Зміст

[1 Вступ 3](#_Toc500864219)

[2 Опис технологій 6](#_Toc500864220)

[2.1. Machine Learning 6](#_Toc500864221)

[2.1.1. Python 6](#_Toc500864222)

[2.1.2. NumPy 6](#_Toc500864223)

[2.1.3. Pandas 7](#_Toc500864224)

[2.1.4. XGBoost 7](#_Toc500864225)

[2.2. Інтерфейс користувача 8](#_Toc500864226)

[2.3. Серверна частина 8](#_Toc500864227)

[2.3.1. Node.js 8](#_Toc500864228)

[2.3.2. MariaDB 9](#_Toc500864229)

[2.4. Хмарні технології 10](#_Toc500864230)

[3 СХЕМА БАЗИ ДАНИХ 11](#_Toc500864231)

[3.1. Загальний опис бази даних 11](#_Toc500864232)

[3.2. Опис таблиць 12](#_Toc500864233)

[3.3. Можливості бази даних 13](#_Toc500864234)

[4 Запити до бази даних 14](#_Toc500864235)

[4.1. Реєстрація користувача 14](#_Toc500864236)

[4.2. Вхід у обліковий запис 15](#_Toc500864237)

[4.3. Перевірка сесії клієнта 16](#_Toc500864238)

[4.4. Додавання пісні до бази данних 17](#_Toc500864239)

[4.5. Видалення музикальної композиці 20](#_Toc500864240)

[4.6. Реєстрація прослуховування музикальної композиці 22](#_Toc500864241)

[4.7. Вибір випадкової множини музикальниї композицій користувача 23](#_Toc500864242)

[4.8. Створення нового списку відтворювань 24](#_Toc500864243)

[4.9. Додавання нової музикальної композиції до списку відтворювань 25](#_Toc500864244)

[4.10. Видалення списку відтворення 27](#_Toc500864245)

[4.11. Видалення музикальної композиції зі списку відтворення 28](#_Toc500864246)

[4.12. Зміна позиції музикальної композиції в списку відтворення 29](#_Toc500864247)

[4.13. Вибірка всі музикальних композицій зі списку відтворення 31](#_Toc500864248)

[4.14. Вибірка всі списку відтворення користувача 32](#_Toc500864249)

[4.15. Вибірка музикальної композиції 33](#_Toc500864250)

[4.16. Вибірка списку прослуханих композицій користувачем 34](#_Toc500864251)

[4.17. Вибірка статистики прослуховувань по кожній музикальнай композиції та їх характеристики 35](#_Toc500864252)

[5 Інструкція користувача 37](#_Toc500864253)

[Висновки 40](#_Toc500864254)

[Перелік посилань 41](#_Toc500864255)

# Вступ

Відповідно до словника Webster's New World, музика – "мистецтво поєднання тонів, з метою формування виразного складу; будь-яка ритмічна послідовність приємних звуків". Однак музика набагато більше, ніж це визначення; музика в усьому навколо нас. Музику можна знайти скрізь у нашому світі. Це допомагає людям знайти себе і допомагає їм у важкі часи, з якими ми всі стикаємося в нашому житті. Музика дає нам спосіб виразити себе і показати, як ми відчуваємо себе всередині себе, що ми зазвичай не бачимо. [1]

Середньостатистичний американець витрачає 4 години на добу, слухаючи музику [2], і 93% американців щодня слухають музику в тій чи іншій мірі. Прослуховування майже половини всієї музики відбувається за допомогою AM / FM-радіо (рисунок 1.1), але частка ринку online сервісів по трансляції музики збільшується і навіть обертається зменшенням доходів класичної музикальної індустрії (рисунок 1.2).

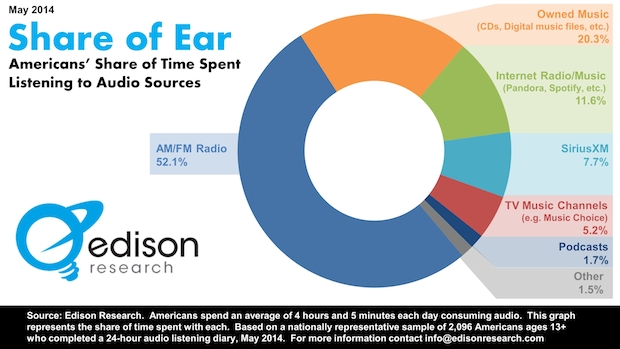


Рисунок 1.1 – Доля способів розповсюдження музики в США в травні 2014

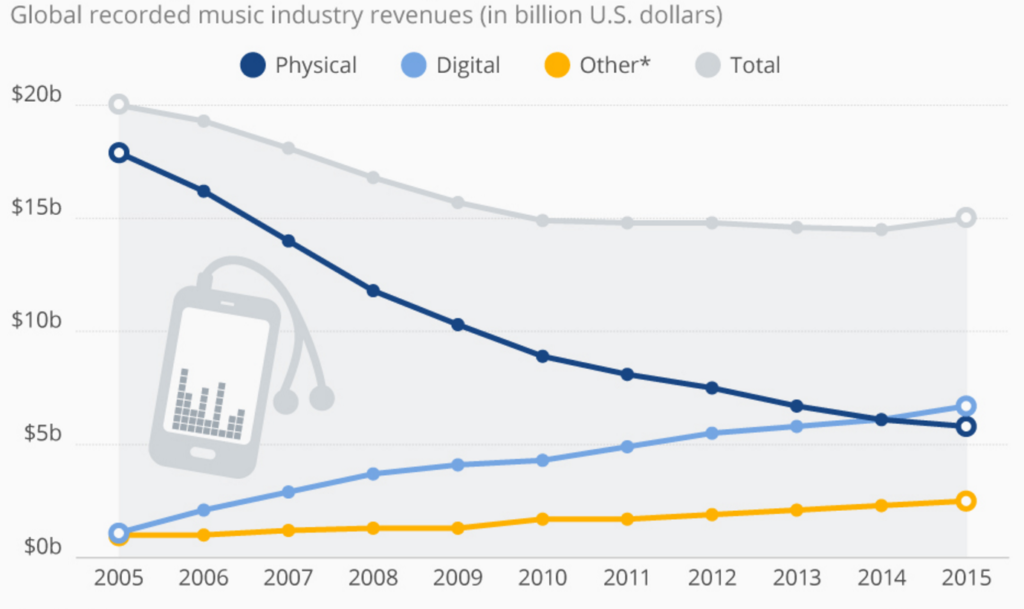


Рисунок 1.2 – Глобальна динаміка виручки музикальної індустрії

Цей успіх породжує конкуренцію серед онлайн сервісів за збільшення кількості клієнтів та запобіганню відтоку зареєстрованих користувачі. Цього вони досягають шляхом створення алгоритмів рекомендацій музики, які дають можливість користувачеві почути саме ті треки, які він бажає.

Дана курсова робота призначена для використання в області мультимедійно-розважальних інформаційних систем. Вона значно полегшить процес пошуку, організації музикальних треків користувача та надання рекомендацій на основі персональних смаків користувача.

Для вирішення даної задачі необхідно розробити зручний для користувача інтерфейс, механізми взаємодії з користувачем та систему рекомендацій. Реалізувати це все в вигляді веб-додатку з можливістю розгортання на одному з популярних хмарних сервісів.

Для реалізації рекомендаційної системи ми використаємо один із алгоритмів машиного навчання на основі градієнтного бустінгу каскаду випадкових дерев рішень з бібліотеки XGBoost, яка є стороньою бібліотекою мови програмуванн Python. Даний алгоритм ми будемо тренерувати на набірі відкрити даних зі змагання Million Song Dataset Challenge на платформі Kaggle.com, який містить інформацію про 1.000.000 треків та історію прослуховувань для 330.000 користувачів. [3]

Для інтерфейсу користувача ми використаємо звичайний набір інструментів: HMTM5/CSS3/ES6/Angular2.

Для обробки всіх запитів зі строни клієнта ми реалізуємо сервер на Node.js. Для збегіння даних користувачів та музикальних треків ми використаємо СУБД MariaDB.

# Опис технологій

## Machine Learning

Ми реалізували рекомендаційну систему в вигляді алгоритму машиного навчання. Модель рекомендацій ми підготували за допомогою мови програмування Python та стороніх бібліотек NumPy, Pandas, XGBoost. Короткий опис цих модулів наведено нижче.

### Python

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня з строгою динамічною типізацією. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python та стандартні бібліотеки доступні як у скомпільованій так і у вихідній формі на всіх основних платформах. В мові програмування Python підтримується декілька парадигм програмування, зокрема: об'єктно-орієнтована, процедурна, функціональна та аспектно-орієнтована. [4]

Переваги: простий та зрозумілий синтаксис, форматування коду – частина синтаксису, швидка розробка будь-яких додатків.

Недоліки: повільне виконання програм.

### NumPy

Оскільки Python — інтерпретована мова, математичні алгоритми, часто працюють в ньому набагато повільніше ніж у компільованих мовах, таких як C або навіть Java. NumPy намагається вирішити цю проблему для великої кількості обчислювальних алгоритмів забезпечуючи підтримку багатовимірних масивів і безліч функцій і операторів для роботи з ними. Таким чином будь-який алгоритм який може бути виражений в основному як послідовність операцій над масивами і матрицями працює в Python настільки швидко як еквівалентний код написаний на C, але зі значно зрозумілішим синтаксисом та меншим об’ємом написаного коду. [5]

Переваги: швидкість робити, простота маніпуляцій зі структурованими даними.

Недоліки: відсутні.

### Pandas

Pandas — програмна бібліотека, написана для мови програмування Python для маніпулювання даними та їхнього аналізу. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами. Pandas є вільним програмним забезпеченням, що випускається за трипунктовою ліцензією BSD. [6]

Переваги: швидкість робити, простота загрузки даних з файлів, підтрима багатьох аналітичних функцій, деяких класичних команд SQL (join, union, intersect, pivot, unpivot, groupby) та побудови графіків, які ілюструють структуру даних.

Недоліки: для більшості операцій створює копію даних, що значно сповільнює процес обробки великих наборів даних.

### XGBoost

XGBoost - це бібліотека програмного забезпечення з відкритим кодом, яка забезпечує підтримку алгоритму градієнтного бустінга для C ++, Java, Python, R та Julia. Вона працює на Linux, Windows та macOS. Метою бібліотеки є забезпечити "Масштабовану портативну та розподілену градієнтну підтримку" (GBM, GBRT, GBDT) ". Крім роботи на одній машині, бібліотека також підтримує розподілені структури обробки Apache Hadoop, Apache Spark і Apache Flink. Вона отримала значну популярність і увагу нещодавно, оскільки цей алгоритм дозволив виграти багато змаганнях з машинного навчання. [7]

Переваги: простота навчання моделей, хороша масштабованість процесу навчання, автоматична преобробна вхідних даних, контроль за відповідність даних під час навчання та під час використаня.

Недоліки: Потребує більше машиного часу та памяті для навчання моділі чим інші відкриті бібліотеки для машиного навчання (LigthGBM, Sklearn).

## Інтерфейс користувача

Для інтерфейсу користувача ми використали звичайний набір інструментів: HMTM5/CSS3/ES6/Angular2.

Переваги: значна поширеність, простота розширення функціоналу, підтримка більшістю сучасних веб-браузерів, в

Недоліки: швидке устарівання технологій

## Серверна частина

Для обробки всіх запитів зі строни клієнта ми реалізуємо сервер на Node.js. Для збегіння даних користувачів та музикальних треків ми використаємо СУБД MariaDB.

### Node.js

Node.js призначений для відокремленого виконання високопродуктивних мережних застосунків на мові JavaScript. Функції платформи не обмежені створенням серверних скриптів для веб, платформа може використовуватися і для створення звичайних клієнтських і серверних мережевих програм. Для забезпечення виконання JavaScript-коду використовується розроблений компанією Google рушій V8.

Для забезпечення обробки великої кількості паралельних запитів у Node.js використовується асинхронна модель запуску коду, заснована на обробці подій в неблокуючому режимі та визначенні обробників зворотніх викликів (callback). Як способи мультиплексування з'єднань підтримується epoll, kqueue, /dev/poll і select. Для мультиплексування з'єднань використовується бібліотека libuv, для створення пулу потоків (thread pool) задіяна бібліотека libeio, для виконання DNS-запитів у неблокуючому режимі інтегрований c-ares. Всі системні виклики, що спричиняють блокування, виконуються всередині пула потоків і потім, як і обробники сигналів, передають результат своєї роботи назад через неіменовані канали (pipe).

За своєю суттю Node.js схожий на фреймворки Perl AnyEvent, Ruby Event Machine і Python Twisted, але цикл обробки подій (event loop) у Node.js прихований від розробника і нагадує обробку подій у веб-застосунку, що працює в браузері. [8]

Переваги: асинхронна однопотокова модель виконання запитів, неблокуючий ввід/вивід, система модулів CommonJS, рушій JavaScript Google V8.

Недоліки: необхідність «розігрівання» коду.

### MariaDB

MariaDB — реляційна система керування базами даних, створена на початку 2009 як відгалуження (форк) MySQL. MariaDB поширюється під вільною та відкритою ліцензією GNU GPL.

Maria — засноване на MyISAM високонадійне сховище, що відрізняються підвищеною стійкістю і збереженню цілісності даних після краху, при повній сумісності з MyISAM. Завдяки веденню логу операцій, у разі краху проводиться відкат результатів виконання поточної операції. Також підтримується можливість відновлення стану з будь-якої точки в балці операцій (включаючи підтримку CREATE / DROP / RENAME / TRUNCATE).

На заміну InnoDB використовується рушій XtraDB, заснований на коді InnoDB-plugin і повністю сумісний з ним, але відрізняється помітно вищою продуктивністю, завдяки інтеграції патчів від компаній Google і Percona. Зокрема, в XtraDB поліпшений механізм роботи з пам'яттю, покращена робота підсистеми введення / виведення InnoDB, додана підтримка декількох нитей читання і запису, підтримка управління пропускною здатністю, реалізація упереджувальною вибіркою даних (read-ahead), адаптивна установка контрольних точок (adaptive checkpointing), розширені можливості з масштабування для великих проектів, система організації блокувань адаптована для роботи на системах з великою кількістю CPU, додані додаткові можливості для накопичення та аналізу статистики. [9]

Переваги: безкоштовна, зворотня сумісність з поширеною СУБД MySQL.

Недоліки: відсутність більшості продвинутих аналітичних функцій, які підтримують інші СУБД.

## Хмарні технології

"Хмара" не є концепцією революційною, що виникла в один момент, але є концепцією еволюційною, виниклою з ідей і технологій, коли власники великих мейн-фреймів прагнули оптимізувати завантаження цих потужностей таким чином, щоб отримати від цього максимум ефективністі і прибутку. Схильність до оптимізації призвела до появи ідеї віддаленого доступу на тимчасовій основі.

Digital Ocean - це платформа, що дозволяє оперативно створювати, впроваджувати, масштабувати веб-сервери та керувати ними. [10]

Ключові можливості Digital Ocean:

* Створення нового сервера відбувається швидше ніж за 55 секунд
* Всі веб-сервери обладнані твердотільними накопичувачами
* Можливість динамічного збільшення розміру накопичувача даних   
  до 16 ТБ
* Швидкісний доступ до серверів
* Автобалансувальник загрузок як сервіс
* Моніторінг та раннє попередження проблем
* Захист від DDos атак
* Підтримка стороннього програмного забезпечення по розгортаню сервісів (Kubernetis, Docker,

# СХЕМА БАЗИ ДАНИХ

## Загальний опис бази даних

Для створення бази даних програмного забезпечення була обрана СУБД MariaDB, оскільки являється вільною та кросс-платформенною.

База даних була створена з метою зберігання інформації про користувачів, треки, списки відтворення та метаданних необхідних для роботи алгоритму рекомендацій.

Cхема бази даних наведена на рисунку 3.1.

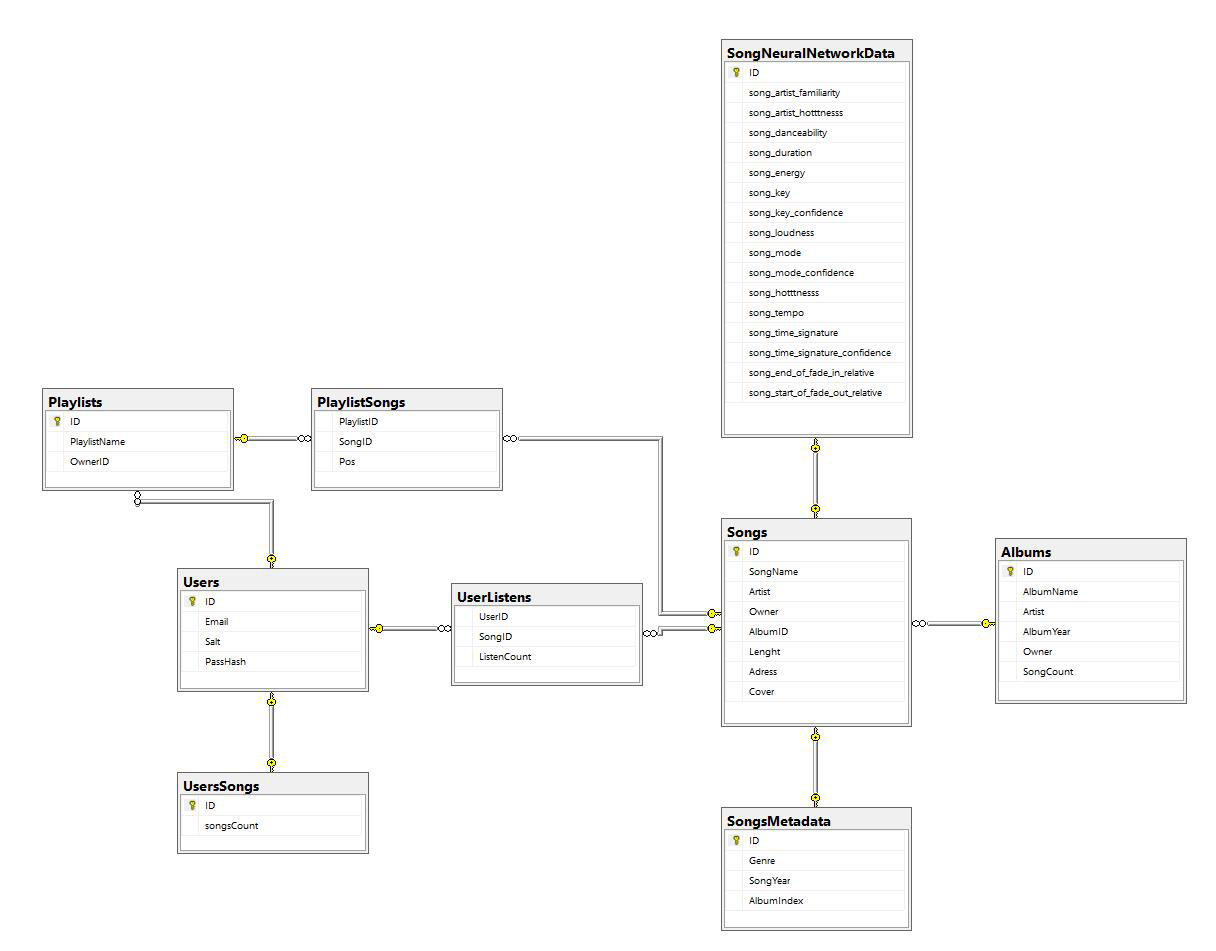


Рисунок 3.1 – Схема бази даних

## Опис таблиць

* Users – містить дані про користувача. Поля таблиці: унікальний індентифікатор, електронна пошта, данні для авторизації;
* UsersSongs – містить дані про кількість завантажених користувачем треків. Поля таблиці: унікальний ідентифікатор користувача, унікальний ідентифікатор музикального треку, кількість загузок даним користувачем даного треку;
* Playlists – містить данні про списки відтворення. Посля таблиці: унікальний ідентифікатор музикального списку відтворення, унікальний ідентифікатор власника даного списку відтворення, назву даного списку відтворення;
* PlaylistSongs – містить данні про пісні списків відтворень та їх порядок. Поля таблиці: унікальний ідентифікатор списку відтворення, унікальний індетифікатор музикального треку, позиці даного музикального треку в даному списку відтворення.
* UserListens – містить данні про пісні прослухані користувачами. Поля таблиці: унікальний ідентифікатор користувача, унікальний ідентифікатор користува, кількість прослуховувань даного треку даним користувачем.
* Songs – містить базові данні про усі треки . Поля таблиці: унікальний ідентифікатор треку, назва треку, назва виконавця, унікальний ідентифікатор користувача, який володіє даним трекуом, унікальний ідентифікатор альбому, довжина треку, шлях в файловій системі до даного треку, шлях в файловій системі до обложки альбому для даного треку.
* SongsMetadata – містить данні про треки що необов’язкові для відображення треків. Поля таблиці: жанр треку, рік виходу альбому, індекс альбому.
* SongNeuralNetworkData – містить данні необхідні для роботи рекомендаційної системи.
* Albums – містить данні про альбоми треків що знаходяться у базі даних. Поля таблиці: унікальний ідентифікатор альбому, назва альбому, рік виходу альбому, власник альбому, кількість музикальних треків, які включає даний альбом.

## Можливості бази даних

Створена база даних реалізує наступні можливості для програмного забезпечення:

• Зберігання та модифікація повної інформації про користувачів

• Зберігання та модифікація повної інформації про музикальні композиції

• Зберігання та модифікація повної інформації про списки відтворення

• Зберігання та модифікація повних метаданих необхідних для роботи рекомендаційної системи

• Обробка та перевірка інформації про сесії користувачів

• Межування доступу до музикальних композицій та списків відтворення різних користувачів

# Запити до бази даних

## Реєстрація користувача

Функція реєструє нового користувача системи та повертає статус оперції (успішна чи ні).

Запит:

|  |
| --- |
|  |
|  | CREATE FUNCTION regUser( |
|  | inEmail VARCHAR(50), |
|  | inPass VARCHAR(50) |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF EXISTS |
|  | ( |
|  | SELECT \* FROM Users WHERE Users.Email = inEmail |
|  | ) |
|  | THEN |
|  | RETURN 0; |
|  | ELSE |
|  | SET @Rnd = FLOOR(RAND()\*2147483647); -- Generate random non negative int |
|  | INSERT INTO Users(Users.Email,Users.Salt,Users.PassHash) Values |
|  | (inEmail, |
|  | @Rnd, |
|  | binary( SHA2 ( CONCAT ( CONVERT ( @Rnd , char(11) ) ,inPass ),256))); |
|  | INSERT INTO UsersSongs(ID, songsCount) VALUES (LAST\_INSERT\_ID(), 0); |
|  | RETURN 1; |
|  | END IF; |
|  | END |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.1.

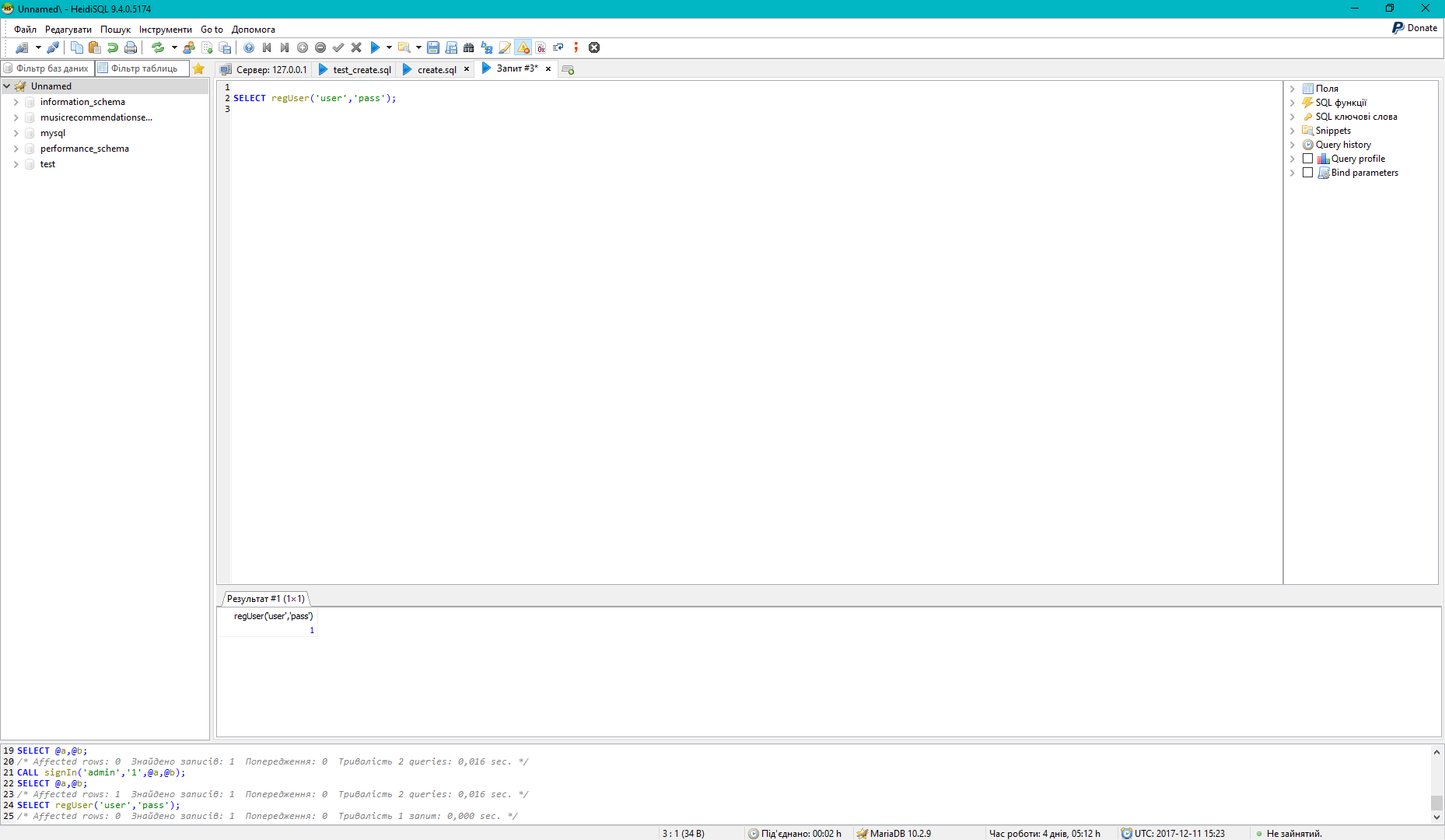


Рисунок 4.1 – Результат виконання запиту по реєстрації користувача

## Вхід у обліковий запис

Функція реєструє вхід користувача системи та повертає данні про сессію або null.

Запит:

|  |
| --- |
|  |
|  | CREATE PROCEDURE signIn( |
|  | IN Email VARCHAR(50), |
|  | IN Pass VARCHAR(50), |
|  | OUT SessionExpire DATETIME, |
|  | OUT SessionToken binary(64) |
|  | ) |
|  | BEGIN |
|  | IF EXISTS |
|  | ( |
|  | SELECT 1 FROM Users |
|  | WHERE Users.Email = Email |
|  | AND Users.PassHash = |
|  | ( |
|  | @Hash:=binary( SHA2 ( CONCAT ( CONVERT ( Users.Salt , char(11) ) ,Pass ),0)) |
|  | ) |
|  | ) |
|  | THEN |
|  | SELECT ADDTIME(NOW(), '14 00:00:00') INTO SessionExpire; |
|  | SELECT SHA2 (CONCAT(SessionExpire,@Hash),0) INTO SessionToken; |
|  | END IF; |
|  | END |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.2.

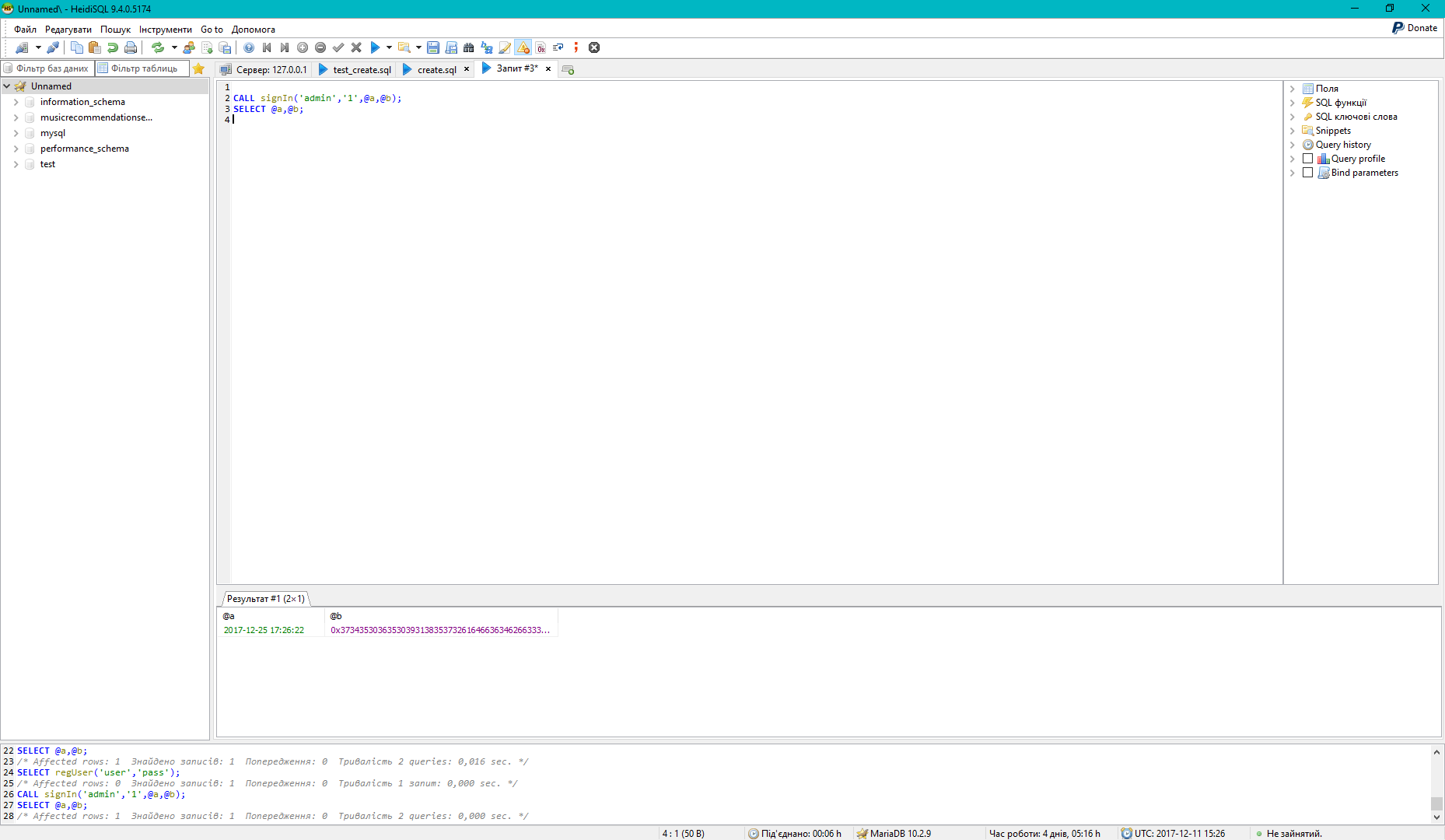


Рисунок 4.2 – Результат виконання запиту по входу в обліковий запис

## Перевірка сесії клієнта

Функція превіряє стан сесії користувача системи та повертає 1 якщо сесія дійсна, або 0 якщо сесія закінчилась чи якщо данні неправильні.

Запит:

|  |
| --- |
|  |
|  | CREATE FUNCTION checkSession( |
|  | Email VARCHAR(50), |
|  | SessionExpire DATETIME, |
|  | SessionToken binary(64) |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF ((SELECT 1 |
|  | FROM Users |
|  | WHERE Users.Email=Email AND |
|  | SHA2(CONCAT(SessionExpire,Users.PassHash),0)=SessionToken) |
|  | AND |
|  | SessionExpire >= NOW() |
|  | ) |
|  | THEN |
|  | RETURN 1; |
|  | ELSE |
|  | RETURN 0; |
|  | END IF; |
|  | END |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.3.

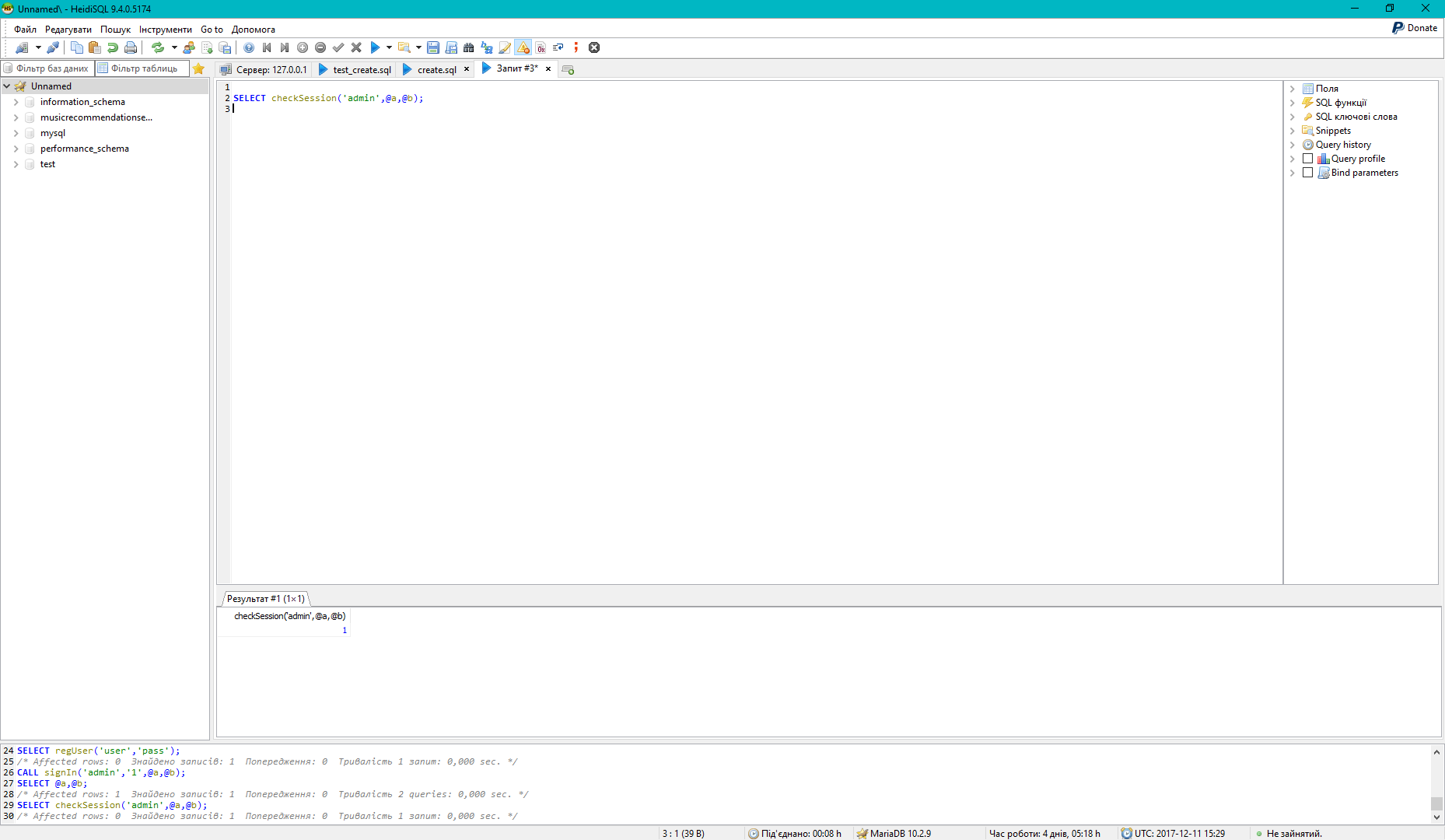


Рисунок 4.3 – Результат виконання запиту по провірці сесії клієнта

## Додавання пісні до бази данних

Функція виконує операцію додавння інформації по музикальнії композиції до бази даних та повертає статус операції (успішна чи ні).

Запит:

|  |
| --- |
|  |
|  | CREATE FUNCTION insertSongWithNeuralData( |
|  | in\_SongName varchar(50), |
|  | in\_Artist varchar(50), |
|  | in\_Owner int, |
|  | in\_Length int, |
|  | in\_Genre varchar(50), |
|  | in\_Year int, |
|  | -- ignored for album if album with specified name and artist exists |
|  | in\_AlbumIndex int, |
|  | in\_AlbumName varchar(50), |
|  | in\_Adress varchar(250), |
|  | in\_Cover varchar(250), |
|  | in\_song\_artist\_familiarity float, |
|  | in\_song\_artist\_hotttnesss float, |
|  | in\_song\_duration float, |
|  | in\_song\_key int, |
|  | in\_song\_key\_confidence float, |
|  | in\_song\_loudness float, |
|  | in\_song\_hotttnesss float, |
|  | in\_song\_tempo float, |
|  | in\_song\_end\_of\_fade\_in\_relative float, |
|  | in\_song\_start\_of\_fade\_out\_relative float |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF EXISTS |
|  | ( |
|  | SELECT songsCount FROM UsersSongs |
|  | WHERE UsersSongs.ID = in\_Owner AND songsCount >= 100 |
|  | ) |
|  | OR ISNULL(in\_SongName) |
|  | OR ISNULL(in\_Length) |
|  | -- check if song limit reached, |
|  | -- won't affect generally availbale songs because NULL != NULL |
|  | -- also do other checks |
|  | THEN |
|  | RETURN 0; |
|  | END IF; |
|  | UPDATE UsersSongs SET songsCount = songsCount + 1 |
|  | WHERE UsersSongs.ID = in\_Owner; |
|  |  |
|  | -- user added songs won't be added to generally available albums |
|  | -- new album will be created instead |
|  | SET @AlbumID = |
|  | ( |
|  | SELECT ID FROM Albums |
|  | WHERE STRCMP(Albums.Artist, in\_Artist) = 0 |
|  | AND STRCMP(Albums.AlbumName, in\_AlbumName) = 0 |
|  | AND ( |
|  | Albums.Owner = in\_Owner |
|  | OR ISNULL(Albums.Owner) = ISNULL(in\_Owner) |
|  | ) |
|  | ); |
|  | -- if artist or album name not specified album won't be created |
|  | IF NOT ISNULL(in\_AlbumName) AND NOT ISNULL(in\_Artist) AND ISNULL(@AlbumID) |
|  |  |
|  | THEN INSERT INTO |
|  | Albums(Albums.AlbumName, Albums.Artist, Albums.AlbumYear, Albums.Owner) |
|  | VALUES (in\_AlbumName, in\_Artist, in\_Year, in\_Owner); |
|  |  |
|  | SET @AlbumID = LAST\_INSERT\_ID(); |
|  | END IF; |
|  |  |
|  | UPDATE Albums SET SongCount = SongCount + 1 WHERE Albums.ID = @AlbumID; |
|  |  |
|  | INSERT INTO |
|  | Songs(Songs.SongName,Songs.Artist,Songs.Owner,Songs.AlbumID, |
|  | Songs.Length,Songs.Adress,Songs.Cover) |
|  | Values |
|  | (in\_SongName,in\_Artist,in\_Owner,@AlbumID,in\_Length,in\_Adress,in\_Cover); |
|  |  |
|  | INSERT INTO |
|  | SongsMetadata(SongsMetadata.ID, SongsMetadata.Genre, SongsMetadata.SongYear, |
|  | SongsMetadata.AlbumIndex) |
|  | VALUES |
|  | (LAST\_INSERT\_ID(), in\_Genre, in\_Year, in\_AlbumIndex); |
|  |  |
|  | INSERT INTO SongNeuralNetworkData VALUES |
|  | (LAST\_INSERT\_ID(), in\_song\_artist\_familiarity, in\_song\_artist\_hotttnesss, |
|  | in\_song\_duration, in\_song\_key, in\_song\_key\_confidence, in\_song\_loudness, |
|  | in\_song\_hotttnesss, in\_song\_tempo, in\_song\_end\_of\_fade\_in\_relative, |
|  | in\_song\_start\_of\_fade\_out\_relative); |
|  |  |
|  | RETURN 1; |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.4.

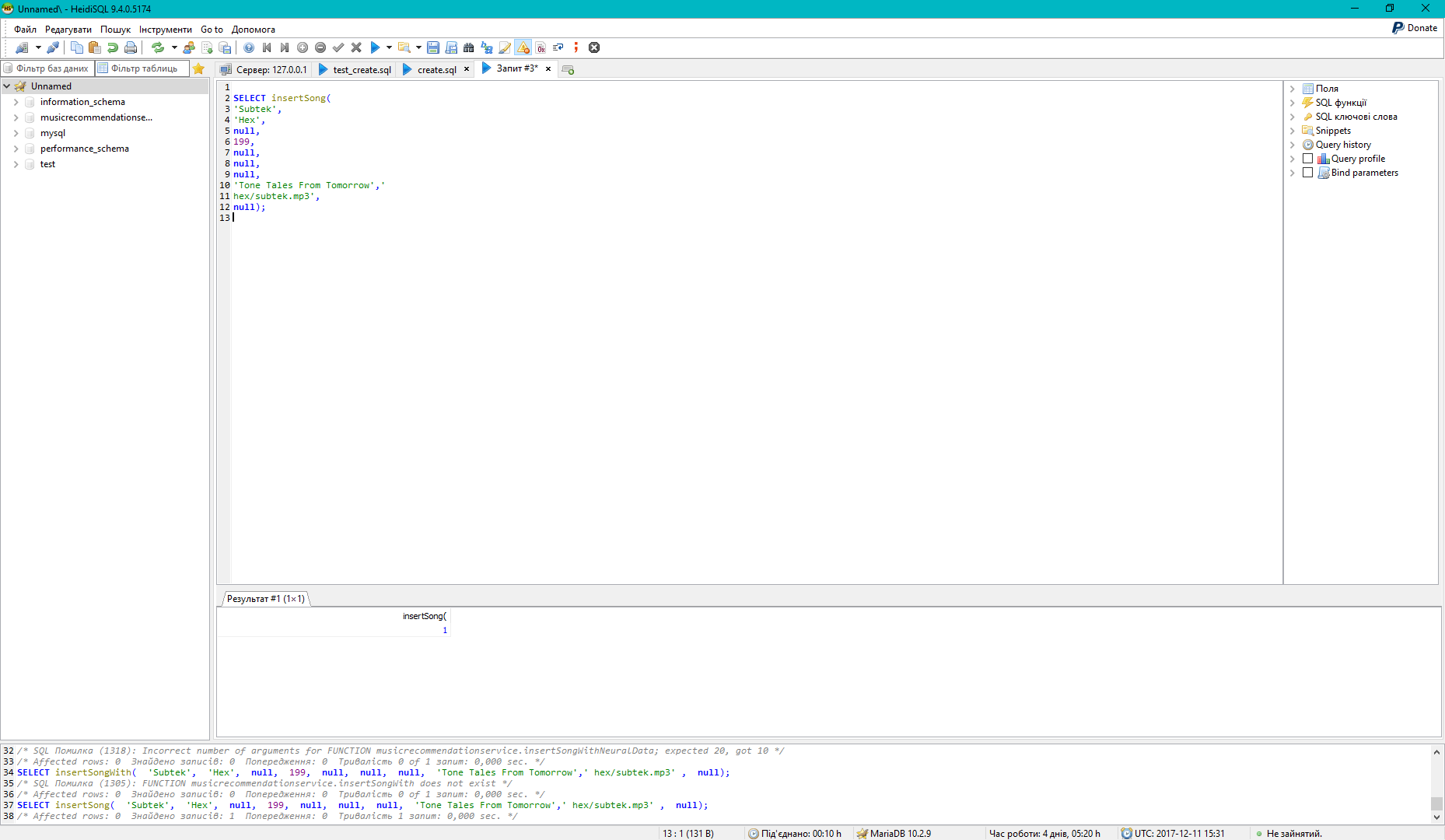


Рисунок 4.4 – Результат виконання запиту по доданю музикального треку в базу даних

## Видалення музикальної композиці

Функція виконує перевірку хто завантажив музикальну композицію та видаляє її з усієї бази данних якщо її завантажив поточний користувачта та повертає статус операції (успішна чи ні).

Запит:

|  |
| --- |
|  |
| |  | | --- | |  | |  | CREATE FUNCTION deleteSong( | |  | InSongID int, | |  | InOwnerID int | |  | ) | |  | RETURNS BOOL | |  | BEGIN | |  | IF NOT EXISTS( | |  | SELECT \* | |  | FROM Songs | |  | WHERE Songs.ID = InSongID AND | |  | Songs.Owner = InOwnerID | |  | ) | |  | THEN | |  | RETURN 0; | |  | END IF; | |  |  | |  | UPDATE UsersSongs SET songsCount = songsCount - 1 | |  | WHERE UsersSongs.ID = (SELECT Owner FROM Songs WHERE Songs.ID = InSongID); | |  |  | |  | SET @InAlbumID = (SELECT AlbumID FROM Songs WHERE ID = InSongID); | |  |  | |  | UPDATE Albums SET SongCount = SongCount - 1 | |  | WHERE Albums.ID = @InAlbumID; | |  |  | |  | DELETE FROM UserListens WHERE SongID = InSongID; | |  | DELETE FROM PlaylistSongs WHERE SongID = InSongID; | |  | DELETE FROM SongsMetadata WHERE ID = InSongID; | |  | DELETE FROM SongNeuralNetworkData WHERE ID = InSongID; | |  | DELETE FROM Songs WHERE ID = InSongID; | |  | DELETE FROM Albums WHERE ID = @InAlbumID AND SongCount = 0; | |  | RETURN 1; | |  | END; | |  | // | |  | DELIMITER ; | |

Результат виконання запиту наведений на рисунку 4.5.

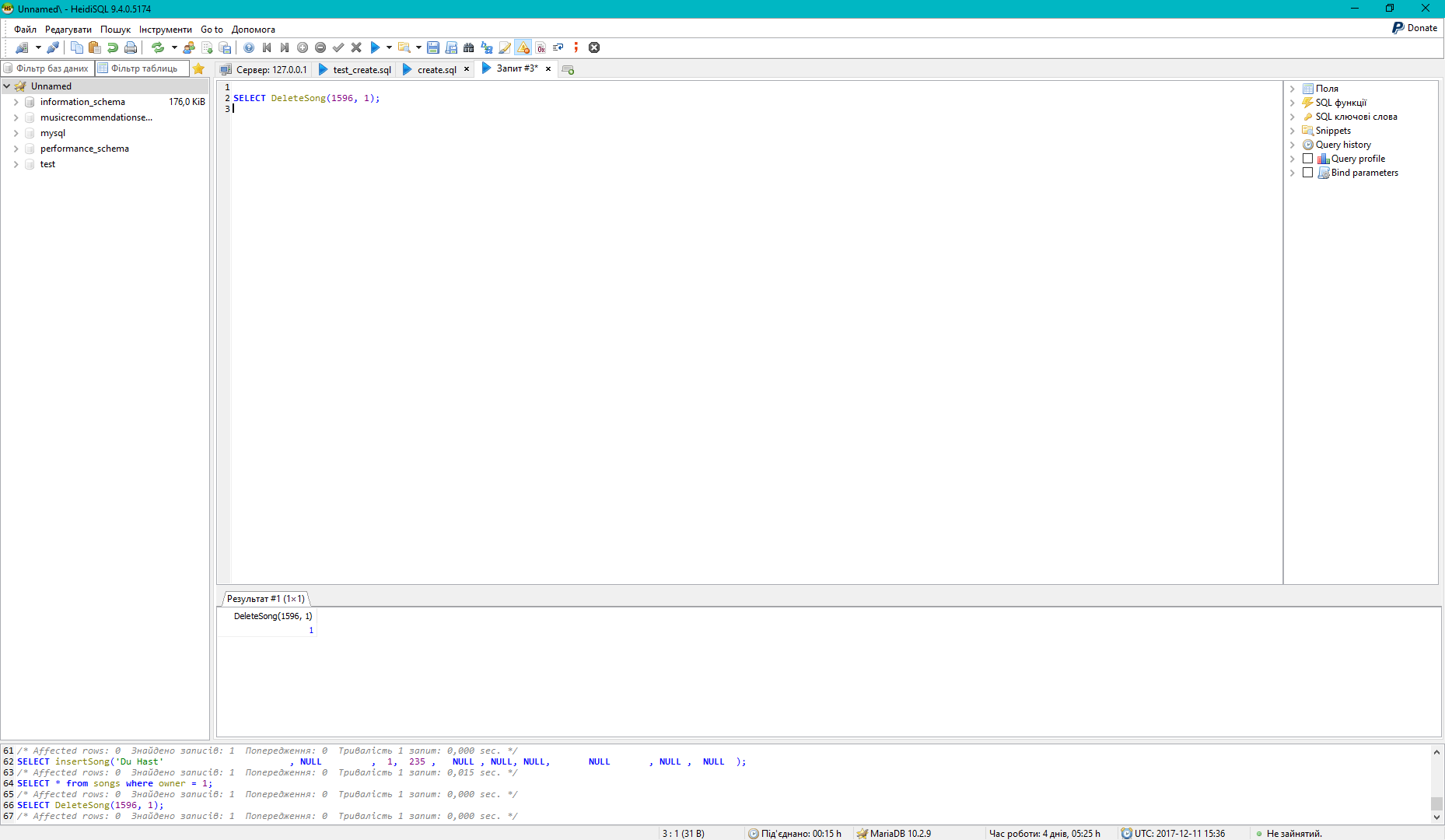


Рисунок 4.5 – Результат виконання запиту по доданю музикального треку в базу даних

## Реєстрація прослуховування музикальної композиці

Функція фіксує факст прослуховування поточної композиції корпистувачем. Потрібна для подальших рекомендацій. Нічого не повертає.

Запит:

|  |  |
| --- | --- |
|  | CREATE PROCEDURE registerListen( |
|  | IN InUserID int, |
|  | IN InSongID int |
|  | ) |
|  | BEGIN |
|  | IF EXISTS( |
|  | SELECT \* FROM UserListens |
|  | WHERE |
|  | UserID = InUserID AND |
|  | SongID = InSongID |
|  | ) |
|  | THEN |
|  | UPDATE UserListens |
|  | SET ListenCount = ListenCount + 1 |
|  | WHERE |
|  | UserID = InUserID AND |
|  | SongID = InSongID; |
|  | ELSE |
|  | INSERT INTO UserListens(UserID, SongID, ListenCount) VALUES |
|  | (InUserID,InSongID,1); |
|  | END IF; |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.6.

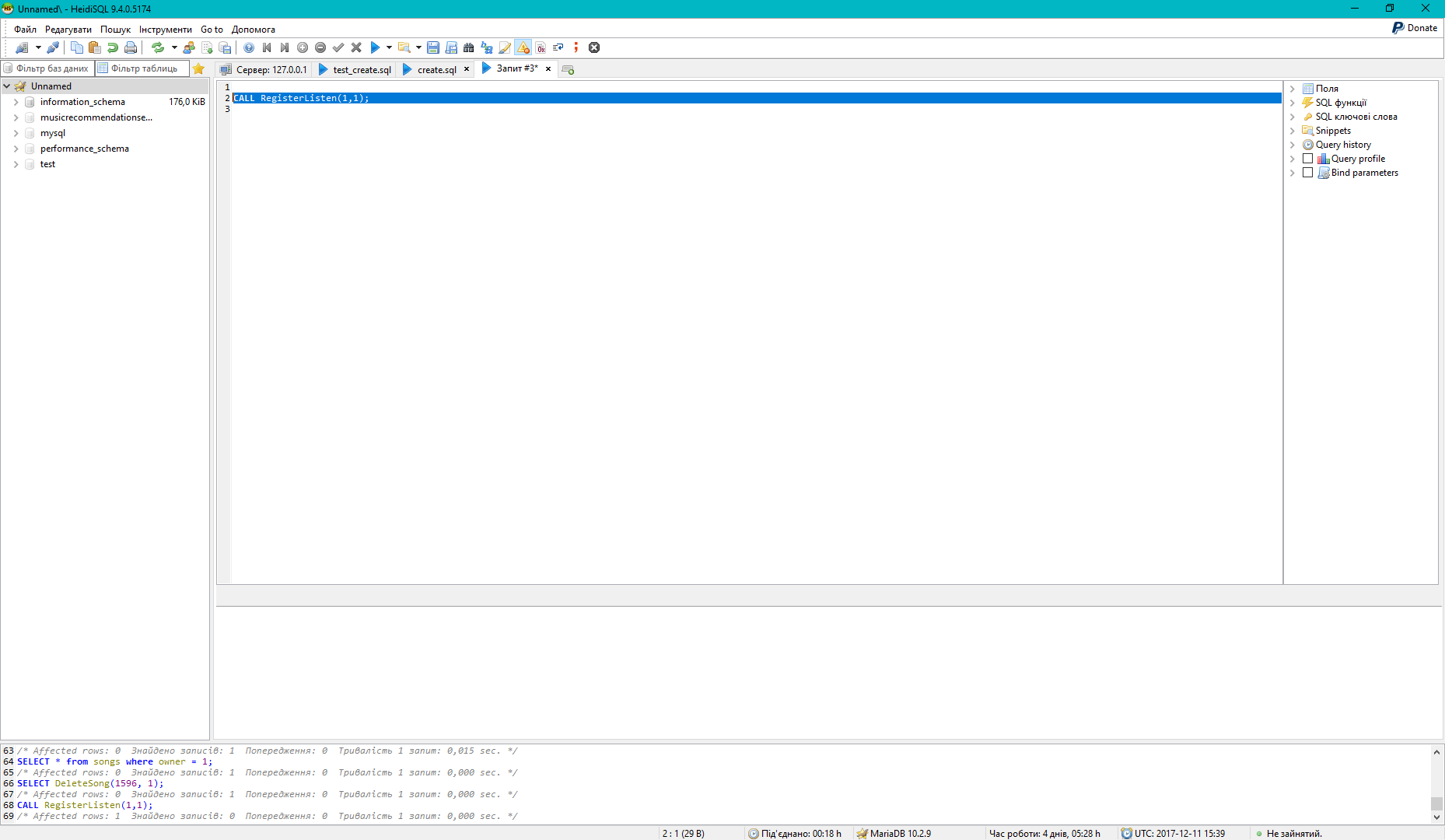


Рисунок 4.6 – Результат виконання реєстрації прослуховування музикальної композиції

## Вибір випадкової множини музикальниї композицій користувача

Функція повертає певну кількість випадкових музикальних композицій серед загальнодоступних та особистих музикальних композицій користувача.

Запит:

|  |
| --- |
|  |
|  | CREATE PROCEDURE getRandomSongs( |
|  | IN InCount int, |
|  | IN InOwnerID int |
|  | ) |
|  | BEGIN |
|  | SELECT ID, SongName, Artist, AlbumID, Length, CONCAT('http://138.68.69.43/music/',Adress) AS Adress , Cover |
|  | FROM Songs |
|  | WHERE Owner IS NULL OR Owner = InOwnerID |
|  | ORDER BY RAND() |
|  | LIMIT InCount; |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.7.

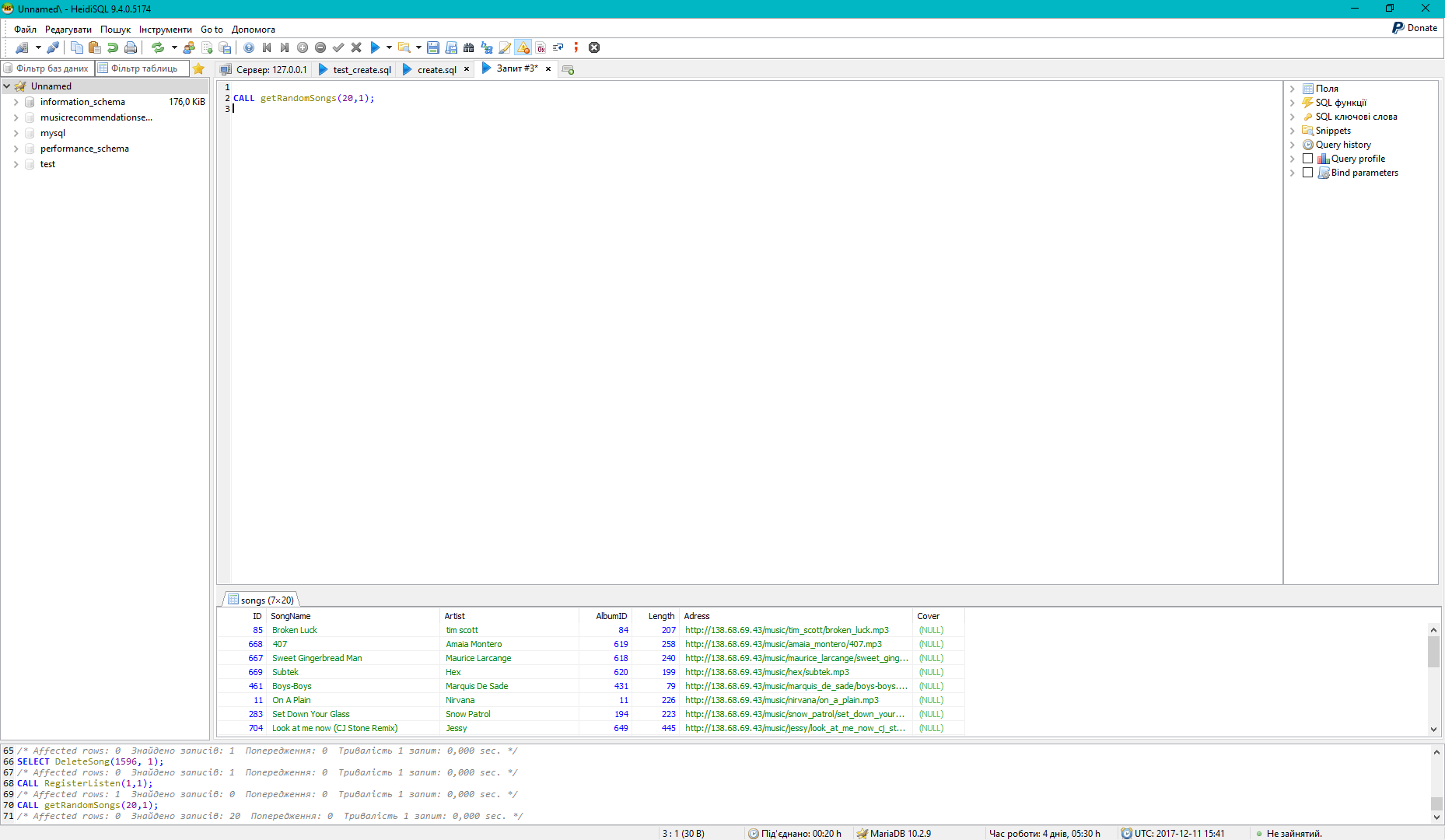


Рисунок 4.7 – Результат виконання генерації випадкової множини музикальних композицій

## Створення нового списку відтворювань

Функція створює список відтворення для поточного користувача та повертає унікальний індентифікатор нового списку відтворення.

Запит:

|  |  |
| --- | --- |
|  | CREATE FUNCTION createPlaylist( |
|  | InPlaylistName varchar(150), |
|  | InOwnerID int |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | INSERT INTO Playlists(PlaylistName,OwnerID) |
|  | VALUES (InPlaylistName,InOwnerID); |
|  | RETURN LAST\_INSERT\_ID(); |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.8.

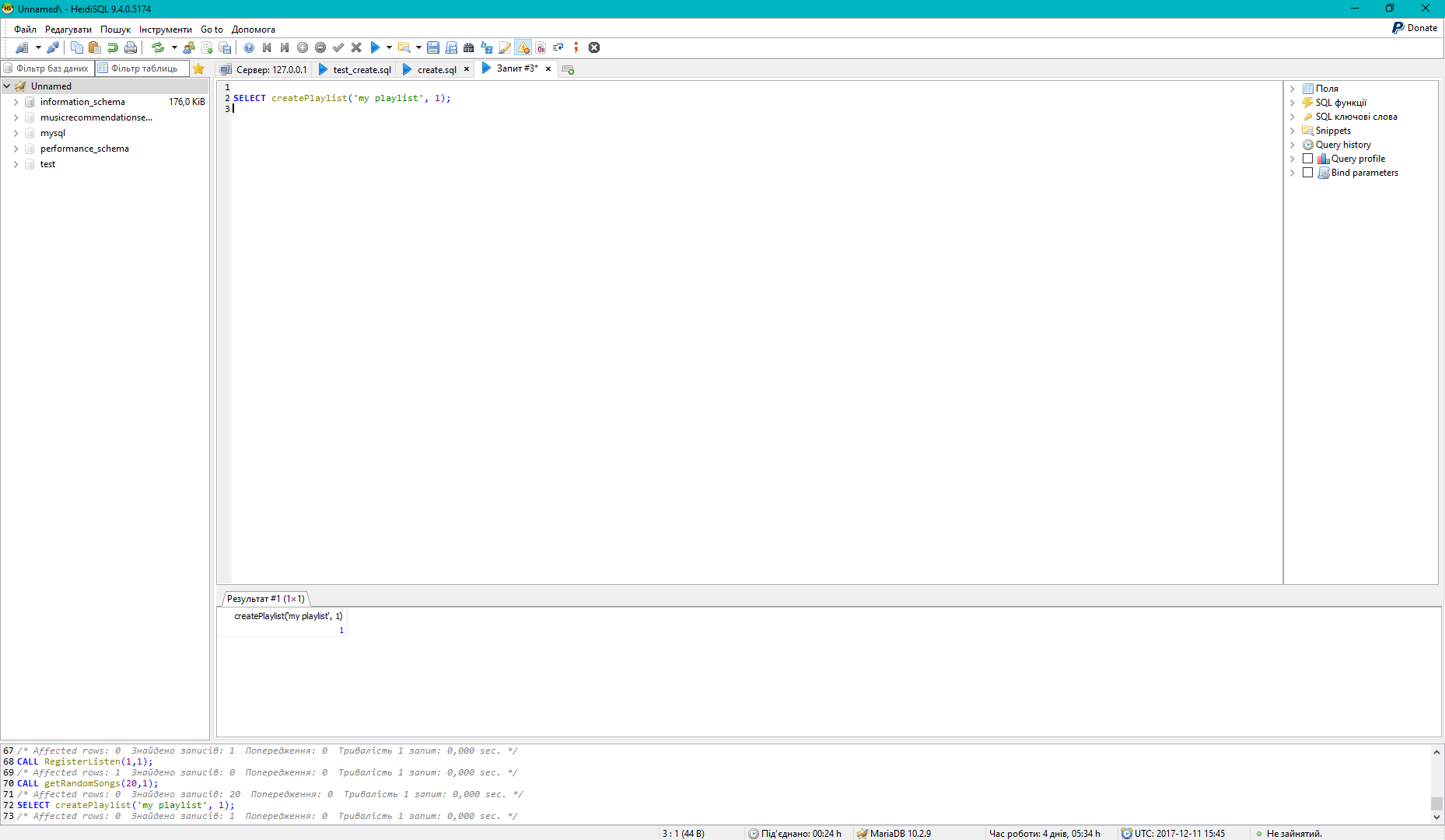


Рисунок 4.8 – Результат виконання запиту по створенню нового списку відтворень

## Додавання нової музикальної композиції до списку відтворювань

Функція перевіряє вхідні данні та додає новий трек до списку відтворення користувача. Повертає статус операції (успішна чи ні).

Запит:

|  |  |
| --- | --- |
|  | CREATE FUNCTION insertInPlaylist( |
|  | InPlaylistID int, |
|  | InSongID int, |
|  | InPrevSongPos float, |
|  | InNextSongPos float, |
|  | InOwnerID int |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF EXISTS ( |
|  | SELECT \* FROM Songs |
|  | WHERE |
|  | ID = InSongID AND |
|  | ( |
|  | Owner = InOwnerID OR |
|  | Owner IS NULL |
|  | ) |
|  | ) |
|  | AND |
|  | EXISTS ( |
|  | SELECT \* FROM Playlists |
|  | WHERE |
|  | OwnerID = InOwnerID |
|  | ) |
|  | THEN |
|  | INSERT INTO PlaylistSongs(PlaylistID,SongID,Pos) |
|  | VALUES (InPlaylistID,InSongID,(InPrevSongPos+InNextSongPos) / 2.0); |
|  | RETURN 1; |
|  | END IF; |
|  | RETURN 0; |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.9.

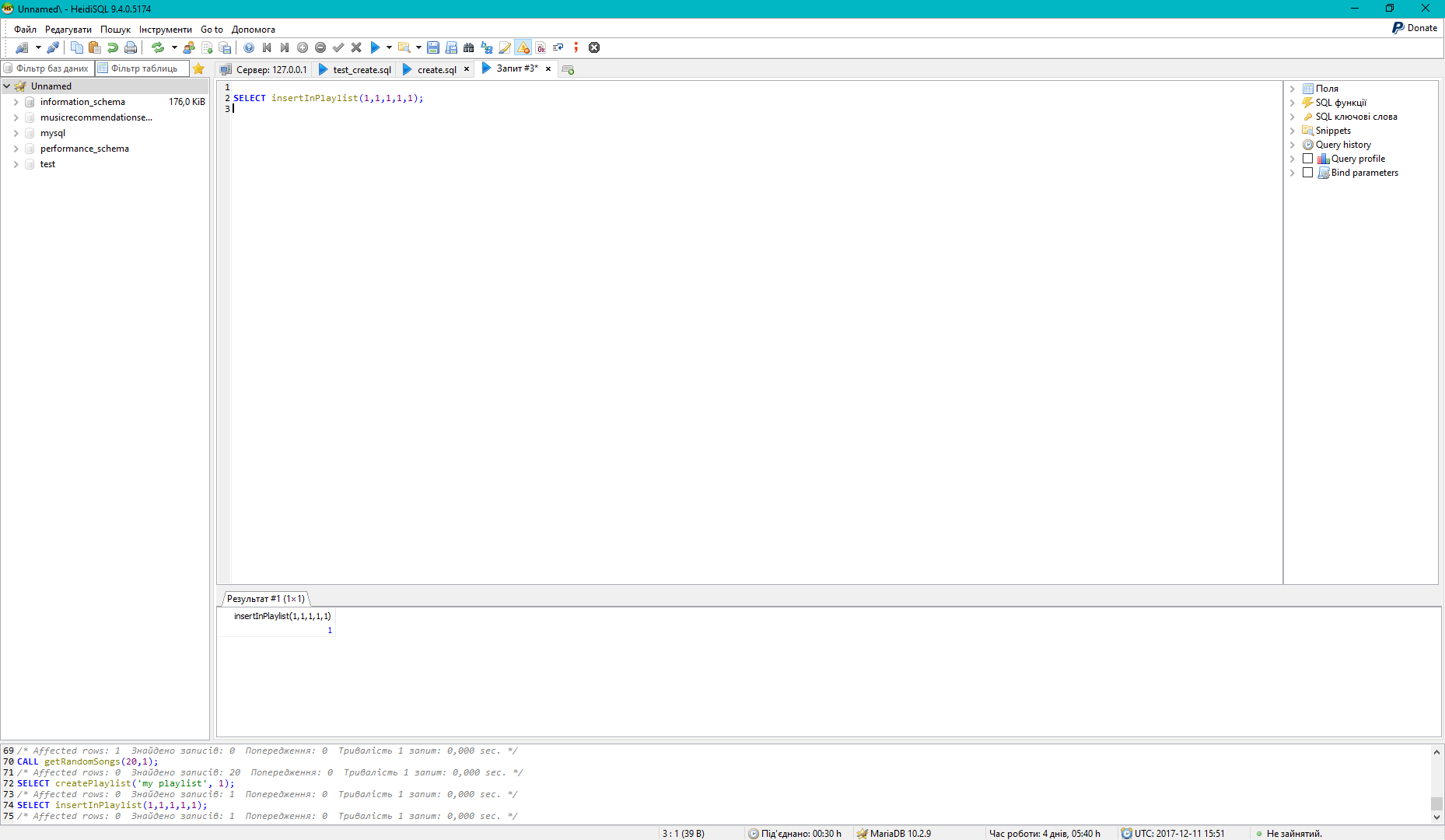


Рисунок 4.9 – Результат виконання запиту по додаванню ново музикальної композиції досписку відтвоерння

## Видалення списку відтворення

Функція перевіряє вхідні данні та видаляє список відтворення користувача та повертає статус операції (успішна чи ні).

Запит:

|  |
| --- |
|  |
|  | CREATE FUNCTION deletePlaylist( |
|  | InPlaylistID int, |
|  | InOwnerID int |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF NOT EXISTS( |
|  | SELECT \* |
|  | FROM Playlists |
|  | WHERE ID = InPlaylistID AND |
|  | OwnerID = InOwnerID |
|  | ) |
|  | THEN |
|  | RETURN 0; |
|  | END IF; |
|  | DELETE FROM PlaylistSongs |
|  | WHERE PlaylistID = InPlaylistID; |
|  | DELETE FROM Playlists |
|  | WHERE ID = InPlaylistID; |
|  | RETURN ROW\_COUNT(); |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.10.

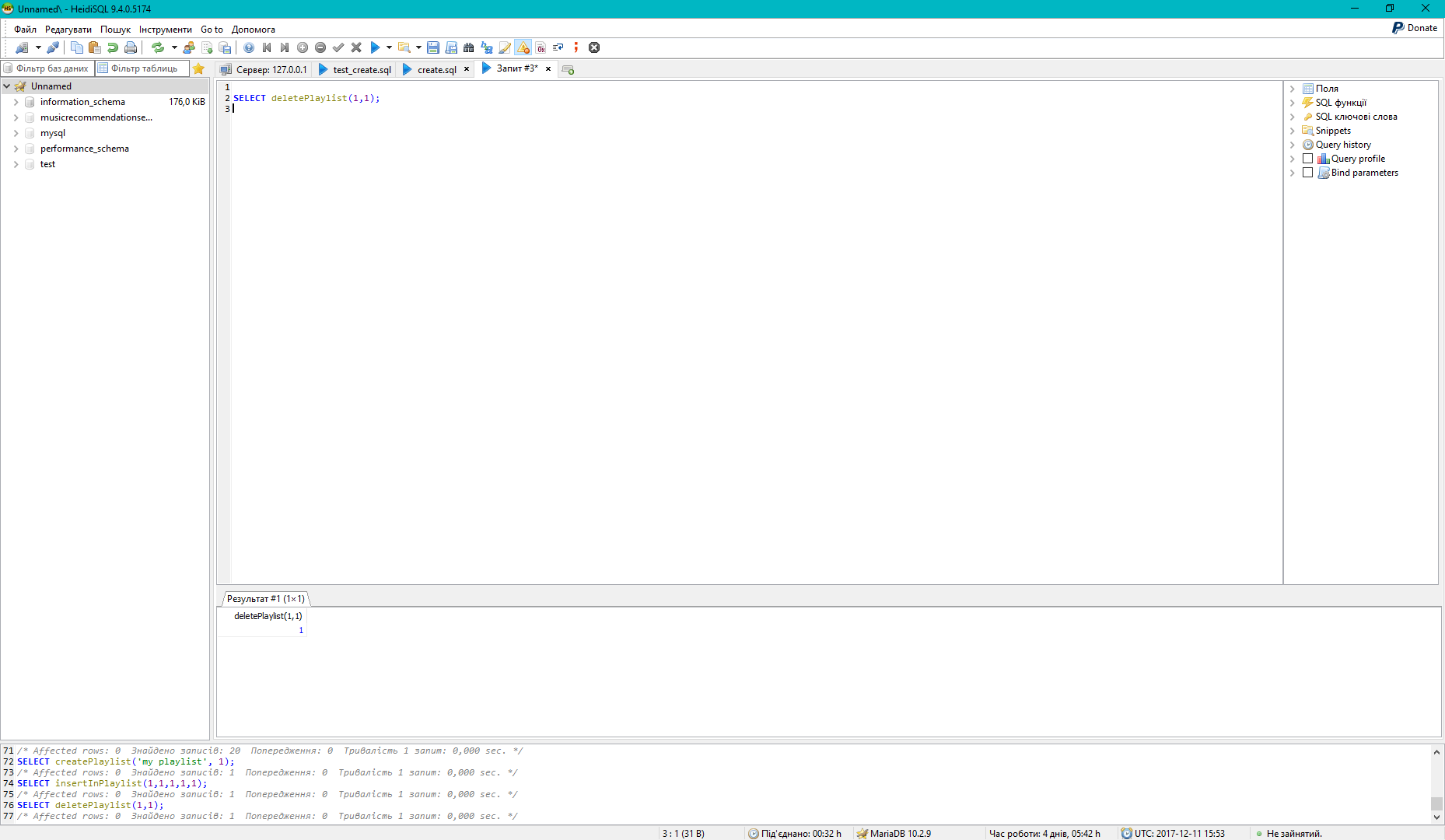


Рисунок 4.10 – Результат виконання запиту по видаленню списку відтворення

## Видалення музикальної композиції зі списку відтворення

Функція перевіряє вхідні данні та видаляє музикальну композицію зі списку відтворення користувача та повертає статус операції (успішна чи ні).

Запит:

|  |
| --- |
|  |
|  | CREATE FUNCTION deletePlaylist( |
|  | InPlaylistID int, |
|  | InOwnerID int |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF NOT EXISTS( |
|  | SELECT \* |
|  | FROM Playlists |
|  | WHERE ID = InPlaylistID AND |
|  | OwnerID = InOwnerID |
|  | ) |
|  | THEN |
|  | RETURN 0; |
|  | END IF; |
|  | DELETE FROM PlaylistSongs |
|  | WHERE PlaylistID = InPlaylistID; |
|  | DELETE FROM Playlists |
|  | WHERE ID = InPlaylistID; |
|  | RETURN ROW\_COUNT(); |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.11.

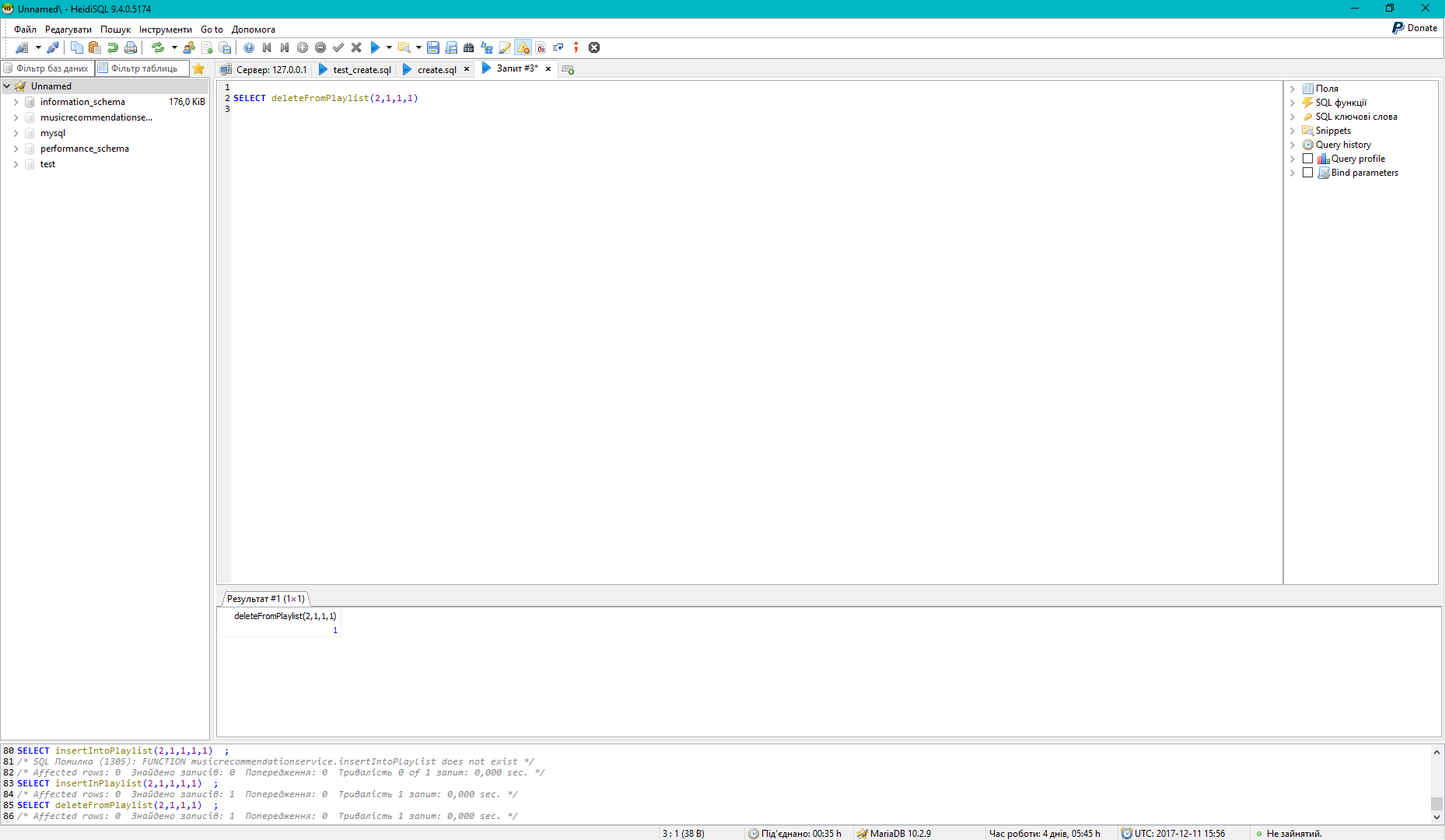


Рисунок 4.11 – Результат виконання запиту по видаленню музикальної композиції зі списку відтворення

## Зміна позиції музикальної композиції в списку відтворення

Функція перевіряє вхідні данні та переміщає пісню у списку відтворення та повертає статус операції (успішна чи ні).

Запит:

|  |
| --- |
|  |
|  | CREATE FUNCTION movePlaylistSong( |
|  | InPlaylistID int, |
|  | InSongID int, |
|  | InOldPos float, |
|  | InNewPos float, |
|  | InOwnerID int |
|  | ) |
|  | RETURNS BOOL |
|  | BEGIN |
|  | IF NOT EXISTS( |
|  | SELECT \* |
|  | FROM Playlists |
|  | WHERE ID = InPlaylistID AND |
|  | OwnerID = InOwnerID |
|  | ) |
|  | THEN |
|  | RETURN 0; |
|  | END IF; |
|  | UPDATE PlaylistSongs |
|  | SET Pos = InNewPos |
|  | WHERE |
|  | PlaylistID = InPlaylistID AND |
|  | SongID = InSongID AND |
|  | Pos = InOldPos; |
|  | RETURN ROW\_COUNT(); |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.12.

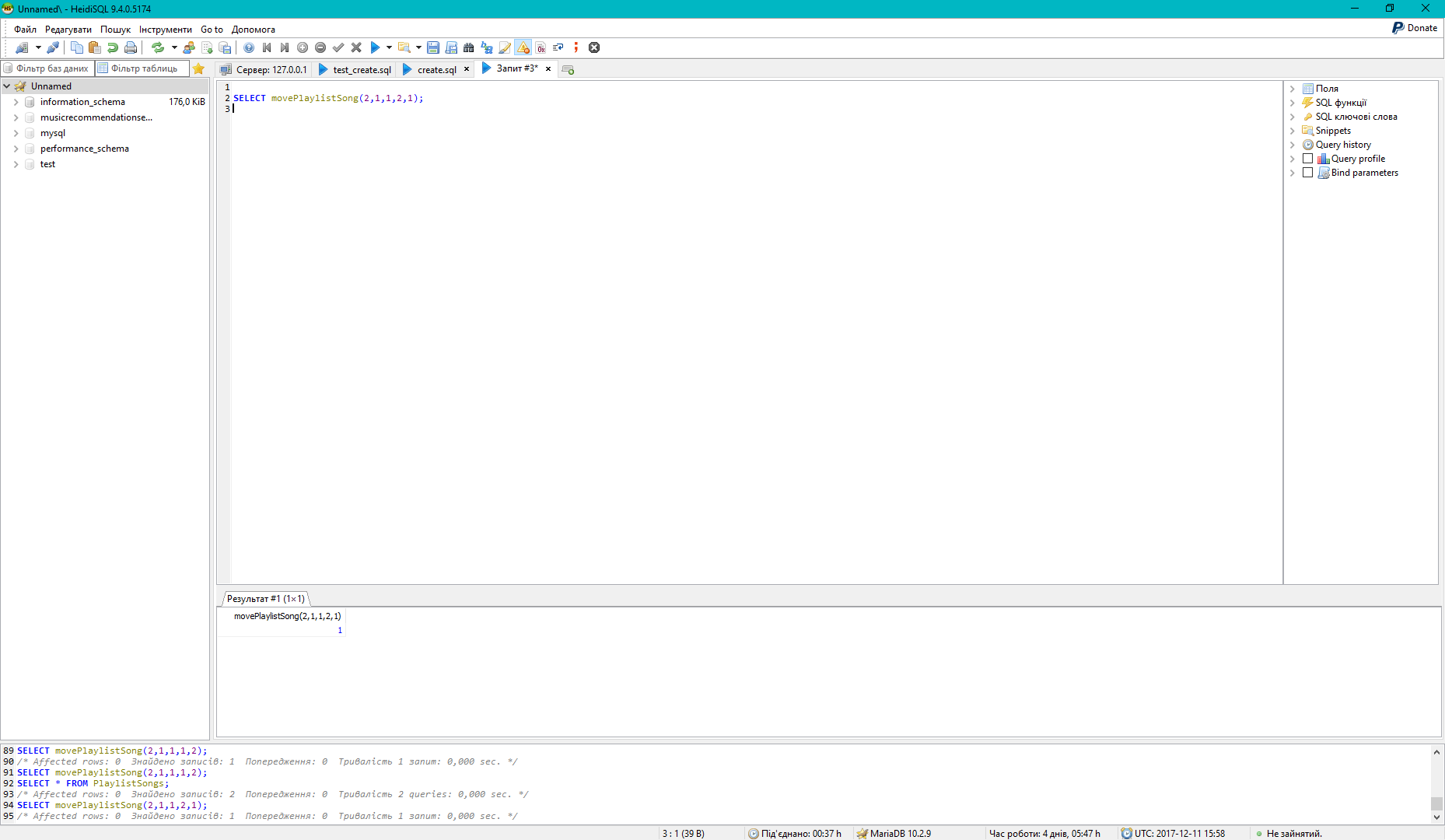


Рисунок 4.12 – Результат виконання запиту по переміщеню музикальної композиції в списку відтворення

## Вибірка всі музикальних композицій зі списку відтворення

Функція перевіряє вхідні данні та повертає усі музикальні композиції зі списку відтворення.

Запит:

|  |
| --- |
|  |
|  | CREATE PROCEDURE getPlaylistSongs( |
|  | IN InPlaylistID int, |
|  | IN InOwnerID int |
|  | ) |
|  | BEGIN |
|  | IF EXISTS( |
|  | SELECT \* |
|  | FROM Playlists |
|  | WHERE ID = InPlaylistID AND |
|  | OwnerID = InOwnerID |
|  | ) |
|  | THEN |
|  | SELECT Songs.ID, Songs.SongName, Songs.Artist, Songs.AlbumID, Songs.Length, |
|  | CONCAT('http://138.68.69.43/music/',Songs.Adress) AS Adress , Songs.Cover, PlaylistSongs.Pos |
|  | FROM PlaylistSongs |
|  | JOIN Songs ON PlaylistSongs.SongID = Songs.ID |
|  | WHERE PlaylistSongs.PlaylistID = InPlaylistID |
|  | ORDER BY PlaylistSongs.Pos ASC; |
|  | END IF; |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.13.

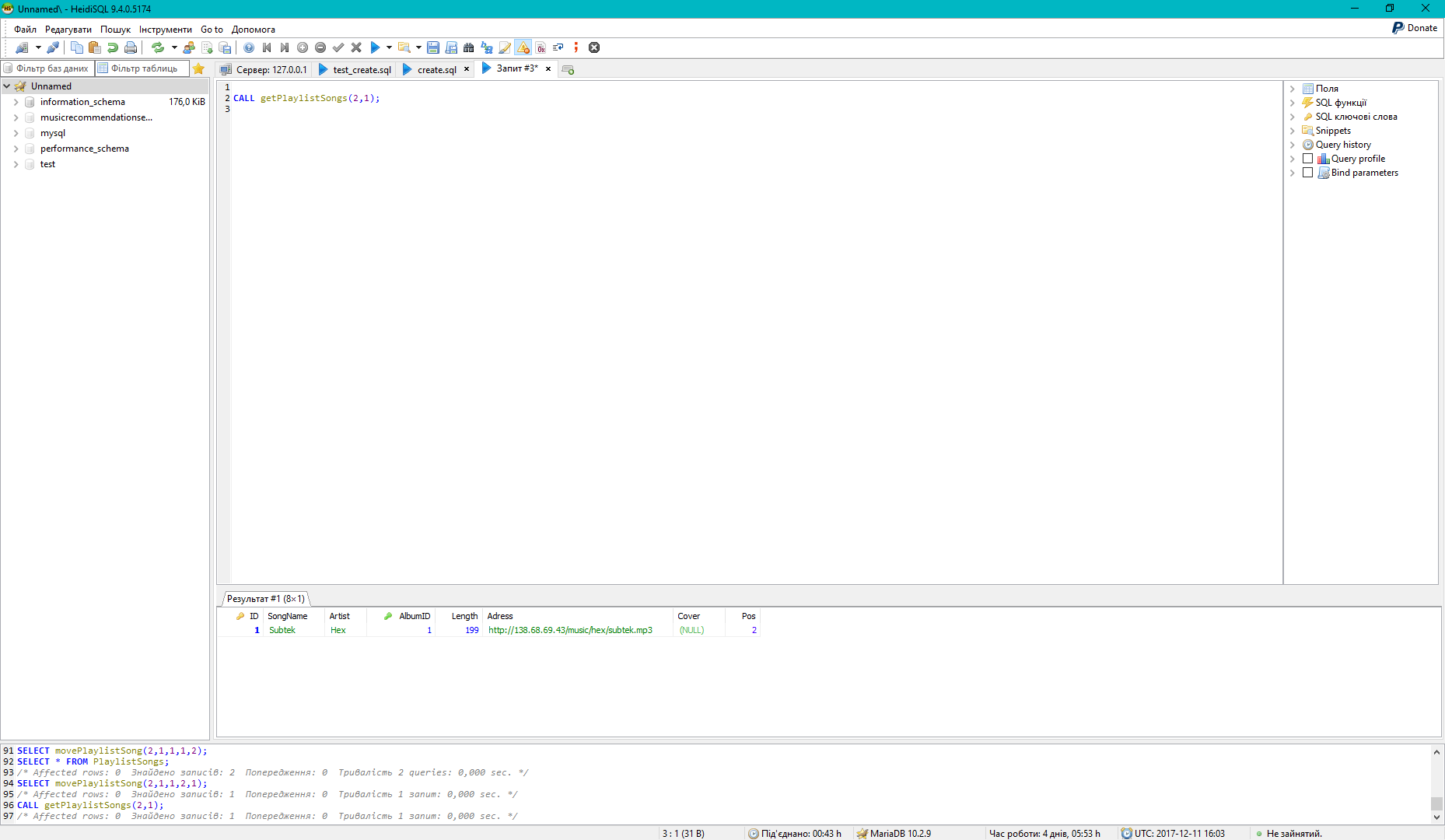


Рисунок 4.13 – Результат виконання запиту по вибіркі усі музикальних композицій зі списку відтворення

## Вибірка всі списку відтворення користувача

Функція повертає унікальні індентифікатори усіх списків відтворення користувача.

Запит:

|  |  |
| --- | --- |
|  | CREATE PROCEDURE listPlaylists( |
|  | IN InOwnerID int |
|  | ) |
|  | BEGIN |
|  | SELECT ID |
|  | FROM Playlists |
|  | WHERE OwnerID = InOwnerID; |
|  | END; |
|  | // |
|  | DELIMITER ; |

Результат виконання запиту наведений на рисунку 4.14.

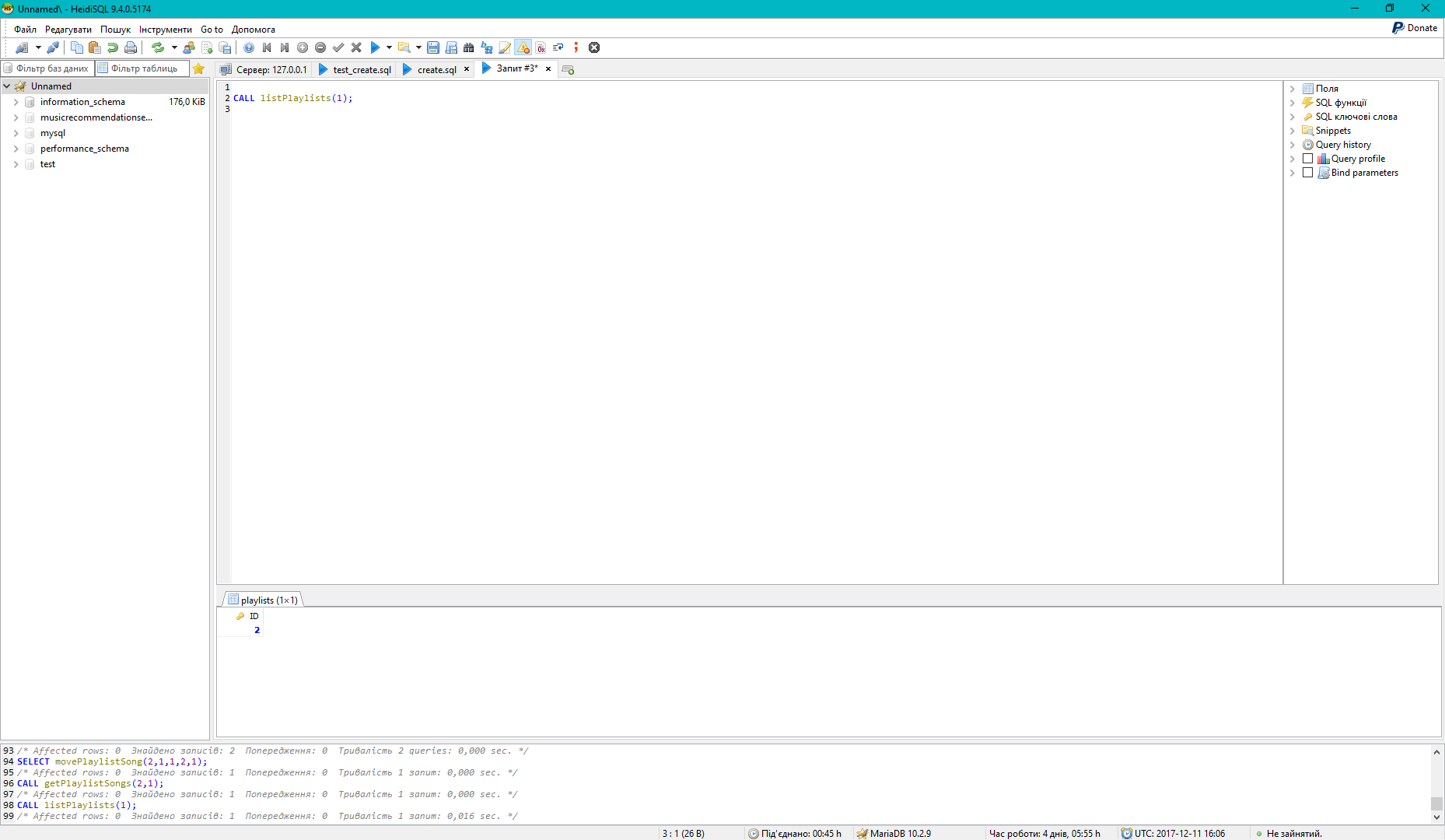


Рисунок 4.14 – Результат виконання запиту по вибіркі усі списків відтворення користувача

## Вибірка музикальної композиції

Функція повертає інформацію по музикальні композиції за її унікальним індентифікаторм.

Запит:

**SELECT** ID, SongName, Artist, AlbumID, **Length**, **concat**('http://138.68.69.43/music/', Adress), Cover **FROM** Songs **WHERE** ID **IN** (1,2,3,4) **AND** ( Owner **IS** **NULL** **OR** Owner = 1 )

Результат виконання запиту наведений на рисунку 4.15.

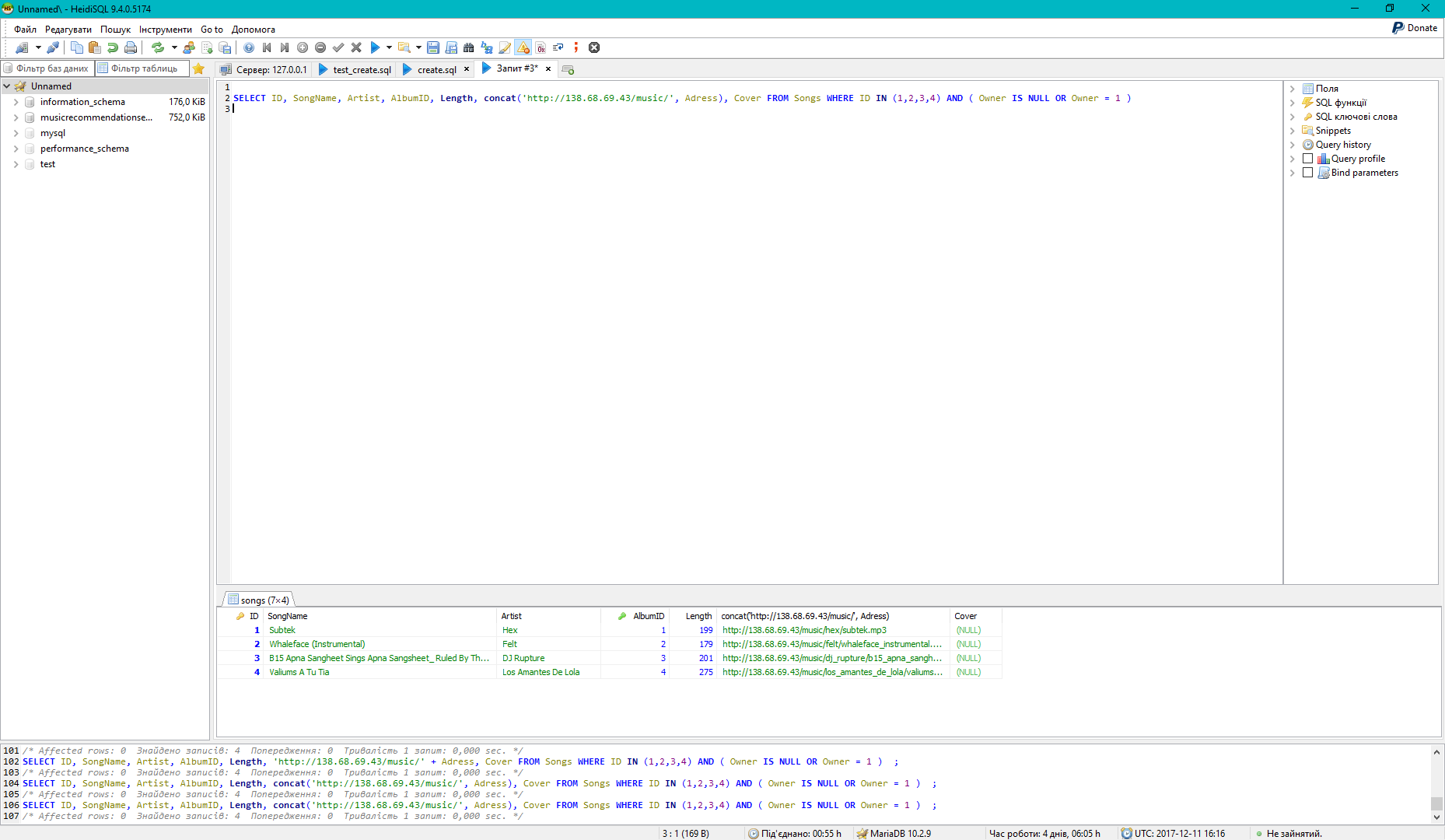


Рисунок 4.15 – Результат виконання запиту по вибіркі музикальної композиції

## Вибірка списку прослуханих композицій користувачем

Функція повертає інформацію по музикальні композиції за її унікальним індентифікаторм.

Запит:

**SELECT** SongID

**FROM** UserListens

**WHERE** UserID = 1

Результат виконання запиту наведений на рисунку 4.16.

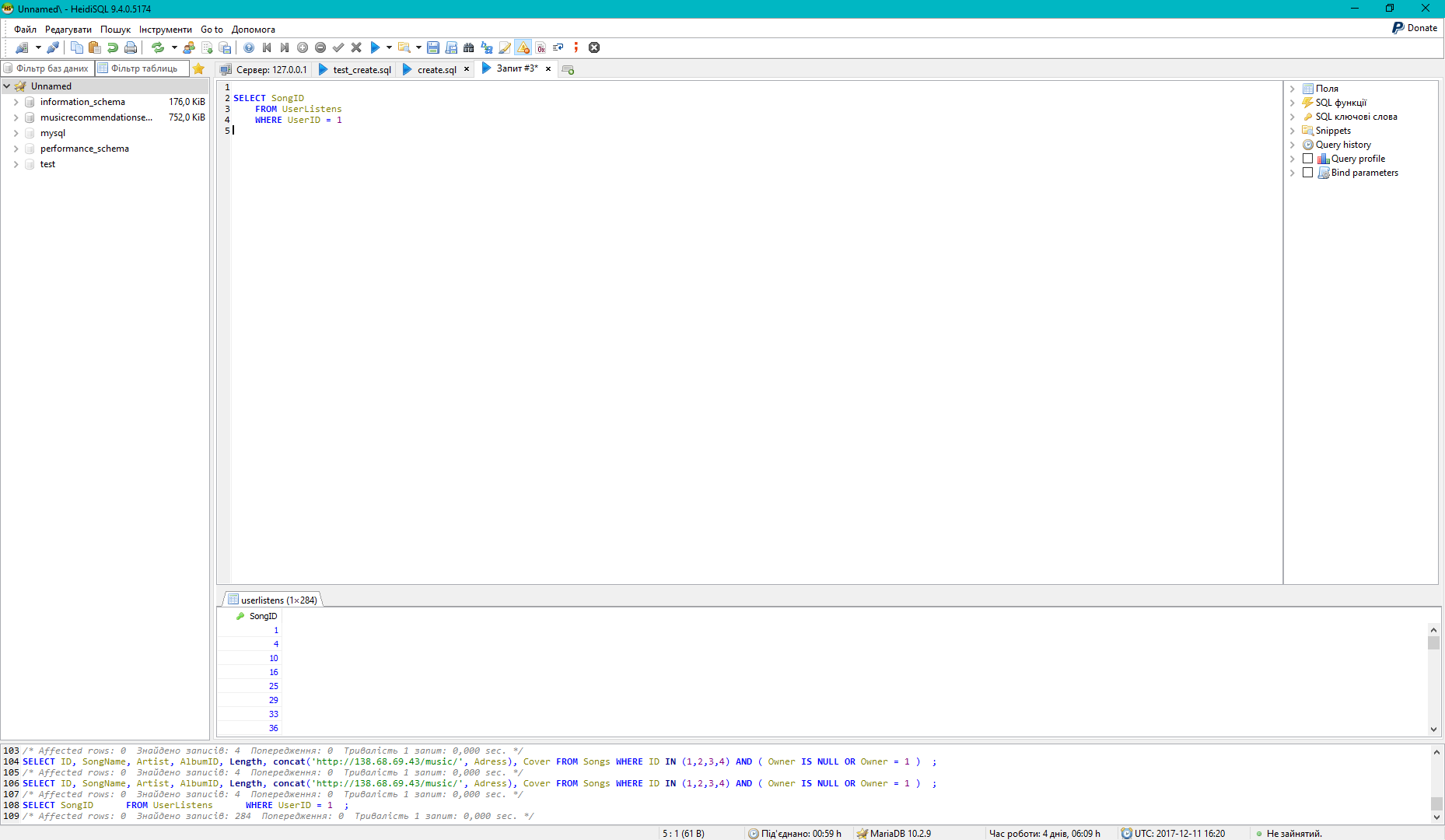


Рисунок 4.16 – Результат виконання запиту по вибіркі прослуханих користувачем композицій

## Вибірка статистики прослуховувань по кожній музикальнай композиції та їх характеристики

Функція повертає статистику прослуховувань по кожній музикальнай композиції та їх характеристики.

Запит:

**SELECT** U.SongID **AS** 'id',

**count**(U.UserID) **AS** 'heard\_by\_unique\_users\_counter',

S.song\_artist\_familiarity **AS** 'artist\_familiarity',

S.song\_artist\_hotttnesss **AS** 'artist\_hotttnesss',

S.song\_duration **AS** 'duration',

S.song\_key **AS** 'key',

S.song\_key\_confidence **AS** 'key\_confidence',

S.song\_loudness **AS** 'loudness',

S.song\_hotttnesss **AS** 'song\_hotttnesss',

S.song\_tempo **AS** 'tempo',

S.song\_end\_of\_fade\_in\_relative **AS** 'end\_of\_fade\_in\_relative',

S.song\_start\_of\_fade\_out\_relative **AS** 'start\_of\_fade\_out\_relative',

S.song\_loudness **AS** 'loudness'

**FROM** UserListens **AS** U

**INNER** **JOIN** SongNeuralNetworkData **AS** S

**ON** S.ID = U.SongID

**WHERE** U.ListenCount > 1

**GROUP** **BY** SongID

Результат виконання запиту наведений на рисунку 4.17.

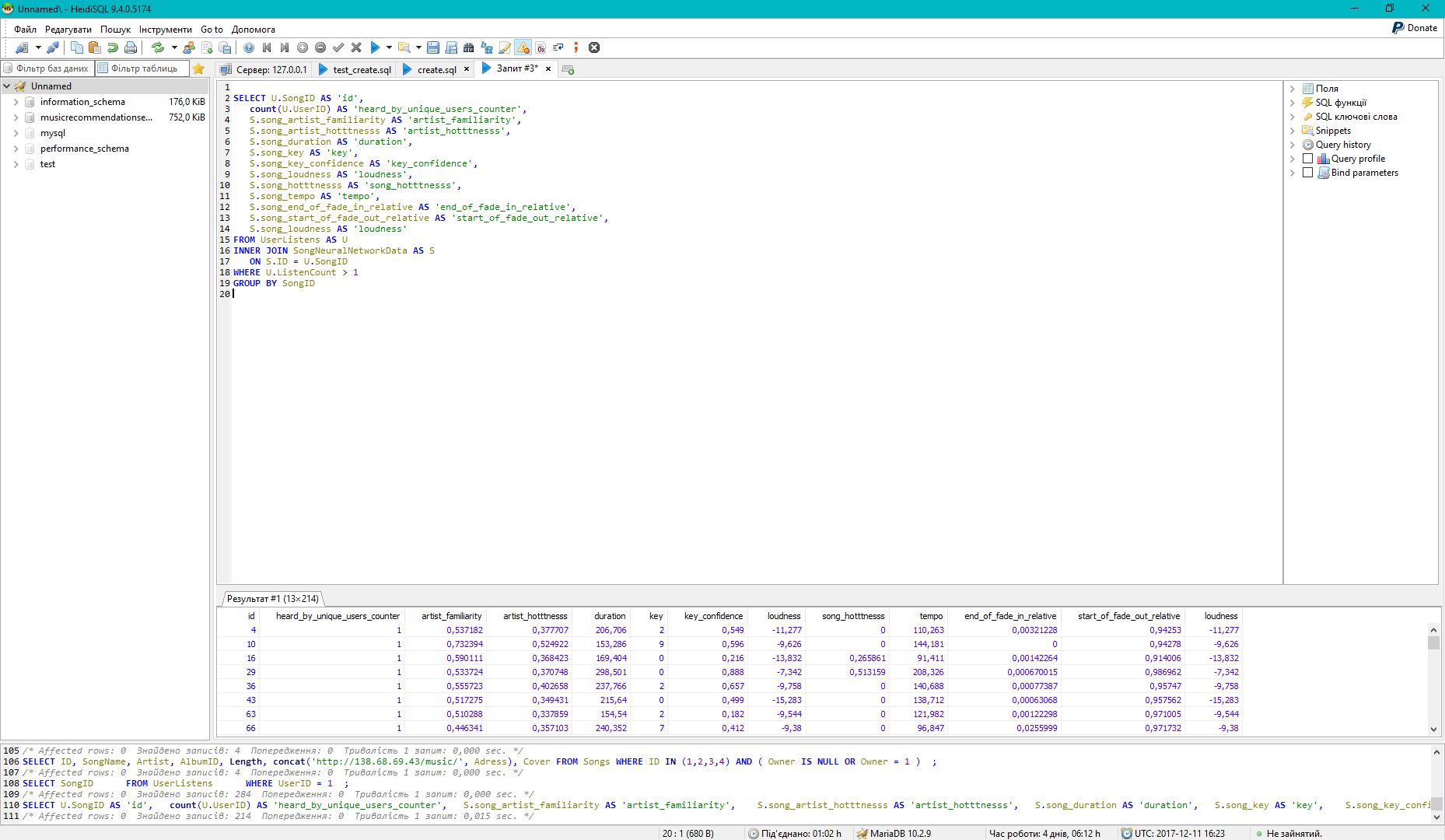


Рисунок 4.17 – Результат виконання запиту по вибіркі статистичних даних та характеристик музикальних композицій

# Інструкція користувача

Розроблене програмне забезпечення передбачає наступні кроки виконання:

1. Завантаження веб-додатку (рисунок 5.1).

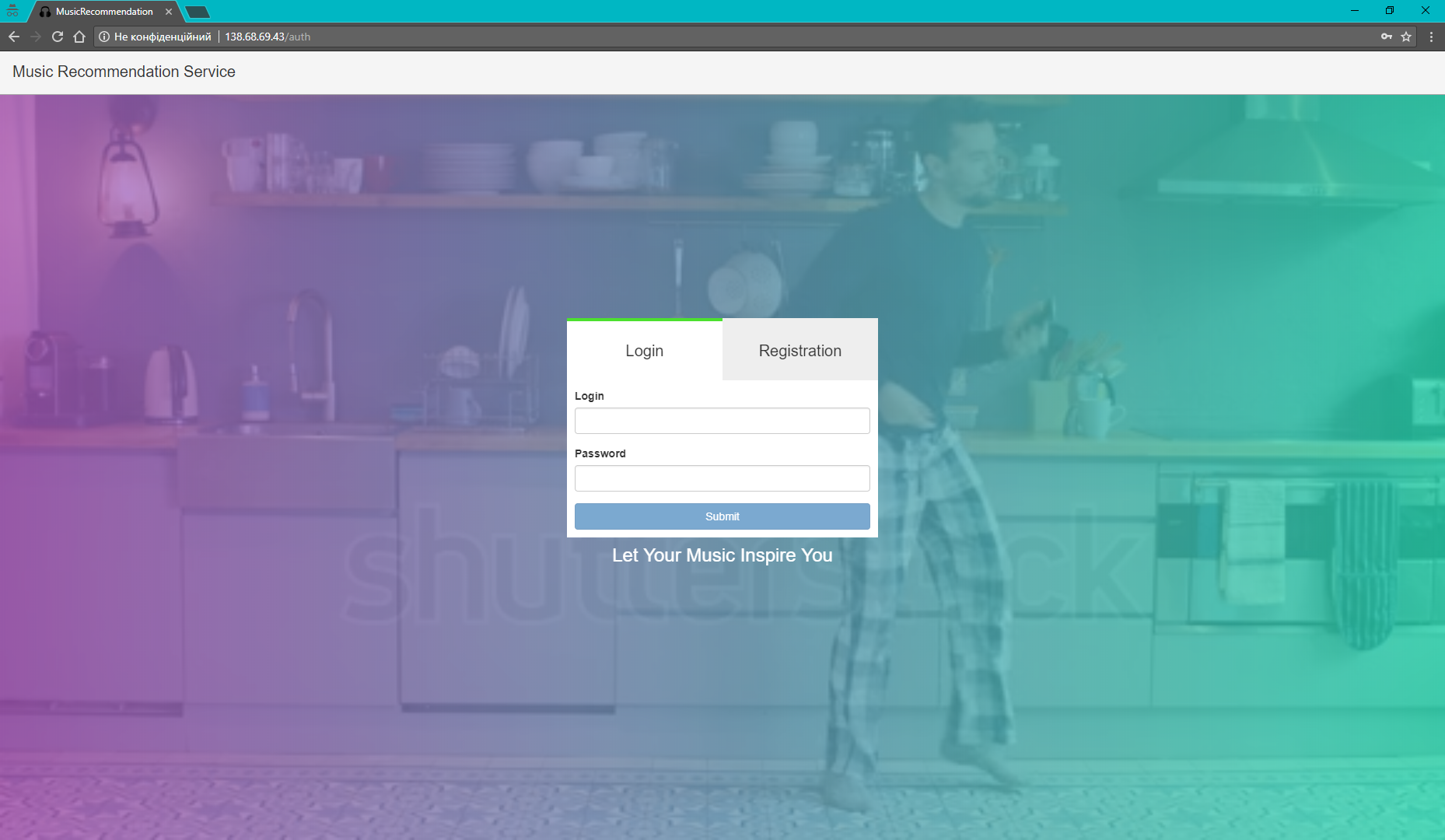


Рисунок 5.1 – Головна сторінка веб-додатку після першого завантаження

1. Вхід в обліковий запис користувача або реєстарація нового користувача (рисунок 5.2).

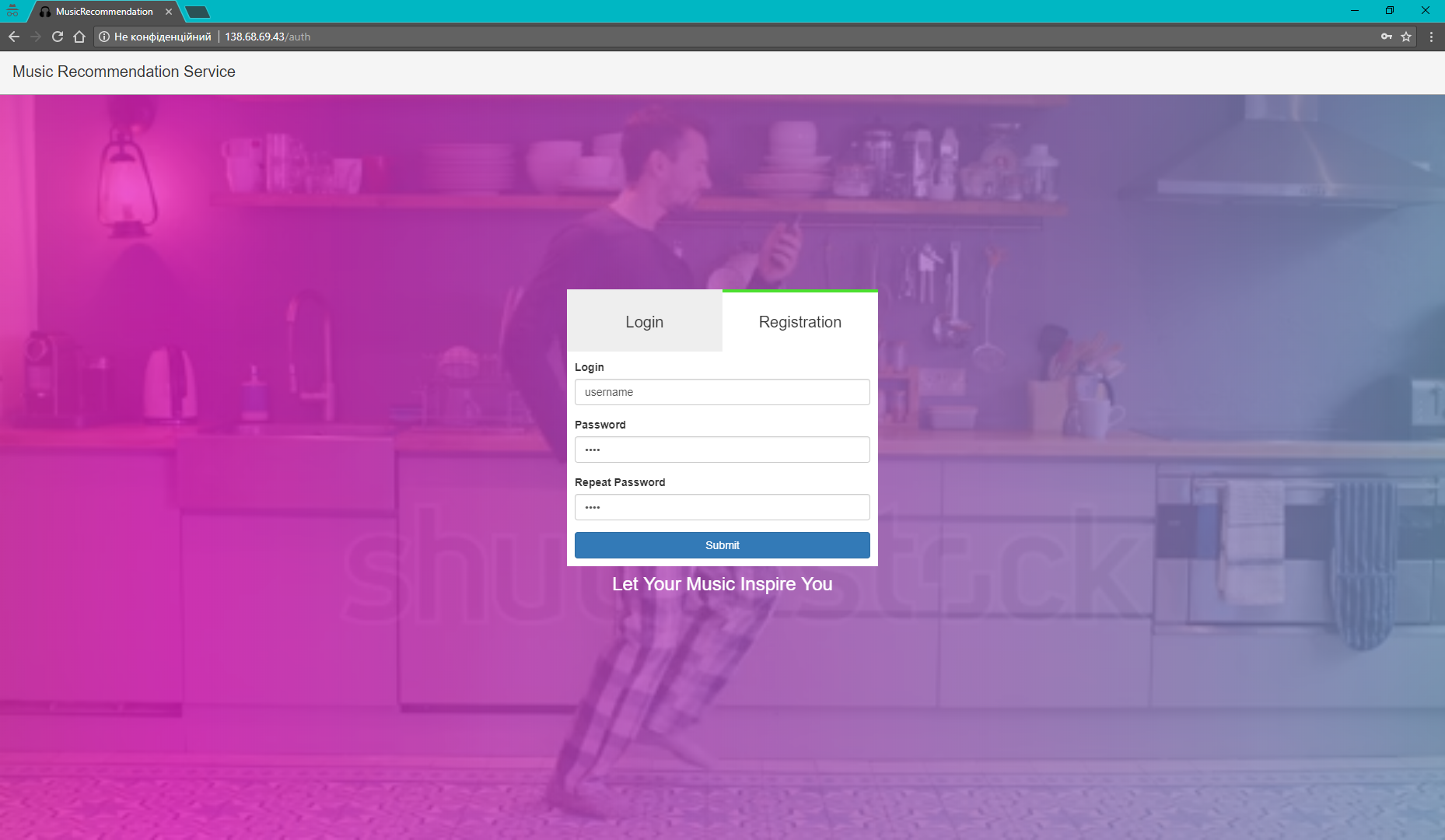


Рисунок 5.2 – Сторінка реєстрації нового користувача

1. Створення нового списку відтворення (опціонально) (рисунок 5.3).

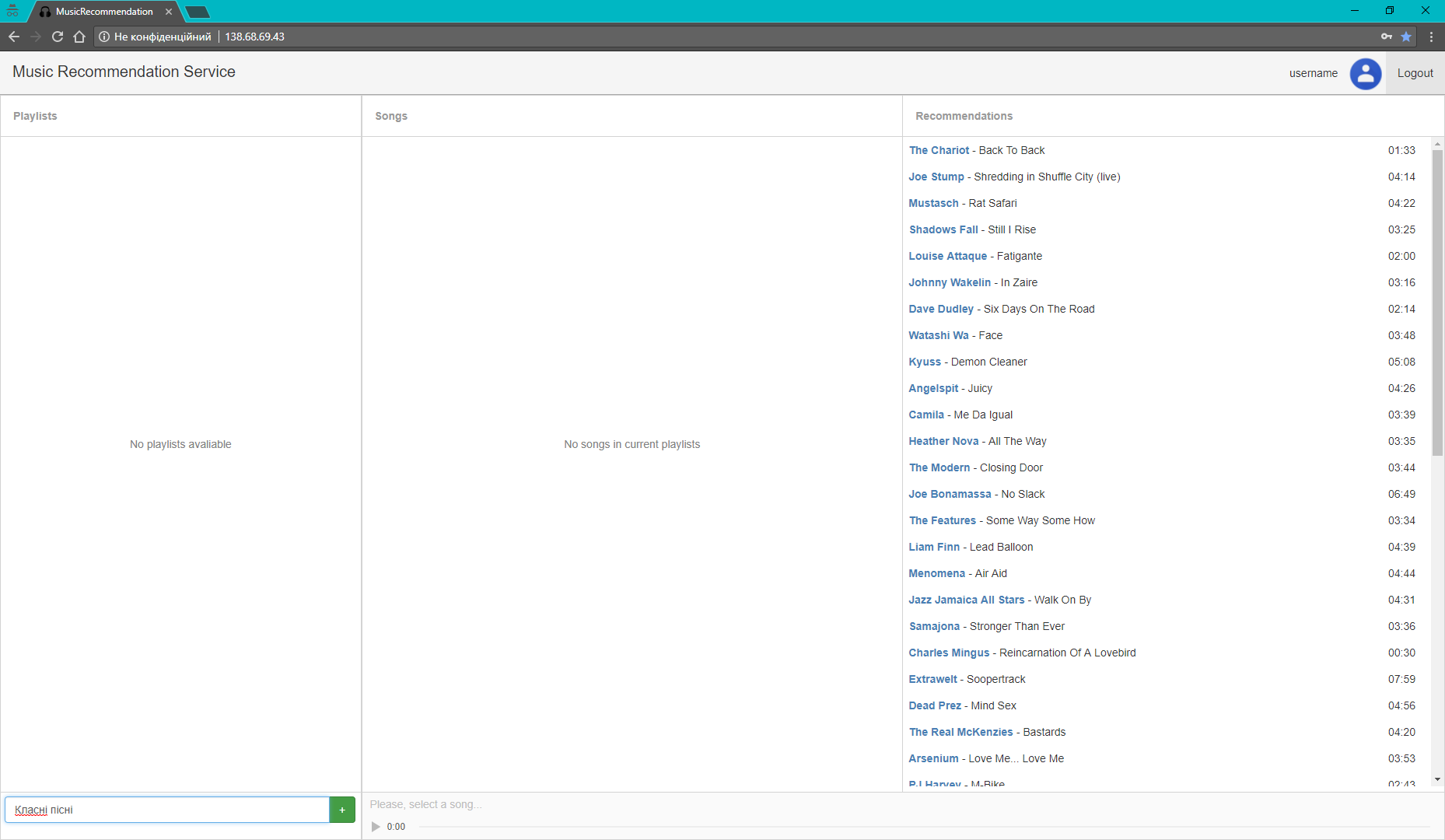


Рисунок 5.3 – Створення нового списку відтворення користувача

1. Обрання музикальної композиції для прослуховування (рисунок 5.4).

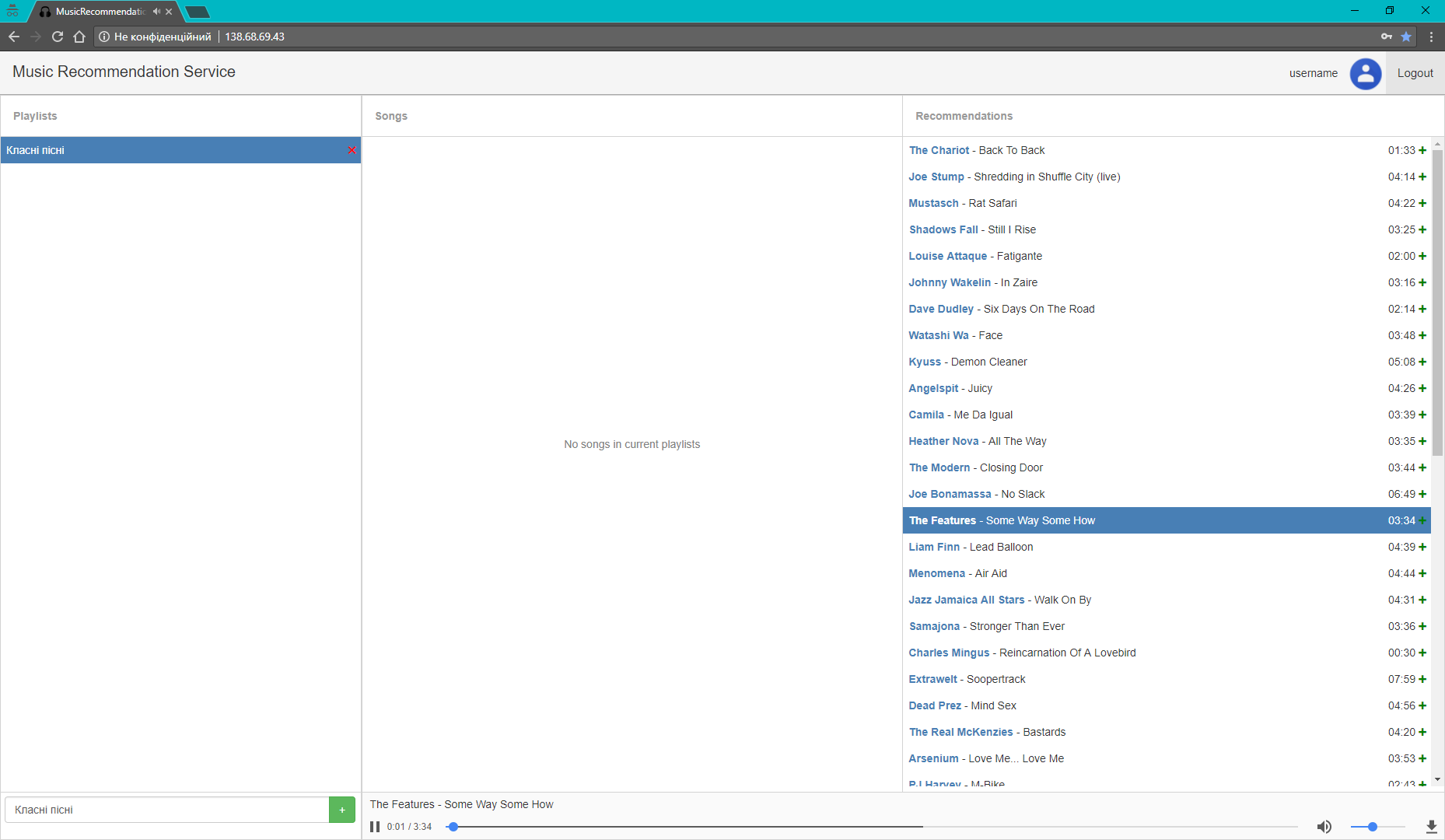


Рисунок 5.4 – Обрання музикальної композиції слухачем

1. Додання музикальної композиції до поточного списку відтворення (опціонально) (рисунок 5.5).

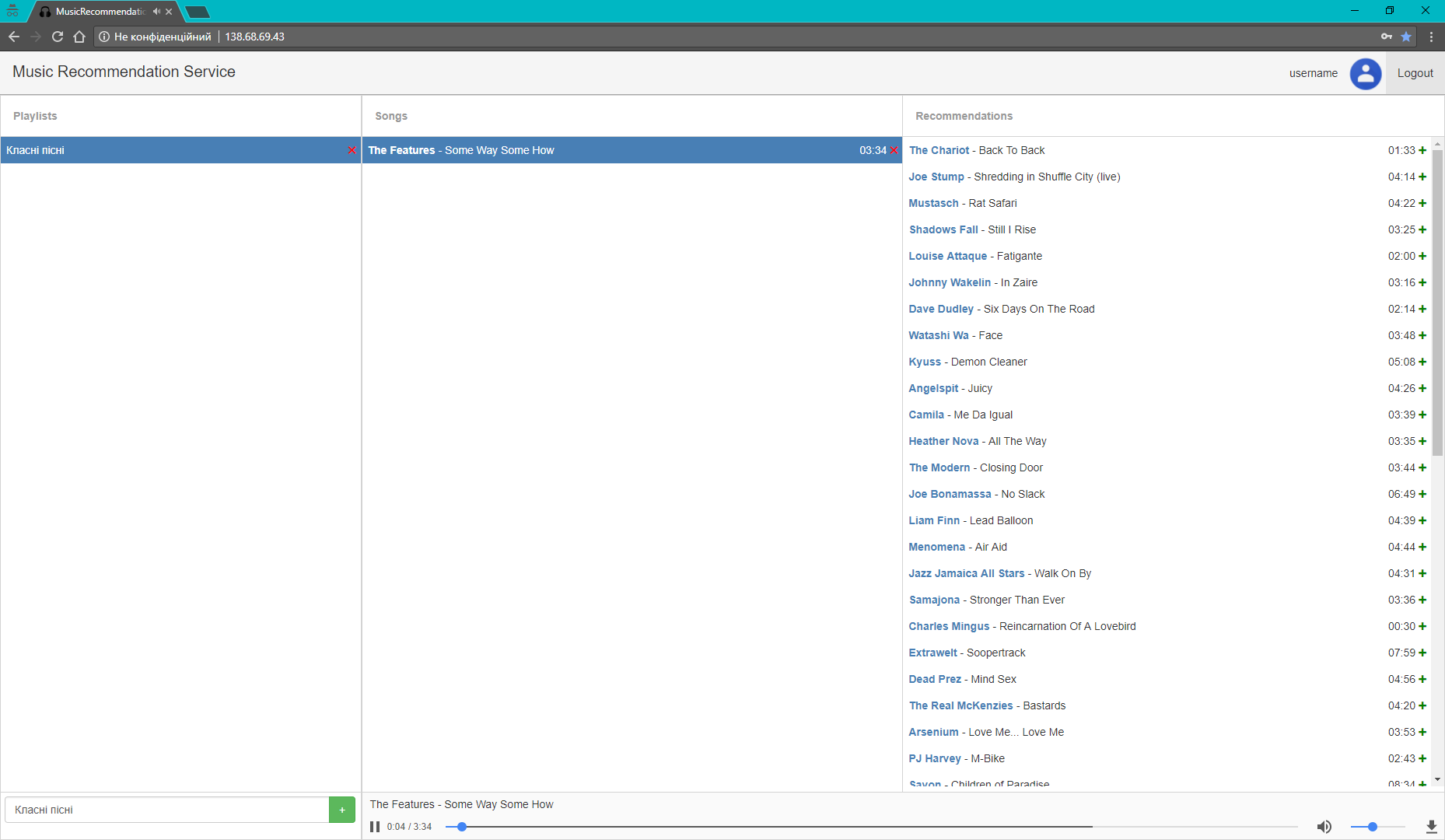


Рисунок 5.5 – Додавання музикальної композиції до поточного списку відтворення

Висновки

В процесі виконання курсової роботи була створена платформа для онлайн прослуховування музикальних композицій з можливістю тонкої настройти списків відтворення музикальних композицій та рекомендаційну систему, яка генерує рекомендації на основі персональних вподобань кожного з користувачів платформи.

Для виконання роботи були отримані теоретичні знання та практичні навички з побудови клієнт-серверної архітектури. Отримані навички створення сервера та бази даних на ньому, роботи з базою даних за допомогою різноманітних програмних засобів, прикладного застосування баз даних.

Отримані навички по роботі з великим набором не структурованих даних: очистка, перетворення, групування. На основі цих даних була побудована рекомендаційна система на основі градієнтного бустінгу каскаду дерев прийняття рішень, яка визначає чи сподобається музикальна композиція користувачеві з точністю 80%.

Створений програмний сервіс вирішує проблему з пошуком та прослуховуванням якісних музикальних композицій.

Перелік посилань

1. Why Is Music Important? [Електронний ресурс] // theodysseyonline.com – Режим доступу:

https://www.theodysseyonline.com/why-is-music-important?utm\_expid=.cZCE7oX8QCubI-ziFLsOXg.0&utm\_referrer=https%3A%2F%2Fwww.google.com.ua%2F

1. The Average American Listens to Four Hours of Music Each Day [Електронний ресурс] // spin.com – Режим доступу:

https://www.spin.com/2014/06/average-american-listening-habits-four-hours-audio-day/

1. Million Song Dataset Challenge [Електронний ресурс] // kaggle.com – Режим доступу:

https://www.kaggle.com/c/msdchallenge/

1. Python (programming language) [Електронний ресурс] // wikipedia.org – Режим доступу:

https://en.wikipedia.org/wiki/Python\_(programming\_language)

1. NumPy [Електронний ресурс] // wikipedia.org – Режим доступу:

https://en.wikipedia.org/wiki/NumPy

1. pandas (software) [Електронний ресурс] // wikipedia.org – Режим доступу:

https://en.wikipedia.org/wiki/Pandas\_(software)

1. Xgboost [Електронний ресурс] // wikipedia.org – Режим доступу:

https://en.wikipedia.org/wiki/Xgboost

1. Node.js [Електронний ресурс] // wikipedia.org – Режим доступу:

https://en.wikipedia.org/wiki/Node.js

1. MariaDB [Електронний ресурс] // wikipedia.org – Режим доступу:

https://en.wikipedia.org/wiki/MariaDB

1. Cloud computing, designed for developers. [Електронний ресурс] // digitalocean.org – Режим доступу:

https://www.digitalocean.com/