

INTERIM PROJECT

Topic: Streaming ETL: Perform Extract-Transform-Load (ETL) operations on streaming data, transforming it into a more structured format or loading it into a database.



Submitted by

Sanjeev M

EMP ID: 2320436

Abstract:

Streaming Extract-Transform-Load (ETL) operations have become essential in modern data processing pipelines, especially with the rise of real-time data analytics and decision-making. This paper explores the concept of Streaming ETL, which involves extracting data from continuous streams, transforming it on-the-fly, and loading it into a structured format or database for further analysis. The primary objective of Streaming ETL is to handle high-volume, high-velocity data streams efficiently, ensuring timely and accurate data processing.

The abstract further delves into the key components and challenges of Streaming ETL. It discusses the importance of stream processing frameworks like Apache Kafka, Apache Flink, and Apache Storm for handling streaming data ingestion and processing. Additionally, it highlights the significance of scalable data transformation techniques, such as stream-based transformations using Apache Beam or Spark Streaming, to handle complex data manipulations in real-time.

Furthermore, the abstract touches upon the role of destination systems such as relational databases, data warehouses, or data lakes in storing and organizing the transformed data for downstream applications. It emphasizes the need for robust error handling, data quality checks, and monitoring mechanisms in Streaming ETL workflows to ensure data integrity and reliability.

Introduction:

In today's data-driven world, the ability to process data in both real-time and batch modes is crucial for organizations aiming to extract valuable insights and make informed decisions. Streaming Extract-Transform-Load (ETL) and batch ETL are two essential methodologies that enable organizations to process data continuously as it flows in real-time or in larger, periodic batches. These methodologies involve extracting raw data, transforming it to a more structured format, and loading it into a database or data warehouse for further analysis.

Table of Contents

CHAPTER	CONTENT
1	Introduction
2	Data Ingestion
3	Batch processing Using ETL
4	Stream Processing Using ETL

Overview:

Streaming ETL (Real-Time):

Real-time data ingestion using Apache Kafka from sensors and IoT devices in stores capturing sales data.

Apache Spark Streaming is used for real-time data processing and transformations on the streaming data.

Transformations include enrichment, aggregation, fraud detection, and inventory management as described earlier.

Enriched and transformed data is loaded into a data warehouse or data lake in near real-time for immediate analysis and reporting.

Batch ETL (Near Real-Time):

In addition to real-time processing, the retail company also performs batch ETL processes at regular intervals (e.g., daily, hourly) to handle historical data and larger volumes.

Batch ETL jobs are scheduled using Apache Airflow or similar workflow management tools to process data in batches.

The batch ETL pipeline includes similar transformations as the real-time pipeline but operates on larger datasets and historical data.

Processed batch data is also loaded into the data warehouse or data lake, merging seamlessly with the real-time data for comprehensive analytics.

Real-Time Streaming ETL

Real-time customer segmentation based on current purchases and behaviours.

Instantaneous fraud detection and prevention for immediate action.

Dynamic pricing adjustments based on real-time market demand signals.

Live inventory management and restocking notifications for store managers.

Batch ETL:

Historical trend analysis to identify seasonal patterns and long-term customer trends.

Large-scale data aggregation for quarterly and annual business reviews.

Regulatory compliance reporting requiring historical data analysis.

Data archiving and backup for long-term data retention and disaster recovery.

Streaming ETL involves processing data in near real-time as it is generated, enabling organizations to derive insights and respond to events promptly. On the other hand, batch ETL processes data in larger volumes and at scheduled intervals, allowing for deeper analysis of historical data and trend identification. Both methodologies play complementary roles in modern data architectures, addressing different use cases and requirements.

Key Components:

1. Data Ingestion: Both streaming and batch ETL begin with data ingestion, where raw data is collected from various sources such as sensors, applications, databases, and external APIs. Streaming ETL uses technologies like Apache Kafka, Amazon Kinesis, or Azure Event Hubs to ingest and buffer real-time data streams. Batch ETL typically involves bulk data extraction from databases, files, or data lakes at predefined intervals.
2. Transformation: After data ingestion, the raw data undergoes transformations to clean, enrich, aggregate, and structure it for analysis. Streaming ETL employs stream processing frameworks like Apache Flink, Apache Spark Streaming, or Apache Storm to perform transformations on-the-fly as data arrives. Batch ETL uses batch processing frameworks like Apache Spark, Apache Hadoop, or Apache Beam to process data in larger chunks and perform complex transformations.
3. Loading: The transformed data is then loaded into a destination system such as a relational database, data warehouse, or data lake for storage and analysis. Both streaming and batch ETL pipelines integrate seamlessly with various storage and

processing systems, ensuring that the transformed data is available for downstream analytics, reporting, and application integration.

CHAPTER 1- Data Ingestion

Data ingestion is the process of importing data from various sources into the system for analysis. In this project, the Unstructured dataset is ingested into the system using Apache Spark, specifically PySpark, which is a powerful data processing framework. PySpark provides functionality to read data from various sources, including CSV files, databases, and distributed storage systems.

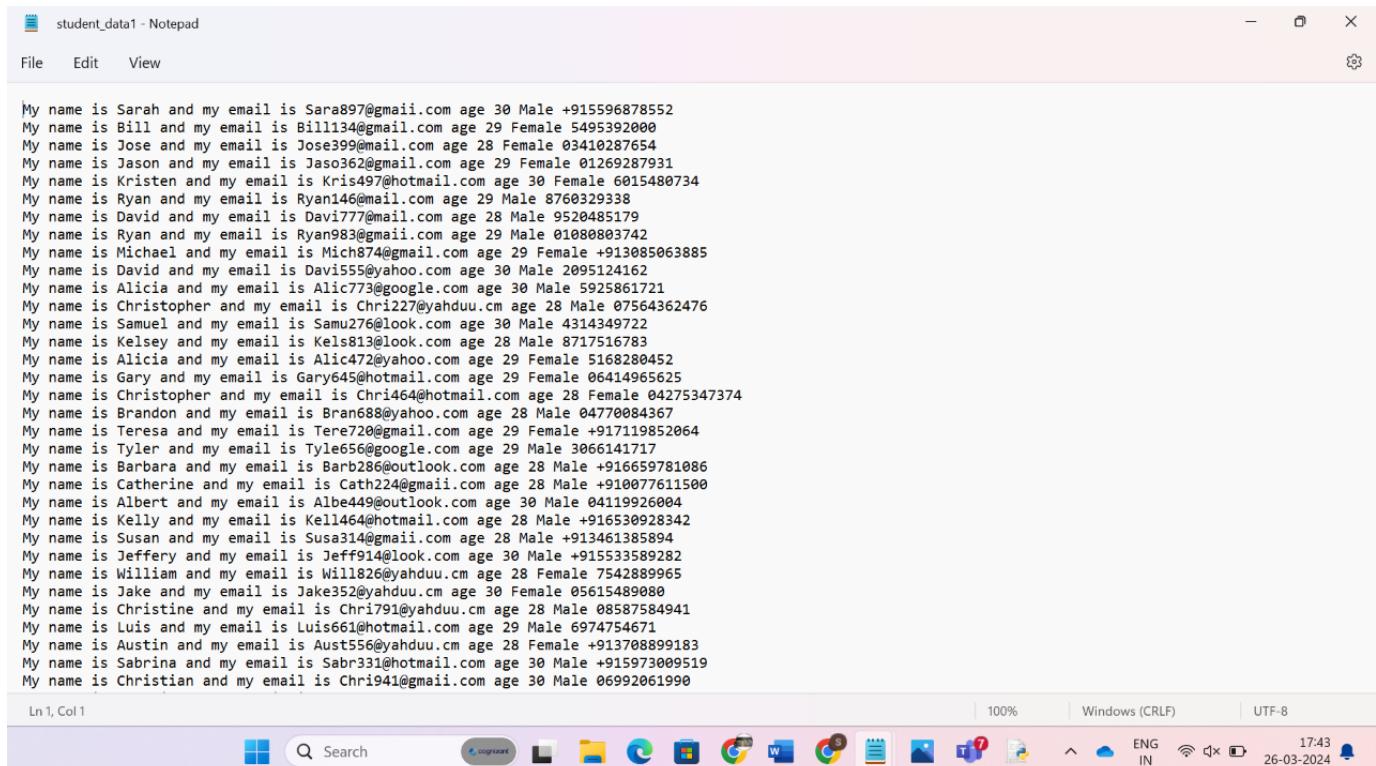
For this project, using Python, I generated the student dataset is generated and stored in a CSV file, and PySpark's DataFrame API is used to read the data from this file into a DataFrame for further analysis.

```
*random.py - C:\Users\2320436\OneDrive - Cognizant\Desktop\random.py (3.11.2)*
File Edit Format Run Options Window Help
import csv
from faker import Faker
import random
import string
# Create Faker instance
fake = Faker()
f=Faker('en_IN')
# Open CSV file in write mode
with open('data.txt', mode='w', newline='') as file:
    # Create CSV writer object
    writer = csv.writer(file)

    for _ in range(3000):
        # Generate random data
        name = fake.first_name()
        age = random.choice([20, 29, 30])
        gender = random.choice(["Male", "Female"])
        em=random.choice(["@gmail.com","@yahoo.com","@hotmail.com","@outlook.com","@gmaiil.com","@yahduu.cm","@mail.com","@look.com","@google.com"])
        n=name[0:4]
        rand=str(random.randint(100,999))
        na=f'{n}{rand}'
        email=f'{na}{em}'
        roll = ''.join(random.choices(string.ascii_letters + string.digits, k=8))
        phone = f.phone_number()

        |
        writer.writerow([f"My name is {name} and my email is {email} age {age} {gender} {phone}"])
print("sucess")
Ln: 26 Col: 8
```

Here input file is in unstructured format. Our goal is unstructured to structured format using spark. In both batch processing and stream processing we are going to do.



The screenshot shows a Windows Notepad window titled "student_data1 - Notepad". The window contains a large amount of unstructured text, which appears to be a list of names, emails, ages, and genders. The text is in a single column and lacks any formal structure like headers or delimiters. The Notepad interface includes a menu bar with File, Edit, and View, and a toolbar with standard icons. The status bar at the bottom shows "Ln 1, Col 1", "100%", "Windows (CRLF)", "UTF-8", and the date/time "26-03-2024 17:43".

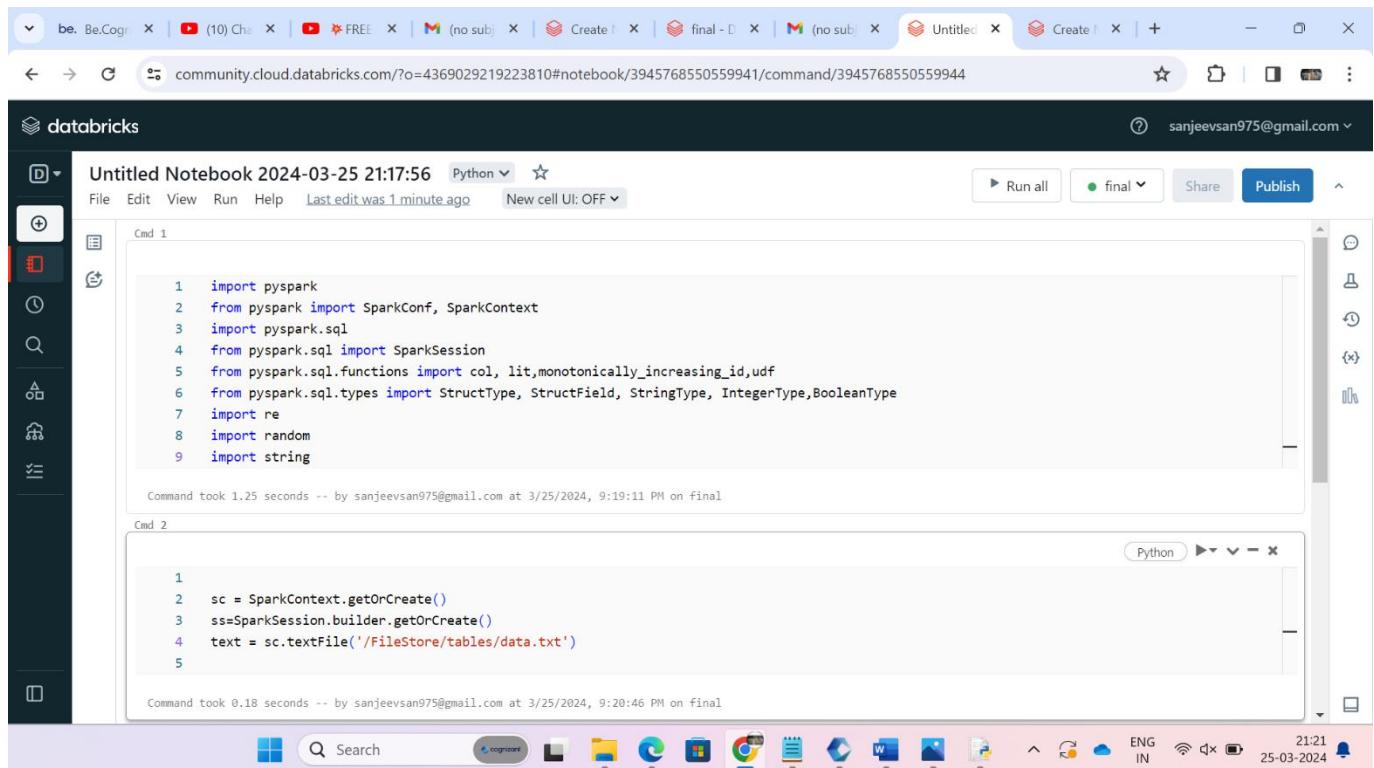
```
My name is Sarah and my email is Sara897@gmail.com age 30 Male +915596878552
My name is Bill and my email is Bill1134@gmail.com age 29 Female 5495392000
My name is Jose and my email is Jose399@gmail.com age 28 Female 03410287654
My name is Jason and my email is Jason362@gmail.com age 29 Female 01269287931
My name is Kristen and my email is Kris497@hotmail.com age 30 Female 6015480734
My name is Ryan and my email is Ryan146@gmail.com age 29 Male 8760329338
My name is David and my email is David777@mail.com age 28 Male 9520485179
My name is Ryan and my email is Ryan983@gmail.com age 29 Male 01080803742
My name is Michael and my email is Mich874@gmail.com age 29 Female +913085063885
My name is David and my email is David555@yahoo.com age 30 Male 2095124162
My name is Alicia and my email is Alicia773@google.com age 30 Male 5925861721
My name is Christopher and my email is Christopher@yahoo.com age 28 Male 07564362476
My name is Samuel and my email is Samu276@look.com age 30 Male 4314349722
My name is Kelsey and my email is Kels813@look.com age 28 Male 8717516783
My name is Alicia and my email is Alicia472@yahoo.com age 29 Female 5168280452
My name is Gary and my email is Gary645@hotmail.com age 29 Female 06414965625
My name is Christopher and my email is Christopher@hotmail.com age 28 Female 04275347374
My name is Brandon and my email is Brandon688@yahoo.com age 28 Male 04770084367
My name is Teresa and my email is Teresa720@gmail.com age 29 Female +917119852064
My name is Tyler and my email is Tyler656@google.com age 29 Male 3066141717
My name is Barbara and my email is Barbara286@outlook.com age 28 Male +916659781086
My name is Catherine and my email is Catherine224@gmail.com age 28 Male +910077611500
My name is Albert and my email is Albert449@outlook.com age 30 Male 04119926004
My name is Kelly and my email is Kelly1464@hotmail.com age 28 Male +916530928342
My name is Susan and my email is Susan314@gmail.com age 28 Male +913461385894
My name is Jeffery and my email is Jeffery914@look.com age 30 Male +915533589282
My name is William and my email is William826@yahoo.com age 28 Female 7542889965
My name is Jake and my email is Jake352@yahoo.com age 30 Female 05615489080
My name is Christine and my email is Christine791@yahoo.com age 28 Male 08587584941
My name is Luis and my email is Luis661@hotmail.com age 29 Male 6974754671
My name is Austin and my email is Austin556@yahoo.com age 28 Female +913708899183
My name is Sabrina and my email is Sabrina331@hotmail.com age 30 Male +915973009519
My name is Christian and my email is Christian941@gmail.com age 30 Male 06992061990
```

Chapter 2

BATCH STREAMING USING ETL:

Importing Libraries:

The code imports necessary libraries for working with Spark, including `pyspark`, `SparkConf`, `SparkContext`, `SparkSession`, and various functions and types from `pyspark.sql`.



```
1 import pyspark
2 from pyspark import SparkConf, SparkContext
3 import pyspark.sql
4 from pyspark.sql import SparkSession
5 from pyspark.sql.functions import col, lit, monotonically_increasing_id, udf
6 from pyspark.sql.types import StructType, StructField, StringType, IntegerType, BooleanType
7 import re
8 import random
9 import string

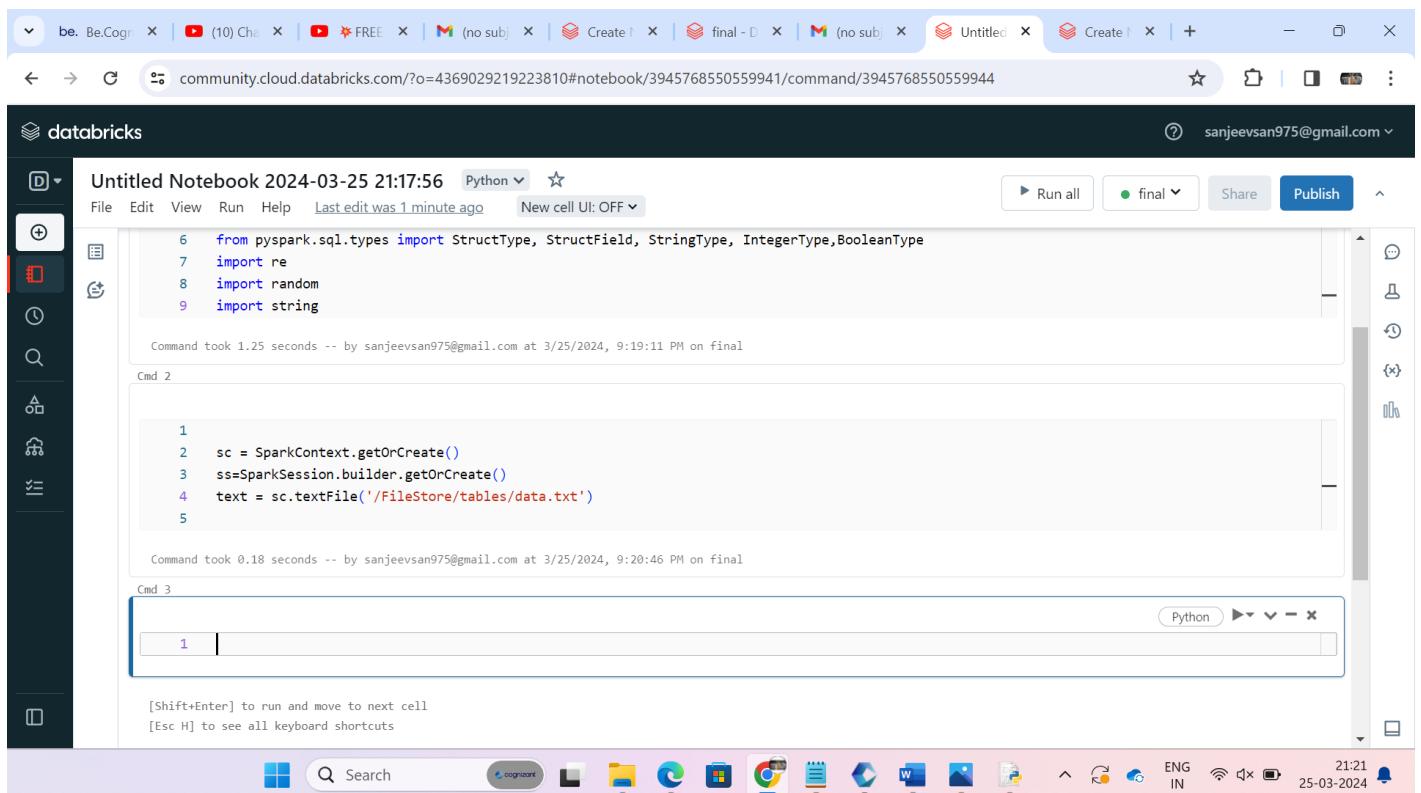
Command took 1.25 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:19:11 PM on final

Cmd 2
1
2 sc = SparkContext.getOrCreate()
3 sss=SparkSession.builder.getOrCreate()
4 text = sc.textFile('/FileStore/tables/data.txt')
5

Command took 0.18 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:20:46 PM on final
```

Spark Context and Session Creation:

It creates a Spark Context ('sc') and Spark Session ('ss') to interact with Spark functionalities.



The screenshot shows a Databricks notebook interface. The top navigation bar includes tabs for 'Be.Cogn' (active), 'Create', 'final - D', 'Untitled', and 'Create'. The URL in the address bar is 'community.cloud.databricks.com/?o=4369029219223810#notebook/3945768550559941/command/3945768550559944'. The left sidebar has icons for file operations like 'New', 'Edit', 'View', 'Run', 'Help', and 'Share'. The main area shows an 'Untitled Notebook 2024-03-25 21:17:56' in Python. Cell 1 contains imports for pyparq, re, random, and string. Cell 2 contains code to get or create SparkContext and SparkSession, and read a text file from '/FileStore/tables/data.txt'. Cell 3 is empty. The bottom status bar shows system icons and the date/time '21:21 IN 25-03-2024'.

```
6  from pyspark.sql.types import StructType, StructField, StringType, IntegerType, BooleanType
7  import re
8  import random
9  import string

Command took 1.25 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:19:11 PM on final

Cmd 2

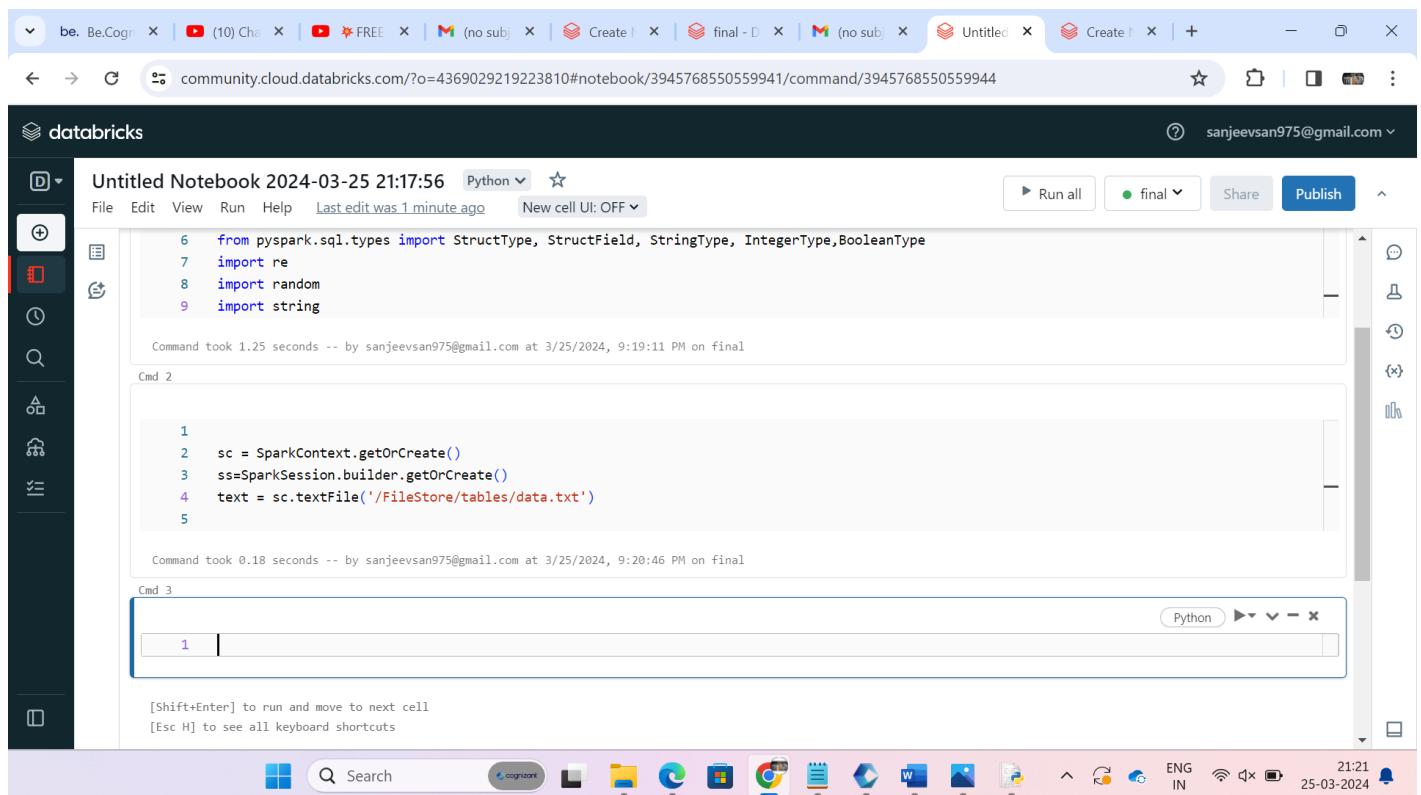
1
2  sc = SparkContext.getOrCreate()
3  ss=SparkSession.builder.getOrCreate()
4  text = sc.textFile('/FileStore/tables/data.txt')

Command took 0.18 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:20:46 PM on final

Cmd 3
```

Reading Text File:

The code reads the content of a text file named `dupli.txt` located in the `/FileStore/tables/` directory using Spark's `textFile` function. This file likely contains comma-separated values (CSV).

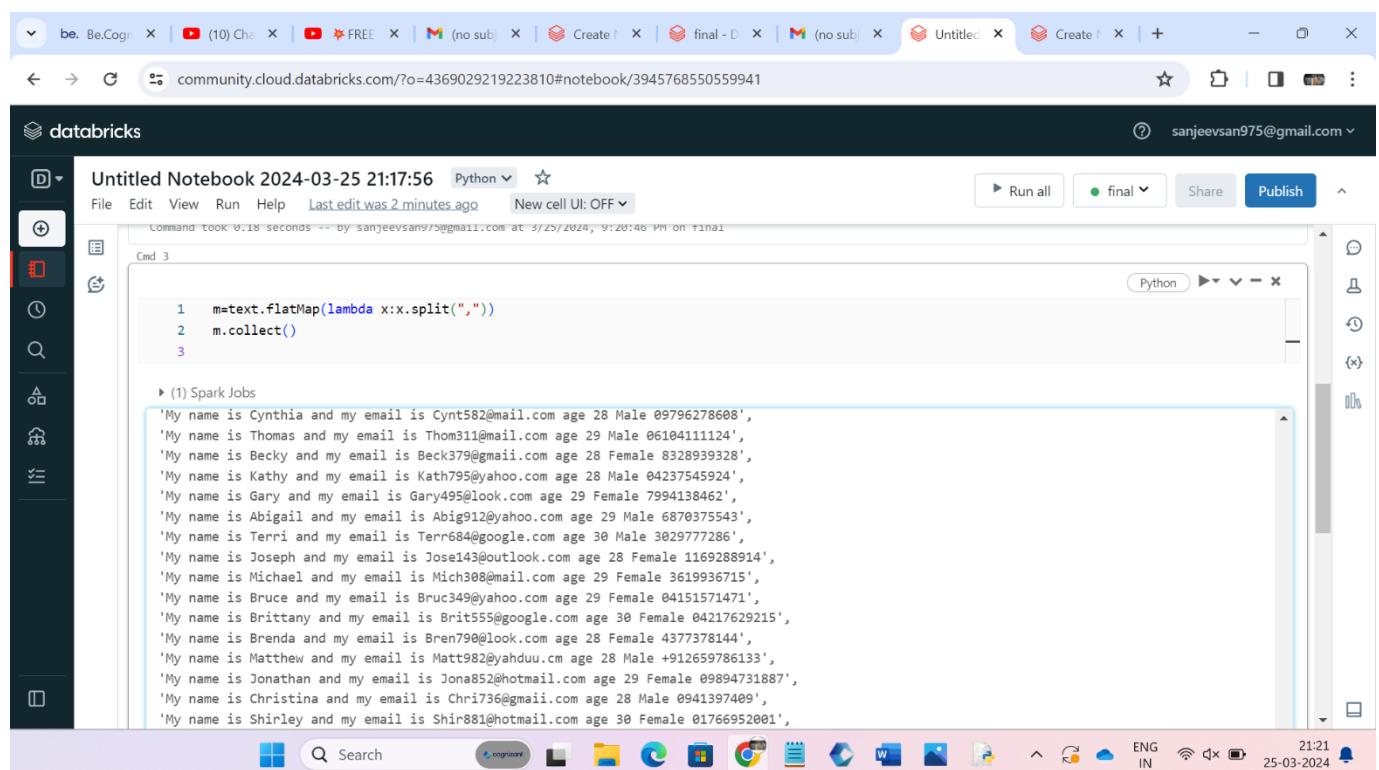


The screenshot shows a Databricks notebook interface with the following details:

- Header:** The URL is community.cloud.databricks.com/?o=436902919223810#notebook/3945768550559941/command/3945768550559944. The user is sanjeevsan975@gmail.com.
- Notebook Information:** Untitled Notebook 2024-03-25 21:17:56, Python, Last edit was 1 minute ago, New cell UI: OFF.
- Code Cells:**
 - Cell 1:** Imports pyparq.sql.types, re, random, and string.
 - Cell 2:** Creates a SparkContext (sc) and SparkSession (ss), then reads the file '/FileStore/tables/data.txt' into a DataFrame.
 - Cell 3:** A blank cell with the number 1 typed into it.
- Toolbar:** Includes Run all, Share, Publish, and other standard notebook controls.
- Bottom Bar:** Shows various icons for Databricks services like Cognizant, Google Sheets, Google Slides, Google Docs, Google Sheets, Google Slides, Google Docs, and Python, along with system status like ENG IN, 21:21, and 25-03-2024.

FlatMapping and Splitting:

The `flatMap` operation is applied to split each line of the text file by comma delimiter, resulting in a flattened RDD of individual data elements.



The screenshot shows a Databricks notebook interface. The notebook title is "Untitled Notebook 2024-03-25 21:17:56". The code cell contains the following Python code:

```
1 m=text.flatMap(lambda x:x.split(","))
2 m.collect()
3
```

The output section displays the results of the `collect` operation, showing a list of names, emails, ages, and genders:

```
'My name is Cynthia and my email is Cynt582@mail.com age 28 Male 09796278608',
'My name is Thomas and my email is Thom311@mail.com age 29 Male 06104111124',
'My name is Becky and my email is Beck379@gmail.com age 28 Female 8328939328',
'My name is Kathy and my email is Kath795@yahoo.com age 28 Male 04237545924',
'My name is Gary and my email is Gary495@look.com age 29 Female 7994138462',
'My name is Abigail and my email is Abig912@yahoo.com age 29 Male 6870375543',
'My name is Terri and my email is Terr684@google.com age 30 Male 3029777286',
'My name is Joseph and my email is Jose143@outlook.com age 28 Female 1169288914',
'My name is Michael and my email is Mich308@mail.com age 29 Female 3619936715',
'My name is Bruce and my email is Bruc349@yahoo.com age 29 Female 04151571471',
'My name is Brittany and my email is Brit555@google.com age 30 Female 04217629215',
'My name is Brenda and my email is Bren790@look.com age 28 Female 4377378144',
'My name is Matthew and my email is Matt982@yahduu.cm age 28 Male +912659786133',
'My name is Jonathan and my email is Jona852@hotmail.com age 29 Female 09894731887',
'My name is Christina and my email is Chri736@gmail.com age 28 Male 0941397409',
'My name is Shirley and my email is Shir881@hotmail.com age 30 Female 01766952001'
```

Data Extraction:

Lambda functions are used to extract specific fields like name, email, age, gender, and phone number from the flattened RDD.

The screenshot shows a Databricks notebook titled "Untitled Notebook 2024-03-25 21:17:56" running in Python. The notebook displays a list of names, emails, ages, genders, and phone numbers extracted from a flattened RDD. Below the notebook, two command cells show the Scala code used for extraction and the resulting RDDs.

```
'My name is Matthew and my email is Matt982@yahoo.com age 28 Male +912659786133',
'My name is Jonathan and my email is Jona852@hotmail.com age 29 Female 09894731887',
'My name is Christina and my email is Christ736@gmail.com age 28 Male 0941397409',
'My name is Shirley and my email is Shir881@hotmail.com age 30 Female 01766952001',
'My name is Brittany and my email is Brit307@gmail.com age 30 Male 04591332394',
'My name is Stephanie and my email is Step916@hotmail.com age 28 Female 6901640117',
'My name is Kathryn and my email is Kathi22@hotmail.com age 29 Female +918877141289',
'My name is Michael and my email is Mich160@look.com age 29 Male 1020532443',
...]
```

Command took 4.40 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:21:36 PM on final

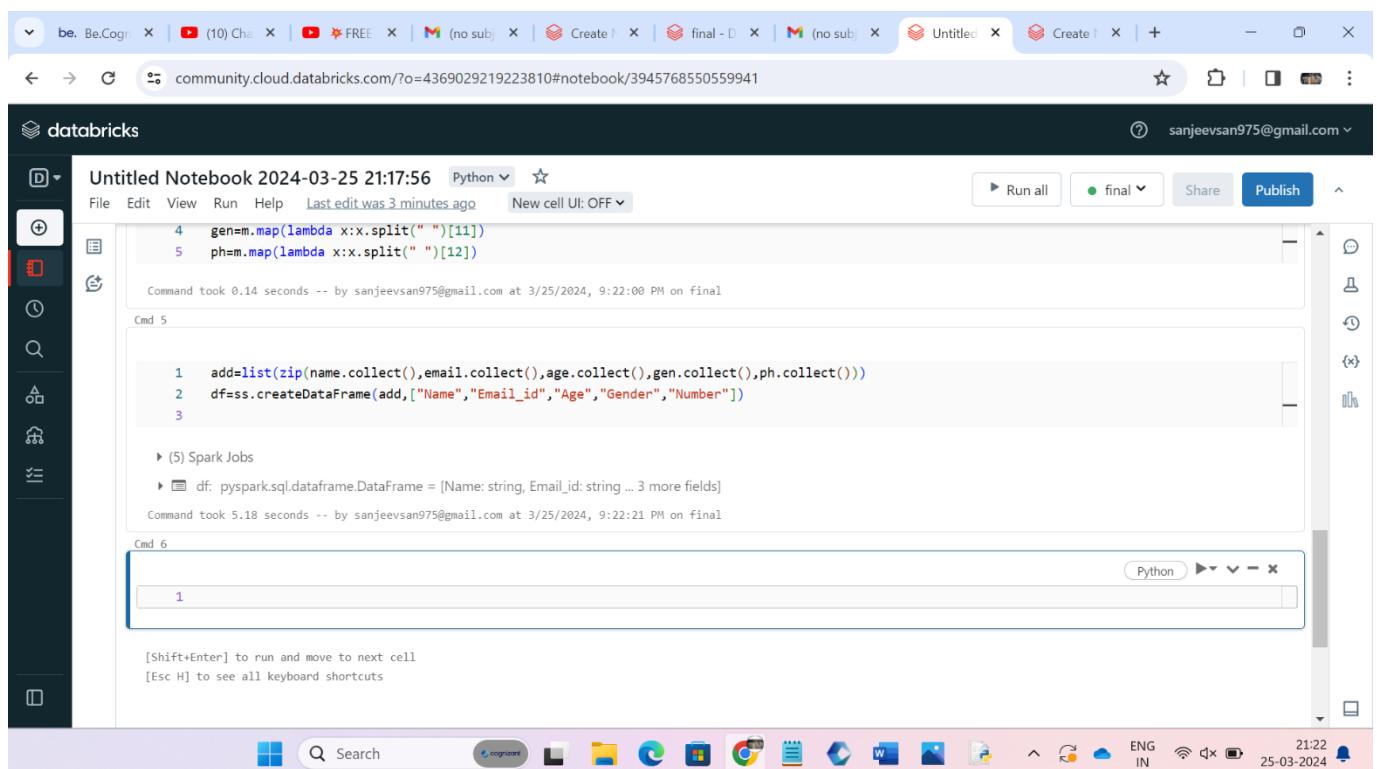
```
1 name=m.map(lambda x: x.split(" ")[3])
2 email=m.map(lambda x: x.split(" ")[8])
3 age=m.map(lambda x: x.split(" ")[10])
4 gen=m.map(lambda x:x.split(" ")[11])
5 ph=m.map(lambda x:x.split(" ")[12])
```

Command took 0.14 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:22:00 PM on final

```
1 add=list(zip(name.collect(),email.collect(),age.collect(),gen.collect(),ph.collect()))
```

Creating DataFrame:

The extracted data is used to create a DataFrame named 'df' with appropriate column names.



The screenshot shows a Databricks notebook titled "Untitled Notebook 2024-03-25 21:17:56" in Python. The notebook contains the following code:

```
4 gen=m.map(lambda x:x.split(" ")[11])
5 ph=m.map(lambda x:x.split(" ")[12])
```

Command took 0.14 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:22:00 PM on final

Cmd 5

```
1 add=list(zip(name.collect(),email.collect(),age.collect(),gen.collect(),ph.collect()))
2 df=ss.createDataFrame(add,[ "Name", "Email_id", "Age", "Gender", "Number"])
3
```

▶ (5) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]

Command took 5.18 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:22:21 PM on final

Cmd 6

```
1
```

[Shift+Enter] to run and move to next cell
[Esc H] to see all keyboard shortcuts

The notebook interface includes a sidebar with icons for file operations, a search bar, and a bottom navigation bar with various application icons.

Untitled Notebook 2024-03-25 21:17:56 Python

```
1 df.show()
```

(1) Spark Jobs

Alex Alex729@yahoo.com 29 Female 0396219178
Maria Mari485@look.com 30 Male 5712320616
Amanda Aman227@look.com 29 Male 06183463156
Cody Cody86@mail.com 30 Male 7035867437
Savannah Savat708@yahduu.cm 29 Male 02378531294
Catherine Cath658@outlook.com 29 Male 8932883922
Robert Robe957@look.com 30 Female 0696925565
James Jame140@yahduu.cm 29 Male +914720157449
Patricia Patr582@yahduu.cm 29 Female +916413416861
Chad Chad343@yahoo.com 29 Female +918141417306
Zachary Zach738@google.com 30 Female 01666478551
John John518@look.com 28 Female +912921407081
Brian Briai865@outlook.com 30 Female 01952326389
Vincent Vinc997@outlook.com 30 Male 06442086564
Thomas Thom987@outlook.com 29 Male 06896219146
Cameron Came859@outlook.com 30 Male 0818649286
Christina Chri534@gmail.com 29 Male 07205430311
Connie Conn843@outlook.com 28 Female 05165399019

21:24 25-03-2024

Untitled Notebook 2024-03-25 21:17:56 Python

```
1 df.show()
```

(1) Spark Jobs

Christina Chri534@gmail.com 29 Male 07205430311
Connie Conn843@outlook.com 28 Female 05165399019

only showing top 20 rows

Command took 1.48 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:24:10 PM on final

Cmd 7

```
1 df.count()
```

(2) Spark Jobs

Out[8]: 3450

Command took 2.65 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:24:34 PM on final

Cmd 8

```
1 dff=df.distinct()
2 dff.show()
```

(2) Spark Jobs

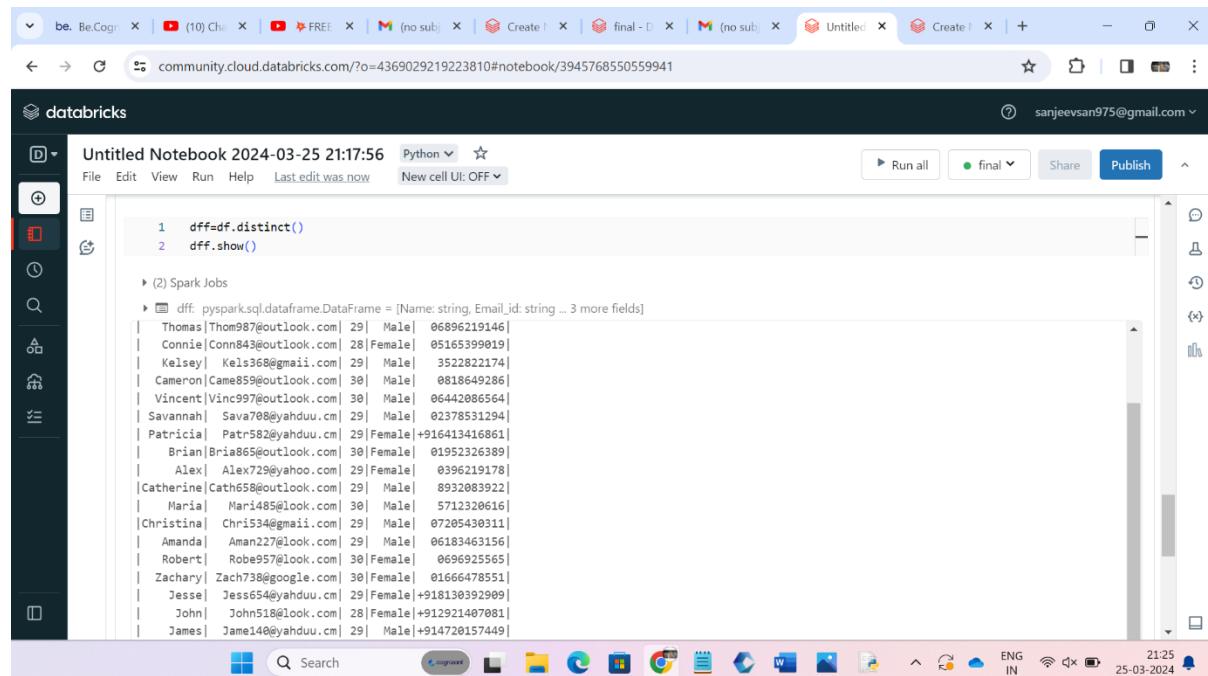
dff: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]

Thomas Thom987@outlook.com 29 Male 06896219146
--

21:25 25-03-2024

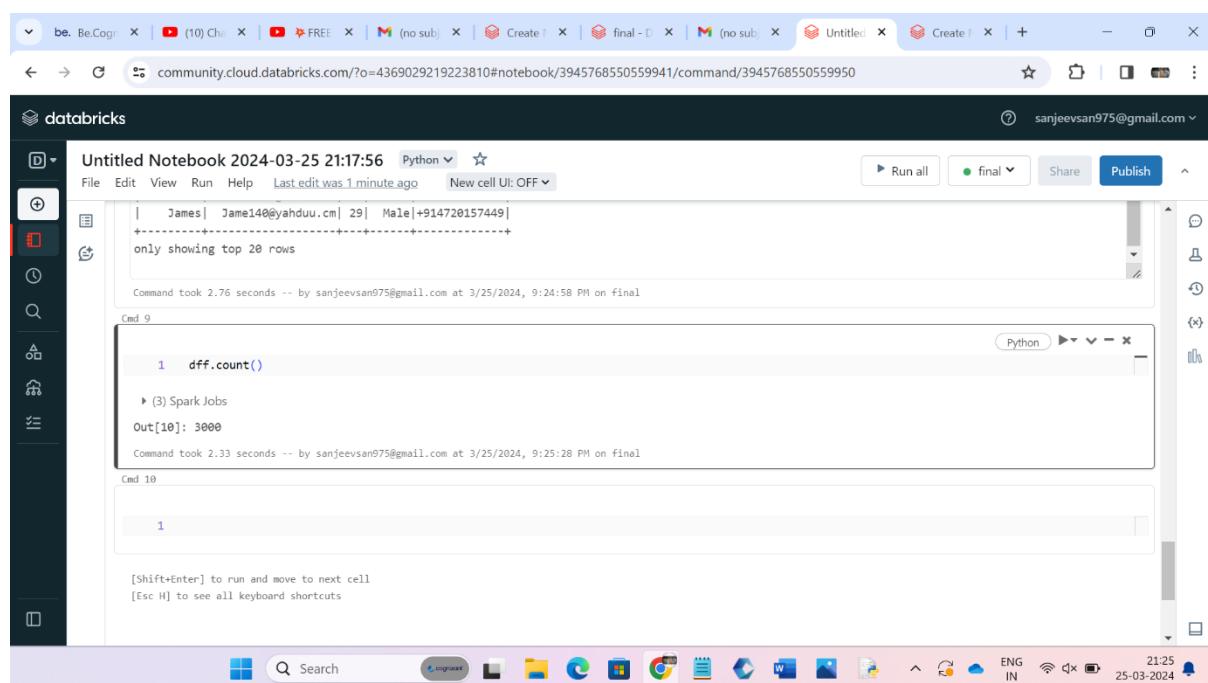
Distinct Records:

Duplicate records are removed from the DataFrame using the `distinct()` function to ensure each record is unique.



The screenshot shows a Databricks notebook titled "Untitled Notebook 2024-03-25 21:17:56" in Python. The code cell contains two lines of code: `1 dff=df.distinct()` and `2 dff.show()`. The output of the second line shows a list of 30 rows from a DataFrame named `dff`, which is a pyspark.sql.dataframe.DataFrame. The columns include Name, Email_id, and several other fields. The output is as follows:

```
| Thomas|Thom98@outlook.com| 29| Male| 06896219146|
| Connie|Conn84@gmail.com| 28|Female| 05165399819|
| Kelsey|Kels36@gmail.com| 29| Male| 3522822174|
| Cameron|Came85@outlook.com| 30| Male| 0818649286|
| Vincent|Vinc997@outlook.com| 30| Male| 06442086564|
| Savannah| Sava708@yahduu.cm| 29| Male| 02378531294|
| Patricia| Patr58@yahduu.cm| 29|Female| +916413416861|
| Brian|Bria865@outlook.com| 30|Female| +91952326389|
| Alex| Alex729@yahoo.com| 29|Female| 0396219178|
| Catherine|Cath658@outlook.com| 29| Male| 8932083922|
| Maria| Mari1485@look.com| 30| Male| 5712320616|
| Christina| Chri534@gmail.com| 29| Male| 07205430311|
| Amanda| Aman227@look.com| 29| Male| 06183463156|
| Robert| Robe957@look.com| 30|Female| 0696925565|
| Zachary| Zach738@google.com| 30|Female| 01666478551|
| Jesse| Jess654@yahduu.cm| 29|Female| +918130392909|
| John| John518@look.com| 28|Female| +912921407081|
| James| Jame140@yahduu.cm| 29| Male| +914728157449|
```



The screenshot shows a Databricks notebook titled "Untitled Notebook 2024-03-25 21:17:56" in Python. The code cell contains one line of code: `1 dff.count()`. The output of the cell is `Out[10]: 3000`. The command took 2.33 seconds. Below the code cell, there is a note: "[Shift+Enter] to run and move to next cell [Esc H] to see all keyboard shortcuts".

Email Validation:

User-defined function (UDF) named 'isValid' is defined to check the validity of email addresses based on a regular expression pattern. This UDF is applied to the DataFrame to create a new column 'validmail' indicating whether each email is valid or not.

The screenshot shows a Databricks notebook titled "Untitled Notebook 2024-03-25 21:17:56" in Python. The code defines a UDF 'isValid' and applies it to a DataFrame 'vm' to add a 'validmail' column. The resulting DataFrame 'vm' is then displayed.

```
1 def isValid(email):
2     p='^[\w+.-_%+-]+@[a-zA-Z\d]+\.(com|org|net)$'
3     return bool(re.match(p,email))
4 vmail=udf(isValid,BooleanType())
5 vm=dff.withColumn("validmail",vmail(dff["Email_ID"]))
6 vm.show()
```

(2) Spark Jobs

vm: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 4 more fields]

Name	Email_id	Gender	Phone	validmail	
Thomas	Thom987@outlook.com	29	Male	06896219146	false
Connie	Conn843@outlook.com	28	Female	05165399019	false
Kelsey	Kels368@gmai.com	29	Male	3522822174	false
Cameron	Came859@outlook.com	30	Male	0818649286	false
Vincent	Vinc997@outlook.com	30	Male	06442886564	false
Savannah	Sava708@ahduu.cm	29	Male	02378531294	false
Patricia	Patr582@ahduu.cm	29	Female	+916413416861	false
Brian	Bria865@outlook.com	30	Female	01952326389	false
Alex	Alex729@yahoo.com	29	Female	0396219178	true
Catherine	Cath658@outlook.com	29	Male	8932083922	false
Maria	Mari485@look.com	30	Male	5712320616	false
Christina	Chri534@gmai.com	29	Male	07205430311	false

Filtering Valid Email Addresses:

Rows with valid email addresses are filtered out from the DataFrame..

The screenshot shows a Databricks notebook interface with the following details:

- Header:** Be.Cognit... (tab), (10) Ch... (tab), FREE (tab), (no sub) (tab), Create (tab), final - D (tab), (no sub) (tab), Untitled (tab), Create (tab).
- User:** sanjeevsan975@gmail.com
- Notebook Title:** Untitled Notebook 2024-03-25 21:17:56
- Cell 10 Output:** A table showing two rows of data:

	John	Johns123@BOOK.COM	28	Female	+912921407051	raise
	James	Jame140@yahoo.cm	29	Male	+914720157449	false

only showing top 20 rows

Command took 2.82 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:25:51 PM on final
- Cell 11 Code:**

```
1 fs=vm.filter(vm["validmail"]== True)
2 res=fs.drop("validmail")
```

fs: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 4 more fields]
res: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]

Command took 0.14 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:26:12 PM on final
- Cell 12:** A new cell with the number 1.
- Bottom Bar:** Search bar, cogwheel icon, file icons (Word, Excel, PDF, etc.), network, battery, ENG IN, 25-03-2024, 21:26.

Untitled Notebook 2024-03-25 21:17:56 Python

```
1 res.show()
```

(2) Spark Jobs

Rachel Rach352@hotmail.com 29 Male +913380570460
Carl Car1654@hotmail.com 29 Female 5147296812
Nicole Nico274@gmail.com 29 Male 06802621288
Jose Jose529@yahoo.com 29 Female +915809481341
Ronald Rona601@yahoo.com 29 Female 2566667833
Alex Alex729@yahoo.com 29 Female 0396219178
Phyllis Phyl1474@hotmail.com 28 Male +914964370948
Cynthia Cynth937@hotmail.com 29 Female +912254429007
Teresa Tere724@yahoo.com 28 Female +917804572063
Xavier Xavi405@gmail.com 30 Female 05949052321
Steven Steve665@hotmail.com 29 Male 06150299231
Katherine Kath696@gmail.com 28 Male +9145348247788
Chad Chad343@yahoo.com 29 Female +918141417306
Susan Susa948@hotmail.com 30 Female 4212300968
Veronica Vero684@gmail.com 30 Male 05717150833
Jessica Jess572@gmail.com 30 Male +913644939979
Stephen Step764@hotmail.com 30 Female +919628815956
Denise Deni889@hotmail.com 29 Male +911486756075

21:26 25-03-2024

Untitled Notebook 2024-03-25 21:17:56 Python

```
only showing top 20 rows
```

Command took 2.38 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:26:24 PM on final

```
1 res.count()
```

(3) Spark Jobs

```
Out[14]: 1011
```

Command took 1.94 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:26:41 PM on final

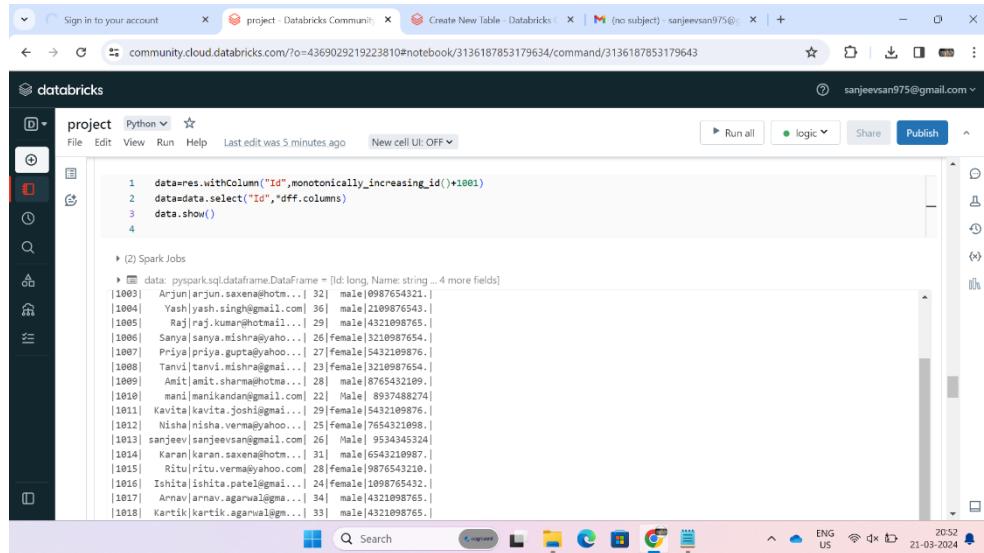
```
1
```

[Shift+Enter] to run and move to next cell
[Esc H] to see all keyboard shortcuts

21:33 25-03-2024

Adding Unique IDs:

Unique IDs are added to each row using the `monotonically_increasing_id()` function, ensuring each row has a distinct identifier.



The screenshot shows a Databricks notebook interface. The code cell contains the following Python code:

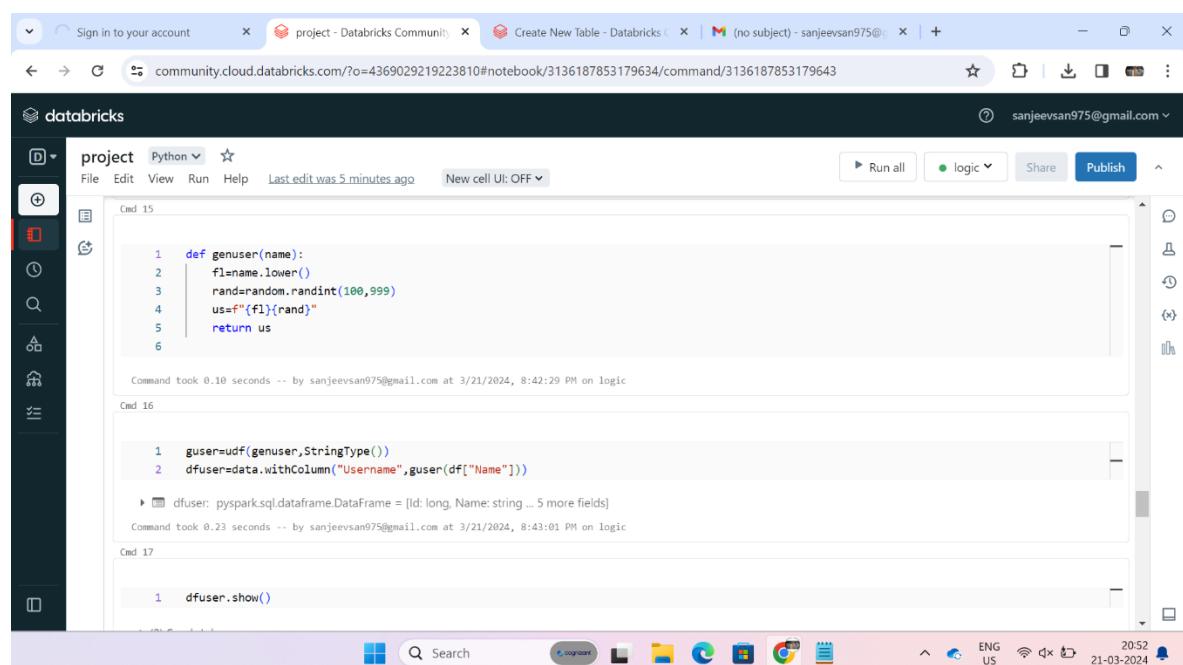
```
1 data=spark.sql("select * from users")
2 data=data.withColumn("Id",monotonically_increasing_id())
3 data.show()
4
```

When run, the cell displays a DataFrame with 18 rows, each containing an ID, name, gender, and age. The IDs are sequential integers starting from 1.

ID	Name	Gender	Age
1	Arjun arjun.saxena@hotmail.com	male	21
2	Yash yash.singh@gmail.com	male	20
3	Raj raj.kumar@hotmail.com	male	29
4	Sanya sanya.mishra@yahoo.com	female	25
5	Priya priya.gupta@yahoo.com	female	27
6	Tanvi tanvi.mishra@gmail.com	female	23
7	Amit amit.sharma@hotmail.com	male	28
8	Mani manikandan@gmail.com	Male	22
9	Kavita kavita.joshi@gmail.com	female	29
10	Nisha nisha.verma@yahoo.com	female	25
11	Sanjeev sanjeevsan@gmail.com	Male	26
12	Karan karan.saxena@hotmail.com	male	21
13	Ritu ritu.verma@yahoo.com	female	28
14	Ishita ishita.patel@gmail.com	female	24
15	Arnav arnav.agarwal@gmail.com	male	34
16	Kartik kartik.agarwal@gmail.com	male	33

Generating Usernames:

Another UDF named `genuser` is defined to generate usernames based on the lowercased first name and a random integer. This UDF is applied to create a new column `Username`.



The screenshot shows a Databricks notebook interface. The code cell contains the following Python code to define a UDF:

```
1 def genuser(name):
2     f1=name.lower()
3     rand=random.randint(100,999)
4     us=f1+str(rand)
5     return us
6
```

Command took 0.10 seconds -- by sanjeevsan975@gmail.com at 3/21/2024, 8:42:29 PM on logic

The next cell shows the application of the UDF:

```
1 guser=udf(genuser,StringType())
2 dfuser=data.withColumn("Username",guser(df["Name"]))
```

Command took 0.23 seconds -- by sanjeevsan975@gmail.com at 3/21/2024, 8:43:01 PM on logic

The final cell shows the resulting DataFrame:

```
1 dfuser.show()
```

```

1   dfuser.show()
(2) Spark Jobs
+---+-----+-----+-----+-----+
| Id|    Name| Email_id|Age|Gender|Number| Username|
+---+-----+-----+-----+-----+
|1001| Sonam| sonam.sharma@yahoo...| 30|female|1098765432.| sonam871|
|1002| Rohit| rohit.verma@hotmail...| 33| male|21098765432.| rohit162|
|1003| Arjun| arjun.saxena@hotmail...| 32| male|0987654321.| arjun512|
|1004| Yash| yash.singh@gmail.com| 36| male|2109876543.| yash471|
|1005| Raj| raj.kumar@hotmail...| 29| male|4321098765.| raj758|
|1006| Sanya| sanya.mishra@yahoo...| 26|female|3210987654.| sanya663|
|1007| Priya| priya.gupta@yahoo...| 27|female|5432109876.| priya666|
|1008| Tanvi| tanvi.mishra@gmail...| 23|female|3210987654.| tanvi529|
|1009| Amit| amit.sharma@hotmail...| 28| male|8765432109.| amit211|
|1010| mani| manikandan@gmail.com| 22| Male| 8937488274| mani555|
|1011| Kavita| kavita.joshi@gmail...| 29|female|5432109876.| kavita436|
|1012| Nisha| nisha.verma@yahoo...| 25|female|7654321098.| nisha287|
|1013| sanjeev| sanjeevsan@gmail.com| 26| Male| 9534345324| sanjeev423|
|1014| Karan| karan.saxena@hotmail...| 31| male|6543210987.| karan471|
|1015| Ritu| ritu.verma@yahoo.com| 28|female|9876543210.| ritu184|

```

Generating Passwords:

Similarly, a UDF named `genpass` is defined to generate random passwords consisting of letters and digits. This UDF is applied to create a new column `Password`

```

1   def genpass():
2       ln=8
3       passc=string.ascii_letters+string.digits
4       p="".join(random.choice(passc) for i in range(ln))
5       return p

```

```

1   genp=udf(genpass,StringType())
2   dfp=dfuser.withColumn("Password",genp())

```

```

1   dfp.show(54)
2

```

The screenshot shows a Databricks notebook interface. The top navigation bar includes tabs for 'Sign in to your account', 'project - Databricks Community', 'Create New Table - Databricks', and '(no subject) - sanjeevsan975@gmail.com'. The main area displays a code cell with the command `dfp.show(54)`. Below the code, the output shows 1052 rows of data from a Spark DataFrame. The columns include names, email addresses, gender, age, and other demographic information. The interface has a dark theme with various toolbars and status bars at the bottom.

```

1 dfp.show(54)
2

(2) Spark Jobs
|1036| Anjali|anjali.ali@gmail.com| 28|female|7654321098.| anjali603|FJj7UF4d|
|1037| Vikram|vikram.singh@gmail...| 35| male|2109876543.| vikram525|jtIaJ8DS|
|1038| Meera|meera.patel@gmail...| 25|female|1098765432.| meera335|NWJ7f1SA|
|1039| Shruti|shruti.sharma@ma...| 30|female|9876543210.| shruti766|eQ9582M1|
|1040| Aditya|aditya.kumar@gmail...| 31| male|0987654321.| aditya822|EgK6Piis|
|1041| Mohit|mohit.verma@gmail...| 33| male|8765432109.| mohit962|Fc8UUPes|
|1042| Sneha|sneha.mishra@gmail...| 24|female|3210987654.| sneha463|SFFFvC55|
|1043| Deepak|deepak.gupta@hotm...| 34| male|8765432109.| deepak272|2HrE6H4W|
|1044| Meera|meera.patel@gmail...| 44|female| 3210987654| meera395|YGMmDB7F|
|1045| Riya|riya.joshi@yahoo.com| 27|female|3210987654.| riya139|ZM7yG1DW|
|1046| Vineet|vineet.agarwal@ho...| 35| male|2109876543.| vineet474|kBxVMTMe|
|1047| Sameer|sameer.verma@hotmail...| 29| male|6543210987.| sameer375|H3ioasc|
|1048| Neha|neha.patel@yahoo.com| 30|female|9876543210.| neha897|QxZNkxj6|
|1049| Pooja|pooja.sharma@yahoo...| 31|female|7654321098.| pooja263|KBD797R|
|1050| Kritika|kritika.ali@yahoo...| 26|female|5432109876.| kritika983|HyqyQt1|
|1051| Anjali|anjali.mishra@yahoo...| 24|female|1098765432.| anjali410|XtpaENn9|
|1052| Mohan|mohan.saxena@hotmail...| 33| male|0987654321.| mohan243|OBUKw3LD|

```

Loading to MySQL Cloud:

Finally, the processed DataFrame (`dfp`) is written to a MySQL database table specified by the `table` parameter using JDBC connection parameters such as driver, URL, username, and password. The data is appended to the table in the database.

For that we are creating a google cloud SQL for loading a data.

MySQL in the cloud offers scalability, managed services, high availability, global accessibility, and cost efficiency. It easily scales to accommodate growing data needs, freeing up time with managed services for backups, maintenance, and updates. High availability features ensure uptime and data durability, while global accessibility fosters collaboration. Cost efficiency stems from a pay-as-you-go model.

Conversely, local MySQL provides performance advantages, data control, predictable costs, customization, and offline access. It offers faster query execution, control over infrastructure, predictable costs, customization options, and ensures uninterrupted access in environments with limited internet connectivity. Organizations often choose based on needs, budget, and regulatory requirements.

1)Create instance in Mysql cloud:

The screenshot shows the Google Cloud Platform interface for creating a MySQL instance. The URL in the browser is `console.cloud.google.com/sql/instances/create;engine=MySQL?project=sanjeev0204-cts`. The page is titled "Create a MySQL instance".

Instance info

- Instance ID *: sanjeev123 (Note: Use lowercase letters, numbers, and hyphens. Start with a letter.)
- Password *: [REDACTED] (Set a password for the root user. [Learn more](#))
- No password
- PASSWORD POLICY**
- Database version *: MySQL 8.0
- Choose a Cloud SQL edition**
- A Cloud SQL edition determines foundational characteristics of your instance and cannot

Pricing estimate

\$2.21 per hour (estimated, without discounts)
That's about \$52.94 per day.
Feature usage and traffic costs aren't included in estimate

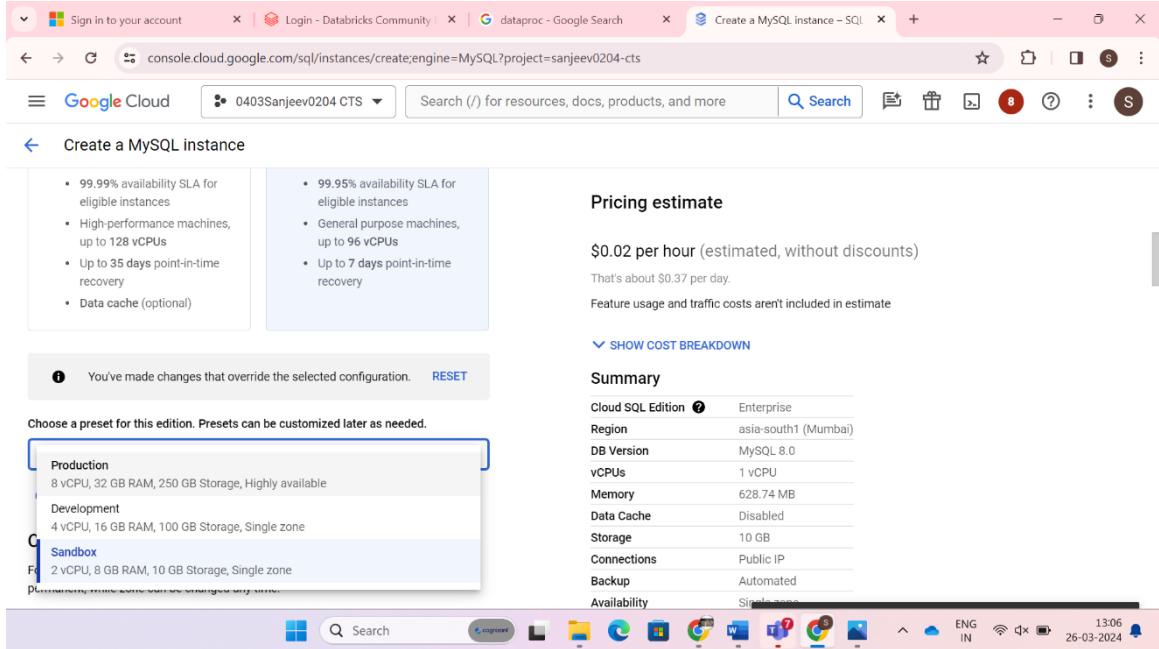
SHOW COST BREAKDOWN

Summary

Cloud SQL Edition	Enterprise Plus
Region	us-central1 (Iowa)
DB Version	MySQL 8.0
vCPUs	8 vCPU
Memory	64 GB
Data Cache	Enabled (375 GB)
Storage	250 GB
Connections	Public IP
Backup	Automated
Availability	Multiple zones (Highly available)

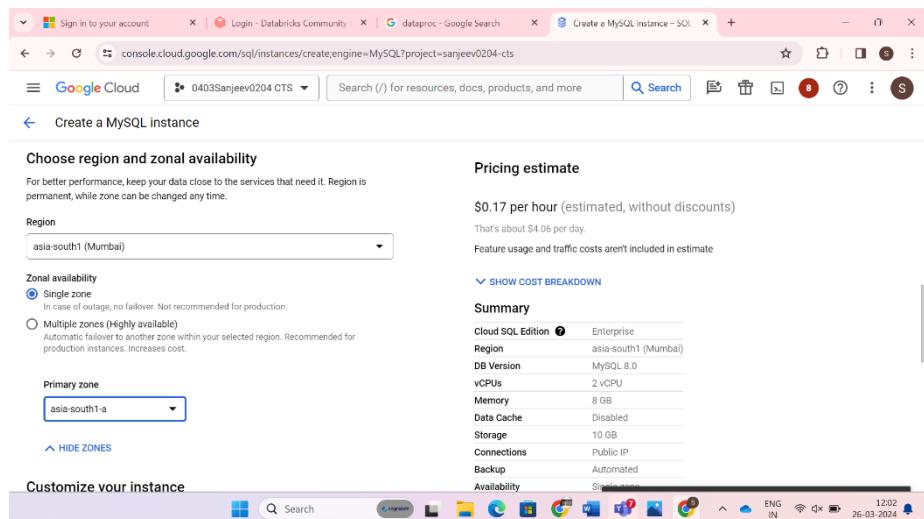
At the bottom, there is a toolbar with icons for file operations, search, and system status (12:01, ENG IN, 26-03-2024).

2) Compare to other production sandbox is less in features and prize.



Region and zone:

A region is a specific geographical location where you can run your resources. Each region is subdivided into several zones. For example, the us-central1 region in the central United States has zones us-central1-a , us-central1-b , us-central1-c , and us-central1-f .



3) Shared Core (vCPU) Instances and Dedicated Core (vCPU) Instances:

Shared Core (vCPU) Instances:

- Shared-core machine types provide **one virtual CPU** that runs for a portion of the time on a single hardware hyper-thread of the host CPU.
- These instances are **cost-effective** for running small, non-resource-intensive applications.

Dedicated Core (vCPU) Instances:

- In contrast, dedicated core instances allocate **exclusive CPU resources** to your virtual machine.
- Each core is reserved for your instance, ensuring consistent performance and security.

So, we using shared core for cost-effective

Sign in to your account | Login - Databricks Community | dataproc - Google Search | Create a MySQL instance - SQL

console.cloud.google.com/sql/instances/create;engine=MySQL?project=sanjeev0204-cts

Google Cloud 0403Sanjeev0204 CTS Search (/) for resources, docs, products, and more Search

Create a MySQL instance

You can also customize instance configurations later

Machine configuration

Machine shapes
Choose from the most commonly used shapes or choose a custom shape. Make sure your selection has enough memory to hold your largest table.

Shared core

1 vCPU, 0.614 GB (selected)

1 vCPU, 1.7 GB

Data cache
Exclusive to Enterprise Plus edition. Adds a local SSD for read caching to reduce read latency and improve performance at additional cost. [Learn more](#)

Enable data cache

Storage
Storage type is SSD. Storage size is 10 GB, and will automatically scale as needed. Google-managed key enabled (most common).

Pricing estimate

\$0.02 per hour (estimated, without discounts)
That's about \$0.37 per day.
Feature usage and traffic costs aren't included in estimate

Show cost breakdown

Summary

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone

ENG IN 26-03-2024 12:03

4) Storage SSD or HDD:

Solid state drives (SSDs) typically use flash-based memory to store data and thus have no moving parts. They have faster read/write speeds than HDDs, lower access times (less latency), and a higher cost per gigabyte of storage.

So, we choose SSD.

Storage

Storage type
Choice is permanent. Storage type affects performance.

SSD (Recommended)
Most popular choice. Lower latency than HDD with higher QPS and data throughput.

HDD
Lower performance than SSD with lower storage rates.

Storage capacity
10 - 65,536 GB. Higher capacity improves performance, up to the limits set by the machine type. Capacity can't be decreased later.

10 GB
 20 GB
 100 GB
 250 GB
 Custom

Enable automatic storage increases
If enabled, whenever you are nearing capacity, storage will be incrementally (and permanently) increased. [Learn more](#)

Pricing estimate
\$0.02 per hour (estimated, without discounts)
That's about \$0.37 per day.
Feature usage and traffic costs aren't included in estimate

[SHOW COST BREAKDOWN](#)

Summary

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single master

5) Network connection :

A public IP address is a unique IP address assigned to your network router by your internet service provider and can be accessed directly over the internet. A private IP address is a unique address that your network router assigns to your device. It is used within a private network to connect securely to other devices.

The screenshot shows a browser window with the following tabs:

- Sign in to your account
- Login - Databricks Community
- dataproc - Google Search
- Create a MySQL instance – SQL

The main content area is titled "Create a MySQL instance". On the left, there's a "Connections" section with "Instance IP assignment" options: "Private IP" (unchecked) and "Public IP" (checked). A note says: "Assigns an external, internet-accessible IP address. Requires using an authorized network or the Cloud SQL Proxy to connect to this instance." Below this is an "Authorized networks" section with a note: "You can specify CIDR ranges to allow IP addresses in those ranges to access your instance." A warning message states: "⚠ You have added 0.0.0.0/0 as an allowed network. This prefix will allow any IPv4 client to pass the network firewall and make login attempts to your instance, including clients you did not intend to allow. Clients still need valid credentials to successfully log in to your instance." On the right, there's a "Pricing estimate" section showing "\$0.02 per hour (estimated, without discounts)" and a "Summary" table:

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone

The status bar at the bottom shows: ENG IN 12:03 26-03-2024.

The screenshot shows a browser window with the same tabs as the previous screenshot.

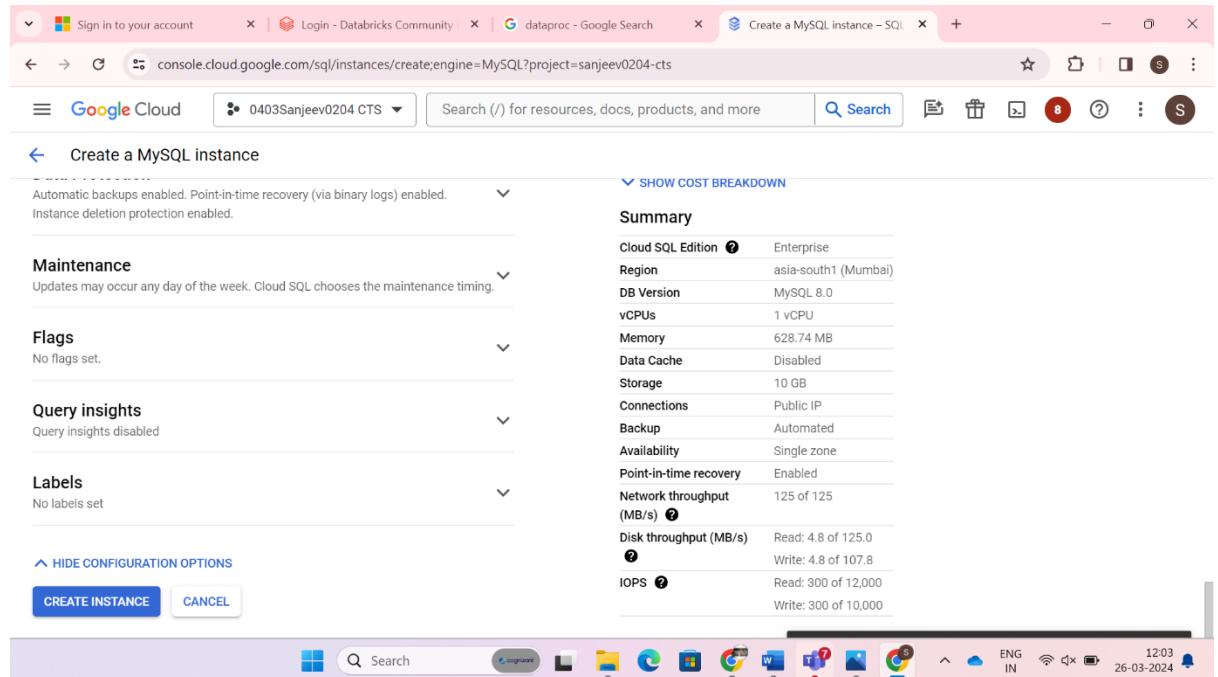
The main content area is titled "Create a MySQL instance". On the left, there's a "New network" dialog with fields for "Name" (set to "all") and "Network" (set to "0.0.0.0/0"). A note below says: "⚠ You have added 0.0.0.0/0 as an allowed network. This prefix will allow any IPv4 client to pass the network firewall and make login attempts to your instance, including clients you did not intend to allow. Clients still need valid credentials to successfully log in to your instance." Below the dialog is an "ADD A NETWORK" button.

On the right, there's a "Pricing estimate" section showing "\$0.02 per hour (estimated, without discounts)" and a "Summary" table:

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone

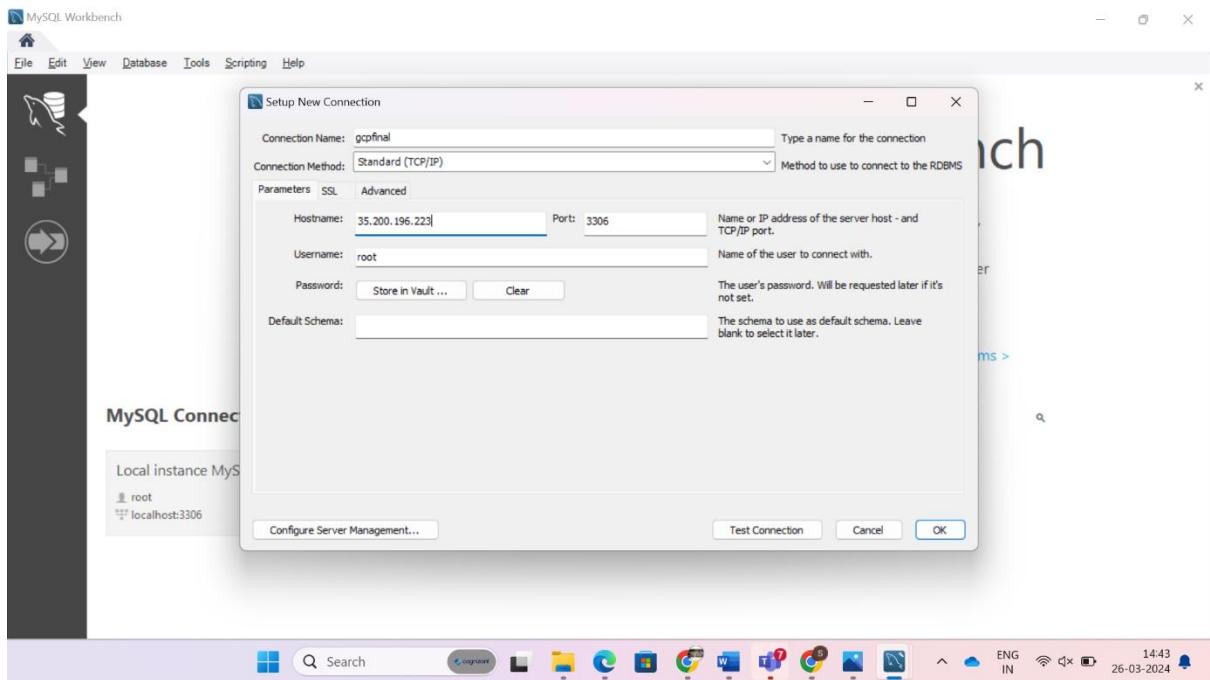
The status bar at the bottom shows: ENG IN 12:03 26-03-2024.

6) Then create a instance:



Now MySql instance is create,

- 1) Open mysql work bench.
- 2) And open setup new connections and enter a connection name.
And Hostname in that we need give mysql instance pubile IP address.
- 3) Enter a password what we are given in mysql instance.
- 4) And check for the connection After the successful connection.
- 5) Then open the mysql work bench and create a database as “demo”.



- We have a code to connect Mysql cloud to Databricks.
- Now we need to give IP address in URL.
- Enter database name and table name in table row.
- Then enter the password and run the program.

A screenshot of a Databricks notebook titled 'Untitled Notebook 2024-03-22 10_38_55'. The notebook is in Python mode. The code cell contains the following Python code:

```
1 driver = "com.mysql.cj.jdbc.Driver"
2 url = "jdbc:mysql://35.200.196.223"
3 table = "demo.demo123"
4 user = "root"
5 password = "sanjeev@123"
6
7 dfp.write.format("jdbc").option("driver", driver).option("url",url).option("dbtable", table).option("mode", "append").option("user",user).option("password", password).save()
```

The cell is run successfully, with the output showing the command took 24.63 seconds and the job status indicating 28 jobs completed and 29 skipped. The notebook interface includes a sidebar with icons for file operations, a search bar, and a top navigation bar with tabs like 'File', 'Edit', 'View', 'Run', 'Help', and 'Last edit was 3 minutes ago'.

Check load data in Mysql cloud:

- Open Mysql work bench and enter the command. Using select statement.
- You will see the loaded data.

The screenshot shows the MySQL Workbench interface. In the Query Grid, a single row of data is selected from a table named 'demo123'. The log output below shows the creation of the database 'demo' failed because it already exists, and then successfully executed the SELECT query.

ID	Name	Email_Id	Age	Gender	Number	Username	Password
1001	Sonam	sonam.sharma@yahoo.com	30	female	1098765432	sonam118	pPozAEH
1002	Rohit	rohit.verma@hotmail.com	33	male	2109876543	rohit935	SAH6KWP2
1003	Arjun	arjun.saxena@hotmail.com	32	male	0987654321	arjun104	eAFCaZ2
1004	Yash	yash.singh@gmail.com	36	male	2109876543	yash909	h1vR9Q8o
1005	Raj	raj.kumar@hotmail.com	29	male	4321098765	raj132	jhqzeYwO

Action Output:

#	Time	Action	Message	Duration / Fetch
1	14:37:09	create database demo	1 row(s) affected	0.078 sec
2	14:38:10	create database demo	Error Code: 1007. Can't create database 'demo': database exists	0.063 sec
3	14:38:53	show databases	5 row(s) returned	0.063 sec / 0.000 sec
4	14:42:50	select * from demo.demo123 LIMIT 0, 1000	54 row(s) returned	0.078 sec / 0.000 sec

The screenshot shows the Google Cloud SQL Instances page. A table lists the instance 'sanjeev123', which is an Enterprise edition running MySQL 8.0. The instance has a public IP address of 35.200.196.223 and is connected to a connection name 'sanjeev0204-cts:asi...'. It is set to High availability 'ENABLE' and is located in 'asia-south1-a'.

Instance ID	Cloud SQL edition	Type	Public IP address	Private IP address	Instance connection name	High availability	Location	Actions
sanjeev123	Enterprise	MySQL 8.0	35.200.196.223		sanjeev0204-cts:asi...	ENABLE	asia-south1-a	⋮

Chapter-3

STREAMING USING ETL:

1)Imports:

It imports necessary modules and functions from PySpark, such as SparkSession, udf, split, col, monotonically_increasing_id, StructType, StructField, StringType, BooleanType, as well as standard Python modules like re (regular expressions), random, and string.

2)SparkSession:

It creates a SparkSession named "StreamingWithETL" or retrieves an existing one if available

The screenshot shows a Databricks notebook titled 'final1' in Python. The notebook contains three code cells:

```
1 from pyspark.sql import SparkSession
2 from pyspark.sql.functions import udf, split, col, monotonically_increasing_id
3 from pyspark.sql.types import StructType, StructField, StringType, BooleanType
4 import re
5 import random
6 import string
7
8 spark = SparkSession.builder.appName("StreamingWithETL").getOrCreate()
```

Command took 0.29 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:08:54 PM on final22

```
1 def is_valid(email):
2     pattern = r'^[A-Za-z0-9.%+-]+@[yahoo|hotmail|gmail]\.(com|org|net)$'
3     return bool(re.match(pattern, email))
```

Command took 0.13 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:09:22 PM on final22

```
1
```

The notebook interface includes a sidebar with icons for file operations, a search bar, and a status bar at the bottom showing system information like battery level, signal strength, and date/time.

3) User-Defined Functions (UDFs):

- `is_valid`: A function to validate email addresses based on a regular expression pattern.
- `generate_username`: A function to generate a username from a given name.
- `generate_password`: A function to generate a random password.

```
1 def is_valid(email):
2     pattern = r'^[A-Za-z0-9.%+-]+@[yahoo|hotmail|gmail]\.(com|org|net)$'
3     return bool(re.match(pattern, email))

Command took 0.13 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:09:22 PM on final22

1 def generate_username(name):
2     n=name.lower()
3     r=random.randint(100, 999)
4     return f"{n}{r}"

Command took 0.10 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:11:21 PM on final22

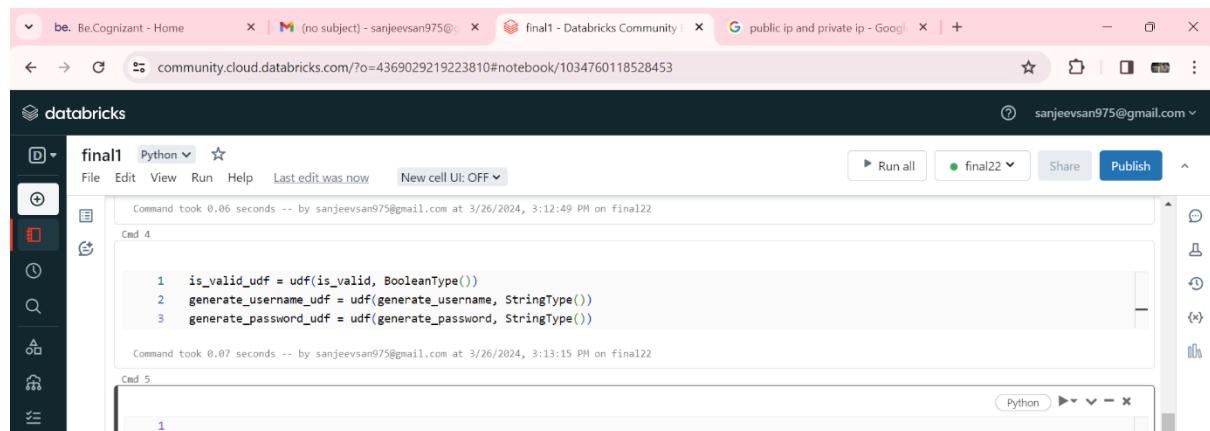
1
```

```
1 def generate_username(name):
2     n=name.lower()
3     r=random.randint(100, 999)
4     return f"{n}{r}"
5 def generate_password():
6     return ''.join(random.choices(string.ascii_letters + string.digits, k=8))

Command took 0.06 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:12:49 PM on final22
```

4) UDF Registration:

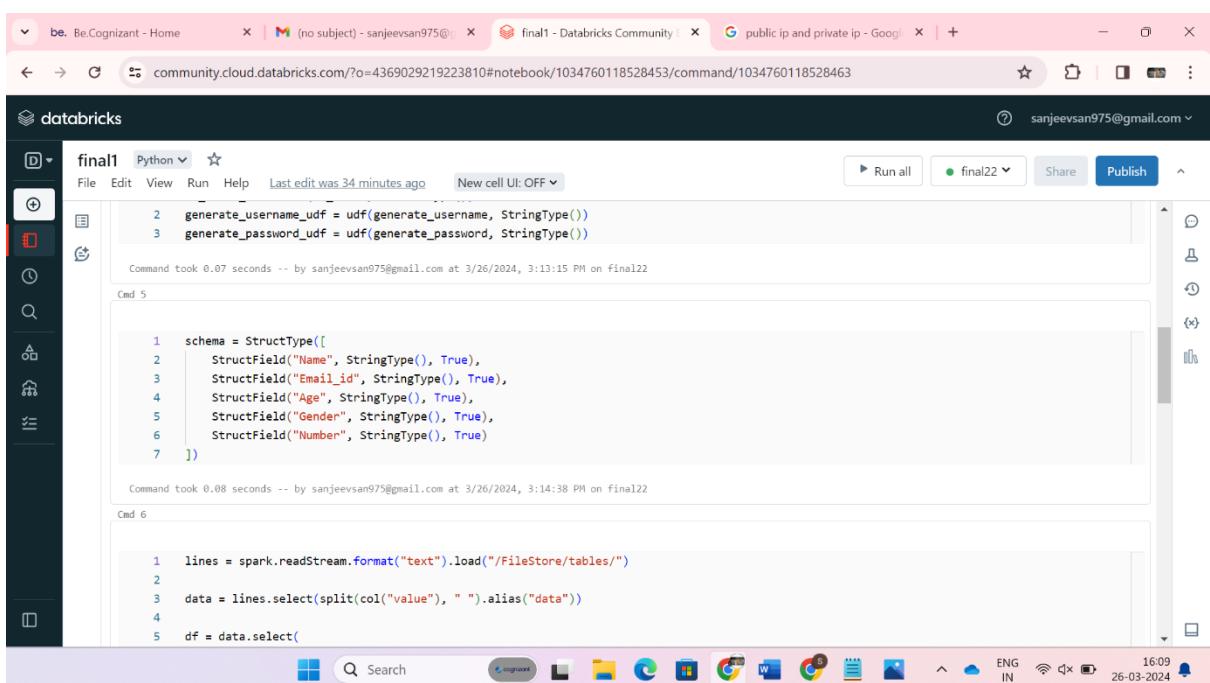
Registers the UDFs using the `udf` function provided by PySpark.



```
is_valid_udf = udf(is_valid, BooleanType())
generate_username_udf = udf(generate_username, StringType())
generate_password_udf = udf(generate_password, StringType())
```

5) Schema Definition:

Defines the schema for the structured streaming data. The schema includes fields for "Name", "Email id", "Gender", and "Number".



```
schema = StructType([
    StructField("Name", StringType(), True),
    StructField("Email_id", StringType(), True),
    StructField("Age", StringType(), True),
    StructField("Gender", StringType(), True),
    StructField("Number", StringType(), True)
])
```

```
lines = spark.readStream.format("text").load("/FileStore/tables/")
data = lines.select(split(col("value"), " ").alias("data"))
df = data.select(
```

6) Reading Stream:

- Reads streaming data from a specified location ("FileStore/tables/") as text files.

Data Transformation:

- Splits the lines into columns using the `split` function and assigns the result to a column named "data".
 - Selects specific columns from the "data" column to create a DataFrame (`df`) with columns "Name", "Email_id", "Age", "Gender", and "Number".

The screenshot shows a Databricks notebook interface. The notebook is titled "final1" and is set to Python. The code in the notebook reads a text stream from a FileStore table and creates a DataFrame with columns: Name, Email_id, Age, Gender, and Number.

```
1 lines = spark.readStream.format("text").load("/FileStore/tables/")
2
3 data = lines.select(split(col("value"), " ").alias("data"))
4
5 df = data.select(
6     col("data").getItem(3).alias("Name"),
7     col("data").getItem(8).alias("Email_id"),
8     col("data").getItem(10).alias("Age"),
9     col("data").getItem(11).alias("Gender"),
10    col("data").getItem(12).alias("Number")
11 )
```

The output of the command shows the resulting DataFrame structure:

- lines: pyspark.sql.dataframe.DataFrame = [value: string]
- data: pyspark.sql.dataframe.DataFrame = [data: array]
- df: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]

Command took 3.70 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:14:48 PM on final22

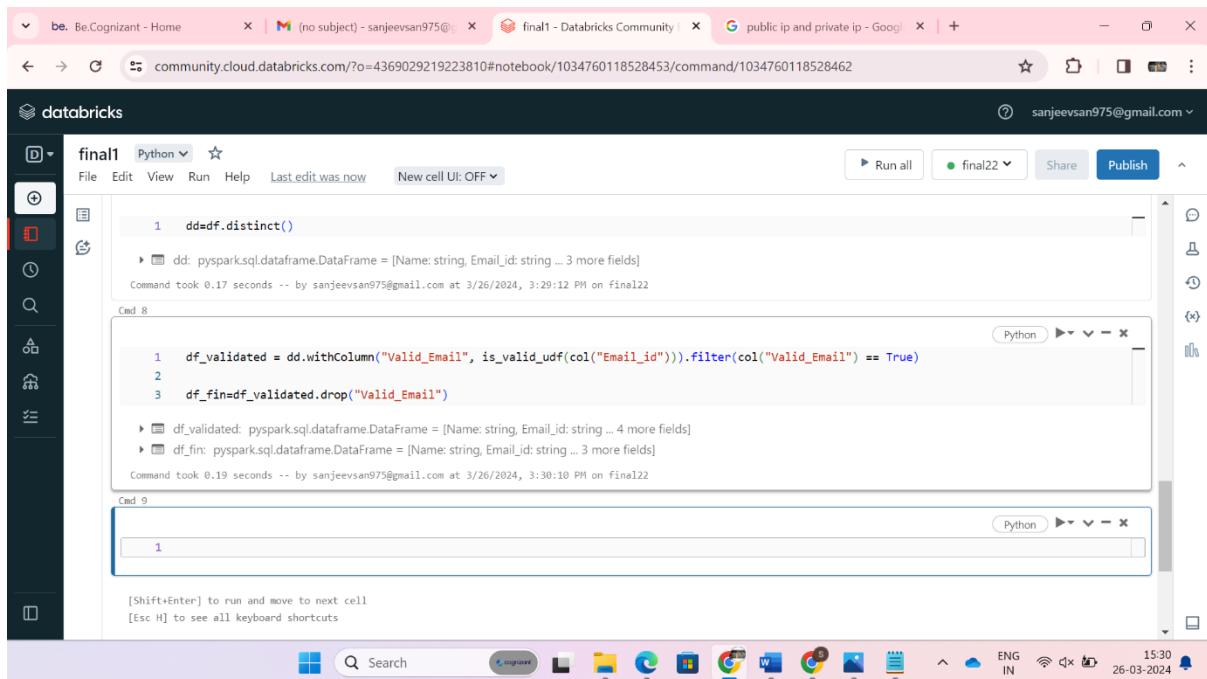
7) Data Deduplication:

- Removes duplicate records from the DataFrame (`'df'`) using the `'distinct'` method.

```
df: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]
Command took 3.70 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:14:48 PM on final122
Cmd 7
1 dd=df.distinct()
Cancel Uploading command
Cmd 8
1
```

8) Data Validation:

- Adds a column "Valid Email" to indicate whether the email addresses are valid by applying the `is_valid_udf` function.
- Filters out rows where the email address is valid.
- Drops the "Valid_Email" column from the DataFrame



The screenshot shows a Databricks notebook titled 'final1' running in Python. The notebook contains three cells:

- Cell 1:

```
1 dd=df.distinct()
```

 This cell creates a distinct DataFrame named 'dd'. The command took 0.17 seconds.
- Cell 2:

```
1 df_validated = dd.withColumn("Valid_Email", is_valid_udf(col("Email_id"))).filter(col("Valid_Email") == True)
2 df_fin=df_validated.drop("Valid_Email")
```

 This cell adds a 'Valid_Email' column using the user-defined function 'is_valid_udf' on the 'Email_id' column. It then filters the DataFrame to keep only rows where 'Valid_Email' is True and drops the 'Valid_Email' column. The command took 0.19 seconds.
- Cell 3: An empty cell with the placeholder [1].

The notebook interface includes a sidebar with various icons for file operations, a top bar with tabs for different notebooks and browser links, and a status bar at the bottom showing the date and time.

Additional Data Transformation:

- Adds a "Username" column by applying the `generate_username_udf` function.
- Adds a "Password" column by applying the `generate_password_udf` function.



The screenshot shows a Databricks notebook titled 'final1' running in Python. The notebook contains two cells:

- Cell 1:

```
1 df_fin: pyspark.sql.DataFrame = [Name: string, Email_id: string ... 3 more fields]
```

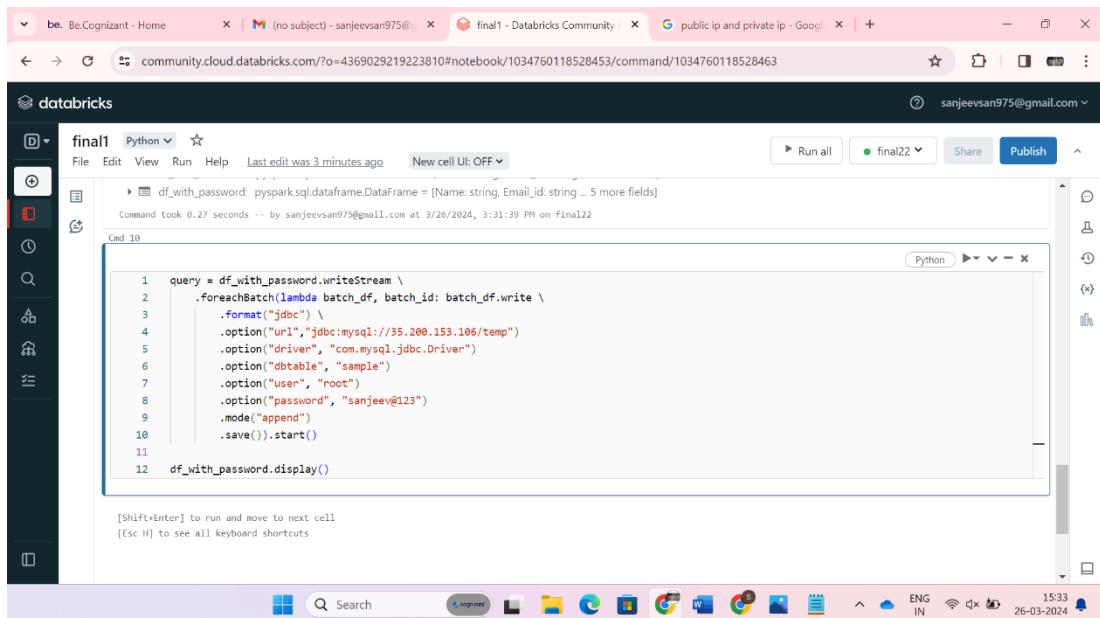
 This cell defines a DataFrame variable 'df_fin' with the specified schema. The command took 0.19 seconds.
- Cell 2:

```
1 df_with_username = df_fin.withColumn("Username", generate_username_udf(col("Name")))
2 df_with_password = df_with_username.withColumn("Password", generate_password_udf())
3 df_with_username: pyspark.sql.DataFrame = [Name: string, Email_id: string ... 4 more fields]
4 df_with_password: pyspark.sql.DataFrame = [Name: string, Email_id: string ... 5 more fields]
```

 This cell adds a 'Username' column to 'df_fin' using the 'generate_username_udf' function on the 'Name' column. It then adds a 'Password' column to the resulting DataFrame using the 'generate_password_udf' function. The final transformed DataFrames are stored in 'df_with_username' and 'df_with_password' respectively. The command took 0.19 seconds.

9) Writing to Sink in MySQL Cloud:

- Writes the processed data to a JDBC sink (MySQL Cloud) using a foreach Batch writer.
- Specifies connection parameters such as URL IP Address, table name, username, and password.
- Create mysql instance in cloud.
- In mysql workbench create new connection and link with Cloud data.

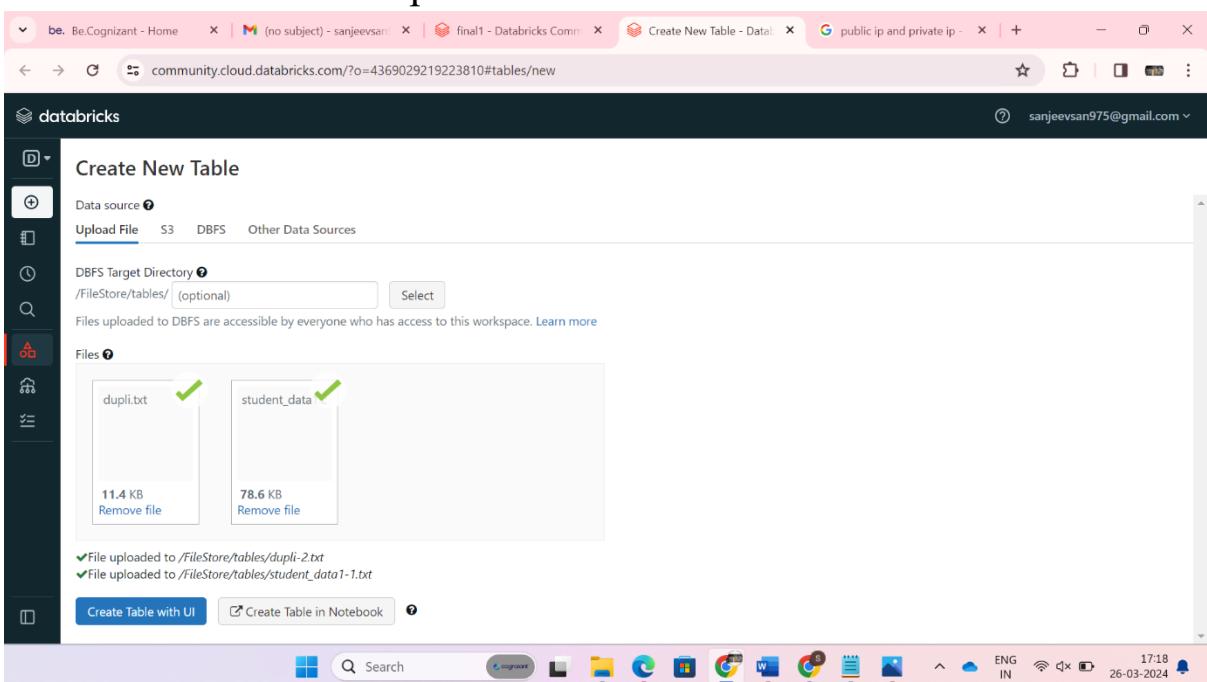


The screenshot shows a Databricks notebook titled "final1" running Python code. The code uses the PySpark DataFrame API to write data to a MySQL database. The code snippet is as follows:

```
query = df_with_password.writeStream \
    .foreachBatch(lambda batch_df, batch_id: batch_df.write \
        .format("jdbc") \
        .option("url", "jdbc:mysql://35.200.153.106/temp") \
        .option("driver", "com.mysql.jdbc.Driver") \
        .option("dbtable", "sample") \
        .option("user", "root") \
        .option("password", "sanjeev@123") \
        .mode("append") \
        .save()).start()
df_with_password.display()
```

The notebook interface includes a sidebar with various icons, a toolbar with "Run all", "Share", and "Publish" buttons, and a status bar at the bottom showing the date and time.

- Now the streaming is started. Now we will insert the file in data bricks as a steam process.



The screenshot shows the "Create New Table" interface in Databricks. Under the "Data source" section, "Upload File" is selected. The "DBFS Target Directory" is set to "/FileStore/tables/". Two files are listed in the "Files" section: "dupli.txt" (11.4 KB) and "student_data" (78.6 KB). Both files have a green checkmark indicating they have been uploaded successfully. At the bottom, there are two buttons: "Create Table with UI" and "Create Table in Notebook".

Databricks Notebook Screenshot:

Name	Email_id	Age	Gender	Number	Username	Password
1 Kartik	kartik.agarwal@gmail.com	33	male	4321098765	kartik214	6d5fwR0
2 Rishi	rishi.saxena@gmail.com	32	male	6543210987	rishi857	dTrdOdeq
3 Priyanka	priyanka.patel@gmail.com	26	female	1098765432	priyanka495	TXrAUCuj
4 Priya	priya.sharma@gmail.com	29	female	9876543210	priya120	iuqtHzF
5 Sonam	sonam.sharma@yahoo.com	30	female	1098765432	sonam578	xLREwbp7
6 Arjun	arjun.gupta@gmail.com	29	male	0987654321	arjun907	8HEW97QE
7 Rohit	rohit.verma@hotmail.com	33	male	2109876543	rohit175	NZ4o2abl

10) Now data will be stored in MySQL cloud. To get that data we need to open MySQL workbench and using select statement we can see that data.

MySQL Workbench Screenshot:

```
Query 1:
1 • create database temp;
2 • select * from temp.sample;
3 • select count(*) from temp.sample;
```

Result Grid:

count(*)
334

Action Output:

#	Time	Action	Message	Duration / Fetch
1	16:06:11	create database temp	1 row(s) affected	0.188 sec
2	16:13:17	select * from temp.sample LIMIT 0, 1000	54 row(s) returned	0.672 sec / 0.000 sec
3	16:15:24	select count(*) from temp.sample LIMIT 0, 1000	1 row(s) returned	1.063 sec / 0.000 sec
4	16:16:44	select count(*) from temp.sample LIMIT 0, 1000	1 row(s) returned	0.109 sec / 0.000 sec

11) before creating a instance in MySQL cloud.

The screenshot shows the Google Cloud Platform Instances page. The URL is `console.cloud.google.com/sql/instances?project=sanjeev0204-cts`. The page displays a table of instances:

Instance ID	Cloud SQL edition	Type	Public IP address	Private IP address	Instance connection name	High availability	Actions
sanjeev123	Enterprise	MySQL 8.0	35.200.153.106		sanjeev0204-cts:asi...	ENABLE	⋮

Below the table, a notification bar indicates: "Created sanjeev123" at "4:03:56 PM GMT+5". The taskbar at the bottom shows various application icons and system status.

Conclusion:

Streaming and batch ETL are indispensable components of modern data processing architectures, offering organizations the flexibility to process data in real-time and batch modes according to their specific use cases and requirements. By leveraging both methodologies, organizations can derive actionable insights, optimize decision-making processes, and unlock new opportunities for innovation and growth in today's data-driven landscape.