ETL: PERFORM EXTRACT-TRANSFORM-LOAD(ETL) OPERATIONS ON DATA, TRANSFORMING IT INTO A MORE STRUCTURED FORMAT OR LOADING IT INTO A DATABASE.

A Project Report Submitted in the fulfilment of the requirements for Interim-Project Evaluation

SUBMITTED BY:
NAME: SNEHA RAMACHANDRAN
EMP ID: 2320393
COHORT CODE: CSDAIA24GP003

METHOD: SPARK ETL

ABSTRACTION:

The Extract, Transform, Load (ETL) process plays a crucial role in data management and analysis by extracting raw data from diverse sources, transforming it into a structured format, and loading it into a database for further analysis and insights generation. In this project, we leverage Apache Spark and DataFrames to perform ETL operations, enabling efficient processing of large-scale datasets and seamless integration with various data sources and storage systems.

The project begins with the extraction phase, where raw data is retrieved from sources such as files, databases, or streaming platforms using Spark's data ingestion capabilities. Leveraging Spark's distributed computing framework, we can efficiently extract data in parallel, enabling scalability and performance optimization for handling massive datasets.

Once the data has been transformed into the desired format, the final step involves loading it into a target database or data warehouse for storage and further analysis. Spark's support for various database connectors and integration with distributed storage systems like Hadoop Distributed File System (HDFS) facilitates seamless data loading and ensures data consistency and reliability.

Throughout the project, we emphasize the importance of data quality, integrity, and efficiency in the ETL process. By leveraging Spark and DataFrames, we achieve scalability, performance, and flexibility, making the ETL pipeline suitable for handling real-world data challenges and enabling organizations to derive actionable insights from their data assets.

Overall, this project showcases the power and versatility of Apache Spark and DataFrames in performing ETL operations, empowering organizations to efficiently manage, transform, and analyze their data for informed decision-making and strategic planni

INTRODUCTION:

Apache Spark is a powerful framework for distributed data processing, offering extensive capabilities for working with structured data through its Data Frame API. One crucial aspect of data processing is managing Data Frame metadata, including column names, data types, and schema information. This report provides an overview of how to access and manipulate Data Frame metadata in Apache Spark using various methods from the Spark SQL and Data Frame API.

FEATURES OF APACHE SPARK

Apache Spark has following features.

- Speed − Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.

- Supports multiple languages − Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.

- Advanced Analytics − Spark not only supports 'Map' and 'reduce'. It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

SPARK DATAFRAME :

In Spark, DataFrames are the distributed collections of data, organized into rows and columns. Each column in a DataFrame has a name and an associated type. DataFrames are similar to traditional database tables, which are structured and concise. We can say that DataFrames are relational databases with better optimization techniques.

Spark DataFrames can be created from various sources, such as Hive tables, log tables, external databases, or the existing RDDs. DataFrames allow the processing of huge amounts of data

WHAT IS DATAFRAME?

When there is not much storage space in memory or on disk, RDDs do not function properly as they get exhausted. Besides, Spark RDDs do not have the concept of schema—the structure of a database that defines its objects. RDDs store both structured and unstructured data together, which is not very efficient.
RDDs cannot modify the system in such a way that it runs more efficiently. RDDs do not allow us to debug errors during the runtime. They store the data as a collection of Java objects.

RDDs use serialization (converting an object into a stream of bytes to allow faster processing) and garbage collection (an automatic memory management technique that detects unused objects and frees them from memory) techniques. This increases the overhead on the memory of the system as they are very lengthy. This was when DataFrames were introduced to overcome the limitations Spark RDDs had. Now, what makes Spark DataFrames so unique? Let's check out the features of Spark DataFrames that make them so popular.

IMPLEMENTATION:

Step 1:

The **%pip** magic command allows you to install packages from PyPI (Python Package Index) directly within your Databricks environment. Once installed, you can use the **faker** package to generate fake data for testing or simulation purposes.

Step 2:

I have imported several libraries and modules in Python for data manipulation using Spark Data Frame ('pyspark.sql'), Faker ('faker'),. Each of these libraries serves a specific purpose in your data analysis workflow. Here's a brief explanation of each import statement and its role.

1.Import Spark Session and Data Types from PySpark:

- **'from pyspark.sql import SparkSession':**
  Imports the 'SparkSession' class from the 'pyspark.sql' module, which is used to interact with Spark SQL and create DataFrame objects.

- **'from pyspark.sql. types import StructType, StructField, StringType, Intege Type, FloatType':**
  Imports various data types and structures from 'pyspark.sql.types' module, such as

  - **StructType** - StructType is a class used to define the structure of a DataFrame schema. It allows you to specify the structure of the DataFrame by defining the names and data types of its columns.

  - **StructField** -StructField is used within StructType to define individual columns of a DataFrame schema. It specifies the name, data type, and nullable property of each column.

- **StringType -** StringType is a data type used to represent string values in Spark Data Frames. It is used for columns containing text or alphanumeric data.

- **IntegerType** - IntegerType is a data type used to represent integer values in Spark DataFrames. It is used for columns containing whole numbers.

- **FloatType -** IntegerType is a data type used to represent floating-point numbers(i.e,numbers with decimal points) in Spark DataFrames. It is used for columns containing whole numbers.

2. Import DataFrame Functions from PySpark:

- **'from pyspark.sql.Functions import col, concat, lit,regexp_replace,avg,expr,when,split':**

  Imports DataFrame functions from 'pyspark.sql. functions' module, including 'col()' for Column Selection , 'concat()' concatenating strings or columns together ,' lit()' add a constant value as a column to a DataFrame, 'regexp_replace()' helps in replacing substrings in string columns based on a regular expression pattern,'avg()' calculates the average value of a numeric column in a DataFrame,'expr()' calculates the average value of a numeric column in a DataFram**,** 'when()' allows you to specify conditions and corresponding values to apply when the condition is true, enabling you to perform conditional transformations based on data conditions,'split()' splits a string column into an array of substrings based on a delimiter.

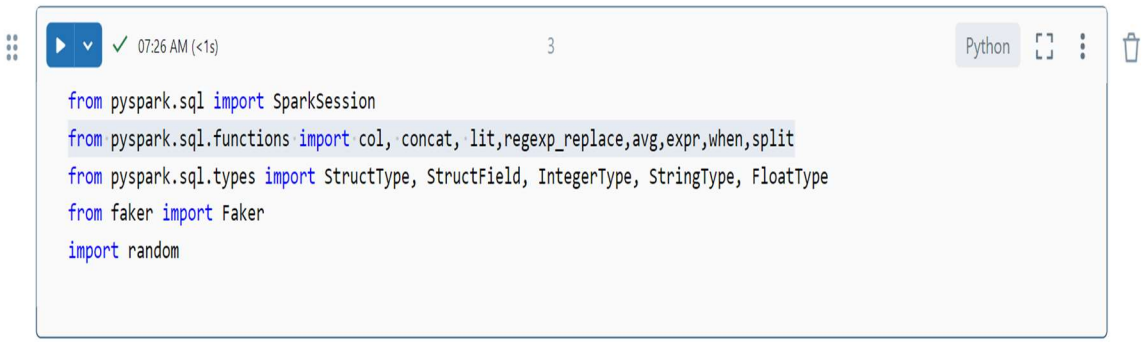
4. Import Faker Library:

- **'from faker import Faker':**

5. Import Random:

- **'import random':**

Imports the 'random' module, which provides functions for generating random numbers, sequences, and making random selections.



By importing these libraries and modules, you have access to a wide range of functionalities for working with data in Spark DataFrames, generating synthetic data with Faker.

Step 3:

This initializes an instance of the Faker class from the faker library in Python. The faker library is commonly used for generating fake data, such as names, addresses, phone numbers, dates, and more, which can be useful for testing, prototyping, and generating sample datasets.

Step 4:

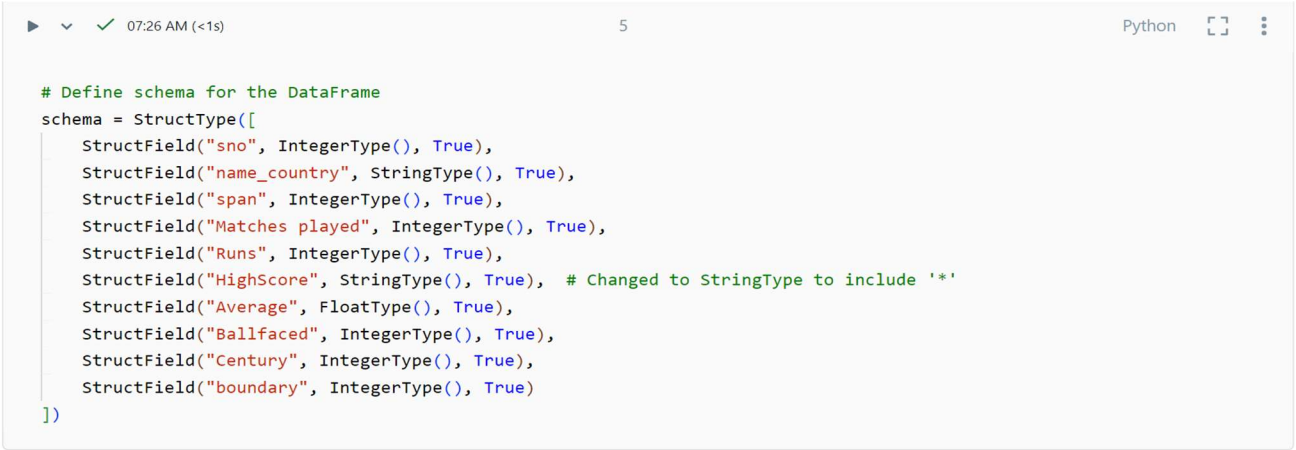
Defining a schema using the StructType and StructField classes from the pyspark.sql.types module in PySpark. This schema is used to define the structure and data types of the columns for a DataFrame in PySpark.

Here's an explanation of each field in the schema:

1. sno: IntegerType - Represents the Serial number

2. name_country: StringType - Represents the Name of the player along with the country they represent

3. span: IntegerType - Represents the Duration of the player's career or playing span
4. Matches played: IntegerType - Represents the Total number of matches played by the player
5. Runs: IntegerType - Represents the Total runs scored by the playerHighScore
6. HighScore: StringType - Represents the Player's highest score in a single match
7. Average : IntegerType - Represents the Batting average of the player
8. Ballfaced: IntegerType - Represents the Total number of balls faced by the player
9. Century: IntegerType - Represents the Number of centuries scored by the player
10. boundary: IntegerType - Total number of boundaries (fours and sixes) hit by the player

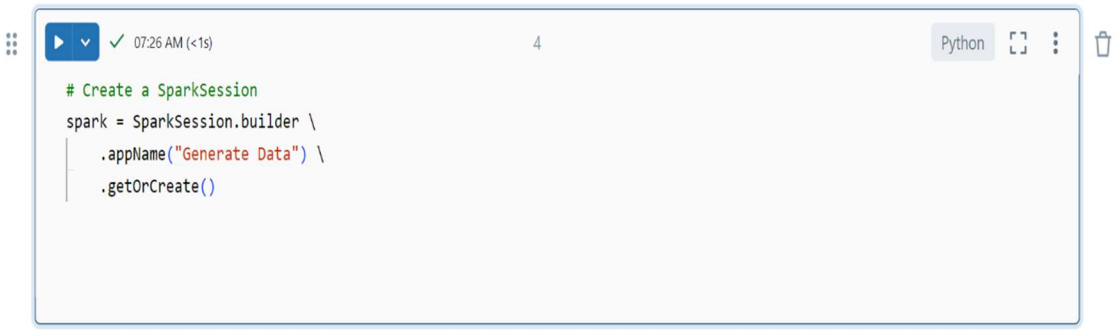


Step : 5

The line spark = SparkSession.builder \.appName("Generate Data").getOrCreate()

creates a Spark session named "GenerateData" using the SparkSession builder in Apache Spark.

1. **SparkSession:**
    - SparkSession is the entry point to programming Spark with the Dataset and DataFrame API.
    - It provides a unified interface for interacting with Spark functionality and allows you to work with structured data.
2. **Builder:**
    - The builder method is used to create a Builder object for configuring and setting up the Spark session.
3. **appName(“GenerateData”):**
    - This sets the name of the Spark application to "Generate Data". It's a user-defined name to identify the Spark job in the Spark UI
4. **getOrCreate ():**
    - The getOrCreate method checks if there is an existing Spark session available.
    - If an active Spark session exists, it returns that session. Otherwise, it creates a new Spark session based on the configuration set using the builder object.



GENERATE DATA :

```
data = []
for i in range(1, 1001):
    name = fake.name()
    country = random.choice(['INDIA', 'Australia', 'England', 'South Africa', 'New Zealand'])
    name_country = f"{name}({country})"
    high_score = random.randint(1, 200)

    # Add '*' randomly to high_score
    if random.choice([True, False]):
        high_score = str(high_score) + '*' if random.choice([True,False]) else str(int(high_score))

    # Create row with random values
    row = (i, name_country, random.randint(1, 20), random.randint(1, 500), random.randint(1, 20000),high_score,random.
    uniform(10, 60) , random.randint(1, 15000), random.randint(0, 10), random.randint(0, 100))
    data.append(row)

columns = ['S.no','name_country','span','Matches played','Runs','HighScore','Average','Ballfaced','Century','boundary']
print(data)
```

Random Data Generation:The code utilizes the random module to generate random values for various attributes such as name, country, span, matches played, runs, high score, average, ball faced, century, and boundary

The fake.name() function from the Faker library generates random names for individuals, adding to the diversity of the generated dataset.Country Selection:The country is randomly chosen from a list of countries including 'INDIA', 'Australia', 'England', 'South Africa', and 'New Zealand', adding geographical diversity to the dataset.

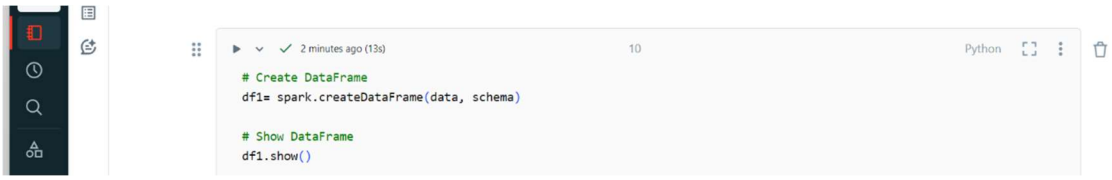
High Score Variation:There is variability introduced in the high score attribute by randomly adding an asterisk ('*') to some high scores based on a random choice of True or False.Data Structure:The generated data is organized into rows, with each row containing values for the different attributes specified in the columns list

Country Selection:The country is randomly chosen from a list of countries including 'INDIA', 'Australia', 'England', 'South Africa', and 'New Zealand', adding geographical diversity to the dataset

Loop Iteration:The for loop iterates 1000 times, generating data for each iteration and appending it to the data list.Data Output:The generated data is printed as a list of tuples, with each tuple representing a row of data.Potential

Improvements:Ensure consistent formatting and indentation to improve code readability.Consider encapsulating the data generation logic within a function for reusability.Add comments to clarify the purpose of each section of the code.

CREATE DATAFRAME:



```
2 minutes ago (13s)                    10                    Python

# Create DataFrame
df1= spark.createDataFrame(data, schema)

# Show DataFrame
df1.show()
```

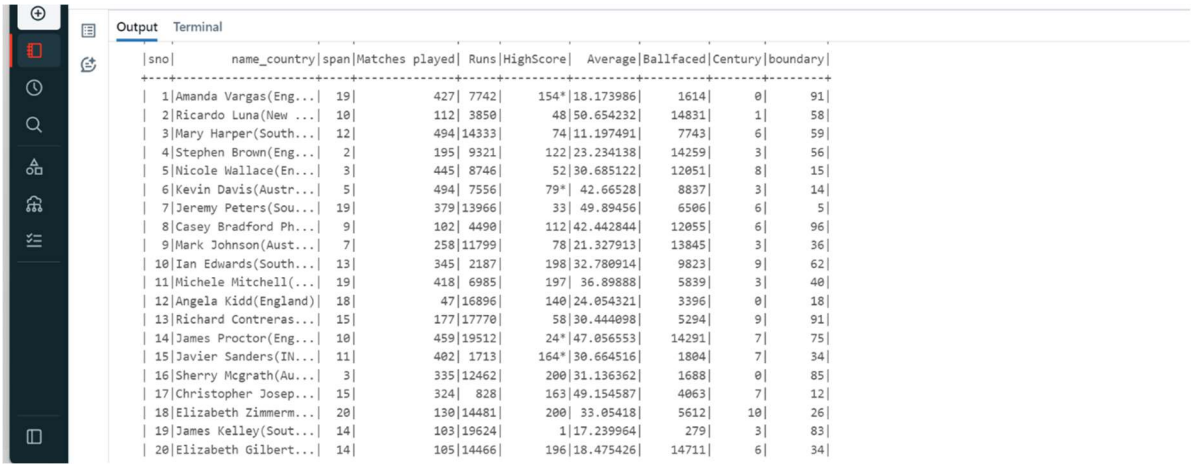
EXPLANATION:

- spark.createDataFrame(data, schema): Creates a DataFrame df1 using the generated data (data) and the specified schema (schema). The data variable contains the generated data, and the schema variable defines the structure of the DataFrame.

- show(): Displays the contents of the DataFrame df1 in a tabular format, showing the first few rows.

This code snippet effectively creates a DataFrame from the generated data and schema and then displays its contents.

INPUT SAMPLE DATA:



Output Terminal

sno	name_country	span	Matches played	Runs	HighScore	Average	Ballfaced	Century	boundary
1	Amanda Vargas(Eng...	19	427	7742	154*	18.173986	1614	0	91
2	Ricardo Luna(New ...	10	112	3850	48	50.654232	14831	1	58
3	Mary Harper(South...	12	494	14333	74	11.197491	7743	6	59
4	Stephen Brown(Eng...	2	195	9321	122	23.234138	14259	3	56
5	Nicole Wallace(En...	3	445	8746	52	30.685122	12051	8	15
6	Kevin Davis(Austr...	5	494	7556	79*	42.66528	8837	3	14
7	Jeremy Peters(Sou...	19	379	13966	33	49.89456	6506	6	5
8	Casey Bradford Ph...	9	102	4490	112	42.442844	12055	6	96
9	Mark Johnson(Aust...	7	258	11799	78	21.327913	13845	3	36
10	Ian Edwards(South...	13	345	2187	198	32.780914	9823	9	62
11	Michele Mitchell(...	19	418	6985	197	36.89888	5839	3	40
12	Angela Kidd(England)	18	47	16896	140	24.054321	3396	0	18
13	Richard Contreras...	15	177	17770	58	30.444098	5294	9	91
14	James Proctor(Eng...	10	459	19512	24*	47.056553	14291	7	75
15	Javier Sanders(IN...	11	402	1713	164*	30.664516	1804	7	34
16	Sherry Mcgrath(Au...	3	335	12462	200	31.136362	1688	0	85
17	Christopher Josep...	15	324	828	163	49.154587	4063	7	12
18	Elizabeth Zimmerm...	20	130	14481	200	33.05418	5612	10	26
19	James Kelley(Sout...	14	103	19624	1	17.239964	279	3	83
20	Elizabeth Gilbert...	14	105	14466	196	18.475426	14711	6	34

Step : 6

EXTRACT COUNTRY FROM name_country COLUMN

Explanation:

- This part of the code creates a new column named "Name" in DataFrame df .It uses the split function to split the values in the "name_country" column based on the regular expression pattern \(, which represents the opening parenthesis.

- The [0] index retrieves the first part of the split result, which corresponds to the player's name. The result is then assigned to the new "Name" column.

- This part of the code creates another new column named "Country" in the DataFrame df. It also uses the split function to split the values in the "name_country" column, this time based on the regular expression pattern " backslash(".

- The [1] index retrieves the second part of the split result, which corresponds to the country abbreviation.

The .substr(1, 3) function is then applied to this part to extract the first three characters, which typically represent the country code or abbreviation.Overall, this line of code is splitting the values in the "name_country" column into separate

columns for the player's name and country abbreviation, and it assigns these values to the new "Name" and "Country" columns, respectively, in the DataFrame df.

```
✓ 07:26 AM (<1s)                                                    17

df=df.drop("name_country")

▸ df: pyspark.sql.dataframe.DataFrame = [sno: integer, span: integer ... 9 more fields]
```

```
✓ 07:26 AM (1s)                                                     18

df.show()

▸ (1) Spark Jobs
```

RESULT:

Output Terminal

```
|sno|span|Matches played| Runs|HighScore|Average|Ballfaced|Century|boundary|              Name|Country|
+---+----+--------------+-----+---------+-------+---------+-------+--------+------------------+-------+
|  1|   9|           155|14445|      183|  55.69|    11971|     10|      23|  Meredith Wiggins|    Eng|
|  2|   6|            38| 3583|      173|  49.31|     5413|      2|      37|       Kevin Black|    New|
|  3|  11|           257|16056|      121|  44.08|    13340|      4|      39|      Joshua Smith|    Eng|
|  4|   2|           346|12802|        9|  44.66|     2335|      2|      88|Elizabeth Friedman|    Eng|
|  5|   8|            20| 9353|      168|  35.21|    13715|      0|       9|     Natalie Scott|    Eng|
|  6|   2|            43|15935|      87*|   20.6|     3186|      6|       8|          Amy Cruz|    Aus|
|  7|   6|           193| 3647|      185|  11.69|     6936|      8|      18|    Karen Costa MD|    Eng|
|  8|   3|           105| 6532|      73*|  10.05|    12404|      6|      94|      Amanda Gates|    IND|
|  9|   7|           487|19690|     176*|  37.42|     5219|      3|      79|      David Flores|    Aus|
| 10|   9|           213|15390|        5|  26.53|     5545|      2|      70|    Ricky Mccarthy|    Eng|
| 11|   5|           429|12760|      43*|  46.57|    13333|      4|      10|  Nichole Mcdonald|    Aus|
| 12|   4|           133|  284|     166*|  25.94|     8628|      6|      37|      Amanda Mills|    Sou|
| 13|   7|           307|  527|       1*|  48.55|     7232|      1|      11|   Zachary Johnson|    Eng|
| 14|   7|           196|  903|      186|  12.21|     1247|      7|      26|   Jessica Maxwell|    Eng|
| 15|  13|           257| 5432|      139|  21.58|     8944|      5|      18|      Anna Douglas|    New|
| 16|  14|           455|15379|       68|  32.01|     2365|      3|      33|  Elizabeth Stokes|    Eng|
| 17|  19|           341| 9600|      33*|  22.47|     4478|      6|      36|    Pamela Schultz|    New|
| 18|   9|           347|15641|     128*|  36.77|     7788|      3|      59|      Robert Dixon|    Sou|
| 19|  19|           189|16752|     187*|  47.79|     9843|      9|      66| Marissa Patterson|    New|
| 20|   7|           454|17440|      93*|  55.95|    13539|      5|      39|   Tasha Rodriguez|    IND|
+---+----+--------------+-----+---------+-------+---------+-------+--------+------------------+-------+
```

Step : 7

ROUND THE AVERAGE COLUMN:

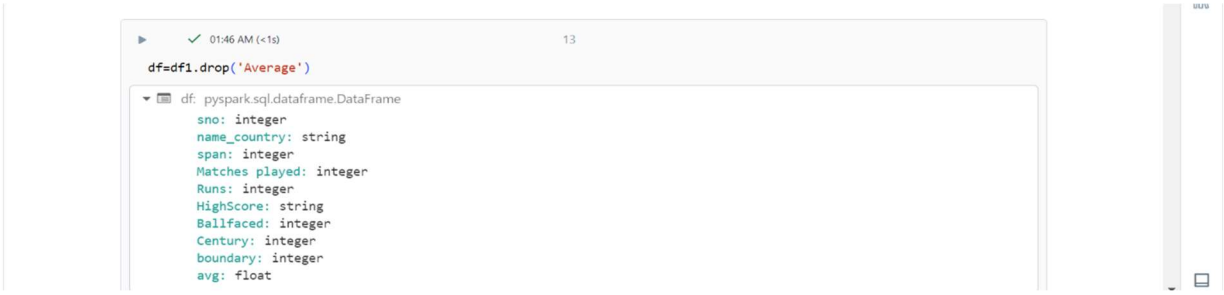
EXPLANATION:

- withColumn("avg", round(col('Average'), 2)): Adds a new column named "avg"to the DataFrame df1, containing rounded values of the "Average" column to two decimal places using the round() function.

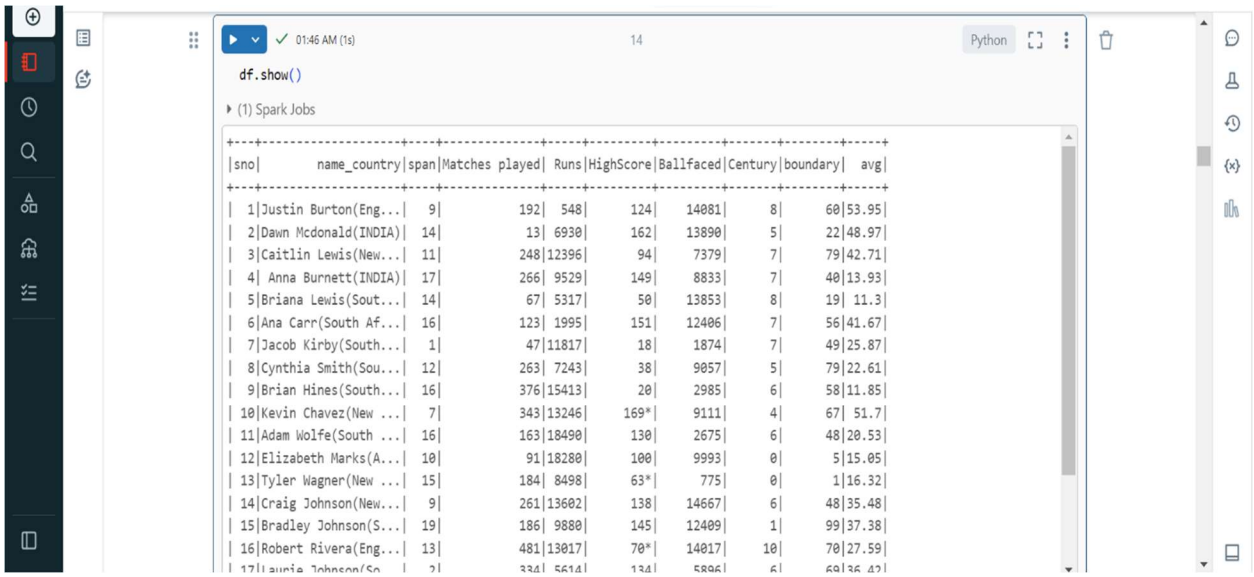
- drop("Average"): Drops the original "Average" column from the DataFrame df1, creating a new DataFrame df without the "Average" column.

This code snippet will add a new column "avg" with rounded values of the "Average" column and drop the original "Average" column from the DataFrame. Make sure to replace "Average" with the actual column name if it's different in your DataFrame.

DROP THE AVERAGE COLUMN



RESULT:



```
df.show()
(1) Spark Jobs
+---+--------------------+----+--------------+-----+---------+---------+-------+--------+-----+
|sno|        name_country|span|Matches played| Runs|HighScore|Ballfaced|Century|boundary|  avg|
+---+--------------------+----+--------------+-----+---------+---------+-------+--------+-----+
|  1|Justin Burton(Eng...|   9|           192|  548|      124|    14081|      8|      60|53.95|
|  2|Dawn Mcdonald(INDIA)|  14|            13| 6930|      162|    13890|      5|      22|48.97|
|  3|Caitlin Lewis(New...|  11|           248|12396|       94|     7379|      7|      79|42.71|
|  4| Anna Burnett(INDIA)|  17|           266| 9529|      149|     8833|      7|      40|13.93|
|  5|Briana Lewis(Sout...|  14|            67| 5317|       50|    13853|      8|      19| 11.3|
|  6|Ana Carr(South Af...|  16|           123| 1995|      151|    12406|      7|      56|41.67|
|  7|Jacob Kirby(South...|   1|            47|11817|       18|     1874|      7|      49|25.87|
|  8|Cynthia Smith(Sou...|  12|           263| 7243|       38|     9057|      5|      79|22.61|
|  9|Brian Hines(South...|  16|           376|15413|       20|     2985|      6|      58|11.85|
| 10|Kevin Chavez(New ...|   7|           343|13246|     169*|     9111|      4|      67| 51.7|
| 11|Adam Wolfe(South ...|  16|           163|18490|      130|     2675|      6|      48|20.53|
| 12|Elizabeth Marks(A...|  10|            91|18280|      100|     9993|      0|       5|15.05|
| 13|Tyler Wagner(New ...|  15|           184| 8498|      63*|      775|      0|       1|16.32|
| 14|Craig Johnson(New...|   9|           261|13602|      138|    14667|      6|      48|35.48|
| 15|Bradley Johnson(S...|  19|           186| 9880|      145|    12409|      1|      99|37.38|
| 16|Robert Rivera(Eng...|  13|           481|13017|      70*|    14017|     10|      70|27.59|
```

Step : 8

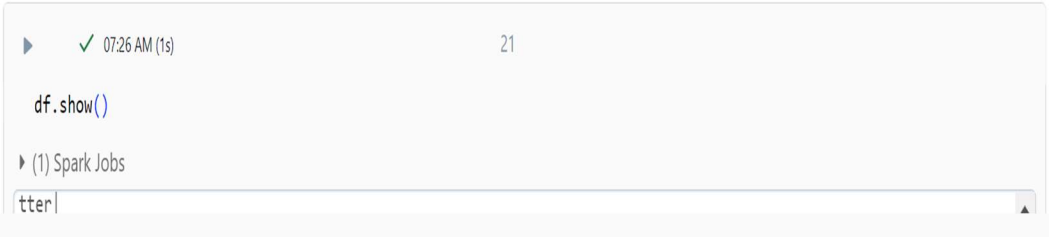
DISPLAY GREATER THAN 10 BOUNDARY (HIGH BOUNDARY HITTER) OR LESS THAN 10 (LOW BOUNDARY HITTER)

when(df['boundary'] >= 10, 'High Boundary Hitter'):

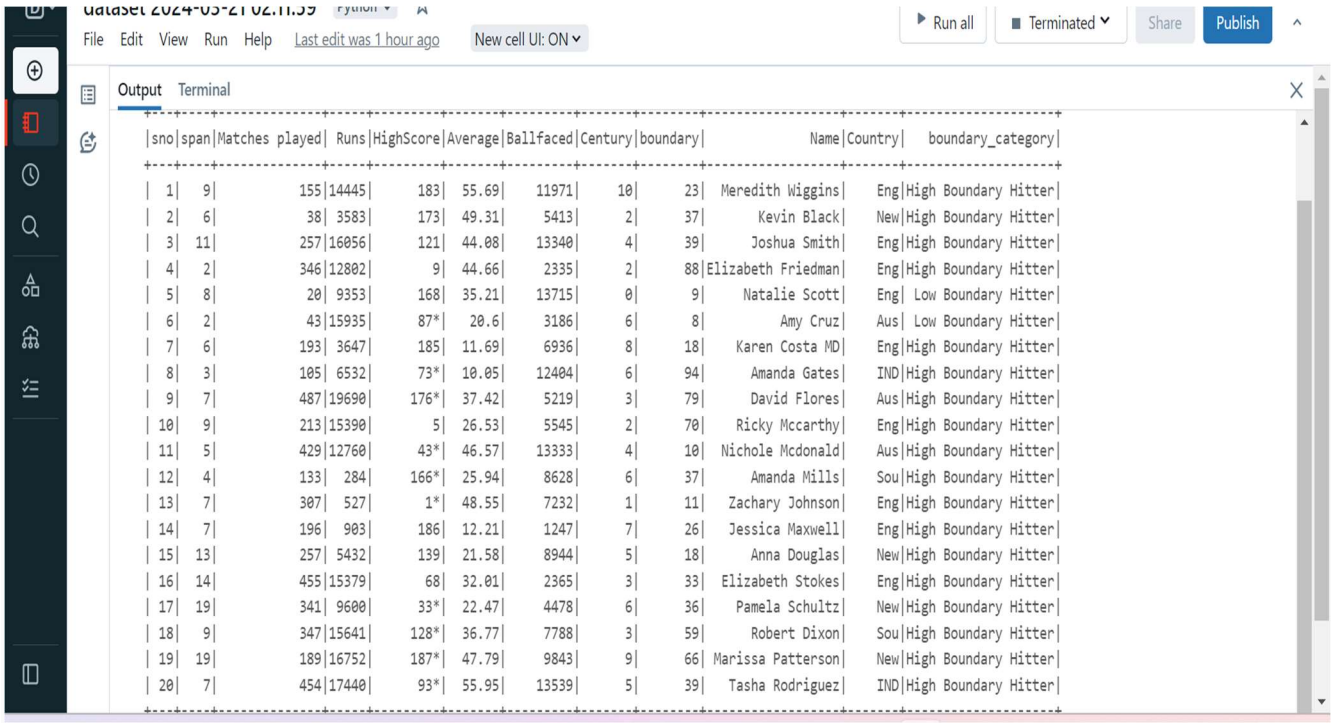
- This part of the code specifies a condition: if the value in the 'boundary' column is greater than or equal to 10, then the value 'High Boundary Hitter' is assigned to the new 'boundary_category' column.

otherwise('Low Boundary Hitter'):

- This part of the code specifies the default value if the condition in the when function is not met.
- If the value in the 'boundary' column is less than 10, then the value 'Low Boundary Hitter' is assigned to the new 'boundary_category' column.



RESULT:



sno	span	Matches played	Runs	HighScore	Average	Ballfaced	Century	boundary	Name	Country	boundary_category
1	9	155	14445	183	55.69	11971	10	23	Meredith Wiggins	Eng	High Boundary Hitter
2	6	38	3583	173	49.31	5413	2	37	Kevin Black	New	High Boundary Hitter
3	11	257	16056	121	44.08	13340	4	39	Joshua Smith	Eng	High Boundary Hitter
4	2	346	12802	9	44.66	2335	2	88	Elizabeth Friedman	Eng	High Boundary Hitter
5	8	20	9353	168	35.21	13715	0	9	Natalie Scott	Eng	Low Boundary Hitter
6	2	43	15935	87*	20.6	3186	6	8	Amy Cruz	Aus	Low Boundary Hitter
7	6	193	3647	185	11.69	6936	8	18	Karen Costa MD	Eng	High Boundary Hitter
8	3	105	6532	73*	10.05	12404	6	94	Amanda Gates	IND	High Boundary Hitter
9	7	487	19690	176*	37.42	5219	3	79	David Flores	Aus	High Boundary Hitter
10	9	213	15390	5	26.53	5545	2	70	Ricky Mccarthy	Eng	High Boundary Hitter
11	5	429	12760	43*	46.57	13333	4	10	Nichole Mcdonald	Aus	High Boundary Hitter
12	4	133	284	166*	25.94	8628	6	37	Amanda Mills	Sou	High Boundary Hitter
13	7	307	527	1*	48.55	7232	1	11	Zachary Johnson	Eng	High Boundary Hitter
14	7	196	903	186	12.21	1247	7	26	Jessica Maxwell	Eng	High Boundary Hitter
15	13	257	5432	139	21.58	8944	5	18	Anna Douglas	New	High Boundary Hitter
16	14	455	15379	68	32.01	2365	3	33	Elizabeth Stokes	Eng	High Boundary Hitter
17	19	341	9600	33*	22.47	4478	6	36	Pamela Schultz	New	High Boundary Hitter
18	9	347	15641	128*	36.77	7788	3	59	Robert Dixon	Sou	High Boundary Hitter
19	19	189	16752	187*	47.79	9843	9	66	Marissa Patterson	New	High Boundary Hitter
20	7	454	17440	93*	55.95	13539	5	39	Tasha Rodriguez	IND	High Boundary Hitter

STEP: 9

DISPLAY THE STRIKERATE:

new_df = df.withColumn("StrikeRate", round(expr("Runs / Ballfaced * 100"), 2)):

withColumn: This is a DataFrame function in Spark that allows adding a new **column to an existing DataFrame.** The withColumn method is used to create a new DataFrame by adding a column to the existing DataFrame df. It doesn't modify the original DataFrame but returns a new DataFrame with the specified transformation.

"StrikeRate": This is the name of the new column that will be added to the DataFrame.

Calculation: The strike rate is calculated by dividing the "Runs" column by the "Ballfaced" column for each row in the DataFrame. The result is multiplied by 100 to convert it into a percentage.

round(expr("Runs / Ballfaced * 100"), 2): This part calculates the strike rate by dividing the "Runs" column by the "Ballfaced" column and then multiplying by 100 to get the percentage.

expr: The expr function allows using SQL expressions within the Spark DataFrame API. The round function rounds the result to two decimal places

new_df.show(): This displays the contents of the DataFrame new_df, showing the newly added "StrikeRate" column along with existing columns

In summary, this code adds a new column called "StrikeRate" to the DataFrame df, which calculates the strike rate based on the "Runs" and "Ballfaced" columns, and then displays the modified DataFrame.

Calculate StrikeRate (runs per 100balls faced):

Strike rate in cricket refers to the number of runs scored by a batsman per 100 balls faced. It is a measure of how quickly a batsman scores runs.

RESULT:

Output Terminal

sno	span	Matches played	Runs	HighScore	Average	Ballfaced	Century	boundary	Name	Country	boundary_category	StrikeRate
1	1	194	8608	157	34.2	13167	2	43	Megan Garcia	New	High Boundary Hitter	65.38
2	15	377	8379	161	35.44	9477	0	49	Thomas Alexander	Aus	High Boundary Hitter	88.41
3	7	439	14389	120	29.35	13557	2	100	Nicholas Gibbs	Sou	High Boundary Hitter	106.14
4	20	397	13807	171	46.49	7353	1	54	Christopher Bailey	Aus	High Boundary Hitter	187.77
5	2	399	18818	36	47.18	4118	4	7	Kenneth Ferguson	Aus	Low Boundary Hitter	456.97
6	11	462	1685	123*	50.27	13660	0	15	Andrew Long	Aus	High Boundary Hitter	12.34
7	9	423	4435	146	58.7	2922	10	62	Gilbert Carlson	New	High Boundary Hitter	151.78
8	18	118	16136	20	22.0	11427	5	21	Regina Ray	IND	High Boundary Hitter	141.21
9	8	10	18591	196*	18.47	2777	9	38	Michelle Lam	Sou	High Boundary Hitter	669.46
10	3	97	17868	8*	30.57	6908	9	78	Sara Lewis	IND	High Boundary Hitter	258.66
11	2	103	9221	36*	50.27	10836	3	7	James Gomez	New	Low Boundary Hitter	85.1
12	6	82	12466	22*	38.67	10927	5	3	Todd Graves	Eng	Low Boundary Hitter	114.08
13	6	16	8772	52*	48.32	7439	10	96	Jessica Price	Aus	High Boundary Hitter	117.92
14	4	324	1129	148*	37.71	10175	7	29	Heidi Brady	New	High Boundary Hitter	11.1
15	4	472	12510	40	22.31	1969	7	66	Steven Flynn	Eng	High Boundary Hitter	635.35
16	5	142	4847	77*	22.57	9313	1	30	Elizabeth Tran	IND	High Boundary Hitter	52.05
17	14	291	6796	165*	40.05	10485	5	81	Lauren Romero	Sou	High Boundary Hitter	64.82
18	3	5	15232	196	39.42	4183	7	21	Amy Anderson	New	High Boundary Hitter	364.14
19	3	19	439	181	34.26	9128	8	27	Tiffany Medina	Eng	High Boundary Hitter	4.81
20	12	189	1699	186*	54.16	12097	3	95	Amy May	IND	High Boundary Hitter	14.04

Step : 10

ADD A NEW COLUMN CATEGORIZING PLAYERS BASED ON BATTING AVERAGE:

EXPLANATION:

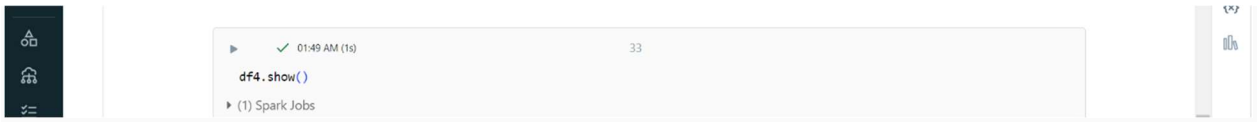
when function:

- The when function in PySpark is used for conditional operations. It evaluates a condition and returns a value if the condition is true.

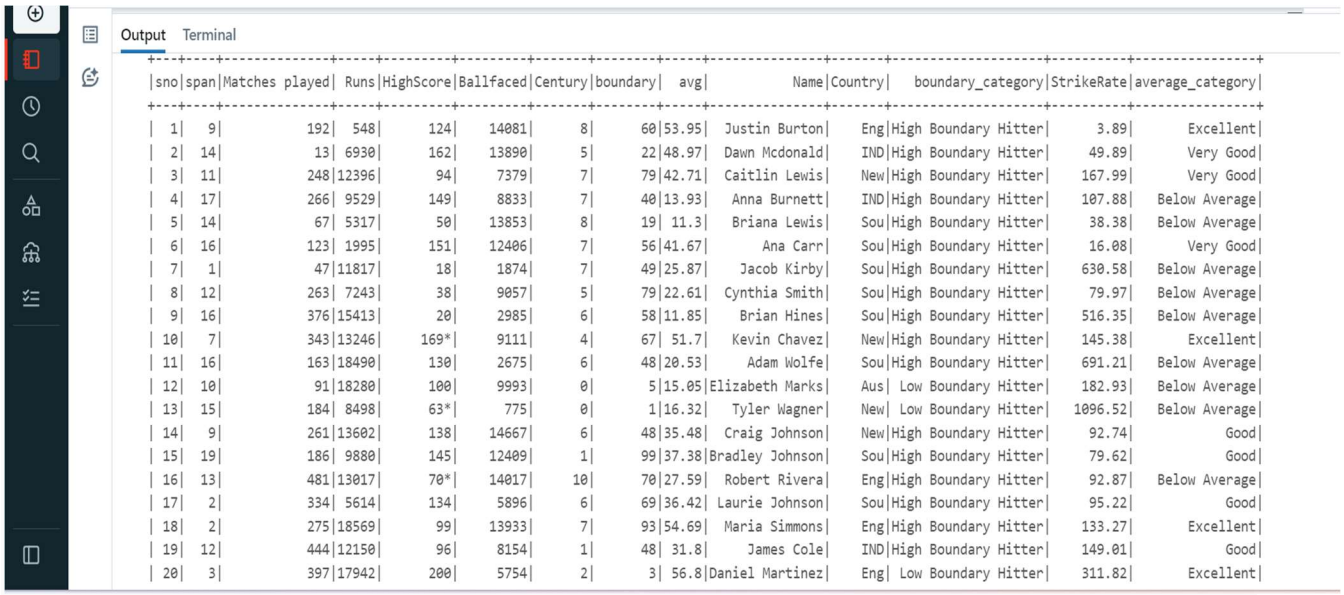
withColumn("average_category", ...):

- This adds a new column named "average_category" to the DataFrame df based on the conditions provided.

- First condition: when(col("Average") >= 50, "Excellent"): If the "Average" column value is greater than or equal to 50, it assigns the value "Excellent" to the "average_category" column.



RESULT:



sno	span	Matches played	Runs	HighScore	Ballfaced	Century	boundary	avg	Name	Country	boundary_category	StrikeRate	average_category
1	9	192	548	124	14081	8	60	53.95	Justin Burton	Eng	High Boundary Hitter	3.89	Excellent
2	14	13	6930	162	13890	5	22	48.97	Dawn Mcdonald	IND	High Boundary Hitter	49.89	Very Good
3	11	248	12396	94	7379	7	79	42.71	Caitlin Lewis	New	High Boundary Hitter	167.99	Very Good
4	17	266	9529	149	8833	7	40	13.93	Anna Burnett	IND	High Boundary Hitter	107.88	Below Average
5	14	67	5317	50	13853	8	19	11.3	Briana Lewis	Sou	High Boundary Hitter	38.38	Below Average
6	16	123	1995	151	12406	7	56	41.67	Ana Carr	Sou	High Boundary Hitter	16.08	Very Good
7	1	47	11817	18	1874	7	49	25.87	Jacob Kirby	Sou	High Boundary Hitter	630.58	Below Average
8	12	263	7243	38	9057	5	79	22.61	Cynthia Smith	Sou	High Boundary Hitter	79.97	Below Average
9	16	376	15413	20	2985	6	58	11.85	Brian Hines	Sou	High Boundary Hitter	516.35	Below Average
10	7	343	13246	169*	9111	4	67	51.7	Kevin Chavez	New	High Boundary Hitter	145.38	Excellent
11	16	163	18490	130	2675	6	48	20.53	Adam Wolfe	Sou	High Boundary Hitter	691.21	Below Average
12	10	91	18280	100	9993	0	5	15.05	Elizabeth Marks	Aus	Low Boundary Hitter	182.93	Below Average
13	15	184	8498	63*	775	0	1	16.32	Tyler Wagner	New	Low Boundary Hitter	1096.52	Below Average
14	9	261	13602	138	14667	6	48	35.48	Craig Johnson	New	High Boundary Hitter	92.74	Good
15	19	186	9880	145	12409	1	99	37.38	Bradley Johnson	Sou	High Boundary Hitter	79.62	Good
16	13	481	13017	70*	14017	10	70	27.59	Robert Rivera	Eng	High Boundary Hitter	92.87	Below Average
17	2	334	5614	134	5896	6	69	36.42	Laurie Johnson	Sou	High Boundary Hitter	95.22	Good
18	2	275	18569	99	13933	7	93	54.69	Maria Simmons	Eng	High Boundary Hitter	133.27	Excellent
19	12	444	12150	96	8154	1	48	31.8	James Cole	IND	High Boundary Hitter	149.01	Good
20	3	397	17942	200	5754	2	3	56.8	Daniel Martinez	Eng	Low Boundary Hitter	311.82	Excellent

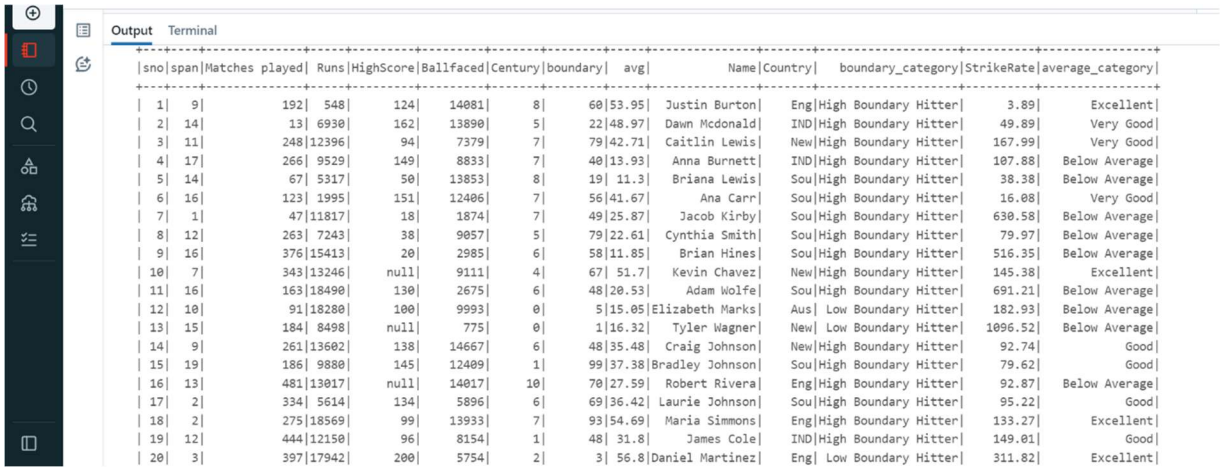
Step : 11

CASTING THE HIGHSCORE COLUMN

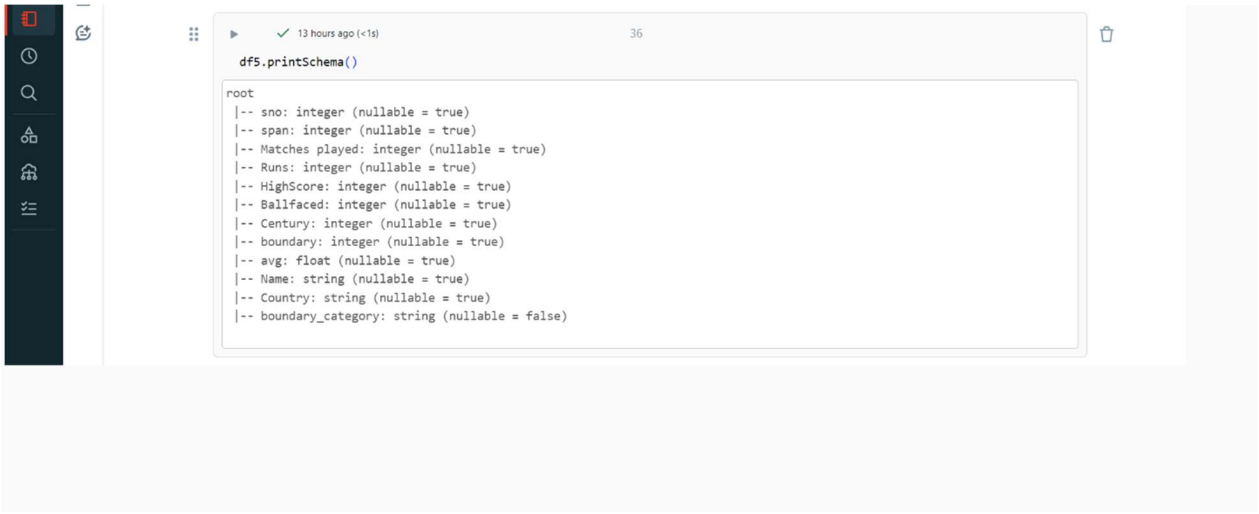
EXPLANATION:

- df4: Represents a DataFrame.
- withColumn(): Method to add a new column or replace an existing one.
- "HighScore": Name of the column to be modified.
- col("HighScore"): Accesses the values in the "HighScore" column.
- cast("int"): Casts the values in the "HighScore" column to integer type.

RESULT:


Output Terminal

sno	span	Matches played	Runs	HighScore	Ballfaced	Century	boundary	avg	Name	Country	boundary_category	StrikeRate	average_category
1	9	192	548	124	14081	8	60	53.95	Justin Burton	Eng	High Boundary Hitter	3.89	Excellent
2	14	13	6930	162	13890	5	22	48.97	Dawn Mcdonald	IND	High Boundary Hitter	49.89	Very Good
3	11	248	12396	94	7379	7	79	42.71	Caitlin Lewis	New	High Boundary Hitter	167.99	Very Good
4	17	266	9529	149	8833	7	40	13.93	Anna Burnett	IND	High Boundary Hitter	107.88	Below Average
5	14	67	5317	50	13853	8	19	11.3	Briana Lewis	Sou	High Boundary Hitter	38.38	Below Average
6	16	123	1995	151	12406	7	56	41.67	Ana Carr	Sou	High Boundary Hitter	16.08	Very Good
7	1	47	11817	18	1874	7	49	25.87	Jacob Kirby	Sou	High Boundary Hitter	630.58	Below Average
8	12	263	7243	38	9057	5	79	22.61	Cynthia Smith	Sou	High Boundary Hitter	79.97	Below Average
9	16	376	15413	20	2985	6	58	11.85	Brian Hines	Sou	High Boundary Hitter	516.35	Below Average
10	7	343	13246	null	9111	4	67	51.7	Kevin Chavez	New	High Boundary Hitter	145.38	Excellent
11	16	163	18490	130	2675	6	48	20.53	Adam Wolfe	Sou	High Boundary Hitter	691.21	Below Average
12	10	91	18280	100	9993	0	5	15.05	Elizabeth Marks	Aus	Low Boundary Hitter	182.93	Below Average
13	15	184	8498	null	775	0	1	16.32	Tyler Wagner	New	Low Boundary Hitter	1096.52	Below Average
14	9	261	13602	138	14667	6	48	35.48	Craig Johnson	New	High Boundary Hitter	92.74	Good
15	19	186	9880	145	12409	1	99	37.38	Bradley Johnson	Sou	High Boundary Hitter	79.62	Good
16	13	481	13017	null	14017	10	70	27.59	Robert Rivera	Eng	High Boundary Hitter	92.87	Below Average
17	2	334	5614	134	5896	6	69	36.42	Laurie Johnson	Sou	High Boundary Hitter	95.22	Good
18	2	275	18569	99	13933	7	93	54.69	Maria Simmons	Eng	High Boundary Hitter	133.27	Excellent
19	12	444	12150	96	8154	1	48	31.8	James Cole	IND	High Boundary Hitter	149.01	Good
20	3	397	17942	200	5754	2	3	56.8	Daniel Martinez	Eng	Low Boundary Hitter	311.82	Excellent



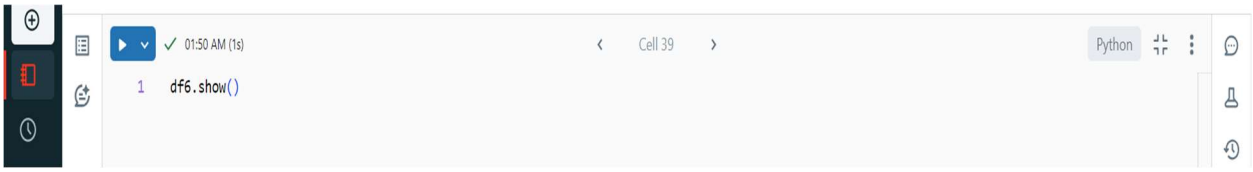
Step : 12

DEFINE THE TRANSFORMATION BINS FOR THE 'SPAN' COLUMN

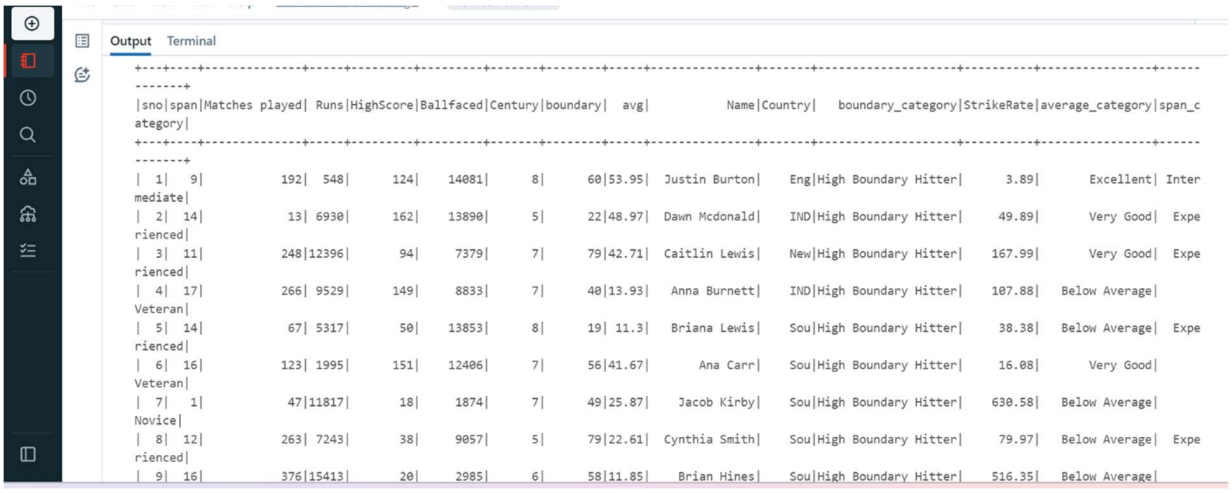
EXPLANATION:

- df5: Represents a DataFrame.
- withColumn(): Method to add a new column or replace an existing one.
- "span_category": Name of the new column to be added.
- when(): Conditional expression function that allows specifying conditions.
- col("span"): Accesses the values in the "span" column.
- otherwise(): Specifies the default value if none of the conditions are met.
- "Novice", "Intermediate", "Experienced", "Veteran": Categories based on the duration of the span.

- After executing this code, the DataFrame df6 will have a new column named "span_category" containing the categories based on the duration specified in the "span" column.



RESULT:



Step : 13

COUNT THE NO OF INDIAN PLAYER:

EXPLANATION:

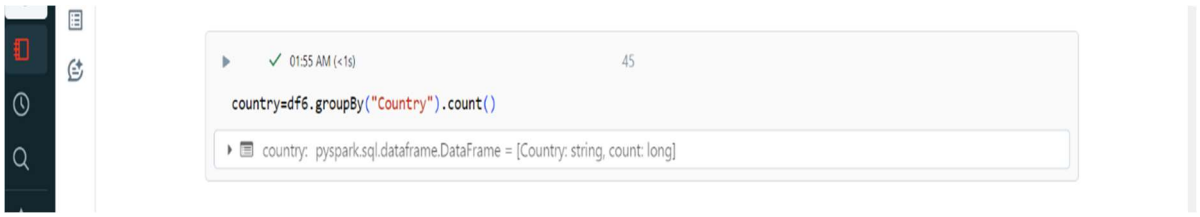
- df6: Represents the original DataFrame with the new "span_category" column.
- filter(col("Country") == "IND"): Filters rows where the "Country" column equals "IND" (India).
- select("Name", "Country"): Selects only the "Name" and "Country" columns from the filtered DataFrame.
- show(): Displays the selected columns in tabular format.

This code snippet will display the names and corresponding countries of the individuals from India in the DataFrame df6. Make sure the column names ("Country" and "Name") match the actual column names in your DataFrame.

RESULT:

Step : 14

COUNT OF EACH COUNTRY



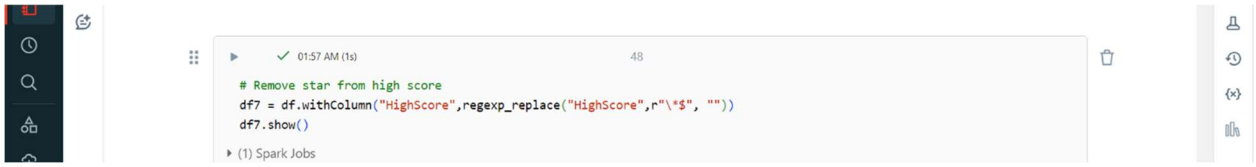
EXPLANATION:

- groupBy("Country"): Groups the DataFrame by the "Country" column.
- count(): Counts the number of occurrences of each group.
- show(): Displays the result with counts for each country.

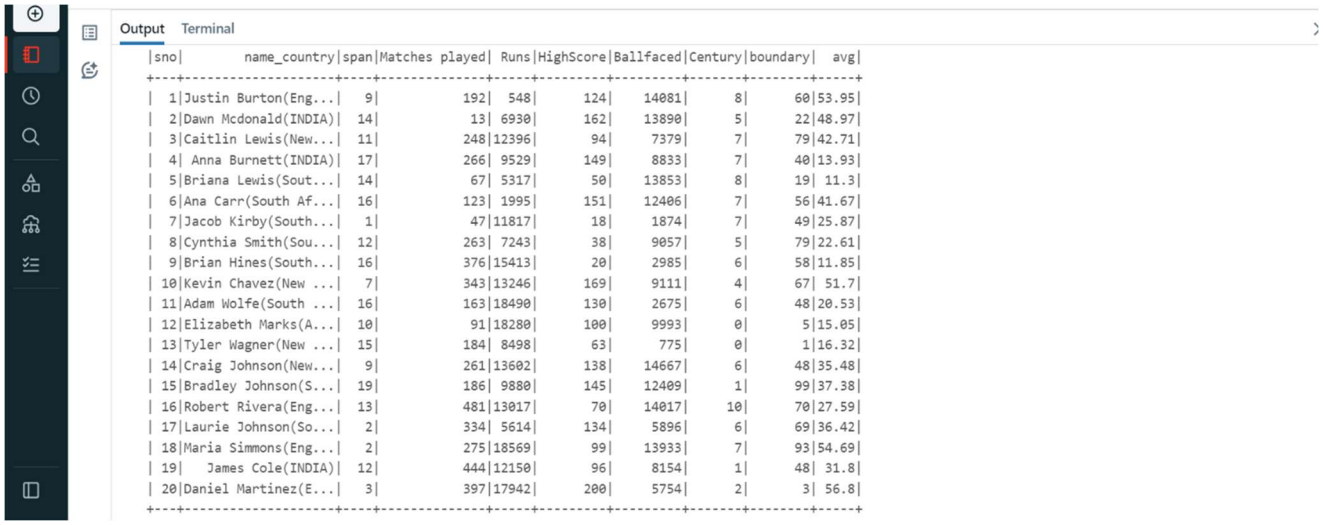
This code snippet will show the count of occurrences for each country in the DataFrame df6. Make sure the column name "Country" matches the actual column name in your DataFrame.

Step : 15

REMOVING *ASTERISK FROM HIGHSCORE COLUMN



RESULT:



EXPLANATION:

- withColumn("HighScore", regexp_replace("HighScore", r"*$", "")):
  Creates a new column "HighScore" in the DataFrame df by replacing the

asterisk character ('*') at the end of each value in the "HighScore" column with an empty string.

- regexp_replace: Function to replace substrings that match a regular expression pattern with a replacement string.
- r"*$": Regular expression pattern to match the asterisk character ('*') at the end of the string.
- " ": Replacement string, in this case, an empty string, indicating that the asterisk character should be removed.
- show(): Displays the resulting DataFrame with the modified "HighScore" column.

Step : 16

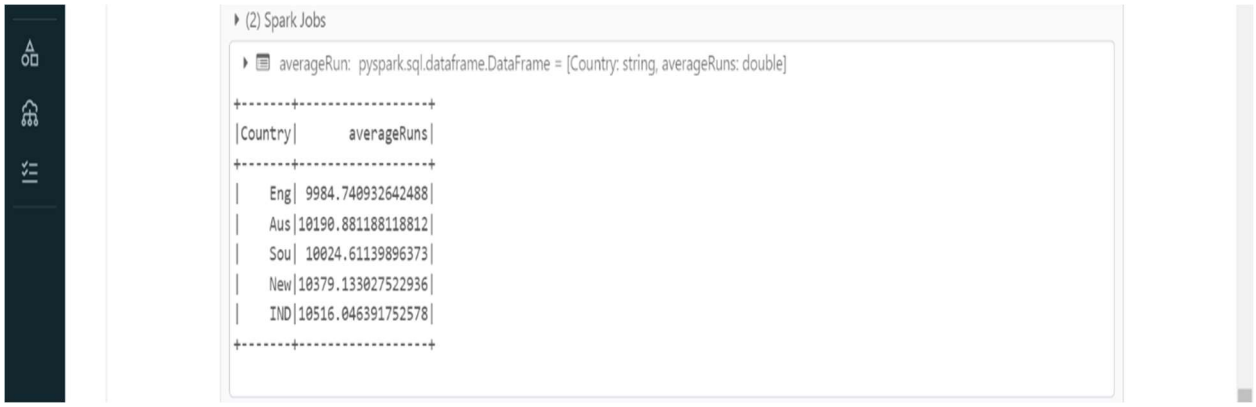
AVERAGE RUNS OF EACH COUNTRY:

EXPLANATION:

- groupBy("Country"): Groups the DataFrame df6 by the "Country" column.

- agg(avg("Runs").alias("averageRuns")): Aggregates the grouped data, calculating the average value of the "Runs" column for each group. The alias("averageRuns") method is used to alias the resulting column as "averageRuns".
- avg("Runs"): Calculates the average value of the "Runs" column within each group.
- show(): Displays the resulting DataFrame with the calculated average runs for each country.

This code snippet will show the average runs scored by players from each country in the DataFrame df6. Make sure the column names ("Country" and "Runs") match the actual column names in your DataFrame



Step : 17

HIGHEST CENTURIES

EXPLANATION:

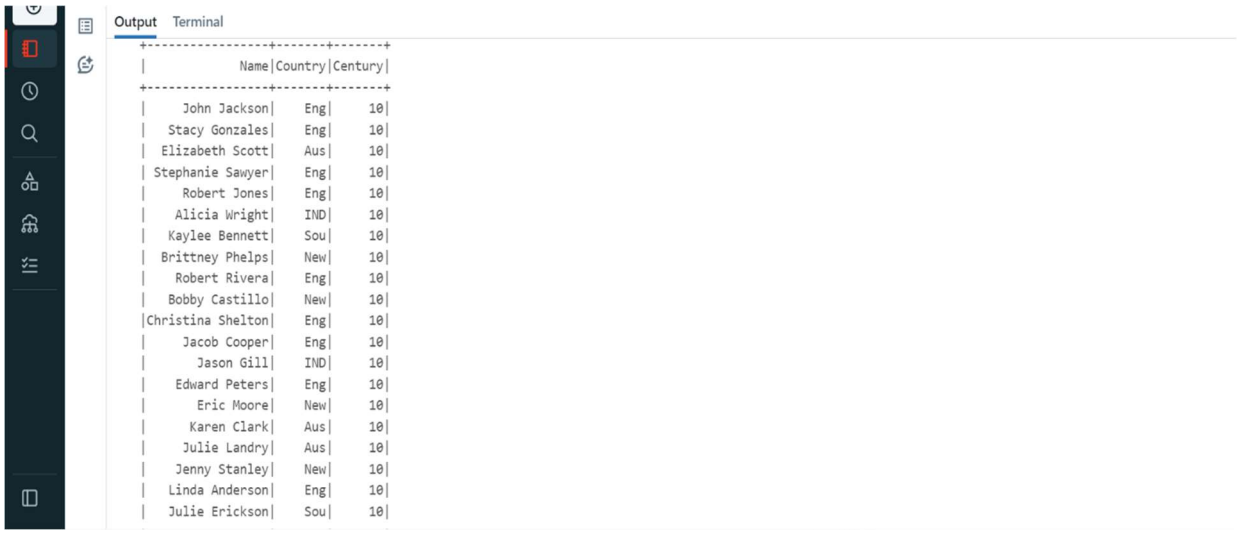
sort(df6["Century"].desc()): Sorts the DataFrame df6 in descending order based on the "Century" column

limit(1): Limits the result to one row, which will contain the player with the most centuries.

select("Name", "Country", "Century"): Selects the specified columns ("Name", "Country", and "Century") from the limited DataFrame.

show(): Displays the resulting DataFrame with the player who has scored the most centuries along with their name, country, and number of centuries.

This code snippet will show the player who has scored the most centuries along with their name, country, and the number of centuries in the DataFrame df6. Make sure the column names ("Name", "Country", and "Century") match the actual column names in your DataFrame.



Step : 18

DISPLAY THE HIGH SPAN:

EXPLANATION:

sort(df6["Century"].desc()): Sorts the DataFrame df6 in descending order based on the "Century" column.

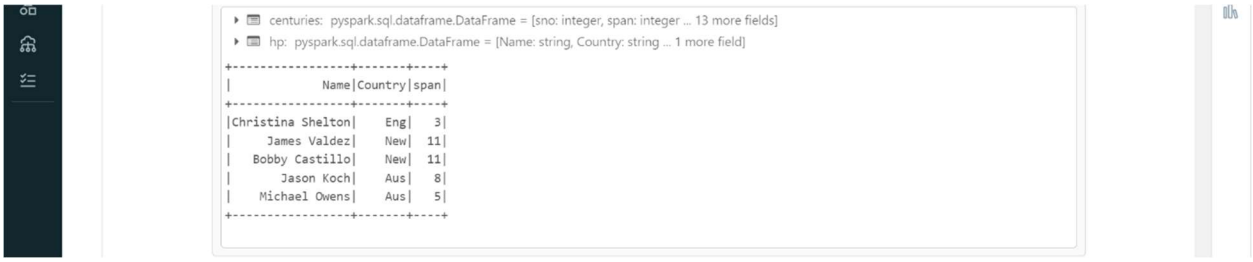
limit(5): Limits the result to the top 5 rows, representing the players with the highest number of centuries.

select("Name", "Country", "span"): Selects the specified columns ("Name", "Country", and "span") from the limited DataFrame.

show(): Displays the resulting DataFrame with the names, countries, and spans of the top 5 players with the highest number of centuries.

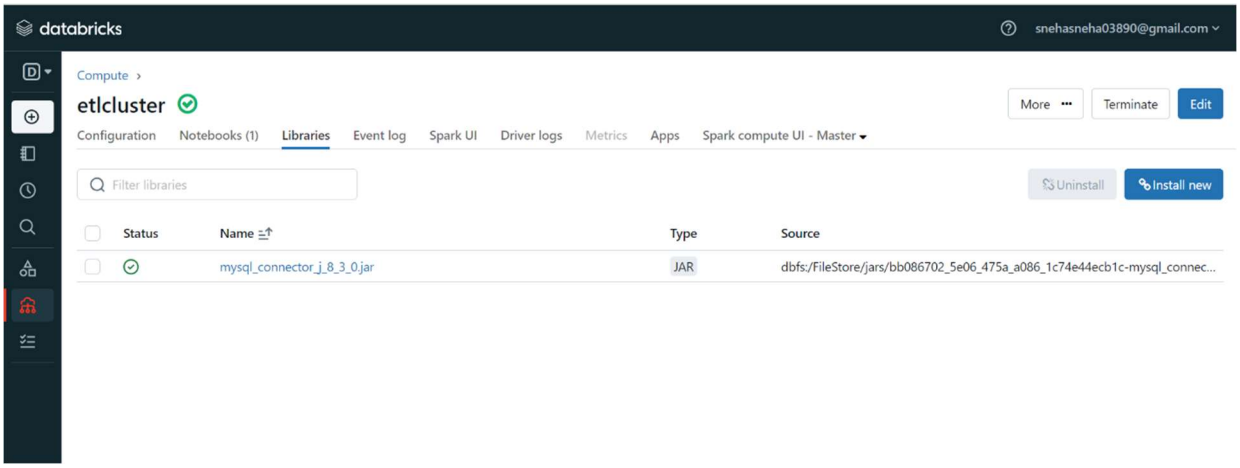
This code will show the top 5 players with the highest number of centuries along with their names, countries, and spans from the DataFrame df6. Make sure the column names ("Name", "Country", and "span") match the actual column names in your DataFrame. If there's any specific error or issue you're facing, please let me know.

RESULT:

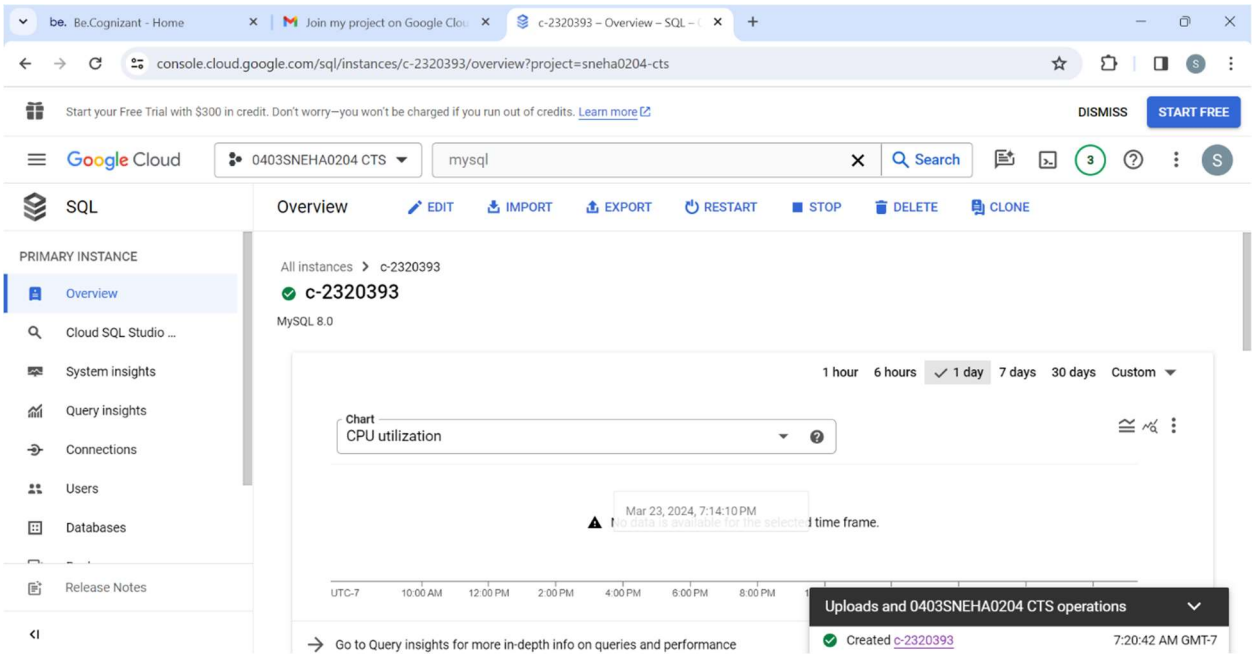


Step 19:

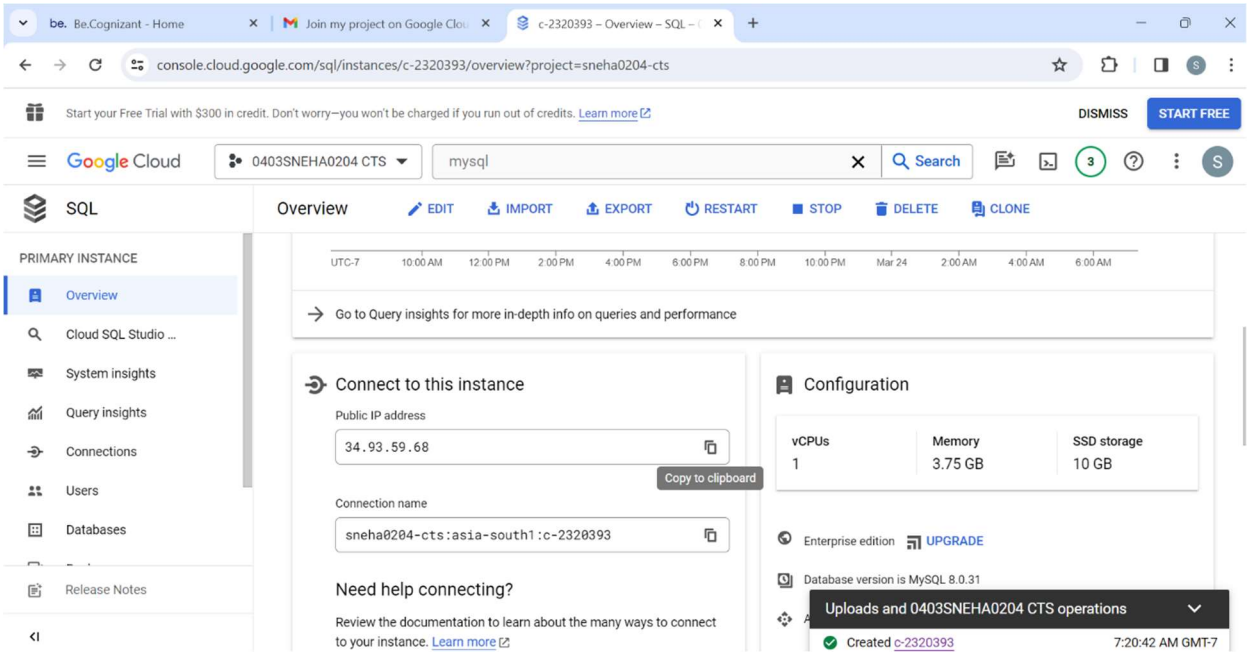
Download To mySQL connector from Google and upload the file in Databricks libraries.



Open the Google cloud console and create CloudSQL instance.



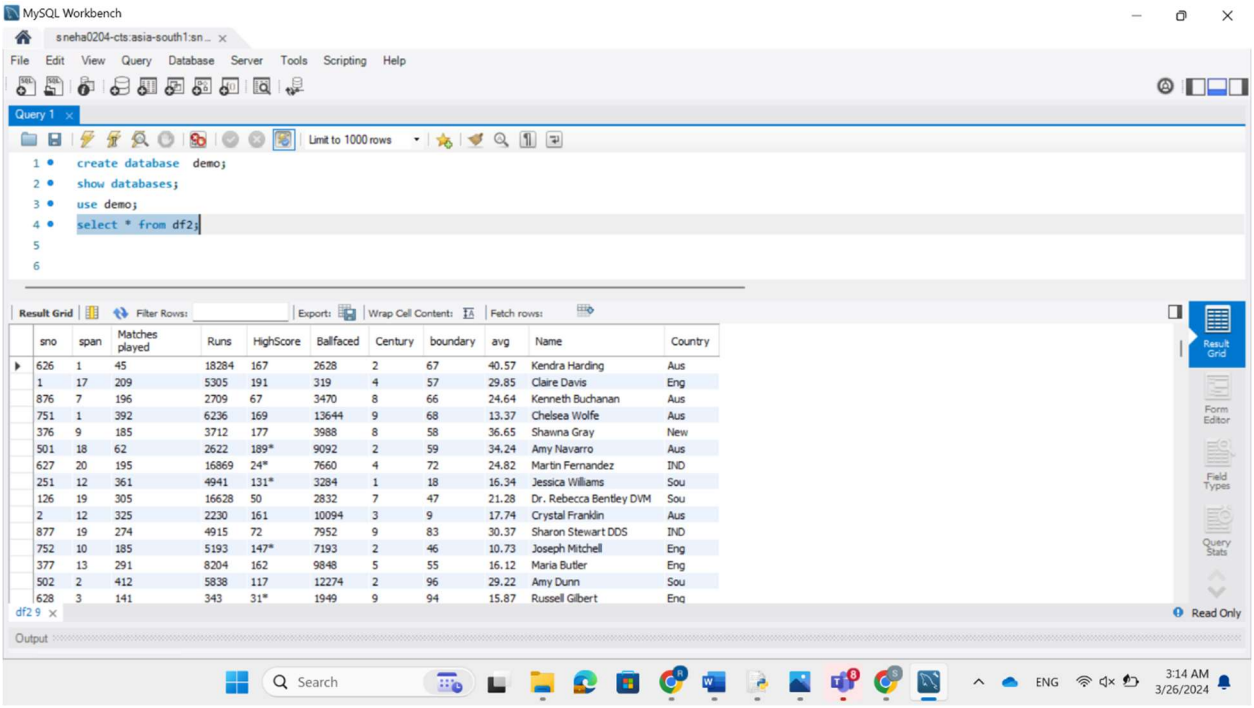
Open the Google Cloud and create the cloudSQL . After creating the cloudSQL , copy the public IP address.

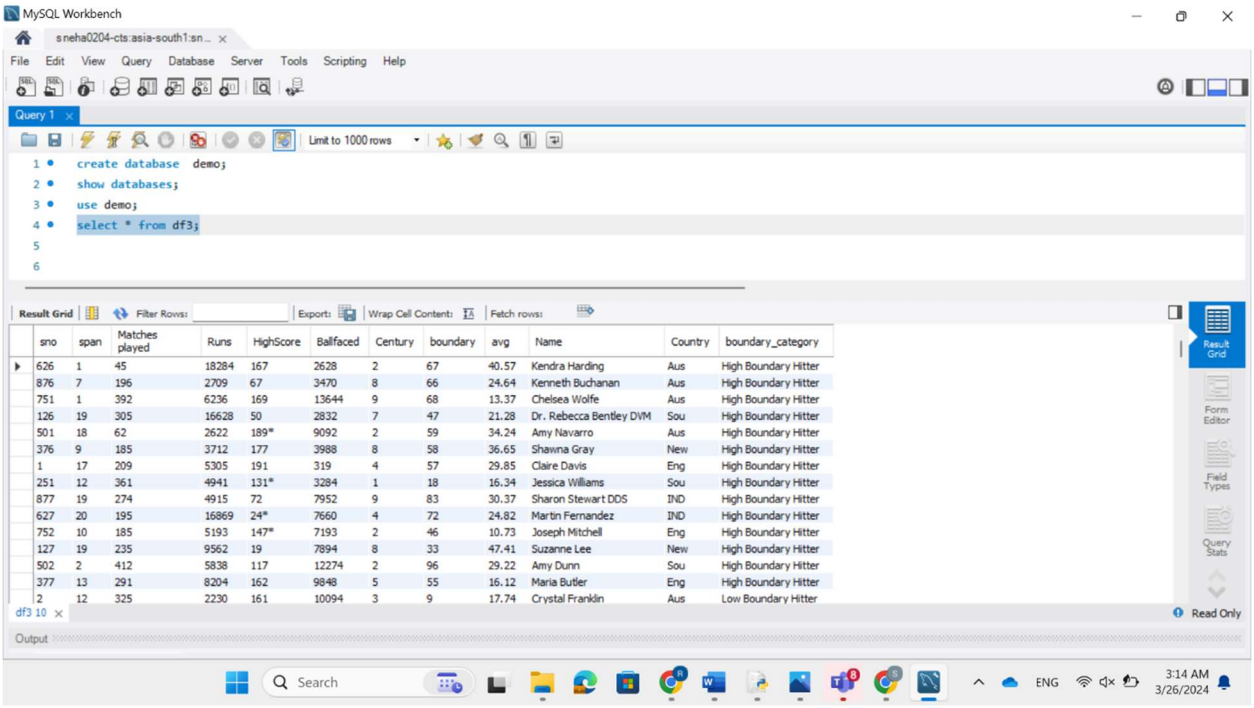


```
driver = "com.mysql.cj.jdbc.Driver"
url = "jdbc:mysql://35.244.33.241/"
table = "demo.demo123"
user = "root"
password = "sneha@123"

df.write.format("jdbc").option("driver", driver).option("url",url).option("dbtable", table).option("mode",
"append").option("user",user).option("password", password).save()

```

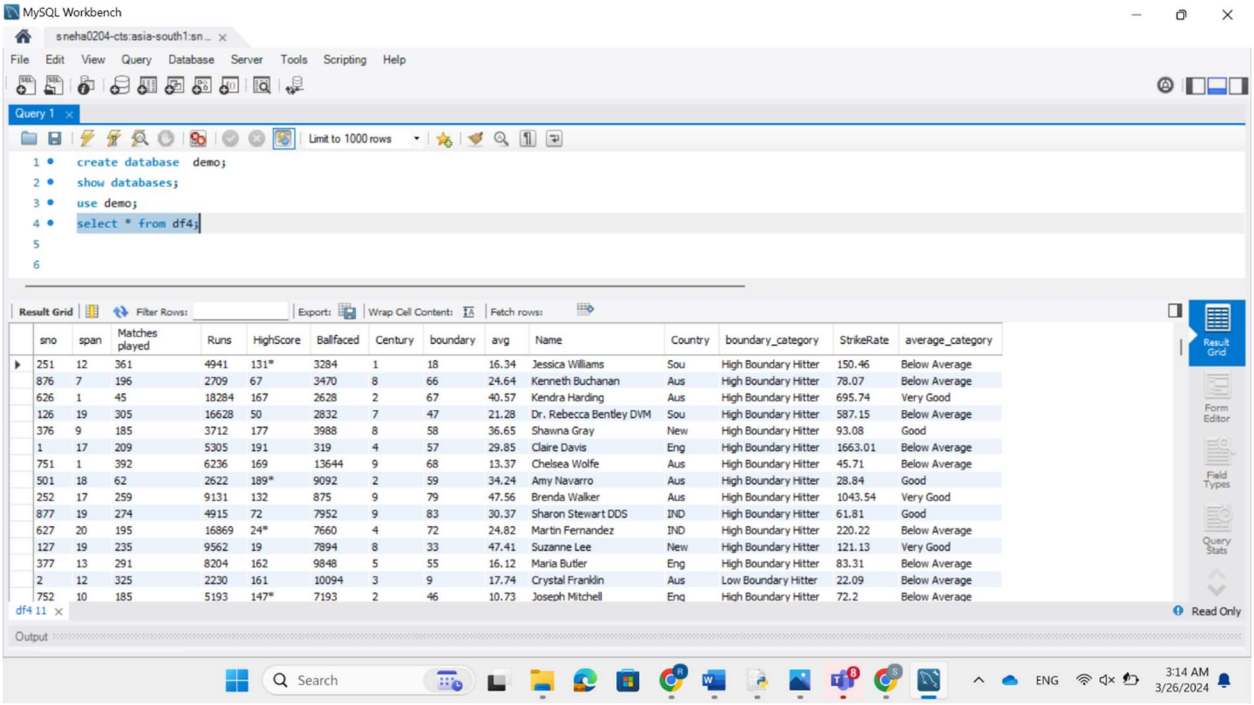
The code aims to save data from a DataFrame into a MySQL database table. It uses the JDBC driver to establish a connection between Python and the MySQL database.

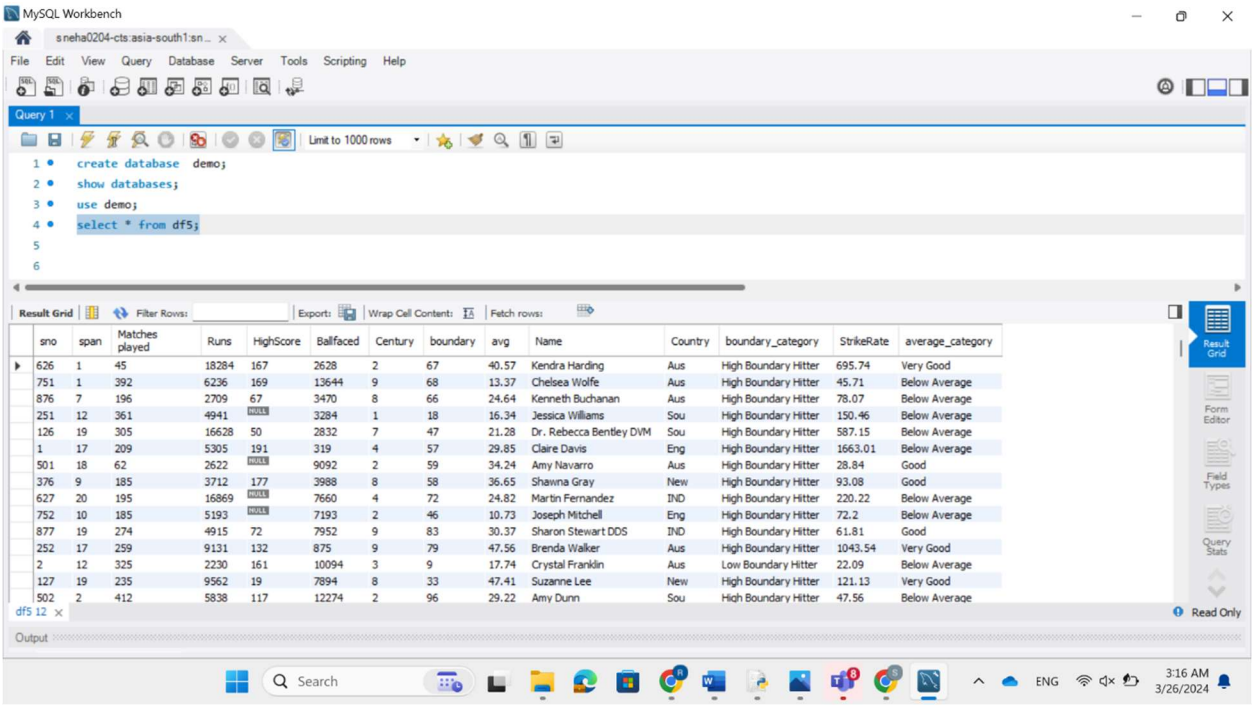
Once connection is done , open MySQL workbench and create a database demo, use show database query to check the available databases. Use the database demo that we created and select all the columns from table demo123.

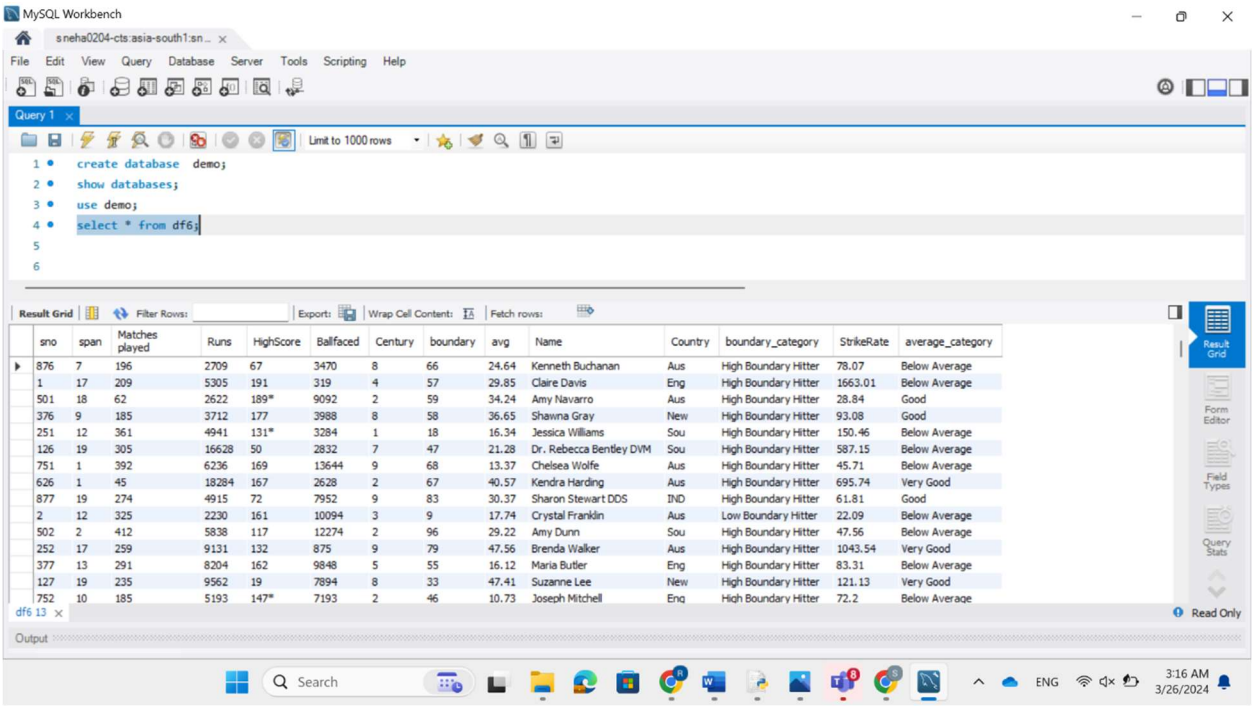


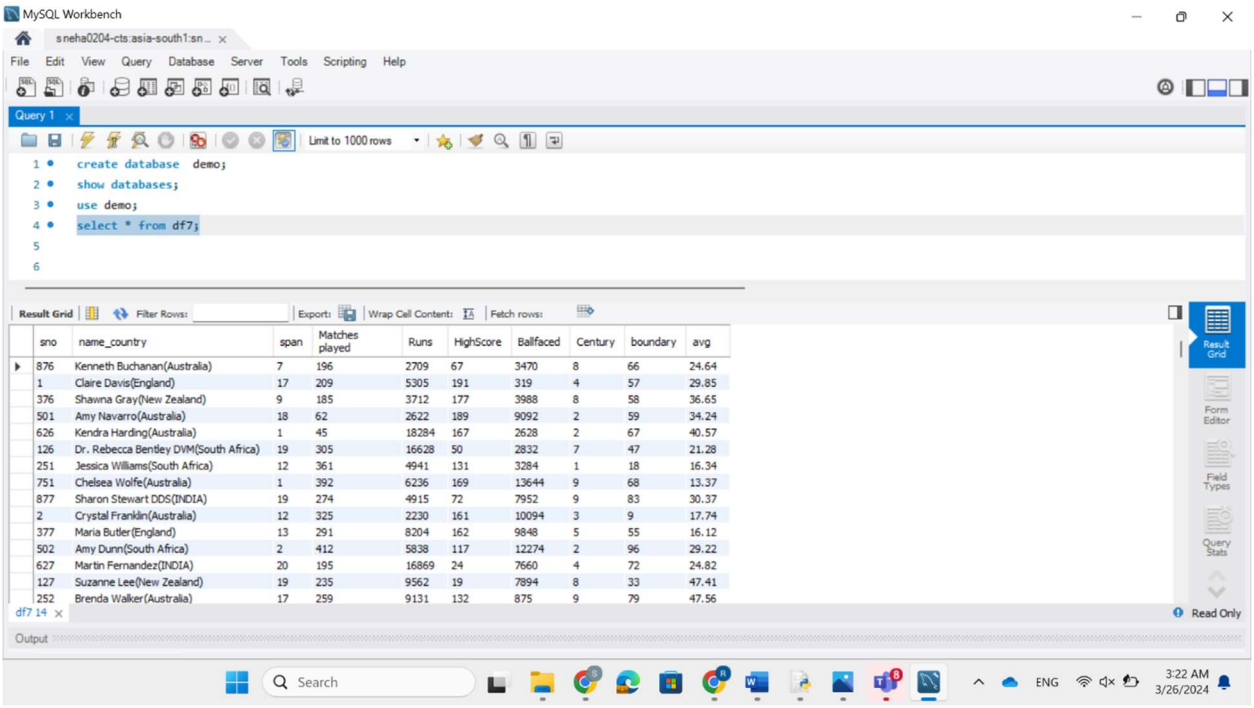


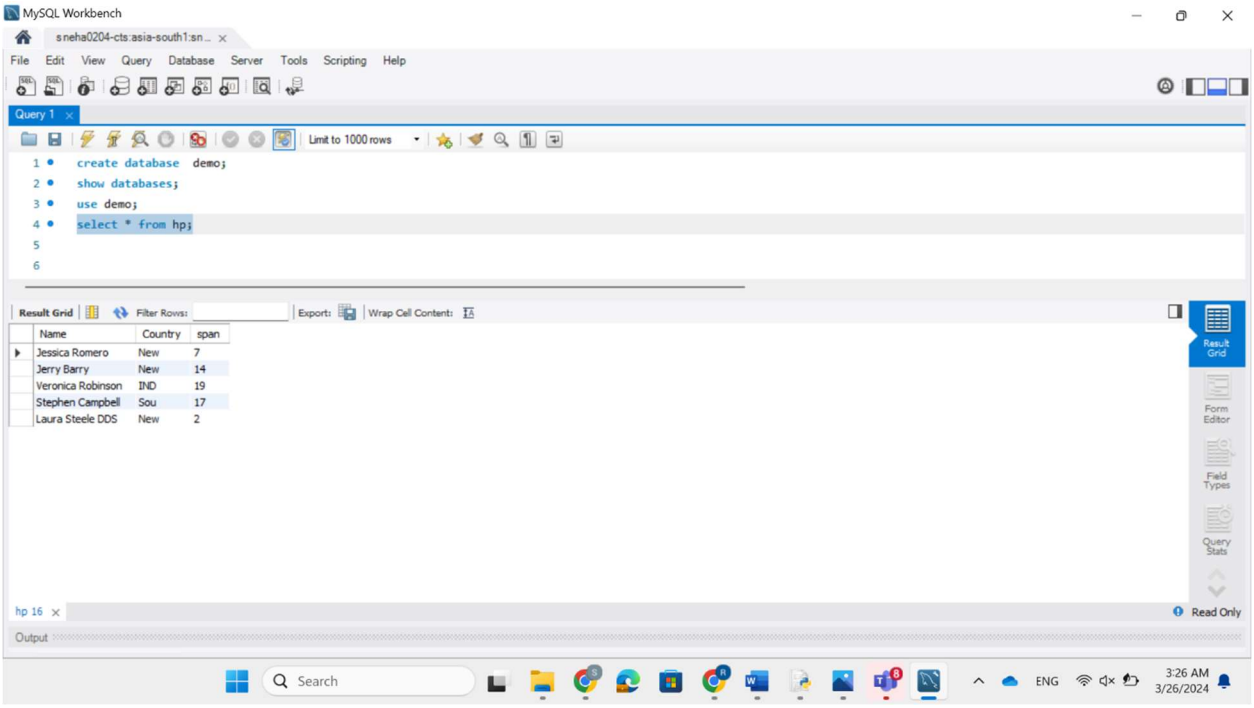
Continue the same process for other tables demo, df4, df5,hpdf6.df7






MySQL Workbench
sneha0204-cts:asia-south1:sn...
File Edit View Query Database Server Tools Scripting Help
Query 1
Limit to 1000 rows
1 create database demo;
2 show databases;
3 use demo;
4 select * from df6;
Result Grid Filter Rows: Export: Wrap Cell Content: Fetch rows:
sno | span | Matches played | Runs | HighScore | Ballfaced | Century | boundary | avg | Name | Country | boundary_category | StrikeRate | average_category
876 | 7 | 196 | 2709 | 67 | 3470 | 8 | 66 | 24.64 | Kenneth Buchanan | Aus | High Boundary Hitter | 78.07 | Below Average
1 | 17 | 209 | 5305 | 191 | 319 | 4 | 57 | 29.85 | Claire Davis | Eng | High Boundary Hitter | 1663.01 | Below Average
501 | 18 | 62 | 2622 | 189* | 9092 | 2 | 59 | 34.24 | Amy Navarro | Aus | High Boundary Hitter | 28.84 | Good
376 | 9 | 185 | 3712 | 177 | 3988 | 8 | 58 | 36.65 | Shawna Gray | New | High Boundary Hitter | 93.08 | Good
251 | 12 | 361 | 4941 | 131* | 3284 | 1 | 18 | 16.34 | Jessica Williams | Sou | High Boundary Hitter | 150.46 | Below Average
126 | 19 | 305 | 16628 | 50 | 2832 | 7 | 47 | 21.28 | Dr. Rebecca Bentley DVM | Sou | High Boundary Hitter | 587.15 | Below Average
751 | 1 | 392 | 6236 | 169 | 13644 | 9 | 68 | 13.37 | Chelsea Wolfe | Aus | High Boundary Hitter | 45.71 | Below Average
626 | 1 | 45 | 18284 | 167 | 2628 | 2 | 67 | 40.57 | Kendra Harding | Aus | High Boundary Hitter | 695.74 | Very Good
877 | 19 | 274 | 4915 | 72 | 7952 | 9 | 83 | 30.37 | Sharon Stewart DDS | IND | High Boundary Hitter | 61.81 | Good
2 | 12 | 325 | 2230 | 161 | 10094 | 3 | 9 | 17.74 | Crystal Franklin | Aus | Low Boundary Hitter | 22.09 | Below Average
502 | 2 | 412 | 5838 | 117 | 12274 | 2 | 96 | 29.22 | Amy Dunn | Sou | High Boundary Hitter | 47.56 | Below Average
252 | 17 | 259 | 9131 | 132 | 875 | 9 | 79 | 47.56 | Brenda Walker | Aus | High Boundary Hitter | 1043.54 | Very Good
377 | 13 | 291 | 8204 | 162 | 9848 | 5 | 55 | 16.12 | Maria Butler | Eng | High Boundary Hitter | 83.31 | Below Average
127 | 19 | 235 | 9562 | 19 | 7894 | 8 | 33 | 47.41 | Suzanne Lee | New | High Boundary Hitter | 121.13 | Very Good
752 | 10 | 185 | 5193 | 147* | 7193 | 2 | 46 | 10.73 | Joseph Mitchell | Eng | High Boundary Hitter | 72.2 | Below Average
df6 13
Read Only
Output
Search
ENG
3:16 AM
3/26/2024


MySQL Workbench
sneha0204-cts:asia-south1:sn...
File Edit View Query Database Server Tools Scripting Help
Query 1
Limit to 1000 rows
1 create database demo;
2 show databases;
3 use demo;
4 select * from df7;
Result Grid Filter Rows: Export: Wrap Cell Content: Fetch rows:
sno | name_country | span | Matches played | Runs | HighScore | Ballfaced | Century | boundary | avg
876 | Kenneth Buchanan(Australia) | 7 | 196 | 2709 | 67 | 3470 | 8 | 66 | 24.64
1 | Claire Davis(England) | 17 | 209 | 5305 | 191 | 319 | 4 | 57 | 29.85
376 | Shawna Gray(New Zealand) | 9 | 185 | 3712 | 177 | 3988 | 8 | 58 | 36.65
501 | Amy Navarro(Australia) | 18 | 62 | 2622 | 189 | 9092 | 2 | 59 | 34.24
626 | Kendra Harding(Australia) | 1 | 45 | 18284 | 167 | 2628 | 2 | 67 | 40.57
126 | Dr. Rebecca Bentley DVM(South Africa) | 19 | 305 | 16628 | 50 | 2832 | 7 | 47 | 21.28
251 | Jessica Williams(South Africa) | 12 | 361 | 4941 | 131 | 3284 | 1 | 18 | 16.34
751 | Chelsea Wolfe(Australia) | 1 | 392 | 6236 | 169 | 13644 | 9 | 68 | 13.37
877 | Sharon Stewart DDS(INDIA) | 19 | 274 | 4915 | 72 | 7952 | 9 | 83 | 30.37
2 | Crystal Franklin(Australia) | 12 | 325 | 2230 | 161 | 10094 | 3 | 9 | 17.74
377 | Maria Butler(England) | 13 | 291 | 8204 | 162 | 9848 | 5 | 55 | 16.12
502 | Amy Dunn(South Africa) | 2 | 412 | 5838 | 117 | 12274 | 2 | 96 | 29.22
627 | Martin Fernandez(INDIA) | 20 | 195 | 16869 | 24 | 7660 | 4 | 72 | 24.82
127 | Suzanne Lee(New Zealand) | 19 | 235 | 9562 | 19 | 7894 | 8 | 33 | 47.41
252 | Brenda Walker(Australia) | 17 | 259 | 9131 | 132 | 875 | 9 | 79 | 47.56
df7 14
Read Only
Output
Search
ENG
3:22 AM
3/26/2024



While doing this process make sure to change the table name and name of dataframe in the jdbc connectivity code for each table that are loaded.

CONCLUSION:

In conclusion, the Extract-Transform-Load (ETL) operations performed on the data aim to transform it into a more structured format or load it into a database. This process involves several key steps:

Extraction: Data is extracted from its source, which could be files, databases, or external systems.

Transformation: The extracted data undergoes transformation operations to clean, validate, and restructure it according to predefined rules and requirements. This may include data type conversions, aggregation, filtering, and enrichment.

Loading: The transformed data is loaded into a target destination, such as a database, data warehouse, or analytical platform, where it can be further analyzed, queried, or visualized.