

---

**SCHOOL OF ENGINEERING AND TECHNOLOGY****ASSESSMENT FOR THE MASTER OF DATA SCIENCE**

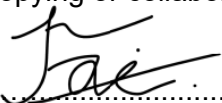
<b>SUBJECT CODE AND TITLE:</b>		<b>MDM5053 BIG DATA MANAGEMENT</b>
<b>ASSESSMENT DUE DATE:</b>		<b>10/12/2024</b>
<b>NO.</b>	<b>STUDENT ID</b>	<b>STUDENT NAME</b>
<b>1</b>	<b>14086334</b>	<b>KAN JUN FAI</b>

**IMPORTANT**

The University requires students to adhere to submission deadlines for any form of assessment. Penalties are applied in relation to unauthorised late submission of work. Coursework submitted after the deadline will be subjected to the prevailing academic regulations. Please check your respective programme handbook.

**Academic Honesty Acknowledgement**

"I **KAN JUN FAI** (name) verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realise the penalties (*refer to the student handbook and undergraduate programme handbook*) for any kind of copying or collaboration."

.......... (Student' Signature / Initial)

## **Dataset Use Case and the Problems:**

[The Spotify dataset from Kaggle](#) was used in our music streaming company simulation, with 50000 records of AI-generated records containing song titles, artist names, genres, (all features shown in image below). The key insights collected through analyzing this dataset using MySQL uncovered success factors of the music streaming industry by examining key features like the average total streams, genre, and release year. However, as the music industry grows, there are massive amounts of user data waiting to be analysed to in order to uncover insights on user preferences, content preferences, and user engagement. There is a growing challenge of effectively analyzing large scale datasets and extracting business insights in a time-efficient manner in this fast-paced digital world. Conventional methods like MySQL face performance challenges in handling large datasets. With the implementation of Big Data tools like PySpark, this project will aim to compare the performance of using MySQL vs. PySpark, in terms of handling large-scale business insights in the most efficient and effective method. The insights gathered will provide the potential advantages of implementing Big Data tools in Big Data management of companies to improve business analytics, leading to improved customer engagement and business growth.

## About Dataset

This dataset contains fictional information about 50,000 songs from various music genres. It includes features such as song popularity, stream count, duration, artists, albums, and languages. **The dataset is generated by ChatGPT and does not contain real data.** It can be used for creative and educational purposes, such as music analysis, trend forecasting, and song popularity studies.

- **song\_id:** The unique identifier for the song.
- **song\_title:** The title of the song.
- **artist:** The artist performing the song.
- **album:** The album where the song is featured.
- **genre:** The music genre of the song.
- **release\_date:** The release date of the song.
- **duration:** The duration of the song (in seconds).
- **popularity:** The popularity score of the song (1-100).
- **stream:** The total number of streams for the song.
- **language:** The language of the song.
- **explicit\_content:** Whether the song contains explicit content (e.g., inappropriate language).
- **label:** The record label that published the song.
- **composer:** The composer of the song.
- **producer:** The producer of the song.
- **collaboration:** Whether the song is a collaboration with other artists.

# Comparing techniques of Conventional (MySQL) vs Big Data (PySpark)

## MySQL

The screenshot shows the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view with 'sunway\_masters' expanded, containing tables like 'artists', 'flight\_data', 'netflix', 'songs', 'spotify', 'steam\_games', 'student\_scores', and 'title\_genres'. The main editor window shows a SQL script with the following queries:

```
1 -- import directory: C:\ProgramData\MySQL\MySQL Server 8.0\Uploads
2
3 • SELECT * FROM spotify LIMIT 10; -- Preview the first 10
4 • SELECT COUNT(*) FROM spotify; -- number of rows: 50000
5
```

Below the script, the 'Result Grid' shows the first 10 records of the 'spotify' table:

song_id	song_title	artist	album	genre	release_date	duration	popularity	stream	language
SP0001	Space executive series.	Sydney Clark	What.	Electronic	1997-11-08	282.0	42	35055874	English
SP0002	Price last painting.	Connor Peters DDS	Nature politics.	Electronic	2015-05-10	127.0	50	9249527	English
SP0003	Piece.	Anna Keith	Visit.	Pop	2024-07-08		10	76669110	English
SP0004	Power industry your.	Zachary Simpson	Behavior evening.	Hip-Hop	2022-08-15	214.0	86	34732016	English
SP0005	Food animal second.	Christopher Mcgee	Front.	Pop	2023-03-05	273.0	63	96649372	English
SP0006	Whatever Mr send.	Nathan King	Various experience.	Folk	2015-11-29	312.0	74	82613530	English
SP0007	Each leg.	Joshua Santos	Determine.	Hip-Hop	1996-03-16	336.0	49	88337653	Japanese
SP0008	Person then enjoy.	Leonard Brown	Sense lot.	Hip-Hop	2017-08-14	237.0	9	95988275	English
SP0009	Poor.	William Ford	Organization.	Pop	2002-02-28		74	67287444	Spanish
SP0010	East husband.	Susan Harrell	List item.	Electronic	2016-12-06	216.0	2	48726791	English

The query above prints the first 10 records out of a total of 50,000 records.

The screenshot shows the MySQL Workbench interface. The main editor window shows a SQL script for data pre-processing and checking for NULL values:

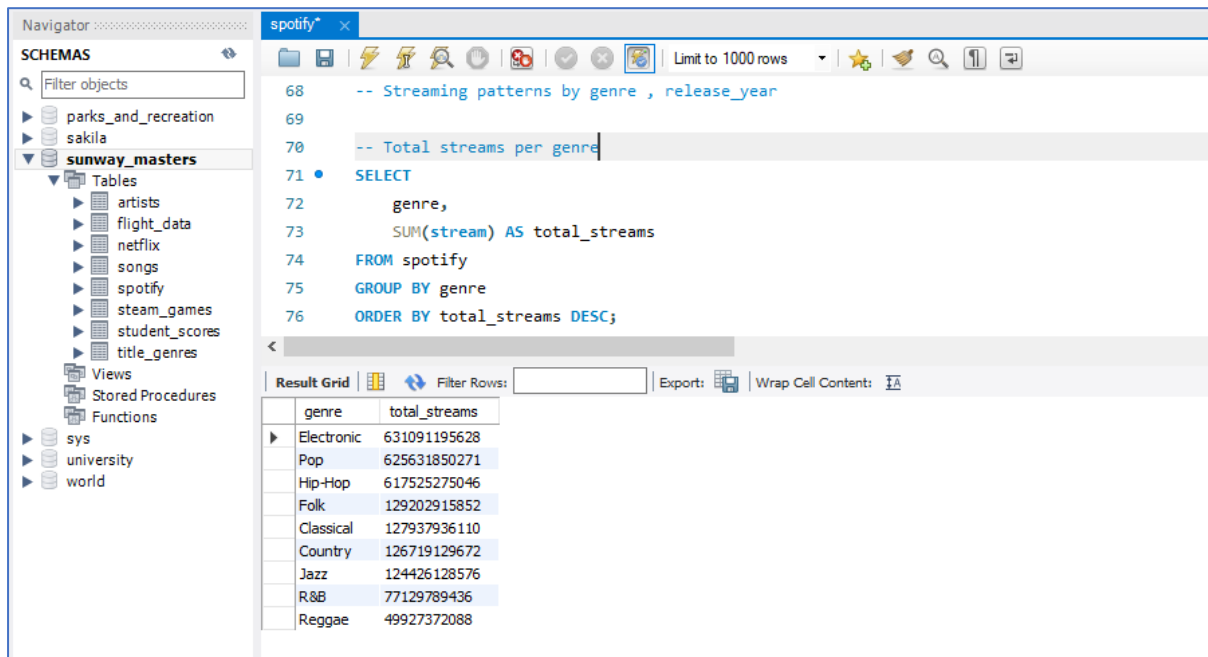
```
7 -- Data Pre-processing
8
9 -- Check for NULL values
10 • SELECT
11     COUNT(*) AS total_rows,
12     SUM(CASE WHEN song_title IS NULL THEN 1 ELSE 0 END) AS missing_song_title,
13     SUM(CASE WHEN artist IS NULL THEN 1 ELSE 0 END) AS missing_artist,
14     SUM(CASE WHEN album IS NULL THEN 1 ELSE 0 END) AS missing_album,
15     SUM(CASE WHEN genre IS NULL THEN 1 ELSE 0 END) AS missing_genre,
16     SUM(CASE WHEN release_date IS NULL THEN 1 ELSE 0 END) AS missing_release_date,
17     SUM(CASE WHEN duration IS NULL THEN 1 ELSE 0 END) AS missing_duration,
18     SUM(CASE WHEN popularity IS NULL THEN 1 ELSE 0 END) AS missing_popularity,
19     SUM(CASE WHEN stream IS NULL THEN 1 ELSE 0 END) AS missing_stream,
20     SUM(CASE WHEN explicit_content IS NULL THEN 1 ELSE 0 END) AS missing_explicit_content,
21     SUM(CASE WHEN label IS NULL THEN 1 ELSE 0 END) AS missing_label,
22     SUM(CASE WHEN composer IS NULL THEN 1 ELSE 0 END) AS missing_composer,
23     SUM(CASE WHEN producer IS NULL THEN 1 ELSE 0 END) AS missing_producer,
24     SUM(CASE WHEN collaboration IS NULL THEN 1 ELSE 0 END) AS missing_collaboration
25 FROM spotify;
26
27
28
```

Below the script, the 'Result Grid' shows the results of the query:

	total_rows	missing_song_title	missing_artist	missing_album	missing_genre	missing_release_date	missing_duration	missing_popularity	missing_stream
▶	50000	0	0	0	0	0	0	0	0

The above query is written such that if there are NULL values in each column, the table will return '1', and '0' when there is no NULL values found. No null values were recorded.

## 1. Total streams per genre



The screenshot shows a database management system interface. On the left is a 'Navigator' pane with a 'SCHEMAS' section. Under 'sunway\_masters', the 'Tables' folder is expanded, showing a list of tables including 'artists', 'flight\_data', 'netflix', 'songs', 'spotify', 'steam\_games', 'student\_scores', 'title\_genres', 'sys', 'university', and 'world'. The 'spotify' table is selected. The main pane displays a SQL query in a text editor. The query is as follows:

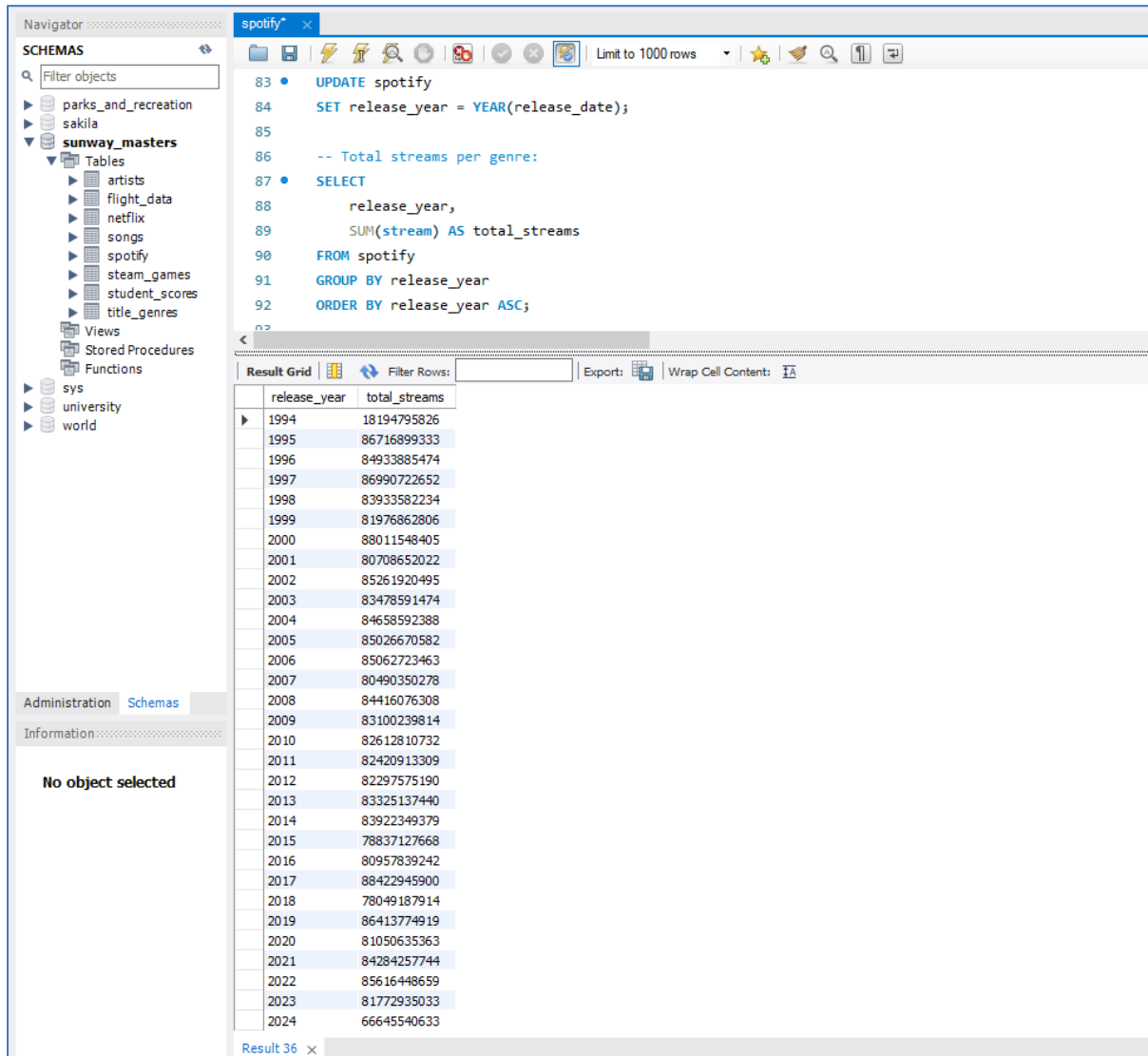
```
-- Streaming patterns by genre , release_year
-- Total streams per genre
SELECT
    genre,
    SUM(stream) AS total_streams
FROM spotify
GROUP BY genre
ORDER BY total_streams DESC;
```

Below the query editor is a 'Result Grid' showing the results of the query. The grid has two columns: 'genre' and 'total\_streams'. The results are sorted in descending order of total streams.

genre	total_streams
Electronic	631091195628
Pop	625631850271
Hip-Hop	617525275046
Folk	129202915852
Classical	127937936110
Country	126719129672
Jazz	124426128576
R&B	77129789436
Reggae	49927372088

The query code above queries for the total streams per genre, grouped by genre and sorted the total streams according to descending order. By interpreting the results, 'Electronic', 'Pop', and 'Hip-Hop' represents the top three most popular genres as they dominate the other genres by an overwhelming amount, followed by 'Folk', 'Classical', 'Country', and 'Jazz', and lastly 'R&B' and 'Reggae'. This pattern is likely due to the high number of released songs based on modern-day trend containing electronic, pop, and hip-hop genres, attracting consumption by global consumers which indicate global appeal. As for business insights obtained from this query, businesses can utilize electronic, pop, and hip-hop genre songs to be utilized in advertisements, radio, and events to attract and funnel attention towards targeted content/advertisements. For Spotify as a streaming platform, content recommendations may be directed towards playlists containing Folk, Classical, Country, Jazz, R&B and Reggae may be encouraged to attract new listeners towards the mentioned genres.

## 2. Total streams per 'release\_year'



The screenshot shows a database management interface with a left-hand 'Navigator' pane and a main query editor. The 'Navigator' pane displays a tree structure of databases and tables, with 'sunway\_masters' expanded to show tables like 'artists', 'flight\_data', 'netflix', 'songs', 'spotify', 'steam\_games', 'student\_scores', and 'title\_genres'. The main query editor contains a SQL script that updates the 'spotify' table to add a 'release\_year' column and then selects the total streams grouped by 'release\_year' in ascending order. Below the query editor, a 'Result Grid' displays the output of the query, showing a table with two columns: 'release\_year' and 'total\_streams'. The results list data for each year from 1994 to 2024.

```
83 • UPDATE spotify
84   SET release_year = YEAR(release_date);
85
86   -- Total streams per genre:
87 • SELECT
88     release_year,
89     SUM(stream) AS total_streams
90   FROM spotify
91   GROUP BY release_year
92   ORDER BY release_year ASC;
```

release_year	total_streams
1994	18194795826
1995	86716899333
1996	84933885474
1997	86990722652
1998	83933582234
1999	81976862806
2000	88011548405
2001	80708652022
2002	85261920495
2003	83478591474
2004	84658592388
2005	85026670582
2006	85062723463
2007	80490350278
2008	84416076308
2009	83100239814
2010	82612810732
2011	82420913309
2012	82297575190
2013	83325137440
2014	83922349379
2015	78837127668
2016	80957839242
2017	88422945900
2018	78049187914
2019	86413774919
2020	81050635363
2021	84284257744
2022	85616448659
2023	81772935033
2024	66645540633

The query code above has two parts. First, a new column “release\_year” was created and added to the spotify table. Secondly, the query then selects for total streams grouped by release year and sorted according to ascending release year. Based on analyzing the volume of streams per year, there is an overall high usage rate of music streaming platforms like Spotify due to the internet boom in 2000s. Since this dataset does not have streaming volume according to varying timestamps, time-series data analysis is not conclusive.

### 3. Average streams per genre

The screenshot shows a database query editor interface. On the left is a 'Navigator' pane with a 'SCHEMAS' tree. The 'spotify' database is expanded, showing tables like 'artists', 'flight\_data', 'netflix', 'songs', 'spotify', 'steam\_games', 'student\_scores', and 'title\_genres'. The main editor area shows a SQL query for the 'spotify' database:

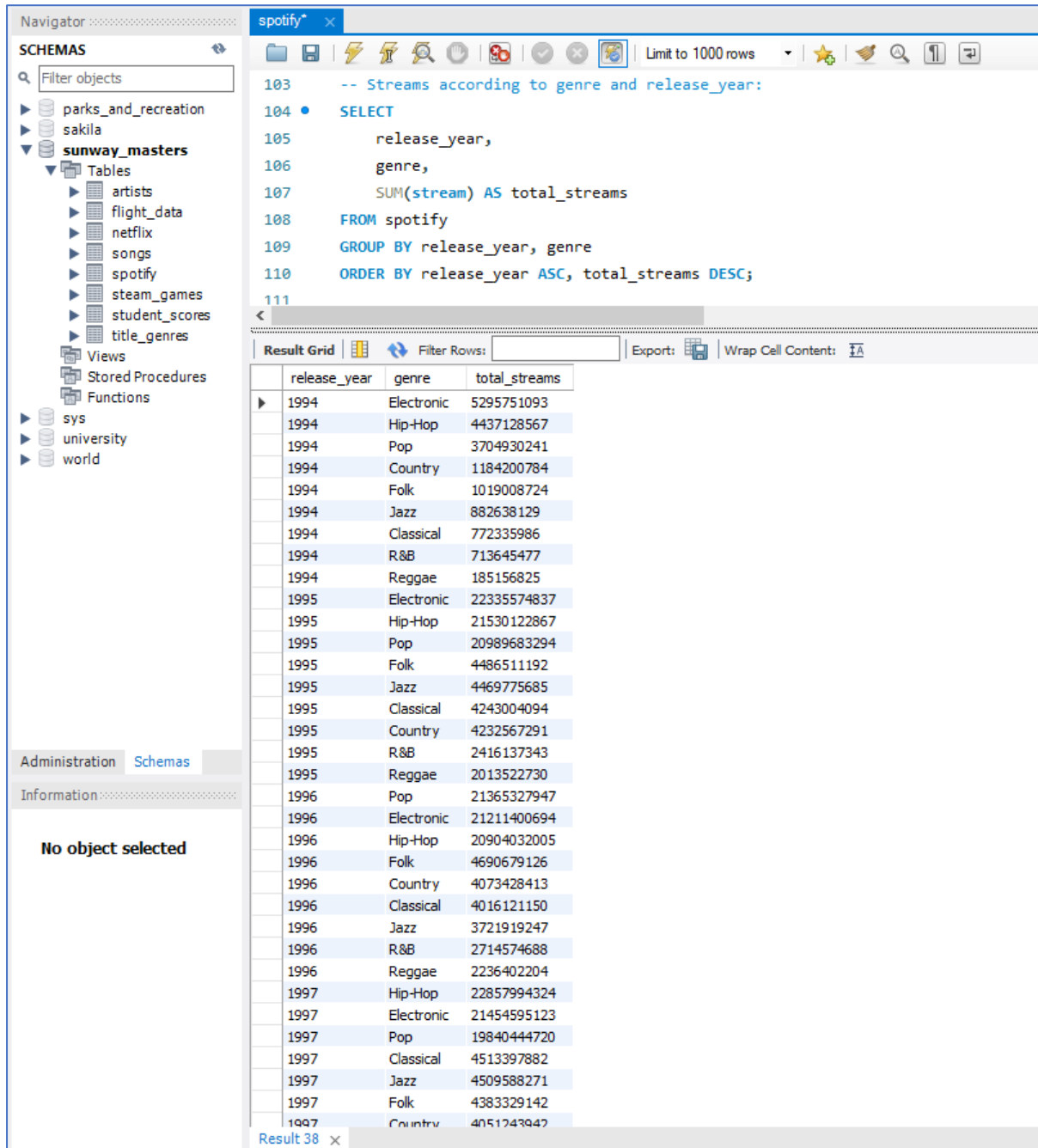
```
-- Avg streams(per year) per genre:
SELECT
    genre,
    AVG(stream) AS avg_streams
FROM spotify
GROUP BY genre
ORDER BY avg_streams DESC;
```

Below the query editor is a 'Result Grid' showing the output of the query. It has two columns: 'genre' and 'avg\_streams'. The results are sorted in descending order of average streams.

genre	avg_streams
Reggae	51418508.8445
R&B	50978049.8586
Folk	50489611.5092
Country	50465603.2147
Classical	50329636.5500
Electronic	50214130.7788
Hip-Hop	50213471.7065
Pop	49914779.8206
Jazz	49690945.9169

The query above queries for average number of streams for each genre, sorted according to descending order of highest to lowest average streams per genre. Reggae, R&B, and Folk are the top three highest streamed genres. Electronic, Pop, and Hip-Hop – where they previously dominate in total streams per genre – are now in the lower performing genres according to average streams. Electronic, Pop, and Hip-Hop may possibly contain a number of songs that are less popular and less streamed, lowering the average score of these genres.

#### 4. Combination of genre and 'release\_year'



The screenshot shows a database interface with a left sidebar containing a 'SCHEMAS' tree. The 'spotify' table is selected under the 'sunway\_masters' schema. The main area displays a SQL query and its results in a 'Result Grid'.

**Query:**

```
-- Streams according to genre and release_year:
SELECT
    release_year,
    genre,
    SUM(stream) AS total_streams
FROM spotify
GROUP BY release_year, genre
ORDER BY release_year ASC, total_streams DESC;
```

**Result Grid:**

release_year	genre	total_streams
1994	Electronic	5295751093
1994	Hip-Hop	4437128567
1994	Pop	3704930241
1994	Country	1184200784
1994	Folk	1019008724
1994	Jazz	882638129
1994	Classical	772335986
1994	R&B	713645477
1994	Reggae	185156825
1995	Electronic	22335574837
1995	Hip-Hop	21530122867
1995	Pop	20989683294
1995	Folk	4486511192
1995	Jazz	4469775685
1995	Classical	4243004094
1995	Country	4232567291
1995	R&B	2416137343
1995	Reggae	2013522730
1996	Pop	21365327947
1996	Electronic	21211400694
1996	Hip-Hop	20904032005
1996	Folk	4690679126
1996	Country	4073428413
1996	Classical	4016121150
1996	Jazz	3721919247
1996	R&B	2714574688
1996	Reggae	2236402204
1997	Hip-Hop	22857994324
1997	Electronic	21454595123
1997	Pop	19840444720
1997	Classical	4513397882
1997	Jazz	4509588271
1997	Folk	4383329142
1997	Country	4051243947

The query above calculates the total number of streams for the combination of release year and genre, sorted in chronological order starting in 1994 to 2024. For each year, the most streamed genre will be the first mentioned (Please see Appendix on last page for full table). Electronic genre is observed to be consistently rank in the top three genres in many years, suggesting



genre dominance, popularity, and global appeal. Business insights for music streaming platforms include themed playlist suggestions to improve user retention rate, and cross-recommendation on niche or less-played songs to introduce less popular songs to new audiences.

# PySpark

```
[34]: from pyspark.sql import SparkSession
      from pyspark.sql.functions import col, avg

      spark = SparkSession.builder.appName("SpotifyAnalysis").getOrCreate()

      spotify = spark.read.csv(r"C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\spotify.csv", header=True, inferSchema=True)
      spotify = spotify.na.drop() # Removing null values
```

```
[12]: spotify.printSchema()

root
 |-- song_id: string (nullable = true)
 |-- song_title: string (nullable = true)
 |-- artist: string (nullable = true)
 |-- album: string (nullable = true)
 |-- genre: string (nullable = true)
 |-- release_date: date (nullable = true)
 |-- duration: double (nullable = true)
 |-- popularity: integer (nullable = true)
 |-- stream: integer (nullable = true)
 |-- language: string (nullable = true)
 |-- explicit_content: string (nullable = true)
 |-- label: string (nullable = true)
 |-- composer: string (nullable = true)
 |-- producer: string (nullable = true)
 |-- collaboration: string (nullable = true)
```

## 1. Total streams per genre

```
[49]: # 1. Total streams per genre

      from pyspark.sql.functions import col, sum

      genre_streams = spotify.groupBy("genre").agg(sum("stream").alias("total_streams"))
      genre_streams.orderBy(col("total_streams").desc()).show()
```

```
+-----+
| genre|total_streams|
+-----+
| Hip-Hop| 162325648559|
| Pop| 161217763634|
| Electronic| 160131619204|
| Folk| 33737729809|
| Jazz| 33389489764|
| Classical| 32493628745|
| Country| 31891760348|
| R&B| 18306966508|
| Reggae| 13272410913|
+-----+
```

## 2. Total streams per 'release\_year'

```
[43]: # 2. Total streams per 'release_year'
      from pyspark.sql.functions import col, year, sum

      # Extract release_year from release_date if it's not already present
      if "release_year" not in spotify.columns:
          spotify = spotify.withColumn("release_year", year(col("release_date")))

      # Group by release_year and calculate total streams
      yearly_streams = spotify.groupBy("release_year") \
          .agg(sum("stream").alias("total_streams")) \
          .orderBy(col("release_year").asc())

      # Show the result
      yearly_streams.show()

+-----+-----+
|release_year|total_streams|
+-----+-----+
|1994|4970176606|
|1995|21918768958|
|1996|22854326284|
|1997|20935050247|
|1998|20206048749|
|1999|22041280411|
|2000|20895045588|
|2001|20506115957|
|2002|23479136462|
|2003|20830218299|
|2004|22350704854|
|2005|20476835829|
|2006|21470829615|
|2007|21072500150|
|2008|22061461357|
|2009|22283010760|
|2010|22340648557|
|2011|23002161955|
|2012|21661786257|
|2013|22069536582|
+-----+-----+
only showing top 20 rows
```

## 3. Average streams per genre

```
[45]: # 3. Average streams per genre

release_trends = spotify.groupBy("release_date").agg(sum("stream").alias("total_streams"))
release_trends.orderBy("release_date").show()

+-----+-----+
|release_date|total_streams|
+-----+-----+
|1994-10-07|98410753|
|1994-10-09|97281219|
|1994-10-10|228253762|
|1994-10-12|132228111|
|1994-10-13|137218441|
|1994-10-14|84012993|
|1994-10-16|42191358|
|1994-10-17|77011515|
|1994-10-19|127839289|
|1994-10-20|250927779|
|1994-10-22|97430984|
|1994-10-23|55695070|
|1994-10-24|86860701|
|1994-10-25|162815559|
|1994-10-26|27375753|
|1994-10-27|3190444|
|1994-10-30|94008260|
|1994-10-31|12917938|
|1994-11-02|38071313|
|1994-11-04|44852177|
+-----+-----+
only showing top 20 rows
```

#### 4. Combination of genre and 'release\_year'

```
[47]: # 4.    Combination of genre and 'release_year'

from pyspark.sql.functions import col, sum

# Group by release_year and genre, then calculate total streams
release_genre_analysis = spotify.groupBy("release_date", "genre") \
    .agg(sum("stream").alias("total_streams")) \
    .orderBy(col("release_date").asc(), col("total_streams").desc())

# Show the result
release_genre_analysis.show()
```

release_date	genre	total_streams
1994-10-07	Electronic	98410753
1994-10-09	Hip-Hop	97281219
1994-10-10	Country	86411334
1994-10-10	Hip-Hop	72978426
1994-10-10	Electronic	68864002
1994-10-12	Electronic	78421076
1994-10-12	Pop	30481138
1994-10-12	Classical	23325897
1994-10-13	Hip-Hop	87377879
1994-10-13	Electronic	49840562
1994-10-14	Electronic	84012993
1994-10-16	Jazz	42191358
1994-10-17	Pop	49806741
1994-10-17	Reggae	27204774
1994-10-19	Pop	96044190
1994-10-19	R&B	21637196
1994-10-19	Folk	10157903
1994-10-20	Jazz	94712595
1994-10-20	Hip-Hop	93860151
1994-10-20	Pop	62355033

only showing top 20 rows

## **Discussion**

The main comparisons to be discussed between MySQL vs. PySpark are the handling of dataset sizes, performance, and scalability, and use-cases. In terms of handling dataset sizes, PySpark is able to manage large datasets with ease, whereas MySQL is more suited for smaller datasets. In terms of performance, MySQL was a few seconds slower than PySpark when handling compute-intensive tasks, whereas PySpark was performing optimally with close to no delay in response. While MySQL has restricted scalability, PySpark's scalability is excellent due to its ability to handle heavy compute tasks. Lastly in terms of use cases, MySQL is useful for smaller datasets and is more conventional in its querying rules. Conversely when using PySpark, similar concept but different language is used compared to MySQL, which requires a strong foundation in coding knowledge in order to effectively utilize PySpark to its full potential. Overall, PySpark is useful in handling massive datasets, providing the ability to perform efficient business analytics. MySQL is excellent at handling smaller structured datasets; however the challenge arises when the data volume increases.

## **Reflection on the Lessons Learned:**

Through the understanding of the listener's content preferences by analysing the seasonal trends of trending genres. Through personalized recommendations, Spotify can tailor and recommend themed playlists like "Back to 2000s", "Electronic Era", making use of nostalgia in order to boost user engagement. Personalized user experiences contribute to longer content engagement time, providing opportunity to boost the reach of less popular songs to users via song discovery. Advertisers may leverage on trends to promote merchandise relating to certain theme of music, further encouraging user engagement.

As the volume of user generated data increases, the challenge comes in how to effectively and efficiently process a massive amount of data in real-time to extract and make use of business insights. Big Data management tools provide the solution to efficiently compile, extract and perform business analytics when data volume is immense. The Big Data tool used is PySpark, while the conventional tool used is MySQL. PySpark can handle large datasets with ease, while MySQL is better suited for smaller datasets. Performance-wise, PySpark outperforms MySQL in compute-intensive tasks, providing near-instantaneous responses, whereas MySQL experiences slight task processing time. Scalability is another key distinction; PySpark offers optimal scalability for heavy compute tasks, while MySQL very much limited. For use cases, MySQL is suitable for handling smaller, structured datasets with conventional querying methods. PySpark, on the other hand, demands solid coding knowledge for processing massive datasets and performing advanced business analytics. Overall, PySpark is best for large-scale data handling, while MySQL is optimized with smaller datasets but struggles as data volume increases.