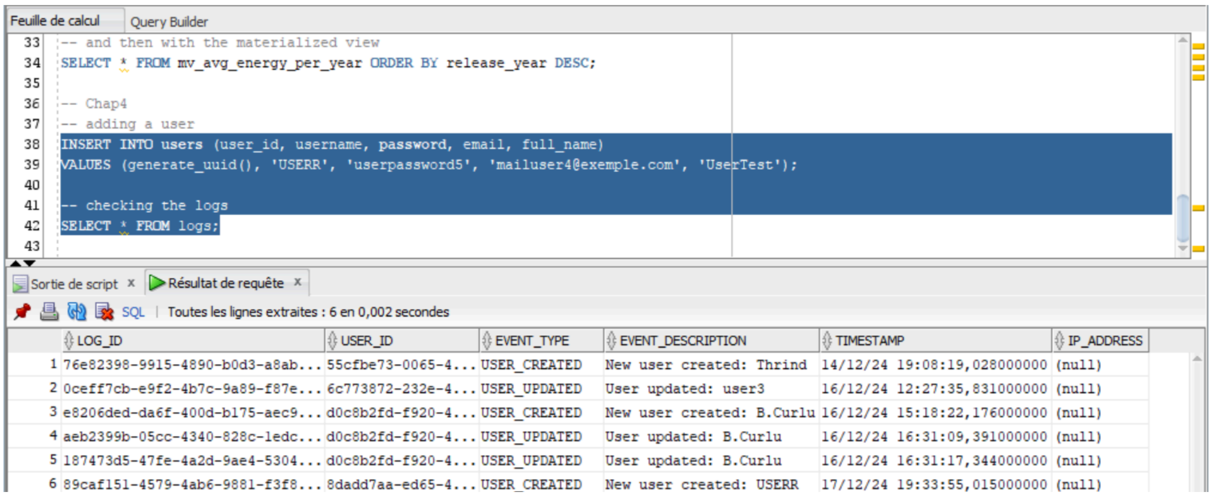
```
CREATE OR REPLACE TRIGGER trg_users_before_insert
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    IF :NEW.user_id IS NULL THEN
        :NEW.user_id := generate_uuid();
    END IF;
    :NEW.salt := DBMS_RANDOM.STRING('X', 64);
    :NEW.password := hash_password(:NEW.password, :NEW.salt);
END;
/
```

```
CREATE OR REPLACE TRIGGER trg_users_after_insert
AFTER INSERT ON users
FOR EACH ROW
BEGIN
```

```
    add_log(:NEW.user_id, 'USER_CREATED', 'New user created: ' || :NEW.username,
NULL);
END;
/
```

Some additional details are available in the sql source file.

For instance, if we add a new user (or update), and then check the logs, we get the table updated :



Feuille de calcul | Query Builder

```
-- and then with the materialized view
SELECT * FROM mv_avg_energy_per_year ORDER BY release_year DESC;

-- Chap4
-- adding a user
INSERT INTO users (user_id, username, password, email, full_name)
VALUES (generate_uuid(), 'USERR', 'userpassword5', 'mailuser4@exemple.com', 'UserTest');

-- checking the logs
SELECT * FROM logs;
```

Sortie de script | Résultat de requête

SQL | Toutes les lignes extraites : 6 en 0,002 secondes

	LOG_ID	USER_ID	EVENT_TYPE	EVENT_DESCRIPTION	TIMESTAMP	IP_ADDRESS
1	76e82398-9915-4890-b0d3-a8ab...	55cfbe73-0065-4...	USER_CREATED	New user created: Thrind	14/12/24 19:08:19,028000000	(null)
2	0ceff7cb-e9f2-4b7c-9a89-f87e...	6c773872-232e-4...	USER_UPDATED	User updated: user3	16/12/24 12:27:35,831000000	(null)
3	e8206ded-da6f-400d-b175-aec9...	d0c8b2fd-f920-4...	USER_CREATED	New user created: B.Curlu	16/12/24 15:18:22,176000000	(null)
4	aeb2399b-05cc-4340-828c-1edc...	d0c8b2fd-f920-4...	USER_UPDATED	User updated: B.Curlu	16/12/24 16:31:09,391000000	(null)
5	187473d5-47fe-4a2d-9ae4-5304...	d0c8b2fd-f920-4...	USER_UPDATED	User updated: B.Curlu	16/12/24 16:31:17,344000000	(null)
6	89caf151-4579-4ab6-9881-f3f8...	8dadd7aa-ed65-4...	USER_CREATED	New user created: USERR	17/12/24 19:33:55,015000000	(null)

2. Stored functions and procedures

Function to compute the total length of a playlist :

```
CREATE OR REPLACE FUNCTION get_playlist_duration(p_playlist_id VARCHAR2)
RETURN NUMBER
IS
  v_total_duration NUMBER;
BEGIN
  SELECT SUM(t.duration_ms)
  INTO v_total_duration
  FROM tracks t
  JOIN playlist_tracks pt ON t.track_id = pt.track_id
  WHERE pt.playlist_id = p_playlist_id;

  RETURN NVL(v_total_duration, 0); -- returns 0 if no track is found
END;
/
```

Function to generate a monthly report on the number of added users :

```
CREATE OR REPLACE FUNCTION generate_monthly_report RETURN VARCHAR2 IS
    v_report VARCHAR2(4000);
BEGIN
```

```
    SELECT "Nombre d utilisateurs ajoutés ce mois-ci : " || COUNT(*)
    INTO v_report
    FROM users
    WHERE EXTRACT(MONTH FROM created_at) = EXTRACT(MONTH FROM SYSDATE)
      AND EXTRACT(YEAR FROM created_at) = EXTRACT(YEAR FROM SYSDATE);

    RETURN v_report;
END;
/
```

To see and check the results, we could use : SELECT generate_monthly_report() FROM DUAL;

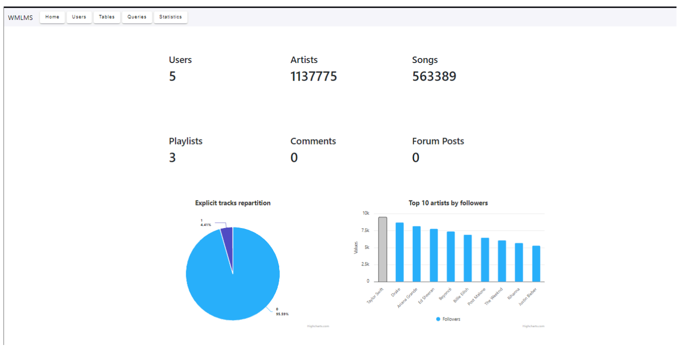
Automatic execution of tasks (PL/SQL procedure) :

```
Job configuration using the DBMS_SCHEDULER package to generate a monthly report
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name        => 'monthly_report_job',
        job_type        => 'PLSQL_BLOCK',
        job_action      => 'BEGIN generate_monthly_report; END;',
        start_date      => SYSTIMESTAMP,
        repeat_interval  => 'FREQ=MONTHLY; BYMONTHDAY=1; BYHOUR=0; BYMINUTE=0;
BYSECOND=0',
        enabled         => TRUE );
END;
/
```

This job will automatically execute the generate_monthly_report procedure at the beginning of each month, allowing for regular, automated reporting.

3. Advanced automation (bonus)

We have a dedicated page on our admin panel to display KPI and charts and gather important information automatically, it automatically refreshes and updates the displayed values by executing queries automatically. On this page, the administrator will have indicators on the status of its database like the number of registered users, the number of registered artists, the numbers of active forums, comments and songs. This statistics panel (in the administration tool) also provide two charts to display some characteristics of the dataset:



Users
5

Artists
1137775

Songs
563389

Playlists
3

Comments
0

Forum Posts
0

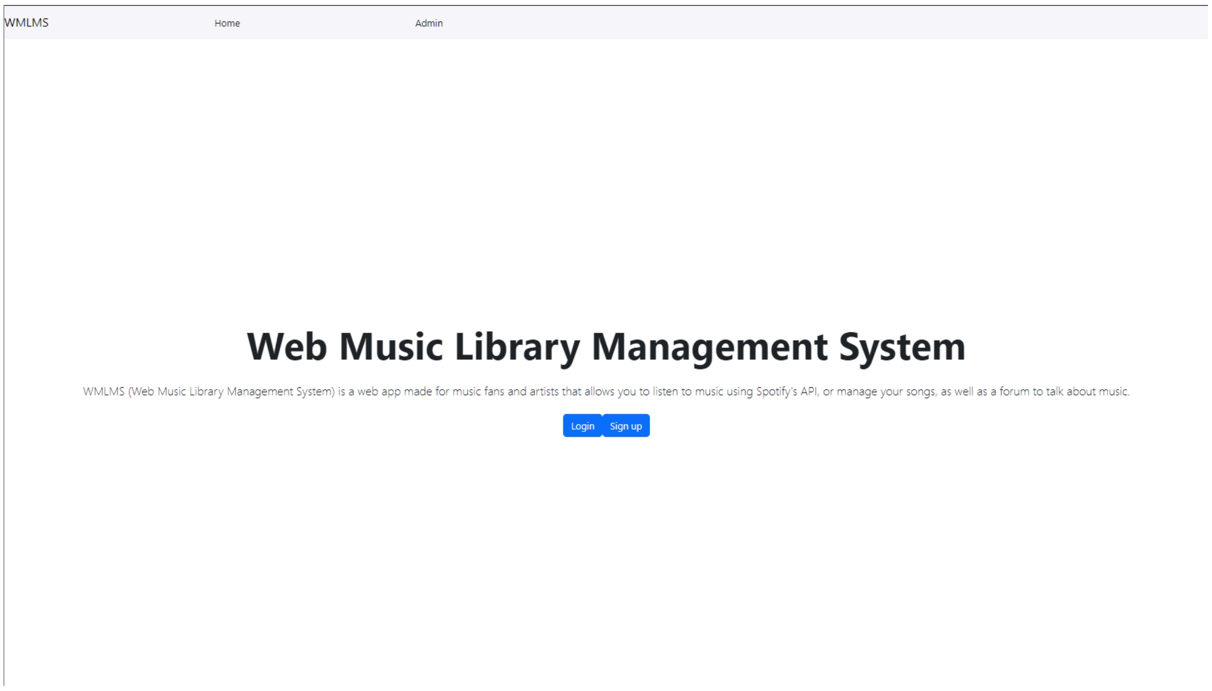
Chapter 5 (bonus) : Graphical interface

1. Creating a graphical interface

You will have access to the presentation video in the Presentation folder of the project.

Our graphical interface is built around administration tasks on the dataset such as table visualisation, user details modification, queries executions and database statistics visualisation. Here is some of the main interfaces you will encounter on this web-app (further user functions are not included in the project since it is a database administration project) :

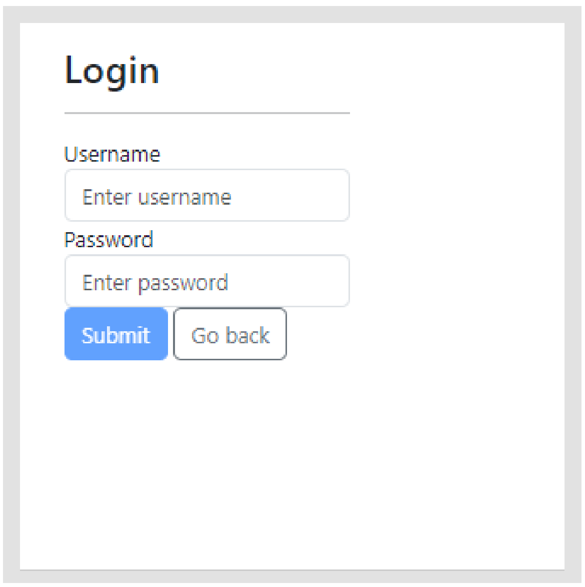
Project home page (login and registration of users) :

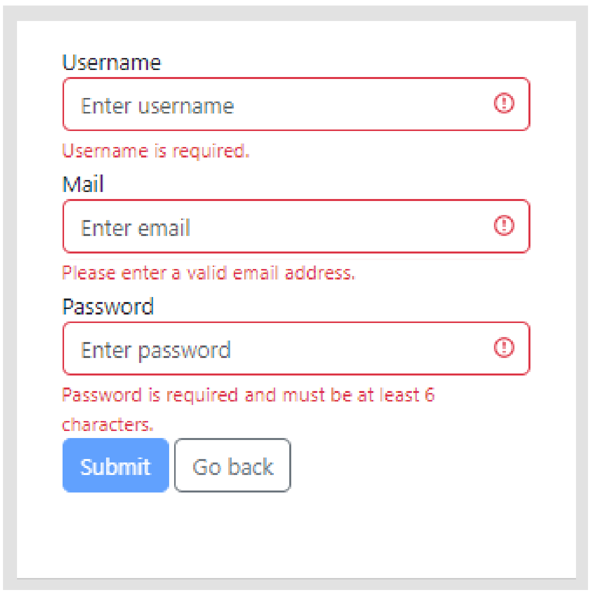


Web Music Library Management System

WMLMS (Web Music Library Management System) is a web app made for music fans and artists that allows you to listen to music using Spotify's API, or manage your songs, as well as a forum to talk about music.

User login and registration :



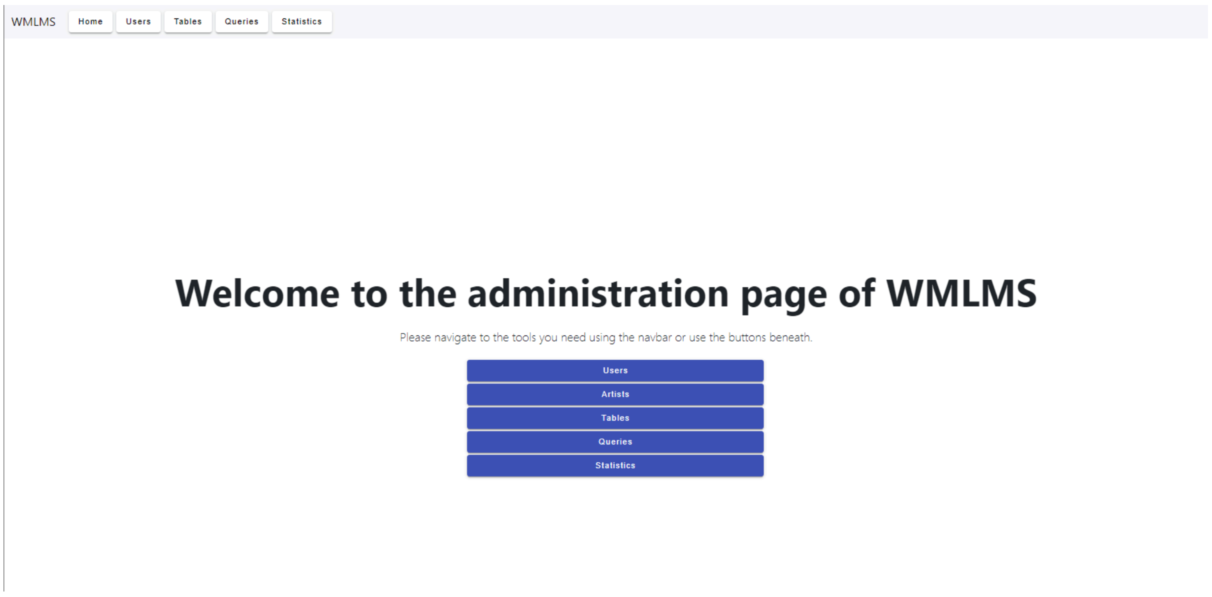


Username is required.

Please enter a valid email address.

Password is required and must be at least 6 characters.

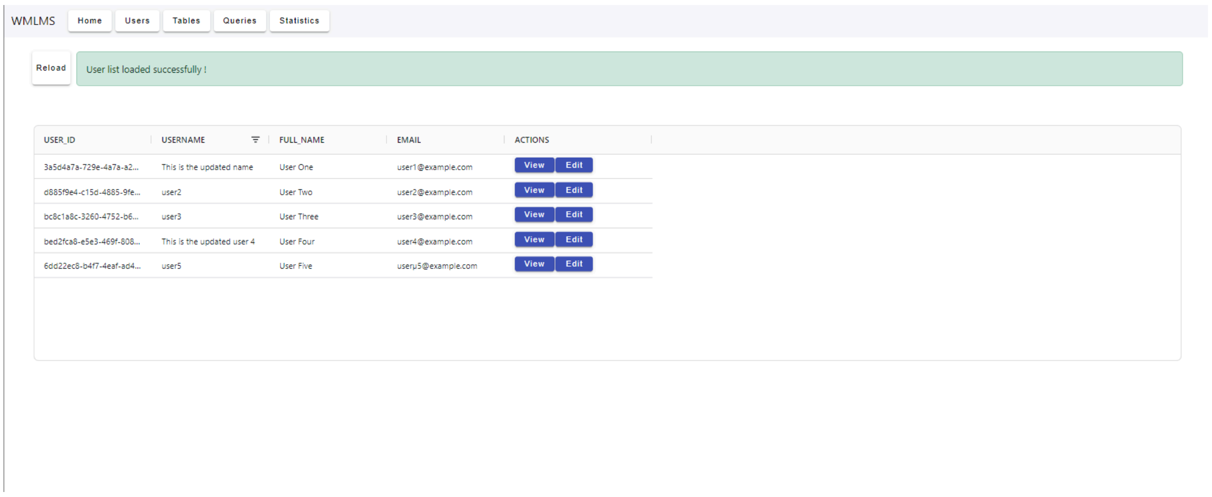
Administration panel home page (after clicking the “Admin” button in the navbar and authentication) :



Welcome to the administration page of WMLMS

Please navigate to the tools you need using the navbar or use the buttons beneath.

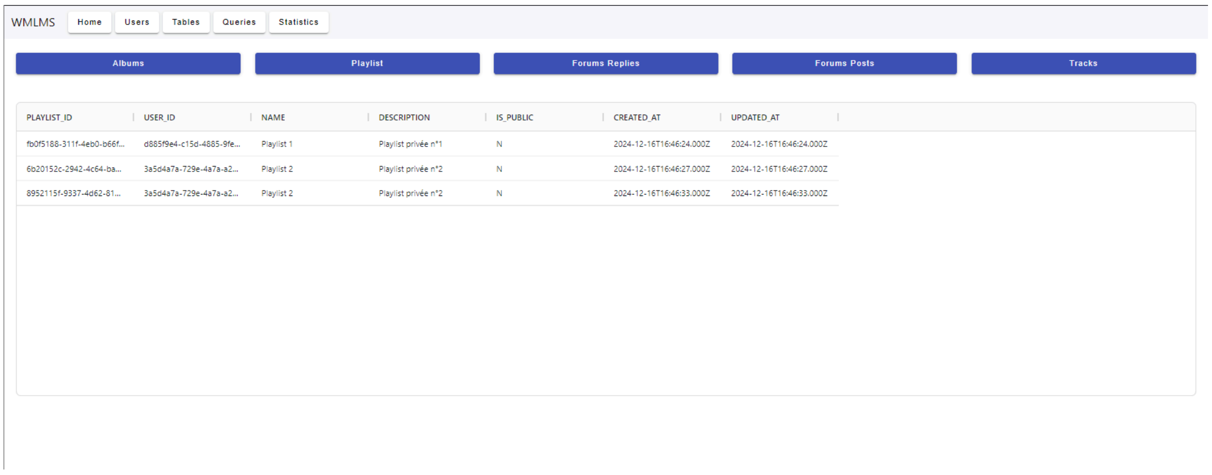
User visualisation and editing :



User list loaded successfully !

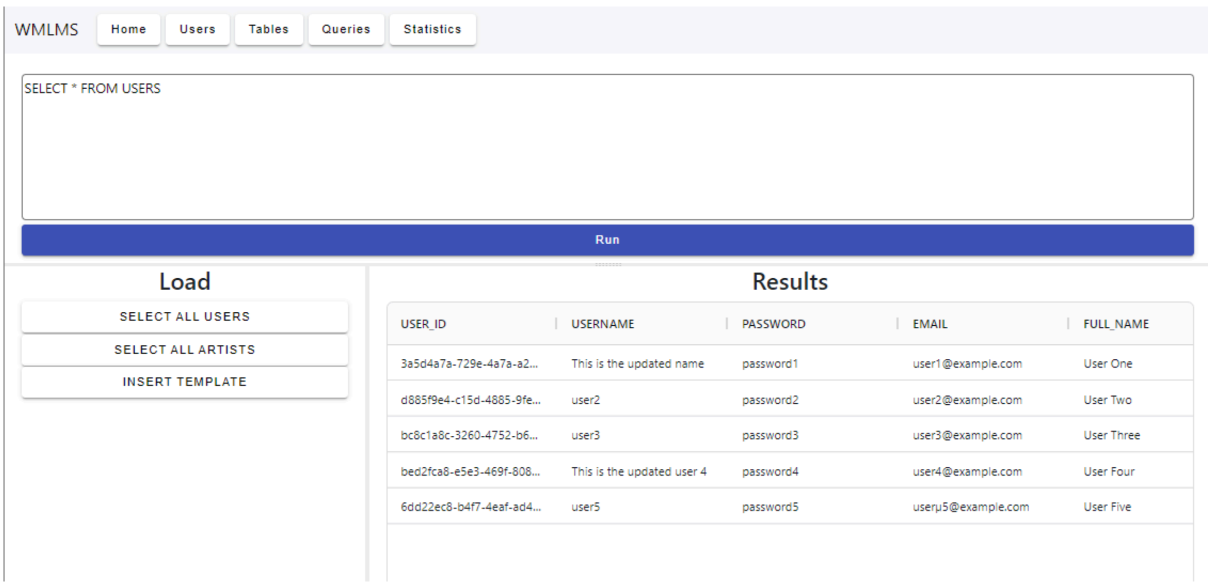
USER_ID	USERNAME	FULL_NAME	EMAIL	ACTIONS
3a5d4a7a-729e-4a7a-a2...	This is the updated name	User One	user1@example.com	View Edit
d885f9e4-c15d-4885-9fe...	user2	User Two	user2@example.com	View Edit
bc8c1a8c-3260-4752-b6...	user3	User Three	user3@example.com	View Edit
bed2fca8-e5e3-469f-808...	This is the updated user 4	User Four	user4@example.com	View Edit
6dd22ec8-b4f7-4eaf-ad4...	user5	User Five	userµ5@example.com	View Edit

Table visualisation panel :



PLAYLIST_ID	USER_ID	NAME	DESCRIPTION	IS_PUBLIC	CREATED_AT	UPDATED_AT
fb0f5188-311f-4eb0-b66f...	d885f9e4-c15d-4885-9fe...	Playlist 1	Playlist privée n°1	N	2024-12-16T16:46:24.000Z	2024-12-16T16:46:24.000Z
6b20152c-2942-4c64-ba...	3a5d4a7a-729e-4a7a-a2...	Playlist 2	Playlist privée n°2	N	2024-12-16T16:46:27.000Z	2024-12-16T16:46:27.000Z
8952115f-9337-4d62-81...	3a5d4a7a-729e-4a7a-a2...	Playlist 2	Playlist privée n°2	N	2024-12-16T16:46:33.000Z	2024-12-16T16:46:33.000Z

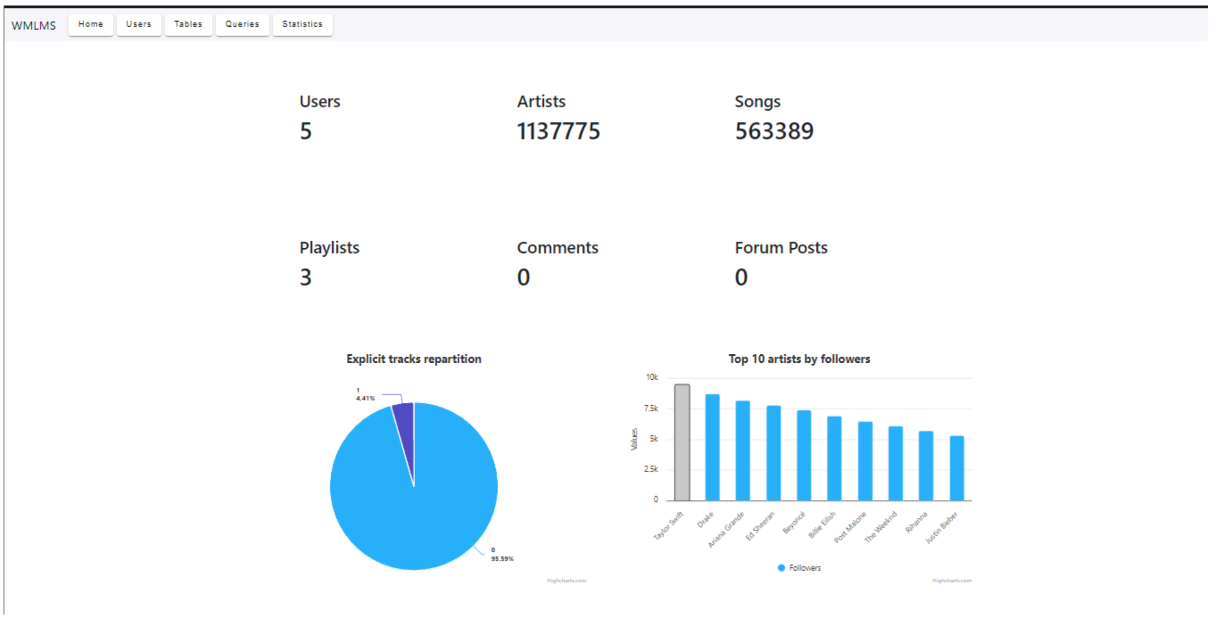
Queries panel :



```
SELECT * FROM USERS
```

USER_ID	USERNAME	PASSWORD	EMAIL	FULL_NAME
3a5d4a7a-729e-4a7a-a2...	This is the updated name	password1	user1@example.com	User One
d885f9e4-c15d-4885-9fe...	user2	password2	user2@example.com	User Two
bc8c1a8c-3260-4752-b6...	user3	password3	user3@example.com	User Three
bed2fca8-e5e3-469f-808...	This is the updated user 4	password4	user4@example.com	User Four
6dd22ec8-b4f7-4eaf-ad4...	user5	password5	userµ5@example.com	User Five

Statistics panel :



Users	Artists	Songs
5	1137775	563389
Playlists	Comments	Forum Posts
3	0	0