UE22CS343BB3 - Database Technologies
Jan – May 2025
ASSIGNMENT #1

Siri N Shetty – PES2UG22CS556

Mini-World Chosen for this Assignment: Music Player Application

(a) Database Preparation:

- Develop a mini-world database by selecting one domain from the provided list or a relevant domain of your choice.
  - Database should be named like "DBT25 A1 SRN Name"
- Create a Relational Schema consisting of at least 5 tables. Draw the Relational Schema diagram.
- Utilize SQL CRUD operations (Create, Read, Insert) to load data into all tables.
- Ensure the tables have only default index on say the Primary Key. Do not create additional indexes.
- Load 10,000+ rows of data into at least 2 of these (transaction?) tables, optionally using distributed storage across multiple drives of your PC.
  - If a program is utilized to create this data, submit the program code along with the assignment.
- Include **your SRN and Name** as part of the data in at least one of the tables while populating the database.

Creation of the database

```
mysql> CREATE DATABASE DBT25_A1_PES2UG22CS556_SIRI_N_SHETTY;
Query OK, 1 row affected (0.02 sec)

mysql>
```

Creating the table to store the details of users

```
mysql> CREATE TABLE Users (
    -> user_id INT AUTO_INCREMENT PRIMARY KEY,
    -> username VARCHAR(50) NOT NULL,
    -> email VARCHAR(100) NOT NULL,
    -> password VARCHAR(255) NOT NULL,
    -> reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    -> is_premium BOOLEAN DEFAULT FALSE,
    -> last_login TIMESTAMP
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```

Creating the table for storing the details of the artists

```
mysql> CREATE TABLE Artists (
    -> artist_id INT AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(100) NOT NULL,
    -> country VARCHAR(200),
    -> bio TEXT,
    -> formed_year INT
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```

Creating the table to store the details of the different albums

```
mysql> CREATE TABLE Albums (
    -> album_id INT AUTO_INCREMENT PRIMARY KEY,
    -> title VARCHAR(100) NOT NULL,
    -> artist_id INT,
    -> release_date DATE,
    -> genre VARCHAR(50),
    -> total_tracks INT,
    -> FOREIGN KEY (artist_id) REFERENCES Artists(artist_id)
    -> );
Query OK, 0 rows affected (0.07 sec)

mysql>
```

Creating the table to store the songs with their details

```
mysql> CREATE TABLE Songs (
    -> song_id INT AUTO_INCREMENT PRIMARY KEY,
    -> title VARCHAR(100) NOT NULL,
    -> album_id INT,
    -> artist_id INT,
    -> duration INT NOT NULL,
    -> track_number INT,
    -> genre VARCHAR(50),
    -> plays INT DEFAULT 0,
    -> FOREIGN KEY (album_id) REFERENCES Albums(album_id),
    -> FOREIGN KEY (artist_id) REFERENCES Artists(artist_id)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```

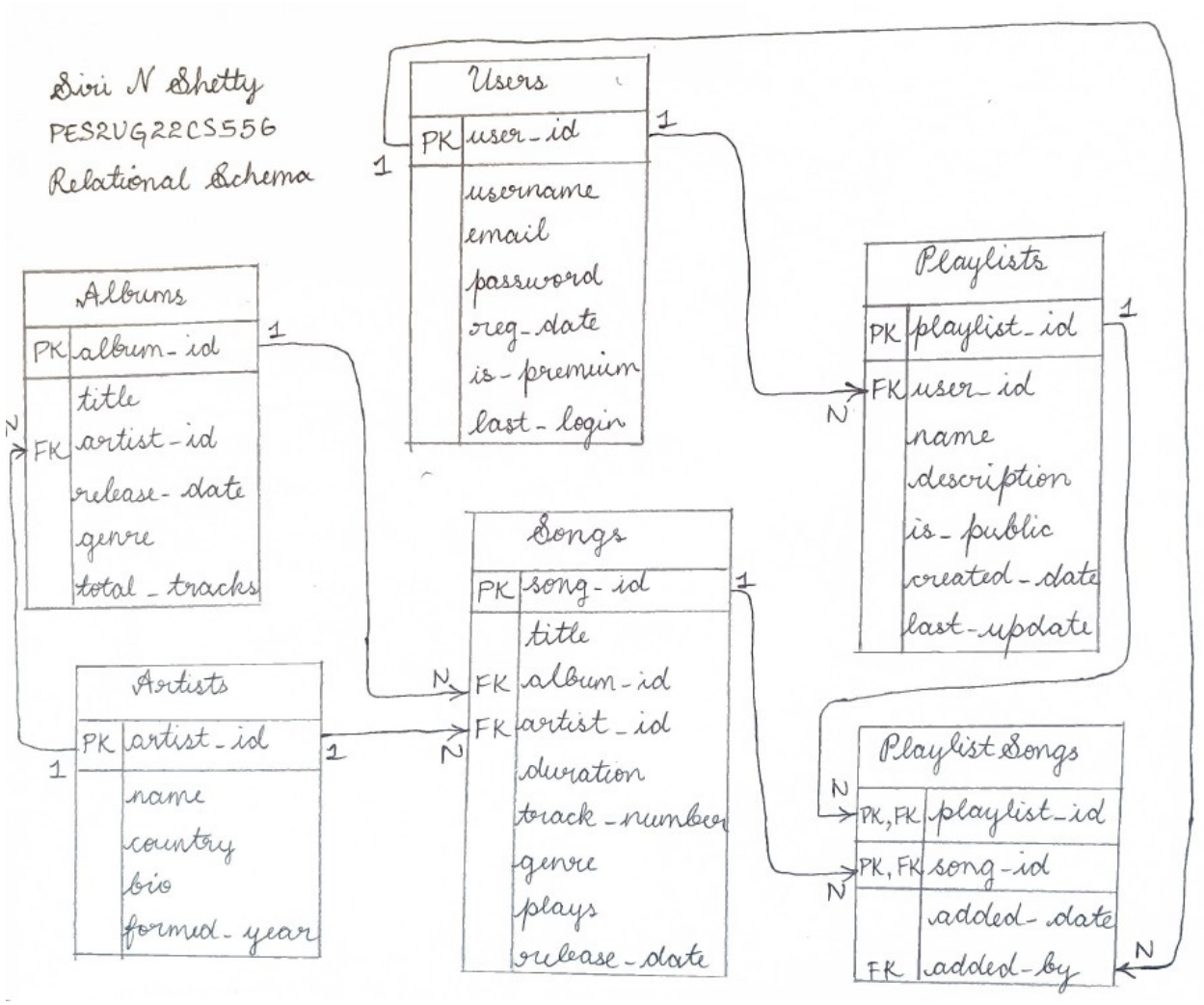
Creating the table for playlists

```
mysql> CREATE TABLE Playlists (
    -> playlist_id INT AUTO_INCREMENT PRIMARY KEY,
    -> user_id INT,
    -> name VARCHAR(100) NOT NULL,
    -> description TEXT,
    -> is_public BOOLEAN DEFAULT TRUE,
    -> created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    -> last_updated TIMESTAMP,
    -> FOREIGN KEY (user_id) REFERENCES Users (user_id)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
```

Creating the table to store the songs in the playlists

```
mysql> CREATE TABLE PlaylistSongs (
    -> playlist_id INT,
    -> song_id INT,
    -> added_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    -> added_by INT,
    -> PRIMARY KEY (playlist_id, song_id),
    -> FOREIGN KEY (playlist_id) REFERENCES Playlists(playlist_id)
    -> , FOREIGN KEY (song_id) REFERENCES Songs(song_id),
    -> FOREIGN KEY (added_by) REFERENCES Users(user_id)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
```

Relational Schema Diagram



Loading Data into Tables

The script **connects to the MySQL database** and uses the **Faker** library to generate realistic random data.
It **inserts bulk data** into tables (Users, Artists, Albums, Songs, Playlists, and PlaylistSongs) using **batch inserts** for efficiency.
The script ensures **foreign key constraints are respected** by fetching MAX(id) values from tables before inserting related data.
Finally, it **handles errors, commits transactions**, and **closes the database connection** after inserting thousands of records.

```
import mysql.connector
import random
import datetime
import string
import hashlib
from faker import Faker
import time

fake = Faker()  # Initialize Faker

conn_params = { # Database connection parameters
    "database": "DBT25_A1_PES2UG22CS556_SIRI_N_SHETTY",
    "user": "root",
    "password": "SiRi123.",
    "host": "localhost",
    "port": 3306
}

# Connect to the database
conn = mysql.connector.connect(**conn_params)
cursor = conn.cursor()

def generate_password_hash(password):
    return hashlib.sha256(password.encode()).hexdigest()

def insert_users(num_users):
    print(f"Inserting {num_users} users...")

    your_srn = "PES2UG22CS556"
    your_name = "Siri N Shetty"

    insert_query = """
    INSERT INTO Users
    (username, email, password, reg_date, is_premium, last_login)
    VALUES (%s, %s, %s, %s, %s, %s)
    """

    cursor.execute(insert_query, (
        your_name,
        f"{your_srn.lower()}@pesu.pes.edu",
        generate_password_hash("password123"),
        fake.date_time_between(start_date="-2y", end_date="now"),
        random.choice([True, False]),
        fake.date_time_between(start_date="-1m", end_date="now")
    ))

    batch_size = 1000       # Batch insert for better performance
    users = []

    for i in range(num_users - 1):
        username = fake.user_name()
        email = fake.email()
```

```
            password = generate_password_hash(fake.password())
            reg_date = fake.date_time_between(start_date="-2y", end_date="now")
            is_premium = random.choice([True, False])
            last_login = fake.date_time_between(start_date="-1m", end_date="now")

            users.append((username, email, password, reg_date, is_premium, last_login))

            if len(users) >= batch_size:
                cursor.executemany(insert_query, users)
                conn.commit()
                users = []

    # Insert any remaining users
    if users:
        cursor.executemany(insert_query, users)
        conn.commit()

    cursor.execute("SELECT MAX(user_id) FROM Users")
    return cursor.fetchone()[0]

def insert_artists(num_artists):
    print(f"Inserting {num_artists} artists...")

    insert_query = """
    INSERT INTO Artists
    (name, country, bio, formed_year)
    VALUES (%s, %s, %s, %s)
    """

    batch_size = 1000
    artists = []

    for i in range(num_artists):
        name = fake.name() if random.random() < 0.7 else fake.company()
        country = fake.country()
        bio = fake.text(max_nb_chars=500)
        formed_year = random.randint(1950, 2023)

        artists.append((name, country, bio, formed_year))

        if len(artists) >= batch_size:
            cursor.executemany(insert_query, artists)
            conn.commit()
            artists = []

    # Insert any remaining artists
    if artists:
        cursor.executemany(insert_query, artists)
        conn.commit()

    cursor.execute("SELECT MAX(artist_id) FROM Artists")
    return cursor.fetchone()[0]
```

```
def insert_albums(num_albums, max_artist_id):
    print(f"Inserting {num_albums} albums...")

    insert_query = """
    INSERT INTO Albums
    (title, artist_id, release_date, genre, total_tracks)
    VALUES (%s, %s, %s, %s, %s)
    """

    genres = ["Rock", "Pop", "Hip Hop", "Jazz", "Classical", "Electronic", "R&B", "Country",
              "Reggae", "Metal", "Folk", "Blues", "Indie", "Alternative", "Dance", "Latin"]

    batch_size = 1000
    albums = []

    for i in range(num_albums):
        title = " ".join(fake.words(nb=random.randint(1, 5))).title()
        artist_id = random.randint(1, max_artist_id)
        release_date = fake.date_between(start_date="-70y", end_date="now")
        genre = random.choice(genres)
        total_tracks = random.randint(1, 20)

        albums.append((title, artist_id, release_date, genre, total_tracks))

        if len(albums) >= batch_size:
            cursor.executemany(insert_query, albums)
            conn.commit()
            albums = []

    # Insert any remaining albums
    if albums:
        cursor.executemany(insert_query, albums)
        conn.commit()

    cursor.execute("SELECT MAX(album_id) FROM Albums")
    return cursor.fetchone()[0]

def insert_songs(num_songs, max_album_id, max_artist_id):
    print(f"Inserting {num_songs} songs...")

    insert_query = """
    INSERT INTO Songs
    (title, album_id, artist_id, duration, track_number, genre, plays, release_date)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """

    genres = ["Rock", "Pop", "Hip Hop", "Jazz", "Classical", "Electronic", "R&B", "Country",
              "Reggae", "Metal", "Folk", "Blues", "Indie", "Alternative", "Dance", "Latin"]

    batch_size = 1000
    songs = []
```

```
    for i in range(num_songs):
        title = " ".join(fake.words(nb=random.randint(1, 6))).title()
        album_id = random.randint(1, max_album_id)
        artist_id = random.randint(1, max_artist_id)
        duration = random.randint(30, 600)  # 30 seconds to 10 minutes
        track_number = random.randint(1, 20)
        genre = random.choice(genres)
        plays = random.randint(0, 1000000)
        release_date = fake.date_between(start_date="-70y", end_date="now")

        songs.append((title, album_id, artist_id, duration, track_number, genre, plays,
release_date))

        if len(songs) >= batch_size:
            cursor.executemany(insert_query, songs)
            conn.commit()
            songs = []

    # Insert any remaining songs
    if songs:
        cursor.executemany(insert_query, songs)
        conn.commit()

    cursor.execute("SELECT MAX(song_id) FROM Songs")
    return cursor.fetchone()[0]

def insert_playlists(num_playlists, max_user_id):
    print(f"Inserting {num_playlists} playlists...")

    insert_query = """
    INSERT INTO Playlists
    (user_id, name, description, created_date, is_public, last_updated)
    VALUES (%s, %s, %s, %s, %s, %s)
    """

    batch_size = 1000
    playlists = []

    for i in range(num_playlists):
        user_id = random.randint(1, max_user_id)
        name = " ".join(fake.words(nb=random.randint(1, 4))).title()
        description = fake.text(max_nb_chars=200) if random.random() > 0.3 else None
        created_date = fake.date_time_between(start_date="-2y", end_date="now")
        is_public = random.random() > 0.2  # 80% are public
        last_updated = fake.date_time_between(start_date=created_date, end_date="now")

        playlists.append((user_id, name, description, created_date, is_public, last_updated))

        if len(playlists) >= batch_size:
            cursor.executemany(insert_query, playlists)
            conn.commit()
```

```
            playlists = []

    # Insert any remaining playlists
    if playlists:
        cursor.executemany(insert_query, playlists)
        conn.commit()

    cursor.execute("SELECT MAX(playlist_id) FROM Playlists")
    return cursor.fetchone()[0]

def insert_playlist_songs(num_playlist_songs, max_playlist_id, max_song_id, max_user_id):
    print(f"Inserting {num_playlist_songs} playlist-song associations...")

    insert_query = """
    INSERT INTO PlaylistSongs
    (playlist_id, song_id, added_date, added_by)
    VALUES (%s, %s, %s, %s)
    """

    # To avoid duplicates, we'll use a set to track combinations we've already inserted
    already_inserted = set()

    batch_size = 1000
    playlist_songs = []

    inserted_count = 0
    attempts = 0
    max_attempts = num_playlist_songs * 2  # Limit attempts to avoid infinite loop

    while inserted_count < num_playlist_songs and attempts < max_attempts:
        attempts += 1
        playlist_id = random.randint(1, max_playlist_id)
        song_id = random.randint(1, max_song_id)

        # Skip if this combination already exists
        if (playlist_id, song_id) in already_inserted:
            continue

        already_inserted.add((playlist_id, song_id))

        added_date = fake.date_time_between(start_date="-2y", end_date="now")
        added_by = random.randint(1, max_user_id)

        playlist_songs.append((playlist_id, song_id, added_date, added_by))
        inserted_count += 1

        if len(playlist_songs) >= batch_size:
            try:
                cursor.executemany(insert_query, playlist_songs)
                conn.commit()
            except mysql.connector.Error as err:
                print(f"Error inserting playlist songs batch: {err}")
```

```
                # Try inserting one by one for problematic batches
                for entry in playlist_songs:
                    try:
                        cursor.execute(insert_query, entry)
                        conn.commit()
                    except mysql.connector.Error as inner_err:
                        print(f"Error inserting individual entry: {inner_err}")
                        conn.rollback()
            playlist_songs = []

    # Insert any remaining playlist songs
    if playlist_songs:
        try:
            cursor.executemany(insert_query, playlist_songs)
            conn.commit()
        except mysql.connector.Error as err:
            print(f"Error inserting final playlist songs batch: {err}")
            # Try inserting one by one
            for entry in playlist_songs:
                try:
                    cursor.execute(insert_query, entry)
                    conn.commit()
                except mysql.connector.Error as inner_err:
                    print(f"Error inserting individual entry: {inner_err}")
                    conn.rollback()

    print(f"Successfully inserted {inserted_count} playlist-song associations after {attempts}
attempts")

def main():
    start_time = time.time()

    try:
        num_users = 1000
        num_artists = 500
        num_albums = 2000
        num_songs = 12000  # One of our large tables (>10,000 rows)
        num_playlists = 3000
        num_playlist_songs = 15000  # Another large table (>10,000 rows)

        # Insert data into each table
        max_user_id = insert_users(num_users)
        print(f"Users inserted. Max user_id: {max_user_id}")

        max_artist_id = insert_artists(num_artists)
        print(f"Artists inserted. Max artist_id: {max_artist_id}")

        max_album_id = insert_albums(num_albums, max_artist_id)
        print(f"Albums inserted. Max album_id: {max_album_id}")

        max_song_id = insert_songs(num_songs, max_album_id, max_artist_id)
        print(f"Songs inserted. Max song_id: {max_song_id}")
```

```
        max_playlist_id = insert_playlists(num_playlists, max_user_id)
        print(f"Playlists inserted. Max playlist_id: {max_playlist_id}")

        insert_playlist_songs(num_playlist_songs, max_playlist_id, max_song_id, max_user_id)
        print("Playlist songs inserted.")

        end_time = time.time()
        print(f"Data generation completed in {(end_time - start_time):.2f} seconds")

    except mysql.connector.Error as err:
        print(f"Database error: {err}")
    except Exception as err:
        print(f"Error: {err}")
    finally:
        # Close the database connection
        cursor.close()
        conn.close()
        print("Database connection closed.")

if __name__ == "__main__":
    main()
```

```
PS D:\Semester6\DBT\Assignment1> python -u "d:\Semester6\DBT\Assignment1\prepare_data.py"
Inserting 1000 users...
Users inserted. Max user_id: 1000
Inserting 500 artists...
Artists inserted. Max artist_id: 500
Inserting 2000 albums...
Albums inserted. Max album_id: 2000
Inserting 12000 songs...
Songs inserted. Max song_id: 12000
Inserting 3000 playlists...
Playlists inserted. Max playlist_id: 3000
Inserting 15000 playlist-song associations...
Successfully inserted 15000 playlist-song associations after 15004 attempts
Playlist songs inserted.
Data generation completed in 7.40 seconds
Database connection closed.
PS D:\Semester6\DBT\Assignment1>
```

(b) Queries Creation and Performance Measurement:

- Execute "SELECT *" queries on all tables to display data and count the rows.
- Craft a variety of queries to exercise both index scans and table scans.
- Also include queries with multi-table joins involving 3 tables; including both "SELECT *" and conditional "SELECT" queries with a subset of columns.
- Run Explain/Analyze Plans for above queries and document each of them.

Executing "SELECT *" queries on Users table to display data:

```
3 •  SELECT * FROM Users;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

user_id	username	email	password	reg_date	is_premium	last_login
1	Siri N Shetty	pes2ug22cs556@pesu.pes.edu	ef92b778bafe771e89245b89ecbc08a44a4e166...	2024-05-21 18:43:11	1	2025-03-16 19:23:49
2	traceyramos	ryanwest@example.com	08c8c1be1897b0e09919703fb14ee1ffdeee717...	2024-02-24 14:34:53	1	2025-03-16 19:23:48
3	thomas78	wallacenicole@example.net	d82c4eb8109b67d5507a4d0454efd007afd6105...	2023-07-02 18:41:10	0	2025-03-16 19:23:07
4	garciawilliam	james72@example.net	944f3896521784f3f8c79a4ac35f1f689c66c6a2...	2023-09-04 18:29:47	0	2025-03-16 19:23:08
5	agriffin	jonestina@example.com	b1536789bc837f4e4857c31c536d0e53af93250...	2025-02-26 21:15:15	0	2025-03-16 19:23:21
6	rodriguezmark	alyssa83@example.net	e60a49750642ac049af655593391ecc019150d2...	2023-12-09 22:07:53	1	2025-03-16 19:23:30
7	hatfieldashley	taylorparker@example.net	1c1cdc5025651487a1ec112218d03c49f0da774...	2024-08-07 04:06:02	1	2025-03-16 19:23:14
8	cnolan	smithkatherine@example.net	05c8e865a0c52c8a9dab40a19262848ef0e2674...	2024-03-10 02:13:34	0	2025-03-16 19:23:08
9	smithaudrey	pamelaedwards@example.net	802cfef07dc3ada140b79292c55904dd824bc63...	2024-03-22 20:55:35	1	2025-03-16 19:23:20
10	hoffmanwilliam	john37@example.org	e268e6792dd897326642cc36e9d39426dde0d8...	2024-04-11 09:31:41	0	2025-03-16 19:23:48
11	romerodarryl	longlinda@example.com	baa3a441f30d78cec1b6faf7626fd5e89bc7439f...	2023-07-04 13:06:35	0	2025-03-16 19:23:19
12	adam58	heatherhill@example.net	2093c6fb38ee5aef78298f6e947a71ba8ac3a34...	2024-03-03 13:15:30	0	2025-03-16 19:23:55
13	raymondgarcia	kwhite@example.net	8fdb09f2f1ac103b56ca3116e89e741ac00d2a7...	2024-02-23 03:33:10	0	2025-03-16 19:23:22
14	colleen16	baileyjacqueline@example.net	ace2af9d52d3aa30d970e7b33d33eace4e8f8d1...	2024-09-17 10:29:49	1	2025-03-16 19:22:58
15	johnstewart	rosslance@example.com	e819b4894bae55114d0857cdf17c0c56fd7e910...	2023-11-18 23:28:55	1	2025-03-16 19:23:06
16	brownwendy	trevorgarcia@example.com	8cdda99fbb9bf4ef9bbc5d49c79ae86b8962498...	2023-08-03 16:26:35	1	2025-03-16 19:23:39
17	floreschristine	taylorrachael@example.com	0181bed84dda6781014a2fe0649cf328dee136c...	2023-06-03 13:13:39	0	2025-03-16 19:23:31
18	jacobthomas	petersrobert@example.com	9ee15868c049ab43cf7f38f8935c2b2431fe3042...	2025-01-30 17:50:05	1	2025-03-16 19:23:37
19	stephaniewat...	charlenemiller@example.org	ee1fdbf812e79dc2b049b399274a4fe6c82912d...	2023-12-23 21:56:24	0	2025-03-16 19:23:00
20	marvin16	xfarrell@example.net	bdef1fcce41b468b6f27ed61360a0dd43a9a222...	2023-07-25 10:35:05	1	2025-03-16 19:23:03

user_id	username	email	password	reg_date	is_premium	last_login
982	epeterson	william06@example.org	063b96a77b7ff8b1deb278e654a46687603b6bd...	2024-07-27 14:12:50	0	2025-03-16 19:23:26
983	cynthiastein	megan94@example.com	0ed80a088d179705abf8969fda41c68a08c01a9...	2023-06-28 00:48:28	1	2025-03-16 19:23:55
984	barbarasimpson	jordanmelanie@example.net	2cc271905b5236004cbd84355256329fa79df3c...	2023-10-24 00:44:23	0	2025-03-16 19:23:26
985	kellylee	ssmith@example.com	19eb451cbfca9f04a0fd841d443a531bd5a048c...	2023-12-13 04:05:29	0	2025-03-16 19:23:56
986	kcobb	scott50@example.net	3fd932fb7599977e634a6a235418f9a18eb3668...	2024-09-20 03:21:17	1	2025-03-16 19:23:28
987	harrissarah	ncantu@example.com	702449c4f0c315116a0097d76027915185f2675...	2024-02-02 12:02:16	1	2025-03-16 19:23:56
988	fisherjeffrey	zhughes@example.org	9788801ceb2147e1e9dfd7a64ba87473bc6e310...	2024-01-26 18:32:26	0	2025-03-16 19:23:06
989	frances08	mark07@example.net	28a24956e19b7e715d7a9f4b04bae1bad6e0c8...	2023-05-10 01:09:43	1	2025-03-16 19:23:52
990	comptonraym...	pampatterson@example.com	4178c8a33a0c5f85f1cf8caafdb6a1f5494d9ff1d...	2023-08-02 17:57:36	1	2025-03-16 19:22:59
991	heidihoward	ericbrown@example.org	973978dc0265bc0d70ae1911f574288f2b28774...	2024-09-29 09:59:20	0	2025-03-16 19:23:47
992	jessicamunoz	jeffrey64@example.org	5d552dda20ebd63ce7c419a327c385a5353c042...	2023-12-09 09:00:25	1	2025-03-16 19:23:27
993	katherine15	joshua33@example.com	8659633185f0ece7d0175845401fd26e0fa530b...	2024-08-29 12:41:50	1	2025-03-16 19:23:22
994	michael42	james94@example.org	b5cee8d1be2bbbc70f78f6b97cc288079395a5a...	2024-11-15 05:21:40	1	2025-03-16 19:23:54
995	halljonathan	jessicafields@example.net	a7be1c2bb6cdf1b9185fb65eaf7fedc055dcb95c...	2024-06-19 13:44:47	0	2025-03-16 19:23:46
996	jasonjones	stephenswilliam@example.com	f668d38051f9481b790e63b7bc54601db07ccaa...	2023-09-07 00:49:35	1	2025-03-16 19:23:32
997	tammysingh	johncaldwell@example.org	707e9ce394241dae543378bdbed63c84fe93252...	2023-08-30 04:01:02	0	2025-03-16 19:23:41
998	nicholasoconnor	smiller@example.org	8b2d3c09ac91e7667e233b81b3dc6febc624574...	2024-05-10 22:43:48	0	2025-03-16 19:23:12
999	mlawrence	nicholasanthony@example.net	7684d5daef19a0df60fc27e37afad65fd30a9eaf...	2025-03-03 01:11:26	1	2025-03-16 19:23:52
1000	clintonkelley	jgeorge@example.net	2f4d0ab5a663d5a79dc6eb40cc0514bb73dc688...	2023-09-11 08:34:01	0	2025-03-16 19:23:05
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Executing "SELECT *" queries on Artists table to display data:

```
3 •  SELECT * FROM Artists;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

artist_id	name	country	bio	formed_year
1	Jim Garza	Greece	I option mention safe along. Because involve tell staff film age. Also fill p...	2006
2	Ann Murphy	Niue	Protect economic yet six game trouble. Minute every our our case. Comm...	1993
3	Johnson-Johnson	Gabon	Activity state everything morning run. Moment mouth pay action itself en...	2004
4	Summers-Rogers	Norway	Small sister throw particularly partner. Off alone reason land. Suddenly p...	1999
5	Horn-Wilkins	Saint Lucia	City floor almost country above. Project another certainly world. War ev...	1956
6	Andrew Anthony	Lao People's Democratic Republic	Unit discover keep college relate huge. Security respond almost machine ...	1950
7	Howard-Moore	Latvia	Threat better season true miss. Thing beautiful yes change general exac...	1970
8	Dyer and Sons	Tokelau	Hundred check head finish evening whom health. Hard attack section tak...	1970
9	Gross, Pierce and Johnson	Reunion	Exist Mrs eye particularly local moment whom. Son quality friend live PM b...	1986
10	Stacy Orozco	Nigeria	His produce unit. During finish exactly black. Indicate ball event out betw...	1999
11	Jordan, Wood and Schnei...	Benin	Painting sport could leader section. Quickly artist open become. Certain a...	2016
12	Katelyn Cook	Sudan	Wonder painting forward worry yet market increase. Man computer prof...	1976
13	Katie Smith	Luxembourg	South respond hope various still report despite. Food be up finally. Back f...	1974
14	Emily Ramirez	Jordan	Trial table environmental before whole particular finally. Quite cause pass...	1965
15	Powell, Garcia and Scott	Martinique	Music pull player serious same. Official born west step sport week loss. B...	1952
16	Brittany Jackson	Indonesia	Good shake film. Voice outside movement reveal and democratic teacher....	2001
17	Justin White	Czech Republic	Class give before suddenly throw perhaps tree. Recognize read offer se...	1954
18	Caldwell Ltd	Togo	Painting bad artist position report. War lay career power read. Son bill de...	2009
19	Cruz, Mccarthy and Mitchell	South Georgia and the South Sa...	Despite challenge office like economic decide three white. Issue reason s...	2021
20	Karen Anderson	Philippines	Music resource garden outside without popular building. Decide laugh ne...	1984

artist_id	name	country	bio	formed_year
482	Chelsea Kelly	Liechtenstein	Perhaps computer concern anything. Agreement become close. Contain ...	1981
483	Stephanie Sullivan	Dominican Republic	Senior write bar yes. Party economy response hospital. Provide subject ...	2023
484	Jacob Savage	Sri Lanka	Market energy finish police record president. Group hospital name full lar...	2012
485	Christy Beasley	Qatar	Various life arrive serve race. Than both fear hospital. Character scientis...	1993
486	Wright-Flowers	Indonesia	Follow despite care someone join. Economic condition else individual they ...	1973
487	Rachel Gregory	Dominican Republic	Begin fast bed while. Long face seek peace require. Modern travel indee...	1951
488	Pam Vega	Togo	Recognize city science anything. Glass drive cover someone fear behavio...	2002
489	Fernandez-Torres	Vietnam	Field dark measure positive. Model candidate across would interest leave...	2018
490	Jennifer Calhoun	Iceland	Deal financial issue college base win. Itself heart mission culture number t...	2018
491	Angela Smith	Swaziland	Thus just human role answer treat magazine. Opportunity join happen sh...	2013
492	Eric Baldwin	Canada	Product class itself best different. Politics center will anyone movie behin...	2015
493	Michael Smith MD	Palestinian Territory	Bed in actually back beat already type. Phone several position everybod...	1995
494	Thomas Patterson	Austria	Kitchen voice kid party. Throughout part alone site summer spend run hu...	2002
495	Christian Wallace	Bulgaria	Its candidate attention development. Drive out face off. Side sense impr...	1982
496	Jordan Price	Tajikistan	Talk population eat discover. Politics thus especially entire administration ...	1971
497	Amanda Molina	Norfolk Island	Country analysis require decade sea see. Wish law special her. Hold care...	1961
498	Harrison-Vasquez	Angola	Travel with when old member. Apply state official watch very like. Nice st...	2011
499	Erica Vargas	French Southern Territories	Really network stay eat analysis. Almost culture church establish interesti...	2020
500	Kylie Morris	Peru	Professional foreign benefit soon huge. Body whether physical certainly. ...	2021
NULL	NULL	NULL	NULL	NULL

Executing "SELECT *" queries on Songs table to display data:

```
3 •  SELECT * FROM Songs;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fet

song_id	title	album_id	artist_id	duration	track_number	genre	plays	release_date
1	Number Think Decide Pm	1713	385	266	12	Electronic	328828	2004-05-13
2	Economy	923	125	66	4	Electronic	617241	1969-07-30
3	Page Owner See	1370	162	358	14	Indie	857794	2004-05-24
4	Value Husband Behavior P...	238	219	76	14	Rock	470468	1959-08-06
5	Establish	295	29	522	1	Country	821194	2022-09-16
6	Read By Room	1006	264	341	11	Latin	793558	1979-07-26
7	Again Watch	982	479	563	15	Electronic	912177	1957-02-01
8	Quality Ever Many Particul...	952	414	243	1	Dance	485898	1960-07-08
9	Season Good Most Western	335	49	30	6	Electronic	766441	1995-06-13
10	Explain Trade Care Four C...	33	115	599	8	Electronic	749439	1987-09-09
11	Spring Model	393	299	486	6	Electronic	104651	1965-06-11
12	Single Establish News	849	270	101	17	Alternative	512215	2012-04-05
13	Increase Fund World Eight...	1039	345	552	7	Folk	21847	1968-02-14
14	They Federal Exactly Fly	296	175	63	8	Jazz	4515	1971-03-18
15	Another Piece They	1678	111	295	19	Classical	214275	1966-11-18
16	Vote Plant A What His Cha...	1552	363	119	17	Jazz	863580	2016-01-22
17	Tax Music	683	178	291	9	Jazz	52797	2011-12-05
18	Floor Per Send Try Position	1805	333	288	8	Hip Hop	77986	1968-02-24
19	Former Board Coach Buy ...	1395	488	598	9	Electronic	715862	1970-04-14
20	Outside War Next According	236	259	260	17	Country	663842	1957-06-29

song_id	title	album_id	artist_id	duration	track_number	genre	plays	release_date
11982	Political Maybe Everyone I...	581	213	407	19	Indie	293470	2015-07-20
11983	Son	1177	94	308	12	Jazz	782667	2016-05-15
11984	Short Me	604	454	245	5	Metal	58648	2013-10-07
11985	Job Own Go Cost Differen...	1027	470	40	13	Pop	433291	1993-06-30
11986	Until	754	225	483	18	Folk	549452	2023-09-05
11987	Institution Guess Heavy R...	364	252	234	14	Folk	498254	2001-08-13
11988	Cell Democrat	521	144	483	15	Alternative	749947	1983-01-25
11989	Management School	521	225	323	3	Jazz	448047	1959-12-07
11990	Method Employee Hundre...	1257	234	331	8	Latin	20693	1962-02-18
11991	Cultural Stock	1842	455	188	15	Alternative	271181	1993-08-12
11992	Sure Organization Debate ...	1076	64	267	6	Rock	870802	2000-11-01
11993	Involve Put Production Inc...	1116	478	503	11	Country	822286	2005-05-07
11994	Energy	1392	428	573	11	Folk	525451	2016-03-15
11995	Building Before Actually He...	1080	7	297	7	Blues	210984	1993-06-22
11996	Even Environment Describ...	447	256	138	14	Blues	184665	1990-09-05
11997	Real	1789	146	88	19	Jazz	304548	1989-07-18
11998	Culture Structure	933	131	583	3	Blues	959183	2022-09-01
11999	Others Film Young	1252	72	481	16	Rock	571739	1971-04-12
12000	Part	1513	347	439	13	Electronic	40190	1988-02-27
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Executing "SELECT *" queries on Albums table to display data:

```
3 •  SELECT * FROM Albums;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

album_id	title	artist_id	release_date	genre	total_tracks
1	Attack Success Heavy	422	2020-09-05	Jazz	16
2	Several Serve	319	2023-04-18	Pop	8
3	Also	347	1964-05-07	Metal	1
4	First State Feel	59	1986-02-22	Country	18
5	Company Model Information White Among	338	2007-03-17	Rock	12
6	Industry	328	2012-09-06	Blues	12
7	President	258	1994-09-09	Country	8
8	Mind Individual Budget	411	1957-01-09	R&B	16
9	Time Name	356	1978-12-18	Classical	16
10	Collection	153	2022-11-10	Latin	4
11	Relationship	97	1961-08-20	Reggae	1
12	Business Environment Child	228	1960-05-31	Jazz	8
13	Owner General	363	2012-12-20	Latin	14
14	Rule	77	2013-08-31	Indie	1
15	Deal Never Move Blue	258	2024-04-07	Jazz	15
16	Station Art	481	1964-11-01	Country	20
17	Art	381	2023-10-27	Country	11
18	Side List	284	2001-08-21	Indie	20
19	Marriage	179	1957-06-11	Indie	18
20	Laugh Event Move Chance	311	1973-01-18	Reggae	3

album_id	title	artist_id	release_date	genre	total_tracks
1982	Camera Control Sure Describe Entire	370	1998-12-21	Country	12
1983	Daughter High Growth	489	1963-03-08	Pop	17
1984	Energy Care	304	1967-05-06	Alterna...	2
1985	Without Democratic	233	2005-08-28	Jazz	11
1986	Fly Leave	233	1981-07-15	Rock	3
1987	Sing Yes	197	2000-11-21	Rock	16
1988	Mr Attorney Figure	54	2011-06-23	Latin	1
1989	Together Deal	162	1964-01-10	Classical	6
1990	Drive Measure Heart	459	2002-05-06	Folk	6
1991	Recent	104	1956-08-27	Dance	14
1992	Ago Be Pretty Church Stage	462	2009-11-27	Metal	12
1993	Indeed Magazine	47	1979-08-13	Folk	19
1994	Response	251	2014-12-14	Indie	3
1995	Southern Moment Human Inside Because	299	1984-05-16	Country	9
1996	These	13	1978-05-13	Metal	8
1997	Bag Increase Nor	490	2019-11-05	Country	8
1998	Receive Fact Discuss Six Police	362	1969-12-03	Pop	11
1999	Short Section Trouble Knowledge	298	2006-07-16	Jazz	2
2000	Indicate Center	110	1985-05-29	Classical	14
NULL	NULL	NULL	NULL	NULL	NULL

Executing "SELECT *" queries on Playlists table to display data:

```
3 •   SELECT * FROM Playlists;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: | Fetch rows:

playlist_id	user_id	name	description	is_public	created_date	last_updated
1	101	Game Those Middle Alone	Six popular rather comm...	1	2024-02-11 15:52:34	2024-12-19 14:04:38
2	973	Able Myself Story Believe	NULL	1	2023-09-22 23:56:43	2025-02-14 21:04:26
3	268	Write During Cultural	Toward charge guess tr...	1	2024-03-08 09:03:02	2024-08-22 04:55:24
4	249	Few	Week any analysis upon...	1	2024-08-31 17:49:48	2024-12-04 14:37:17
5	287	Conference School Value Effort	NULL	0	2025-01-14 19:06:14	2025-02-01 09:47:20
6	49	Debate Method	Why individual anything ...	1	2023-04-18 08:04:00	2024-08-10 01:28:16
7	455	Analysis	Live suddenly structure l...	1	2024-11-14 15:06:27	2024-12-05 02:02:39
8	48	Table	NULL	1	2024-12-16 11:27:35	2025-01-12 18:42:07
9	378	Sure Region	NULL	1	2023-12-20 10:59:55	2024-01-13 06:23:38
10	131	Thing Give Goal Medical	Identify sure leg hour. ...	1	2023-11-28 16:36:44	2024-08-07 16:17:56
11	739	Force Itself	NULL	0	2023-08-28 13:01:11	2024-09-19 18:41:50
12	624	Age Difference Tv	Design success yard kee...	1	2024-09-09 19:41:05	2024-12-25 17:27:56
13	552	Into No	Huge military late toget...	1	2024-06-10 22:38:22	2024-07-05 14:55:04
14	36	Agree Magazine Once	Travel to music challeng...	1	2024-09-21 17:54:16	2024-12-04 15:58:51
15	335	Rather	Above culture compare ...	1	2023-07-29 03:10:29	2024-12-17 12:22:56
16	172	Partner	Painting kitchen cause h...	1	2024-12-10 09:29:40	2025-03-05 22:41:37
17	836	Around Instead	NULL	1	2023-09-27 05:22:18	2024-09-12 20:51:41
18	879	Him Throughout	NULL	1	2023-07-19 23:02:24	2023-11-01 16:24:54
19	211	Have Everybody Man	Society garden peace w...	1	2023-08-24 01:43:55	2024-12-31 13:00:04
20	122	House Rate	Election sense court ow...	1	2023-09-29 13:29:45	2024-06-09 06:53:06

playlist_id	user_id	name	description	is_public	created_date	last_updated
2982	368	Probably Glass Southern	NULL	1	2024-12-16 06:56:29	2025-01-18 03:51:55
2983	569	Quickly After	NULL	1	2024-10-01 16:15:05	2024-11-06 07:02:51
2984	823	Majority	NULL	1	2023-11-12 06:01:32	2024-11-08 18:56:12
2985	92	Condition Work	Small word issue. Art tre...	1	2023-09-22 15:06:03	2024-03-14 04:53:12
2986	133	Wind By	Between car I control. T...	1	2025-03-15 14:02:38	2025-03-15 16:08:15
2987	11	Necessary Treatment Thus	NULL	1	2023-09-17 23:17:15	2024-01-21 11:21:40
2988	952	Start	Community cover situati...	1	2024-09-25 21:56:17	2025-02-28 05:28:22
2989	261	Where	Compare sea live save d...	1	2023-12-21 11:42:34	2025-01-07 18:26:24
2990	864	Cold	Local writer hot surface ...	1	2024-07-23 14:31:31	2025-01-25 17:58:26
2991	600	Nature Most Such	Again open possible incr...	1	2023-07-16 22:05:56	2024-03-10 20:59:00
2992	128	Design Though Cut Thus	Feel like market capital ...	0	2023-07-27 03:36:30	2025-02-14 04:34:05
2993	934	Region	Discuss point prove. Rig...	1	2023-08-15 19:01:53	2024-08-06 10:02:34
2994	484	Wish Government Sea Look	How college exactly mo...	1	2023-08-25 15:14:30	2024-05-31 13:11:54
2995	806	Foot Class	Movie themselves foreig...	1	2024-07-09 23:05:46	2024-10-12 19:55:52
2996	722	Course Good Policy	NULL	1	2024-08-12 11:42:27	2024-09-04 20:39:20
2997	165	Perhaps Half	Talk very senior. Fast a...	1	2023-07-31 05:30:22	2024-07-21 07:34:16
2998	811	Friend Key Seem Anything	Gas myself appear war. ...	1	2023-11-02 17:25:07	2024-05-17 09:24:32
2999	33	Again Painting	NULL	1	2023-09-09 00:46:21	2024-05-31 13:58:22
3000	903	Democratic	NULL	1	2024-12-27 19:59:26	2025-01-28 20:53:52
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Executing "SELECT *" queries on PlaylistSongs table to display data:

```
3 •   SELECT * FROM PlaylistSongs;
```

Result Grid | Filter Rows: | Edit:

playlist_id	song_id	added_date	added_by
1	9759	2024-12-22 22:36:08	77
1	9833	2024-05-16 06:30:33	881
1	11694	2023-05-05 22:10:03	84
2	2045	2024-11-18 05:55:04	949
2	3947	2024-07-06 00:35:52	111
2	5648	2023-06-02 00:21:53	532
2	5858	2023-04-16 09:50:00	117
2	10795	2024-03-01 02:31:42	60
3	20	2023-07-11 08:26:05	329
3	72	2024-03-22 14:03:02	55
3	3283	2024-10-06 16:28:32	986
3	6842	2024-07-24 03:21:34	466
3	7395	2024-05-06 01:13:57	611
3	7596	2023-08-02 00:20:43	895
4	492	2024-10-02 17:39:07	309
4	1135	2023-03-31 14:57:11	333
4	2809	2024-03-12 21:19:39	53
4	3850	2023-06-01 09:19:15	264
4	6290	2024-03-26 07:00:58	980
4	6322	2023-04-05 18:39:58	195

playlist_id	song_id	added_date	added_by
2996	10980	2024-09-19 16:28:56	603
2996	11500	2024-08-29 04:08:09	788
2996	11947	2023-04-01 11:17:29	145
2997	2030	2024-05-07 19:53:34	506
2997	5731	2023-05-31 18:01:50	669
2997	9585	2023-04-24 03:58:18	265
2998	2007	2024-04-13 01:43:56	494
2998	3973	2024-06-07 02:39:42	552
2998	4023	2023-05-30 22:19:15	765
2998	6956	2024-10-16 04:35:51	101
2998	7323	2025-03-04 23:47:16	540
2998	8803	2024-09-13 16:06:16	939
2999	9154	2025-02-14 14:17:00	87
2999	10157	2024-03-21 02:26:04	86
3000	851	2023-07-12 20:14:41	893
3000	4444	2024-05-04 06:07:43	255
3000	6162	2023-11-06 00:07:25	820
3000	10210	2024-10-03 01:16:17	18
3000	11876	2024-03-09 05:57:12	168
NULL	NULL	NULL	NULL

Executing "SELECT *" queries on all tables to count the rows:

```
mysql> SELECT COUNT(*) FROM Users;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Artists;
+----------+
| COUNT(*) |
+----------+
|      500 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Albums;
+----------+
| COUNT(*) |
+----------+
|     2000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Songs;
+----------+
| COUNT(*) |
+----------+
|    12000 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM Playlists;
+----------+
| COUNT(*) |
+----------+
|     3000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM PlaylistSongs;
+----------+
| COUNT(*) |
+----------+
|    15000 |
+----------+
1 row in set (0.01 sec)
```

Index scan queries (using primary key)
Accessing data using user_id from Users table

```
mysql> SELECT * FROM Users WHERE user_id = 100;
+---------+-------------+--------------------------+------------------------------------------------------------------+---------------------+------------+---------------------+
| user_id | username    | email                    | password                                                         | reg_date            | is_premium | last_login          |
+---------+-------------+--------------------------+------------------------------------------------------------------+---------------------+------------+---------------------+
|     100 | turnerpatrick | campbelljulia@example.com | f8c37f5e694b5d5baff3d60d9c6a7d5c794e192986f5d718710397cf0d659098 | 2023-07-30 20:40:40 |          0 | 2025-03-16 19:23:21 |
+---------+-------------+--------------------------+------------------------------------------------------------------+---------------------+------------+---------------------+
1 row in set (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM Users WHERE user_id = 100;
+-------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                               |
+-------------------------------------------------------------------------------------------------------+
| -> Rows fetched before execution  (cost=0..0 rows=1) (actual time=100e-6..200e-6 rows=1 loops=1)
 |
+-------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Accessing data using song_id from Songs table

```
mysql> SELECT * FROM Songs WHERE song_id = 5000;
+---------+-----------------------------+----------+-----------+----------+--------------+-------+---------+--------------+
| song_id | title                       | album_id | artist_id | duration | track_number | genre | plays   | release_date |
+---------+-----------------------------+----------+-----------+----------+--------------+-------+---------+--------------+
|    5000 | Despite Land Onto Structure |     1994 |       317 |      188 |           12 | Indie | 152381 | 2010-01-13   |
+---------+-----------------------------+----------+-----------+----------+--------------+-------+---------+--------------+
1 row in set (0.00 sec)

mysql> EXPLAIN ANALYZE SELECT * FROM Songs WHERE song_id = 5000;
+-------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                               |
+-------------------------------------------------------------------------------------------------------+
| -> Rows fetched before execution  (cost=0..0 rows=1) (actual time=100e-6..100e-6 rows=1 loops=1)
 |
+-------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

In the given query execution, the cost and time are low because indexed scans are being utilized. Both **user_id** and **song_id** are primary keys, meaning they have built-in indexes that allow for efficient lookups. When a query searches for a specific primary key value, the database can quickly locate the required row using index-based retrieval, rather than scanning the entire table.

Table scan queries (not using indexed columns)

Accessing data of a particular genre (here: Rock)

```
mysql> SELECT * FROM Songs WHERE genre = 'Rock';
```

song_id	title	album_id	artist_id	duration	track_number	genre	plays	release_date
4	Value Husband Behavior Public Compare Organization	238	219	76	14	Rock	470468	1959-08-06
53	Still Leg Outside Change	1493	75	57	16	Rock	196707	2004-05-11
61	Again	1477	450	470	12	Rock	726857	2019-03-13
64	West Late	1117	248	126	5	Rock	832046	1986-02-24
90	Onto	547	261	178	6	Rock	840920	1993-06-08
94	Lawyer Expect Actually Cost Level	1968	201	386	3	Rock	853568	1969-08-31
99	Apply Of	994	44	182	11	Rock	128112	2005-11-29
103	Now Month Situation Purpose	354	422	97	1	Rock	667193	1960-10-30
128	Major Who	25	269	170	4	Rock	915788	1969-08-10
163	Different Wall May Film Base Establish	554	168	587	19	Rock	215687	2013-01-04
164	Strategy Total Deal Never	1642	16	297	18	Rock	762554	1994-08-23
166	Oil Who	1708	34	445	20	Rock	961080	2002-04-08
179	Yet Later Save Unit Never Red	1301	493	549	20	Rock	904812	2023-03-22
184	Treat Summer American President	378	31	41	18	Rock	380438	1977-07-10
277	Couple Beautiful Analysis Audience Very	138	378	84	18	Rock	143121	2004-10-07
287	Material Truth Stand Fish Project	638	223	557	8	Rock	303193	1965-07-28
296	Stock Discuss	598	143	122	18	Rock	917688	2011-10-28
301	Toward Although Leg She Stage Tough	769	8	148	11	Rock	884973	1985-12-07
310	Trial Reflect High	610	14	309	14	Rock	628898	1980-04-04
315	Lawyer Out Military Offer Job	129	49	483	7	Rock	974360	1983-03-13
345	Check Huge	1023	279	536	15	Rock	359095	1996-06-07
425	List Occur Everybody	321	148	560	9	Rock	108439	2014-09-09
428	Image To Become Out	1202	74	55	4	Rock	116278	1987-10-19
455	Generation	997	349	174	2	Rock	11986	2010-10-17
470	Also	772	268	434	19	Rock	797611	1965-08-25
484	Long Guess Her	1395	251	251	9	Rock	148002	1960-03-21
499	Bit Though Tend	1842	5	479	1	Rock	309809	2023-10-23
520	Summer Become Very Senior	597	156	304	17	Rock	338213	1985-12-27
553	Sport Old Benefit	688	370	160	5	Rock	809969	1970-10-27
556	Push	937	357	451	2	Rock	453001	1984-08-02
589	Event Concern	545	478	115	9	Rock	493656	1980-10-27
595	How Yes War	783	366	412	8	Rock	914416	2013-09-03
598	Century	519	24	432	9	Rock	248058	1991-02-23
606	Recently	512	245	493	13	Rock	696429	1957-04-12
623	Modern	1864	364	181	3	Rock	522995	2000-02-24
632	Case Firm Beautiful Between	414	417	144	3	Rock	945702	1965-06-25
645	Simply Moment With Respond Feel Fill	1811	187	281	18	Rock	529021	1999-10-15
652	Front	939	23	411	17	Rock	906772	1974-03-12
656	Better Painting Name Education Wide	102	325	584	6	Rock	930070	1977-07-23
665	If Win Per New Government Too	328	283	312	20	Rock	775213	2002-01-31

(The output has 735 rows)

```
mysql> EXPLAIN ANALYZE SELECT * FROM Songs WHERE genre = 'Rock';
+-----------------------------------------------------------------------------------------------
| EXPLAIN
+-----------------------------------------------------------------------------------------------
| -> Filter: (songs.genre = 'Rock')  (cost=1233 rows=1209) (actual time=0.14..16.6 rows=735 loops=1)
    -> Table scan on Songs  (cost=1233 rows=12090) (actual time=0.133..14.6 rows=12000 loops=1)
 |
+-----------------------------------------------------------------------------------------------
1 row in set (0.02 sec)
```

Accessing data of Users who have a premium plan

```
mysql> SELECT * FROM Users WHERE is_premium = TRUE;
```

user_id	username	email	password	reg_date	is_premium	last_login
1	Siri N Shetty	pes2ug22cs556@pesu.pes.edu	ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f	2024-05-21 18:43:11	1	2025-03-16 19:23:49
2	traceyramos	ryanwest@example.com	08c8c1be1897b0e09919703fb14ee1ffdeee717ce8ec174cf7282b2887493bcb	2024-02-24 14:34:53	1	2025-03-16 19:23:48
6	rodriguezmark	alyssa83@example.net	e60a49750642ac049af655593391ecc019150d2ea684767a61d1bae0ce4836fb	2023-12-09 22:07:53	1	2025-03-16 19:23:30
7	hatfieldashley	taylorparker@example.net	1c1cdc5025651487a1ec112218d03c49f0da77452825aeabfd8629b64e37aa24	2024-08-07 04:06:02	1	2025-03-16 19:23:14
9	smithaudrey	pamelaedwards@example.net	802cfef07dc3ada140b79292c55904dd824bc63496fe03e55adbafbef3dda85b	2024-03-22 20:55:35	1	2025-03-16 19:23:20
14	colleen16	baileyjacqueline@example.net	ace2af9d52d3aa30d970e7b33d33eace4e8f8d113920f5194ae38b1c94b23f3e	2024-09-17 10:29:49	1	2025-03-16 19:22:58
15	johnstewart	rosslance@example.com	e819b4894bae55114d0857cdf17c0c56fd7e91013534d8349605299fab9e2740	2023-11-18 23:28:55	1	2025-03-16 19:23:06
16	brownwendy	trevorgarcia@example.com	8cdda99fbb9bf4ef9bbc5d49c79ae86b89624981c0c3c9d64a1eca8030a7e69e	2023-08-03 16:26:35	1	2025-03-16 19:23:39
18	jacobthomas	petersrobert@example.com	9ee15868c049ab43cf7f38f8935c2b2431fe3042f8357bc8442b77a9d06fdf4c	2025-01-30 17:50:05	1	2025-03-16 19:23:37
20	marvin16	xfarrell@example.net	bdef1fcce41b468b6f27ed61360a0dd43a9a222cc55bd13d40358b6bbf00b5c7	2023-07-25 10:35:05	1	2025-03-16 19:23:03
23	raymondwallace	smurray@example.net	30a2b672698e656fd69fd494ea5e8d9a356a5ef9bea58bb25173603e5191573a	2024-08-05 00:06:40	1	2025-03-16 19:23:21
24	wesleycervantes	iayala@example.com	a8c9b811c082d2ff86f2a1ed17b3e5e2ed84220cdb88fe232bce877974157fc0	2023-10-25 12:21:27	1	2025-03-16 19:23:26
26	nkelly	haynesmichael@example.org	5b272e3dd7c6c7c56bce3b1b63ff3ad89ded021a9e4d3b3171a23f9b75b22192	2024-06-06 12:02:03	1	2025-03-16 19:23:04
27	ycurry	jessegrant@example.net	5c0bc278233d5c60424b70d11e33e1505a86ceb2e7e6374769158ea62581ac2e	2023-08-18 17:58:39	1	2025-03-16 19:23:13
28	jenningssean	shawnpatel@example.com	242017892dcd49e8cd2ae07175525b75b7dc1f66eec4cf899368af0ffb56a834	2024-06-26 19:37:42	1	2025-03-16 19:23:46
29	leestephanie	proberts@example.com	650e9b975b39ef9e89cb91743b022007ea96610e9f3f945a8541a1999c2eea95	2023-07-20 00:00:06	1	2025-03-16 19:23:55
30	walkermatthew	mreeves@example.org	a3460bb87d86bb4719f9a417cd745a547439613f9618e683be04dd853d13a7e7	2024-04-05 15:46:04	1	2025-03-16 19:22:56
33	steve17	yolandagonzalez@example.com	6f878dbbd0dd99a72d2ad0aa368c841a6c1acec5fc4d42a4c909ad0b83ecfcd0	2024-04-04 18:26:12	1	2025-03-16 19:23:41
38	pstanton	sgalloway@example.org	07b1d8ceadf1e845cf657000ba3a2eb1d1e4c2f1121668625699f6509b3ce2d2	2024-03-16 04:25:59	1	2025-03-16 19:23:20
39	sarah33	rachelcooper@example.org	9df1932e718720981475ef683689dd3f4a6eece6e6b5d1338377ad5b0a39eab5	2024-04-12 16:59:45	1	2025-03-16 19:23:28
43	vsanders	cervantescatherine@example.com	a737703ee9724b0e32818117aaade9ac1784645c14a439e6ab8c182a3888cee9	2025-03-11 15:12:29	1	2025-03-16 19:23:08
44	joshuagutierrez	rschultz@example.net	4f4d23b250e0f6b3560ce4e09c2a846e50911f48e75cf073be4215dfdd059b42	2025-02-02 20:25:45	1	2025-03-16 19:23:34
45	gavila	sabrinamorales@example.com	accf37b77d3fe8b49642a474789b1437490b1f0cac2041ba2c2fa91970da0941	2023-04-18 10:46:31	1	2025-03-16 19:23:50
47	lindatorres	ubarber@example.com	c0bc31f1def5ee67a2dfcb8c381a0b4fd5b647df366793979daf265e43ad3576	2023-11-06 17:05:34	1	2025-03-16 19:23:06
49	terryward	markmeadows@example.net	131fbeea55160c27fc48a6e7787a2ca040eb873b160bb55aca3604e082b4faac	2025-02-05 02:49:43	1	2025-03-16 19:23:41
52	qharrington	jeremyyang@example.net	875553ca3c0d04b1b8916de27e78a62bee3682b7b161ee246e6a130a9af757f5	2024-05-13 04:35:21	1	2025-03-16 19:23:24
54	christopher91	jamesstewart@example.net	a11882bccf20829fe6a2a64c0ad7e13cc09718e9d6baac2fd3dc7efab195185c	2024-06-28 13:36:14	1	2025-03-16 19:23:38
55	emilyclark	dominique35@example.org	235c81849bbbf5aafa7670296e76bf9fc1e590d16a13e48ccb76972102113b96	2024-06-14 11:07:59	1	2025-03-16 19:23:38
59	lauren15	mary33@example.com	8e0b2d7fd05c471aaf85e0ab12c2b3e7d9884d1083e599b8bbcb6dcb54e3b1a9	2023-10-03 06:52:12	1	2025-03-16 19:23:47
60	foleymonica	wilsonlisa@example.net	a62bf7ef42dceeab7b2a722f76a6d842f0b20118bf94660ed2362fb6465012dd	2024-06-04 13:54:52	1	2025-03-16 19:23:13
63	mcbridebrian	hortoncaitlin@example.net	eca0038cdc2fadd3b509015c2434e2dce5add8aec7481cce479f635c707e4b28	2024-09-26 10:56:30	1	2025-03-16 19:23:29
65	lsalazar	lowealexa@example.com	671b28cd81d1edc623906098964cae812a713d6c03cd6323c9cf750fc7768887	2024-08-22 05:41:03	1	2025-03-16 19:23:33
67	cobbvalerie	adamskeith@example.org	34be628bc837879a282b1325894dca25f9ae225ac5d27472f8acef6062d31a6a	2023-04-04 08:44:58	1	2025-03-16 19:23:44
70	carlsoncourtney	tanya21@example.com	5f31df039b0d41249232ac5285d7843c46effb6b08d7b67073462190a0dfe445	2023-09-21 06:24:04	1	2025-03-16 19:23:36
71	lcunningham	fosterdanny@example.net	2b5462ce74bd8c74708322bdbc56f425d1168328a52462621ea21bbc1ba5f80d	2024-04-01 20:03:47	1	2025-03-16 19:23:54
73	rickyking	yjohnson@example.com	bd617b27c0f15c7b4d59afd6da2deda4fe2b73288eeb8048d52e194b8021a0d5	2024-07-03 19:42:15	1	2025-03-16 19:23:28
75	jonathanwright	qdavis@example.org	b7ae321f11e57b85a0445f4e9f407be803daa098230a2c1b8babe06e028f6563	2023-09-02 11:20:24	1	2025-03-16 19:23:11
76	leejennifer	apalmer@example.org	d06cf624c206c160c0f23388c34f8b8a38791391c0ee0d50170aea5c4ba85191	2023-03-24 05:57:26	1	2025-03-16 19:23:51
79	dholland	bmitchell@example.org	f5b540a9acddb88dfd10ba721a826e7b6f199a3b38a0f1d91f8f36b63c07ddcd	2024-03-31 23:09:39	1	2025-03-16 19:23:56
80	briana13	barnesjames@example.net	cf3878c5225230637644d2d17797bb8a763173ffde6e3bf1966c0d0f8795b407	2024-08-18 05:50:10	1	2025-03-16 19:23:37
84	uturner	kristin52@example.org	c0aea14749eaa884c6ec7a39a97080fd7e5ddf74af5405f57289e0c44d54e06d	2024-04-29 11:45:22	1	2025-03-16 19:23:06

(The output has 490 rows)

```
mysql> EXPLAIN ANALYZE SELECT * FROM Users WHERE is_premium = TRUE;
+----------------------------------------------------------------------------------------------------------
| EXPLAIN
+----------------------------------------------------------------------------------------------------------
| -> Filter: (users.is_premium = true)  (cost=103 rows=100) (actual time=0.0789..0.606 rows=490 loops=1)
    -> Table scan on Users  (cost=103 rows=1000) (actual time=0.0765..0.538 rows=1000 loops=1)
 |
+----------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

These queries take more time because they perform table scans, meaning the database must check every row. The title search lacks an index, causing a high cost. **This makes execution significantly slower.**

Three-table join with SELECT * involving 3 tables

```sql
SELECT *
FROM Playlists p
JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
JOIN Songs s ON ps.song_id = s.song_id
WHERE p.user_id = 50;
```

playlist_id	user_id	name	description	is_public	created_date	last_updated	playlist_id	song_id	added_date	added_by	song_id	title	album_id	artist_id	duration	track_number	genre	plays	release_date
209	50	Production Their	Suggest any moment su...	1	2024-09-16 05:19:48	2024-12-18 07:37:10	209	3336	2024-04-12 19:43:50	90	3336	Board Support Us List Six	509	163	345	6	Electronic	91561	1984-02-04
209	50	Production Their	Suggest any moment su...	1	2024-09-16 05:19:48	2024-12-18 07:37:10	209	7052	2023-09-22 19:26:38	160	7052	Ground Add	959	72	163	4	Jazz	189178	1990-05-26
209	50	Production Their	Suggest any moment su...	1	2024-09-16 05:19:48	2024-12-18 07:37:10	209	8095	2023-07-23 13:24:26	245	8095	Mouth Surface Start Orga...	289	423	597	12	Classical	72049	1984-01-15
209	50	Production Their	Suggest any moment su...	1	2024-09-16 05:19:48	2024-12-18 07:37:10	209	8519	2023-10-08 07:43:52	398	8519	Section Black Hundred Tra...	1826	462	296	9	Electronic	943860	1975-02-03
209	50	Production Their	Suggest any moment su...	1	2024-09-16 05:19:48	2024-12-18 07:37:10	209	11614	2024-04-09 18:00:02	313	11614	War Think Risk Important	1963	353	329	16	Jazz	977431	2002-04-23
254	50	Level Without Condition	Decision which know bac...	1	2024-09-04 12:26:30	2024-09-29 10:17:28	254	1347	2024-09-15 06:43:23	835	1347	Physical Thus Would Entire	1719	406	365	5	Electronic	399573	1957-08-04
254	50	Level Without Condition	Decision which know bac...	1	2024-09-04 12:26:30	2024-09-29 10:17:28	254	10462	2023-06-12 03:04:31	848	10462	Hair	1086	486	95	19	Metal	206446	1981-04-08
254	50	Level Without Condition	Decision which know bac...	1	2024-09-04 12:26:30	2024-09-29 10:17:28	254	10630	2024-06-19 14:21:40	222	10630	Ready Force Pressure Water	1279	221	174	20	Latin	501770	2000-09-06
1132	50	Himself	NULL	1	2023-09-06 07:03:25	2024-05-28 18:52:19	1132	1423	2024-01-16 00:05:34	108	1423	Explain Possible Guess Speak	1229	241	430	15	Dance	190981	1974-07-14
1132	50	Himself	NULL	1	2023-09-06 07:03:25	2024-05-28 18:52:19	1132	5721	2024-07-30 07:56:18	375	5721	Especially	224	341	455	6	Country	120850	1968-10-03
1132	50	Himself	NULL	1	2023-09-06 07:03:25	2024-05-28 18:52:19	1132	10151	2024-09-13 13:00:34	464	10151	Take Last Store	1374	338	93	15	Indie	233283	1998-07-03
1168	50	Side Detail Set Recognize	Industry stock mean ow...	1	2024-10-10 02:24:20	2024-11-14 06:53:02	1168	120	2023-05-03 07:24:30	451	120	Radio Owner Area Also Co...	311	400	439	19	Classical	706668	2000-11-03
1168	50	Side Detail Set Recognize	Industry stock mean ow...	1	2024-10-10 02:24:20	2024-11-14 06:53:02	1168	6097	2023-12-02 22:36:19	636	6097	Go Not	753	268	50	13	R&B	899459	2017-05-01
1168	50	Side Detail Set Recognize	Industry stock mean ow...	1	2024-10-10 02:24:20	2024-11-14 06:53:02	1168	9880	2023-12-09 04:28:22	309	9880	Boy Where His Pull Science...	553	466	290	20	Electronic	26394	1986-04-01
1246	50	Fire	Of life food operation pr...	0	2023-06-09 13:19:36	2024-06-11 08:37:49	1246	202	2024-05-27 17:41:20	926	202	Stage Arrive System Liste...	881	356	588	4	Electronic	727764	1979-07-01
1246	50	Fire	Of life food operation pr	0	2023-06-09 13:19:36	2024-06-11 08:37:49	1246	462	2024-07-03 21:10:14	129	462	Add Hope Economy Yeah	1867	202	143	5	Metal	744247	1984-03-01

```
mysql> EXPLAIN ANALYZE
    -> SELECT *
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN Songs s ON ps.song_id = s.song_id
    -> WHERE p.user_id = 50;
+------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=19.4 rows=33.9) (actual time=0.0522..0.204 rows=26 loops=1)
    -> Nested loop inner join  (cost=7.6 rows=33.9) (actual time=0.0436..0.117 rows=26 loops=1)
        -> Index lookup on p using user_id (user_id=50)  (cost=2.45 rows=7) (actual time=0.0288..0.0499 rows=7 loops=1)
        -> Index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.321 rows=4.84) (actual time=0.00687..0.00904 rows=3.71 loops=7)
    -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.253 rows=1) (actual time=0.00306..0.00308 rows=1 loops=26)
 |
+------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

Higher Execution Time & Cost

- More tables joined: Playlists, PlaylistSongs, Songs, and Artists.
- Filtering Conditions: p.is_public = TRUE and s.genre = 'Pop'.
- Nested Loop Joins:
  - Multiple loops due to filtering on non-indexed columns (s.genre = 'Pop').
  - Filtering happens late, causing unnecessary row retrieval.
- High Execution Time: Filtering + table scan + multiple loops = **Slower performance**.

Three-table join with conditional SELECT and subset of columns involving 3 tables:

```
mysql> SELECT p.name AS playlist_name, s.title AS song_title, s.duration, a.name AS artist_name
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN Songs s ON ps.song_id = s.song_id
    -> JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE AND s.genre = 'Pop';
+-------------------------------------------+-----------------------------------------------------+----------+---------------------------------+
| playlist_name                             | song_title                                          | duration | artist_name                     |
+-------------------------------------------+-----------------------------------------------------+----------+---------------------------------+
| Able Myself Story Believe                 | Agreement Necessary Pattern Almost                  |      169 | Geoffrey Kelley                 |
| Analysis                                  | There                                               |      359 | Edward Jones                    |
| Sure Region                               | Group That Perform Accept Remain Third              |      516 | Santiago-Cooke                  |
| Sure Region                               | Image Statement                                     |       98 | Williams-Mcguire                |
| Around Instead                            | Generation                                          |      204 | Richard Matthews                |
| Around Instead                            | Price Day Common Democratic Morning                 |      404 | Freeman-Juarez                  |
| Him Throughout                            | Guy Decade Former                                   |      122 | Misty Snyder                    |
| Him Throughout                            | Level Make Structure Possible Today                 |      240 | Simon-Mitchell                  |
| Have Everybody Man                        | Management View Sure Economic                       |      480 | Terry Walker                    |
| House Rate                                | Skill                                               |      234 | Melissa Peterson                |
| House Rate                                | Special Role Career Art                             |      416 | Bruce Torres                    |
| Serious Often                             | Power Like Color Create                             |       51 | Osborne-Williams                |
| Determine Current                         | Heart Seven Speak Arm Participant Training          |      121 | Ann Murphy                      |
| Keep Anything                             | Run Community Condition More Rock Purpose           |      579 | Daniel Rowe                     |
| Official Too                              | Race Goal Win Late Its Operation                    |      375 | Munoz PLC                       |
| Story Use Could Issue                     | Voice As Executive Side                             |      249 | Bethany Herrera                 |
| Tv Win Fund Body                          | Detail                                              |      588 | Ariana Watson                   |
| Which Result                              | Into Key Film                                       |      208 | Welch PLC                       |
| Which Result                              | Page Education Strong Congress Pass                 |      227 | Stacy Galvan                    |
| Option Evening Sell                       | Myself Feel Truth                                   |       70 | Daniel Anderson                 |
| Option Evening Sell                       | Pick Team President On Citizen Radio                |      103 | Donald Reynolds                 |
| Share Second Find Mission                 | Check Color True Site                               |      457 | Dawson-Martin                   |
| Present Rock                              | Run Community Condition More Rock Purpose           |      579 | Daniel Rowe                     |
| Exist Sing Exist                          | Certainly This Name Nice                            |      164 | David Vazquez                   |
| Audience Inside Later                     | Agency Middle Seat Sound                            |      222 | Boyer, Lawson and Hayden        |
| Audience Inside Later                     | Office Current Official                             |      326 | James Bell                      |
| Subject                                   | American Charge Fill                                |      418 | Lindsey Miller                  |
| Investment Say People                     | Trial Who                                           |      171 | Eddie Valenzuela                |
| Trouble                                   | Indicate Fish                                       |      223 | Cook PLC                        |
```

(The output has 749 rows)

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, s.duration, a.name AS artist_name
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN Songs s ON ps.song_id = s.song_id
    -> JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE AND s.genre = 'Pop';
+---------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+---------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=1069 rows=143) (actual time=0.151..56.5 rows=749 loops=1)
    -> Nested loop inner join  (cost=1019 rows=143) (actual time=0.146..54.9 rows=749 loops=1)
        -> Nested loop inner join  (cost=520 rows=1425) (actual time=0.108..20.9 rows=12006 loops=1)
            -> Filter: (p.is_public = true)  (cost=303 rows=295) (actual time=0.0859..2.81 rows=2419 loops=1)
                -> Table scan on p  (cost=303 rows=2947) (actual time=0.0841..2.4 rows=3000 loops=1)
            -> Covering index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.254 rows=4.84) (actual time=0.00446..0.00696 rows=4.96 loops=2419)
        -> Filter: ((s.genre = 'Pop') and (s.artist_id is not null))  (cost=0.25 rows=0.1) (actual time=0.00269..0.00269 rows=0.0624 loops=12006)
            -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.25 rows=1) (actual time=0.00225..0.00228 rows=1 loops=12006)
    -> Single-row index lookup on a using PRIMARY (artist_id=s.artist_id)  (cost=0.251 rows=1) (actual time=0.00193..0.00196 rows=1 loops=749)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.06 sec)
```

Lower Execution Time & Cost

- Fewer tables joined: Playlists, PlaylistSongs, and Songs.
- Filtering Condition: user_id = 50.
- Efficient Index Usage:
    - Uses an index on user_id, leading to direct lookups.
    - No unnecessary row scans, reducing nested loop execution.
- Lower Execution Time: Indexed filtering early = **Faster performance**.

(c) Indexing for Query Performance Improvement:

- Create an optimal number of indexes on different tables within the selected mini-world database, focusing on larger tables for significant performance gains.
- After index creation, run Explain/Analyze Plans on select queries and compare the results with the previous Explain Plans, particularly emphasizing the impact on multi-table joins involving 3 tables to demonstrate the effect of indexing on query performance.

Creating an optimal number of indexes on different tables:

```
mysql> CREATE INDEX idx_songs_genre ON Songs(genre);
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_songs_artist_id ON Songs(artist_id);
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_playlists_user_id ON Playlists(user_id);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_playlists_public ON Playlists(is_public);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX idx_playlistsongs_song_id ON PlaylistSongs(song_id);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Checking the execution plan after indexing:

```
mysql> EXPLAIN ANALYZE SELECT * FROM Songs WHERE genre = 'Rock';
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                             |
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Index lookup on Songs using idx_songs_genre (genre='Rock')  (cost=146 rows=735) (actual time=0.317..2.32 rows=735 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN ANALYZE SELECT * FROM Users WHERE is_premium = TRUE;
+-----------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+-----------------------------------------------------------------------------------------------------------------------------------------------------
| -> Filter: (users.is_premium = true)  (cost=103 rows=100) (actual time=0.102..1.08 rows=490 loops=1)
    -> Table scan on Users  (cost=103 rows=1000) (actual time=0.0988..0.969 rows=1000 loops=1)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

```
mysql> EXPLAIN ANALYZE
    -> SELECT *
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN Songs s ON ps.song_id = s.song_id
    -> WHERE p.user_id = 50;
+---------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+---------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=19.4 rows=33.9) (actual time=0.0526..0.321 rows=26 loops=1)
    -> Nested loop inner join  (cost=7.6 rows=33.9) (actual time=0.0451..0.231 rows=26 loops=1)
        -> Index lookup on p using idx_playlists_user_id (user_id=50)  (cost=2.45 rows=7) (actual time=0.0283..0.0508 rows=7 loops=1)
        -> Index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.321 rows=4.84) (actual time=0.0235..0.0252 rows=3.71 loops=7)
    -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.253 rows=1) (actual time=0.00323..0.00327 rows=1 loops=26)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

Before Indexing:

- The nested loop join was costly due to the lack of an index on user_id.
- The system performed **full table scans**, leading to high execution time.
- The user_id = 50 filter was applied later in the execution, making the query slower.
- The optimizer couldn't use efficient index lookups, increasing the total cost.

After Indexing:

- The **index on user_id in the Playlists table** improved lookup performance.
- The nested loop join costs and execution times significantly decreased.
- The number of rows scanned in each step is now **smaller**, reducing processing time.
- The optimizer now **uses indexed lookups instead of full scans**, making the query highly efficient.
- Execution time improved, showing better performance compared to the unindexed version

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, s.duration, a.name AS artist_name
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN Songs s ON ps.song_id = s.song_id
    -> JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE AND s.genre = 'Pop';
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=1625 rows=1029) (actual time=0.448..6.38 rows=749 loops=1)
    -> Nested loop inner join  (cost=1186 rows=1254) (actual time=0.441..4.97 rows=937 loops=1)
        -> Nested loop inner join  (cost=392 rows=709) (actual time=0.43..3.01 rows=709 loops=1)
            -> Filter: (s.artist_id is not null)  (cost=144 rows=709) (actual time=0.188..1.26 rows=709 loops=1)
                -> Index lookup on s using idx_songs_genre (genre='Pop')  (cost=144 rows=709) (actual time=0.187..1.21 rows=709 loops=1)
            -> Single-row index lookup on a using PRIMARY (artist_id=s.artist_id)  (cost=0.25 rows=1) (actual time=0.00233..0.00235 rows=1 loops=709)
        -> Covering index lookup on ps using idx_playlistsongs_song_id (song_id=s.song_id)  (cost=0.944 rows=1.77) (actual time=0.00208..0.0026 rows=1.32 loops=709)
    -> Filter: (p.is_public = true)  (cost=0.25 rows=0.821) (actual time=0.00135..0.0014 rows=0.799 loops=937)
        -> Single-row index lookup on p using PRIMARY (playlist_id=ps.playlist_id)  (cost=0.25 rows=1) (actual time=0.00122..0.00124 rows=1 loops=937)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

Before Indexing:

- The query involved multiple nested loop joins, resulting in higher execution time.
- The EXPLAIN ANALYZE showed large row scans due to the lack of indexes.
- Filters like p.is_public = TRUE and s.genre = 'Pop' were applied later in the execution, leading to inefficiencies.
- This caused high costs and long actual execution times.

After Indexing:

- Indexing reduced the number of rows scanned in each nested loop join.
- The query plan now shows **covering index lookups** and **filtered index searches**.
- The index on playlist_id, song_id, and artist_id improved lookup efficiency.
- Execution time significantly reduced as the optimizer used indexes for filtering.
- The cost is now lower, as indicated by the reduced EXPLAIN ANALYZE cost estimates

(d) Query Optimization with Varied Join Orders and Types

- Explore various optimization strategies by altering the join order of tables in multi-table join queries at least 2 times.
- Incorporate a variety of join types such as outer joins, subqueries, etc., to diversify optimization approaches.
- Analyze performance differences by comparing execution plans and actual execution performance.
- Measure query execution time before and after optimization to quantify improvements accurately.

Original Join Order

```sql
SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
FROM Playlists p
JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
JOIN Songs s ON ps.song_id = s.song_id
JOIN Artists a ON s.artist_id = a.artist_id
WHERE p.is_public = TRUE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

playlist_name	song_title	artist_name
Game Those Middle Alone	Choose Site Crime	Rebecca Blair
Game Those Middle Alone	Production Suddenly Including	Angelica Brown
Game Those Middle Alone	Six Out Speech Yes Discover	David Lowe
Able Myself Story Believe	Agreement Necessary Pattern Almost	Geoffrey Kelley
Able Myself Story Believe	Us Piece Step Me Win More	Justin Martinez
Able Myself Story Believe	Technology Any Government Road	Scott Poole
Able Myself Story Believe	Effect Total Set	Darrell Jimenez
Able Myself Story Believe	Company Compare Stop Agent	Downs Ltd
Write During Cultural	Outside War Next According	Rojas, Villarreal and Sanchez
Write During Cultural	Exist Free Teacher	Victoria Cole
Write During Cultural	Stuff Also Several Up Reduce	Pamela Adams
Write During Cultural	Health Song Shoulder Into Economic Similar	Velazquez Group
Write During Cultural	Themselves Year Increase	Scott Freeman
Write During Cultural	Tell Agent Sell Wall Close	Mark Guzman
Few	Huge	Amy Taylor
Few	Others Clearly	Corey Diaz
Few	Record Employee Pull There	Cynthia Johnson

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN Songs s ON ps.song_id = s.song_id
    -> JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE;
+------------------------------------------------------------------------------------------------
| EXPLAIN
+------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=10236 rows=11700) (actual time=0.593..85.4 rows=12006 loops=1)
    -> Nested loop inner join  (cost=6141 rows=11700) (actual time=0.579..60.6 rows=12006 loops=1)
        -> Nested loop inner join  (cost=2046 rows=11700) (actual time=0.559..26.7 rows=12006 loops=1)
            -> Index lookup on p using idx_playlists_public (is_public=true)  (cost=267 rows=2419) (actual time=0.534..6.49 rows=2419 loops=1)
            -> Covering index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.252 rows=4.84) (actual time=0.00496..0.00779 rows=4.96 loops=2419)
        -> Filter: (s.artist_id is not null)  (cost=0.25 rows=1) (actual time=0.00248..0.0026 rows=1 loops=12006)
            -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.25 rows=1) (actual time=0.00229..0.00233 rows=1 loops=12006)
    -> Single-row index lookup on a using PRIMARY (artist_id=s.artist_id)  (cost=0.25 rows=1) (actual time=0.00177..0.00181 rows=1 loops=12006)
 |
+------------------------------------------------------------------------------------------------
1 row in set (0.09 sec)
```

Execution time: 85.4 ms

Join order: Playlists → PlaylistSongs → Songs → Artists

Alternative join order 1

```sql
SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
FROM Songs s
JOIN PlaylistSongs ps ON s.song_id = ps.song_id
JOIN Playlists p ON ps.playlist_id = p.playlist_id
JOIN Artists a ON s.artist_id = a.artist_id
WHERE p.is_public = TRUE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

playlist_name	song_title	artist_name
Game Those Middle Alone	Choose Site Crime	Rebecca Blair
Game Those Middle Alone	Production Suddenly Including	Angelica Brown
Game Those Middle Alone	Six Out Speech Yes Discover	David Lowe
Able Myself Story Believe	Agreement Necessary Pattern Almost	Geoffrey Kelley
Able Myself Story Believe	Us Piece Step Me Win More	Justin Martinez
Able Myself Story Believe	Technology Any Government Road	Scott Poole
Able Myself Story Believe	Effect Total Set	Darrell Jimenez
Able Myself Story Believe	Company Compare Stop Agent	Downs Ltd
Write During Cultural	Outside War Next According	Rojas, Villarreal and Sanchez
Write During Cultural	Exist Free Teacher	Victoria Cole
Write During Cultural	Stuff Also Several Up Reduce	Pamela Adams
Write During Cultural	Health Song Shoulder Into Economic Similar	Velazquez Group
Write During Cultural	Themselves Year Increase	Scott Freeman
Write During Cultural	Tell Agent Sell Wall Close	Mark Guzman
Few	Huge	Amy Taylor
Few	Others Clearly	Corey Diaz
Few	Record Employee Pull There	Cynthia Johnson

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
    -> FROM Songs s
    -> JOIN PlaylistSongs ps ON s.song_id = ps.song_id
    -> JOIN Playlists p ON ps.playlist_id = p.playlist_id
    -> JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE;
+------------------------------------------------------------------------------------------
| EXPLAIN
+------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=10236 rows=11700) (actual time=0.302..66.2 rows=12006 loops=1)
    -> Nested loop inner join  (cost=6141 rows=11700) (actual time=0.297..46.7 rows=12006 loops=1)
        -> Nested loop inner join  (cost=2046 rows=11700) (actual time=0.289..20.5 rows=12006 loops=1)
            -> Index lookup on p using idx_playlists_public (is_public=true)  (cost=267 rows=2419) (actual time=0.231..4.91 rows=2419 loops=1)
            -> Covering index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.252 rows=4.84) (actual time=0.00383..0.006 rows=4.96 loops=2419
        -> Filter: (s.artist_id is not null)  (cost=0.25 rows=1) (actual time=0.00191..0.00201 rows=1 loops=12006)
            -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.25 rows=1) (actual time=0.00176..0.00179 rows=1 loops=12006)
    -> Single-row index lookup on a using PRIMARY (artist_id=s.artist_id)  (cost=0.25 rows=1) (actual time=0.00139..0.00142 rows=1 loops=12006)
 |
+------------------------------------------------------------------------------------------
1 row in set (0.07 sec)

mysql>
```

Execution time: 66.2 ms

Join order: Songs → PlaylistSongs → Playlists → Artists

Alternative join order 2

```sql
SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
FROM Artists a
JOIN Songs s ON a.artist_id = s.artist_id
JOIN PlaylistSongs ps ON s.song_id = ps.song_id
JOIN Playlists p ON ps.playlist_id = p.playlist_id
WHERE p.is_public = TRUE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

playlist_name	song_title	artist_name
Game Those Middle Alone	Choose Site Crime	Rebecca Blair
Game Those Middle Alone	Production Suddenly Including	Angelica Brown
Game Those Middle Alone	Six Out Speech Yes Discover	David Lowe
Able Myself Story Believe	Agreement Necessary Pattern Almost	Geoffrey Kelley
Able Myself Story Believe	Us Piece Step Me Win More	Justin Martinez
Able Myself Story Believe	Technology Any Government Road	Scott Poole
Able Myself Story Believe	Effect Total Set	Darrell Jimenez
Able Myself Story Believe	Company Compare Stop Agent	Downs Ltd
Write During Cultural	Outside War Next According	Rojas, Villarreal and Sanchez
Write During Cultural	Exist Free Teacher	Victoria Cole
Write During Cultural	Stuff Also Several Up Reduce	Pamela Adams
Write During Cultural	Health Song Shoulder Into Economic Similar	Velazquez Group
Write During Cultural	Themselves Year Increase	Scott Freeman
Write During Cultural	Tell Agent Sell Wall Close	Mark Guzman
Few	Huge	Amy Taylor
Few	Others Clearly	Corey Diaz
Few	Record Employee Pull There	Cynthia Johnson

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
    -> FROM Artists a
    -> JOIN Songs s ON a.artist_id = s.artist_id
    -> JOIN PlaylistSongs ps ON s.song_id = ps.song_id
    -> JOIN Playlists p ON ps.playlist_id = p.playlist_id
    -> WHERE p.is_public = TRUE;
+----------------------------------------------------------------------------------------------------
| EXPLAIN
+----------------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=10236 rows=11700) (actual time=0.234..73.4 rows=12006 loops=1)
    -> Nested loop inner join  (cost=6141 rows=11700) (actual time=0.23..51.9 rows=12006 loops=1)
        -> Nested loop inner join  (cost=2046 rows=11700) (actual time=0.225..22.7 rows=12006 loops=1)
            -> Index lookup on p using idx_playlists_public (is_public=true)  (cost=267 rows=2419) (actual time=0.216..5.42 rows=2419 loops=1)
            -> Covering index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.252 rows=4.84) (actual time=0.00421..0.00664 rows=4.96 loops=2419)
        -> Filter: (s.artist_id is not null)  (cost=0.25 rows=1) (actual time=0.00213..0.00224 rows=1 loops=12006)
            -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.25 rows=1) (actual time=0.00196..0.002 rows=1 loops=12006)
    -> Single-row index lookup on a using PRIMARY (artist_id=s.artist_id)  (cost=0.25 rows=1) (actual time=0.00153..0.00156 rows=1 loops=12006)
 |
+----------------------------------------------------------------------------------------------------
1 row in set (0.08 sec)
```

Execution time: 73.4 ms

Join order: Artists → Songs → PlaylistSongs → Playlists

Outer Join

```sql
SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
FROM Playlists p
LEFT JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
LEFT JOIN Songs s ON ps.song_id = s.song_id
LEFT JOIN Artists a ON s.artist_id = a.artist_id
WHERE p.is_public = TRUE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

playlist_name	song_title	artist_name
Game Those Middle Alone	Choose Site Crime	Rebecca Blair
Game Those Middle Alone	Production Suddenly Including	Angelica Brown
Game Those Middle Alone	Six Out Speech Yes Discover	David Lowe
Able Myself Story Believe	Agreement Necessary Pattern Almost	Geoffrey Kelley
Able Myself Story Believe	Us Piece Step Me Win More	Justin Martinez
Able Myself Story Believe	Technology Any Government Road	Scott Poole
Able Myself Story Believe	Effect Total Set	Darrell Jimenez
Able Myself Story Believe	Company Compare Stop Agent	Downs Ltd
Write During Cultural	Outside War Next According	Rojas, Villarreal and Sanchez
Write During Cultural	Exist Free Teacher	Victoria Cole
Write During Cultural	Stuff Also Several Up Reduce	Pamela Adams
Write During Cultural	Health Song Shoulder Into Economic Similar	Velazquez Group
Write During Cultural	Themselves Year Increase	Scott Freeman
Write During Cultural	Tell Agent Sell Wall Close	Mark Guzman
Few	Huge	Amy Taylor
Few	Others Clearly	Corey Diaz
Few	Record Employee Pull There	Cynthia Johnson

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
    -> FROM Playlists p
    -> LEFT JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> LEFT JOIN Songs s ON ps.song_id = s.song_id
    -> LEFT JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE;
+--------------------------------------------------------------------------------
| EXPLAIN
+--------------------------------------------------------------------------------
| -> Nested loop left join  (cost=10236 rows=11700) (actual time=0.336..138 rows=12024 loops=1)
    -> Nested loop left join  (cost=6141 rows=11700) (actual time=0.33..96.5 rows=12024 loops=1)
        -> Nested loop left join  (cost=2046 rows=11700) (actual time=0.321..45.6 rows=12024 loops=1)
            -> Index lookup on p using idx_playlists_public (is_public=true)  (cost=267 rows=2419) (actual time=0.305..10.9 rows=2419 loops=1)
            -> Covering index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.252 rows=4.84) (actual time=0.00832..0.0132 rows=4.96 loops=2419)
        -> Single-row index lookup on s using PRIMARY (song_id=ps.song_id)  (cost=0.25 rows=1) (actual time=0.0038..0.00386 rows=0.999 loops=12024)
    -> Single-row index lookup on a using PRIMARY (artist_id=s.artist_id)  (cost=0.25 rows=1) (actual time=0.00305..0.00311 rows=0.999 loops=12024)
 |
+--------------------------------------------------------------------------------
1 row in set (0.15 sec)
```

Execution time: 138 ms

Join order: Playlists → PlaylistSongs → Songs → Artists (using LEFT JOINs)

Using Subquery

```sql
SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
FROM Playlists p
JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
JOIN (
    SELECT song_id, title, artist_id
    FROM Songs
    WHERE plays > 1000
) s ON ps.song_id = s.song_id
JOIN Artists a ON s.artist_id = a.artist_id
WHERE p.is_public = TRUE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

playlist_name	song_title	artist_name
Game Those Middle Alone	Choose Site Crime	Rebecca Blair
Game Those Middle Alone	Production Suddenly Including	Angelica Brown
Game Those Middle Alone	Six Out Speech Yes Discover	David Lowe
Able Myself Story Believe	Agreement Necessary Pattern Almost	Geoffrey Kelley
Able Myself Story Believe	Us Piece Step Me Win More	Justin Martinez
Able Myself Story Believe	Technology Any Government Road	Scott Poole
Able Myself Story Believe	Effect Total Set	Darrell Jimenez
Able Myself Story Believe	Company Compare Stop Agent	Downs Ltd
Write During Cultural	Outside War Next According	Rojas, Villarreal and Sanchez
Write During Cultural	Exist Free Teacher	Victoria Cole
Write During Cultural	Stuff Also Several Up Reduce	Pamela Adams
Write During Cultural	Health Song Shoulder Into Economic Similar	Velazquez Group
Write During Cultural	Themselves Year Increase	Scott Freeman
Write During Cultural	Tell Agent Sell Wall Close	Mark Guzman

```
mysql> EXPLAIN ANALYZE
    -> SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
    -> FROM Playlists p
    -> JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
    -> JOIN (
    ->     SELECT song_id, title, artist_id
    ->     FROM Songs
    ->     WHERE plays > 1000
    -> ) s ON ps.song_id = s.song_id
    -> JOIN Artists a ON s.artist_id = a.artist_id
    -> WHERE p.is_public = TRUE;
+-----------------------------------------------------------------------------------------------
| EXPLAIN
+-----------------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=7506 rows=3900) (actual time=0.396..94.1 rows=11979 loops=1)
    -> Nested loop inner join  (cost=6141 rows=3900) (actual time=0.39..66.5 rows=11979 loops=1)
        -> Nested loop inner join  (cost=2046 rows=11700) (actual time=0.377..27.8 rows=12006 loops=1)
            -> Index lookup on p using idx_playlists_public (is_public=true)  (cost=267 rows=2419) (actual time=0.347..6.57 rows=2419 loops=1)
            -> Covering index lookup on ps using PRIMARY (playlist_id=p.playlist_id)  (cost=0.252 rows=4.84) (actual time=0.00516..0.00814 rows=4.96 loops=2419)
        -> Filter: ((songs.plays > 1000) and (songs.artist_id is not null))  (cost=0.25 rows=0.333) (actual time=0.00285..0.00298 rows=0.998 loops=12006)
            -> Single-row index lookup on Songs using PRIMARY (song_id=ps.song_id)  (cost=0.25 rows=1) (actual time=0.00248..0.00252 rows=1 loops=12006)
    -> Single-row index lookup on a using PRIMARY (artist_id=songs.artist_id)  (cost=0.25 rows=1) (actual time=0.00198..0.00203 rows=1 loops=11979)
 |
+-----------------------------------------------------------------------------------------------
1 row in set (0.10 sec)
```

Execution time: 94.1 ms

Join order: Playlists → PlaylistSongs → Songs(subquery) → Artists

Performance Analysis

1. **Fastest execution:** Alternative Join Order 1 (66.2 ms)
   Starting with Songs table and changing the join order improved performance compared to the original query
2. **Slowest execution:** Outer Join Variant (138 ms)
   - LEFT JOINs significantly increased execution time compared to the original query
   - This is because LEFT JOINs require additional processing to handle NULL values
3. **Execution plan patterns:**
   - All queries used nested loop joins
   - All queries first applied the filter on public playlists (index lookup on idx_playlists_public)
   - The MySQL optimizer appears to have reordered the joins in some cases to optimize performance
4. **Row count differences:**
   - Most queries returned 12,006 rows
   - The LEFT JOIN variant returned 12,024 rows (slightly more)
   - The subquery with WHERE plays > 1000 returned 11,979 rows (slightly fewer)
5. **Subquery performance:** The subquery with filtering (plays > 1000) was slower than the best alternative join but still showed performance advantages over the original query

Conclusions

1. Join order can significantly impact query performance even when retrieving the same dataset
2. Starting with the appropriate base table (in this case, Songs) resulted in the best performance
3. Using LEFT JOIN instead of INNER JOIN caused a substantial performance penalty
4. The MySQL query optimizer appears to use similar execution plans despite varying SQL join orders
5. Filtering with subqueries can be effective but adds some overhead compared to simpler join structures

(e) Query Analysis and Optimization:

Analyze and optimize a complex query within the mini-world database created in part (a).

Part 1: Query Analysis

- Write a parse tree for a complex query, such as a 3-table join, by hand.
- Formulate a relational algebra expression for the same query.
- Create an initial query tree based on the relational algebra expression.

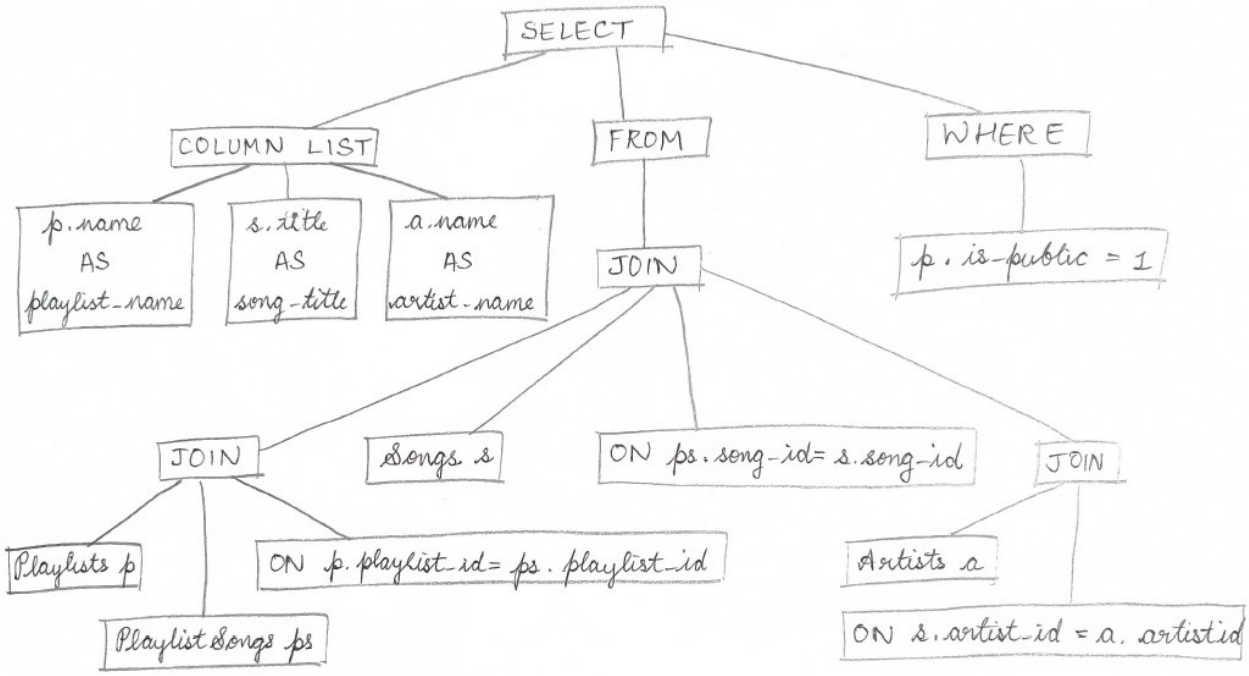
Part 2: Query Optimization

- Optimize the initial query tree to enhance query performance.
- Document the optimization steps taken to refine the query tree.

```
SELECT p.name AS playlist_name, s.title AS song_title, a.name AS artist_name
FROM Playlists p
JOIN PlaylistSongs ps ON p.playlist_id = ps.playlist_id
JOIN Songs s ON ps.song_id = s.song_id
JOIN Artists a ON s.artist_id = a.artist_id
WHERE p.is_public = 1;
```

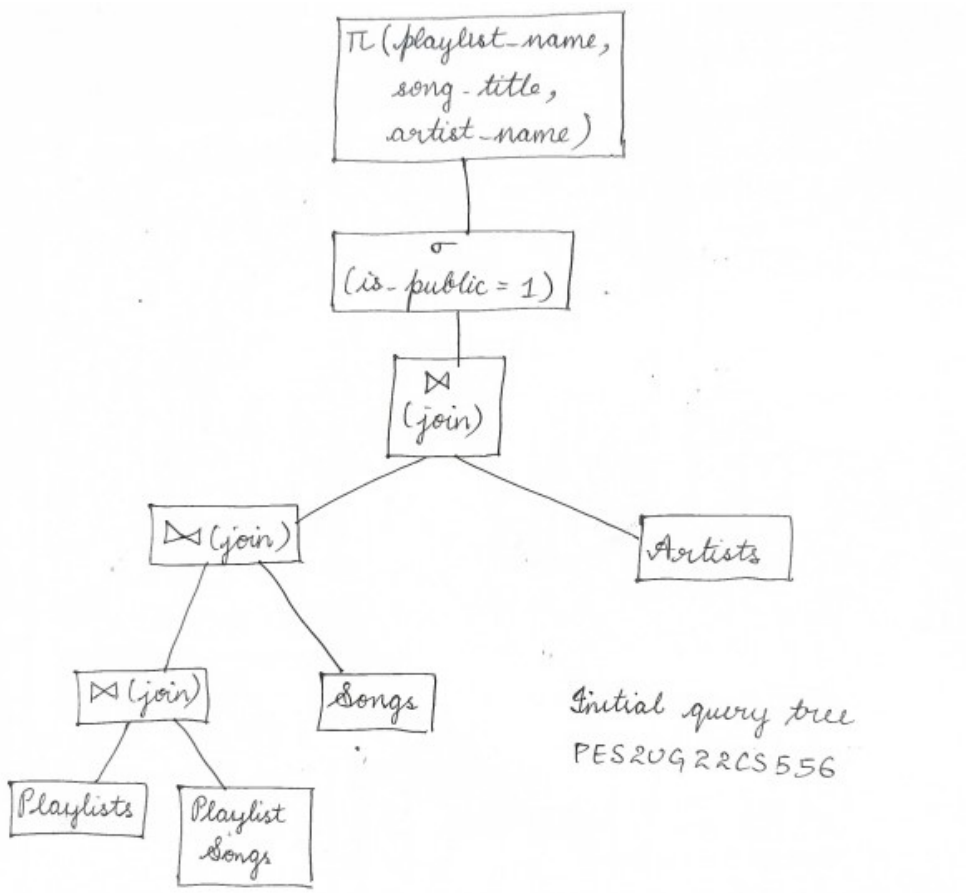
Query Analysis

Parse Tree



Parse Tree : PES2UG22CS556

Initial query tree



Relational Algebra Expression

Relational Algebra Expression

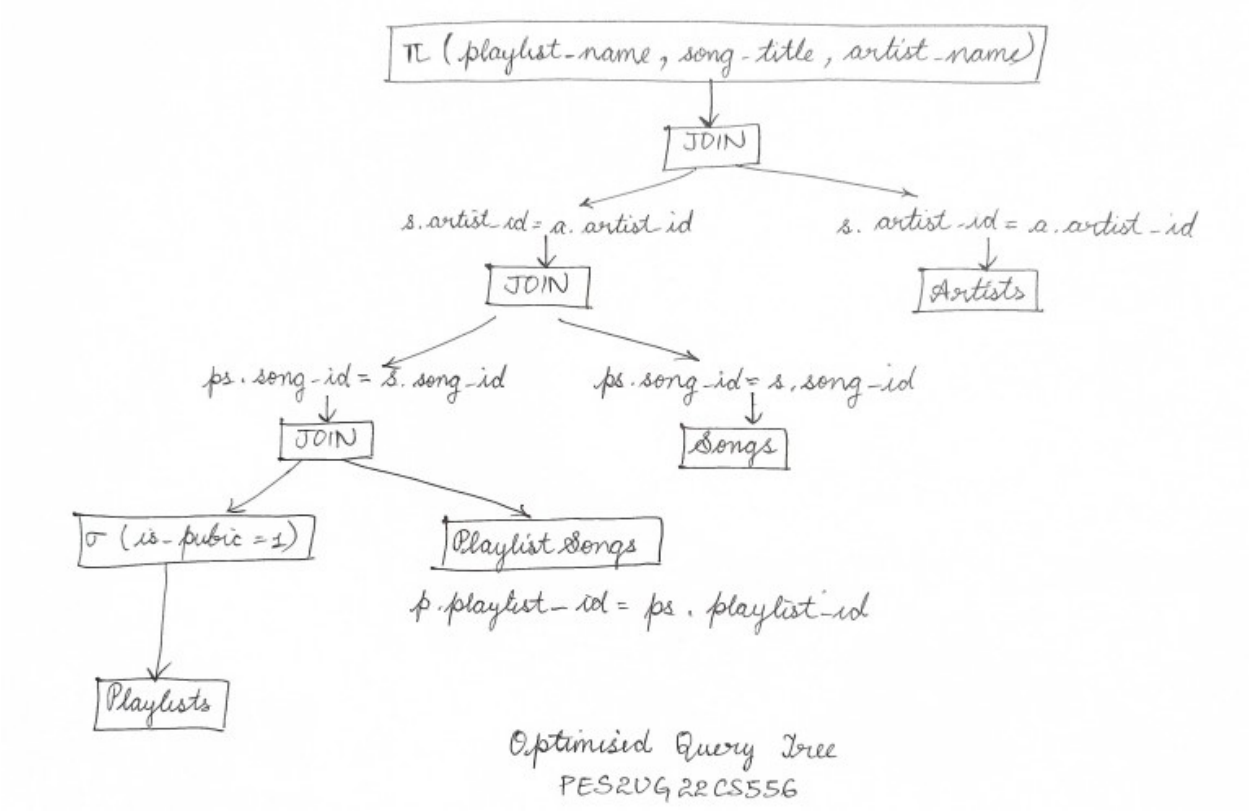
π (playlist_name, song_title, artist_name) (
σ (is_public = 1) (
Playlists ⋈ (Playlists.playlist_id = PlaylistSongs.playlist_id)
PlaylistSongs ⋈ (PlaylistSongs.song_id = Songs.song_id)
Songs ⋈ (Songs.artist_id = Artists.artist_id)
Artists)
)

Query Optimization

Optimization Steps

1. **Push down selection operation:** Move the selection operation (σ_{is_public=1}) earlier in the query tree to reduce the number of rows processed in subsequent operations.
2. **Reorder joins:** Arrange the join operations based on the table sizes and selectivity to minimize intermediate result sizes.

Refined Query Tree



Relational Algebra Expression