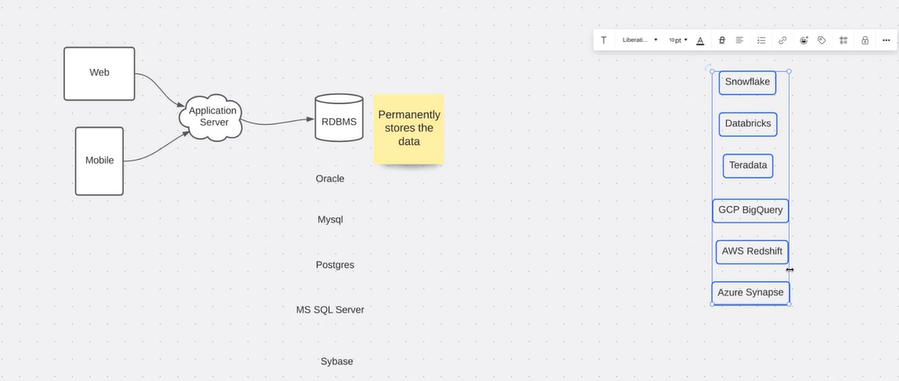
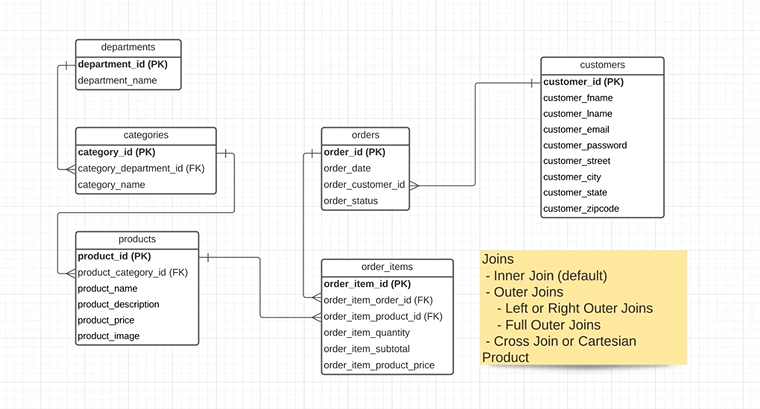
Python for Data Engineer Data warehouse Tools:1) Snow Flake (Recommended)2) Big Query3) Red shift Data Processing Fundamentals:1)Batch Processing - means data in Batch - Apache Spark Fundamental and Apache Spark is a Tool and managed version of it is Databricks,Aws emr, Aws glue and cloud data proc 2) Real Time Stream Processing Data - Apache Kafka Tool and Apache Flink ToolWorkflow Management - Apache AirflowCloud Platform - for data storage ie Big Data Data Ops - means Devops ie developement and operations Modern Data Stack DBT - data build tool

<https://docs.google.com/document/d/1MEFXQhtcOZrJGztsAhAu7SJj181toHL0IEVNbVmr61o/edit>

<https://www.learnpython.org/en/Variables_and_Types>

 **Database Data Warehouse or (**MPP massively parallel processing)

**Data Model Diagram**:

**Purpose Built Databases:**

Ex:

1. LinkedIn :

* For activity – NOSQL
* For Subscription – RDBMS i.e. Transaction related
* Profile – NOSQL
* Jobs – NOSQL or Search Based Databases
* Here for each purpose we are using different types of databases.
* Data Warehouse:
  + Structured and organized data.
  + Predefined schema.
  + Designed for reporting and structured data analysis.
  + Ex: Imagine a retail company that wants to analyze its sales data. The company extracts data from various sources like point-of-sale systems, e-commerce platforms, and customer databases. They clean and structure this data, removing duplicates and inconsistencies, and load it into a data warehouse. Analysts can then easily query the data warehouse to generate reports on sales trends, inventory management, and customer behavior.
* Data Lake:
  + Flexible and unstructured data.
  + Schema-on-read, not predefined.
  + Suited for storing and processing a wide range of data types, especially in big data and analytics scenarios.
  + Ex : social media company that collects a vast amount of data daily, including text, images, videos, user interactions, and more. Instead of structuring the data beforehand, they store it in a data lake. This unstructured data can later be used for various purposes like sentiment analysis, image recognition, and user behavior analysis. The data lake allows them to store and process massive volumes of data without needing to define a schema upfront.

Difference between Data warehouse and RDBMS:

1. **RDBMS**: tables, columns, data types, rows and columns.

* Transactional (OLTP – online transaction processing) (build mobile or web applications)
* SQL

1. **Data Warehouse**: tables, columns, data types, rows and columns.

* DSS (decision support system) or OLAP ( online analytical processing) (build reports or dashboards)
* SQL

<https://github.com/dgadiraju/data>

**Git commands: (Windows Power shell)**

Mkdir

Cd

Git clone (paste git URL)

**CREATE DATABASE** itversity\_user\_retail;

**CREATE USER** itversity\_user **WITH ENCRYPTED PASSWORD** 'Dai@2020';

**GRANT ALL ON DATABASE** itversity\_user\_retail **TO** itversity\_user;

**Top 10 records in postgress sql:**

Select \* from categories order by category\_id asc limit 10

**Execution of order:**

From -- > Where (filtering) --> Group by and having clause --> Select -- >Order by

Aggregating functions are not used in **where clause** as per rules and restrictions

Use **Having Clause** to filter the **aggregated data**

* + - Here alias is not allowed instead we need to use aggregated function like **count** (\*) > 123.

**Joints**

**Views**

* + - **create or replace view** view\_name

**As**

**View** does not hold data and it is not a physical structure and we can use view in multiple queries.

Where as **Table** is a Physical Structure and holds data.

**CTE: (common table expression) –**

**With cte\_name as**

(Condition)

**Cumulative Aggregations:**

**CTAS** - (create a table structure but no data to be inserted)

Create table orders\_stg

As

Select \* from orders where 1=2; or false (in postgress)

**Ranking in SQL: (analytical or windows functions)**

* Rank over() -
* Dense rank over() -
* Global ranking over() -
* Ranking based on key or partition key

**SQL Troubleshooting issue:**

1. **Database connectivity issue: telnet** is powerful tool for solving this issue.
2. **Syntax and semantic issues:** syntax is keyword and semantic means table, column names.
3. **Bugs in queries:**
4. **Performance issues**

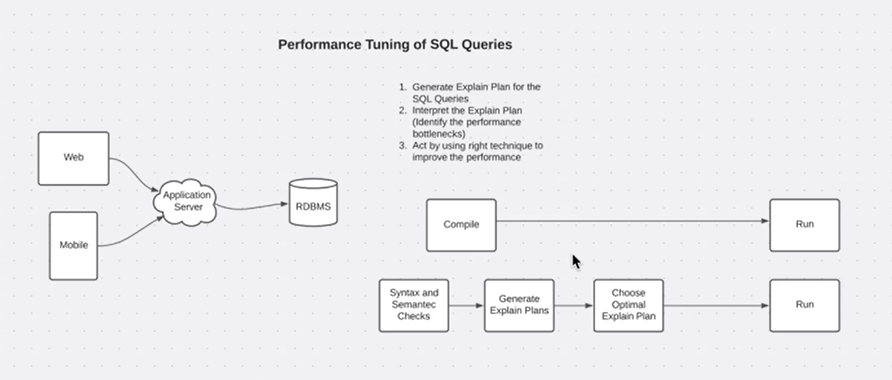
**telnet** localhost (ip address) 5433 -- **enter** – if it is blank means no issues

Check with **DBA** weather **database** **server** is running or not

Check weather **firewall** is blocking the connection btw server and pgadmin

**Bugs**: output is different from our requirement then it is called bugs.

**Performiming tuning in sql server:**



* Using **Explain** command we do the **stuff.**

**Note:** whenever we have a primary and unique key we have index associated with in it.

**Index scan:**

* **Description:** An index is a data structure that allows the database engine to locate and retrieve specific rows efficiently.

**Ex**: SELECT email FROM users WHERE user\_id = 123; filter based in index

**Sequential Scan (full table scan):** A sequential scan involves reading each row in a table one by one, from the beginning to the end, without using an index.

**Ex**: SELECT email FROM users WHERE username = 'john\_doe'; filter based on name

**Create index** index\_name **on** table\_name (column name)

**COALESCE:** to replace the null values we are using this function.

SELECT id, **COALESCE** (first\_name, 'Unknown') || ' ' || **COALESCE** (last\_name, 'Unknown') AS full\_name

FROM employees;

**Create a virtual environment:**

Python -m venv pr-venv

**Activate python virtual environment**

C:\Python-data-engineer\pr-venv\Scripts\activate.ps1

**Jupiter notebook install in VS code**

Install the Jupiterextensions

Create a file with extensions - **.ipynb**

**Shift + enter -** to run the code

**Ipython** - is much more power, robust

**Variables** – are like a placeholder for any values.

**Help command**–related to class, function and variables

Python collections: Even though all the four python collections are important, we see **list** and **dict** more often when it comes to **actual data processing**. In data engineering projects we use **pandas**, **spark**, **etcetera**. We don't use **Python collections** for **data processing**, but when it comes to dealing with **metadata**, we have to use Python collections.

1. List – List items are ordered, changeable, homogenous and allow duplicate values.
2. Tuples – Tuple items are ordered, unchangeable, heterogeneous and allow duplicate values.

* Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

1. Set – items are unordered, unchangeable, and do not allow duplicate values.
2. Dict- group of key-value pairs and key is always unique

**Lambda Functions:** are the functions **without** **names**.

* If you have a scenario where your function use in 1 place and will **not** **use** anywhere else then we use this **lambda function.**
* Lambda function does not have a **name.**
* We simply directly pass the **arguments** directly.

**Overview of JSON Strings and Files:**

* Metadata and properties data are available in json.
* Yes, you are correct. **json**.**load** () (when working with a **file**) or **json.load**s () (when working with a **string**) is used to convert JSON-formatted data into the respective equivalent Python data types. The mapping between JSON data types and Python data types is as follows:

JSON object ({}) maps to Python dictionary (dict)

JSON array ([]) maps to Python list (list).

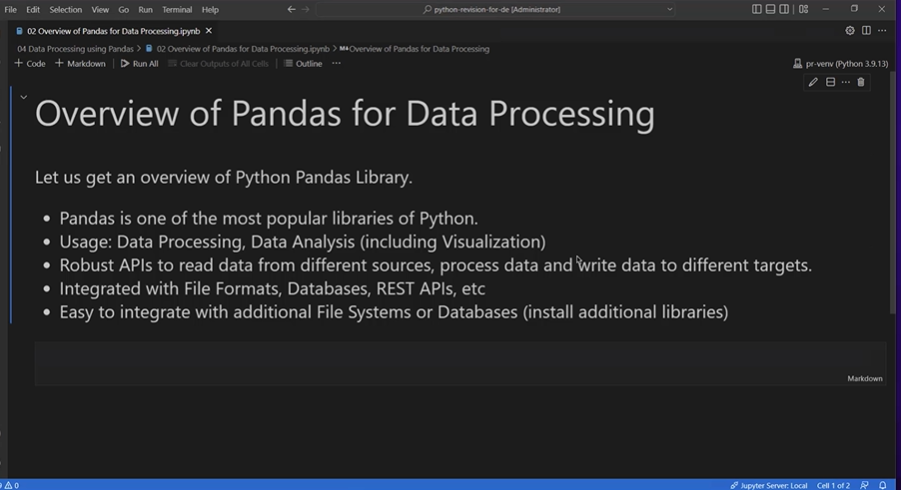
JSON string maps to Python string (str).

JSON number (integer or float) maps to Python int or float.

JSON true/false maps to Python True or False

JSON null maps to Python None

**Overview of Pandas:**



* Pip install pandas
* Pip show pandas

**Key Points:**

* When working with pandas we have multiple formats :
* **Queries** –to interact with the pandas
* **Boolean , and operator and in operator**
* **Ex: pd.query(‘expr == “condition”’)**

**Csv File to Json file converter:**

* Python
* Python –m venv ffc-venv
* Choose the interpreter by selecting the option called ctr + shift + p
* pip show pandas | Select-String Version -- to check the pandas version
* requirements.txt – where you can specify the dependencies and version of the library we need to install and
* pip install –r .\requirements.txt - to install the library from requirements.txt file
* exit – to come out from the virtual environment

**Dynamically pass the values to your application using Environment Variables:**

* **SDLC life cycle**: stands for Software Development Life Cycle

**How to pass run time argument in python applications:**

* **By sys module**
* import sys

args = sys.argv

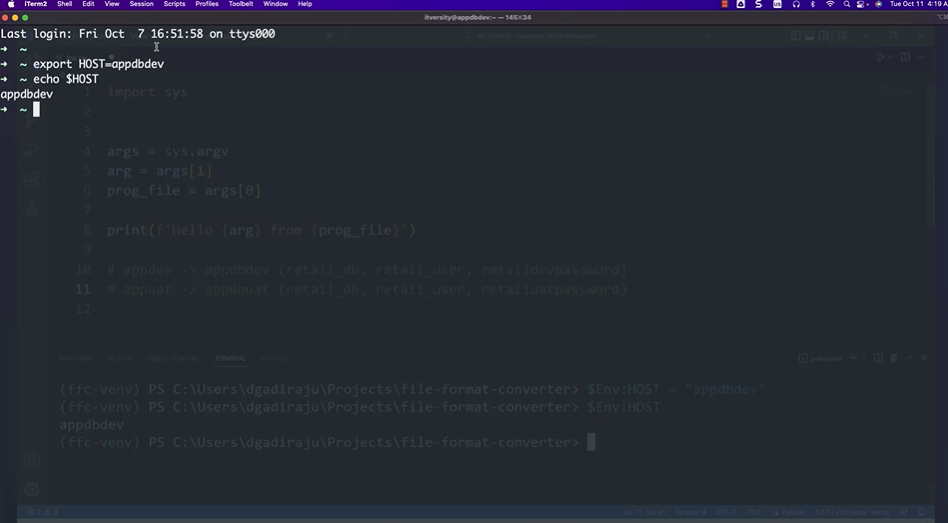
arg = args[1]

prgm\_file = args[0] #means program file

print(f"Hello {arg} from {prgm\_file}")

* Sys has an **agrv** which contains the **application** **name** and **arguments** that are passed during the **runtime**.

**Setting environment variable for mac or linux:**

****

* **To set Environment variable in windows power shell:**
* **$Env:Host = "appdbdev"**
* **Sys** for accessing command line arguments.

>>> import os

>>> Type (os.environ)

<Class 'os.\_Environ'>

>>> os.environ.get ("HOST")

'appdbdev'

**Project 1 - Files to json converter**

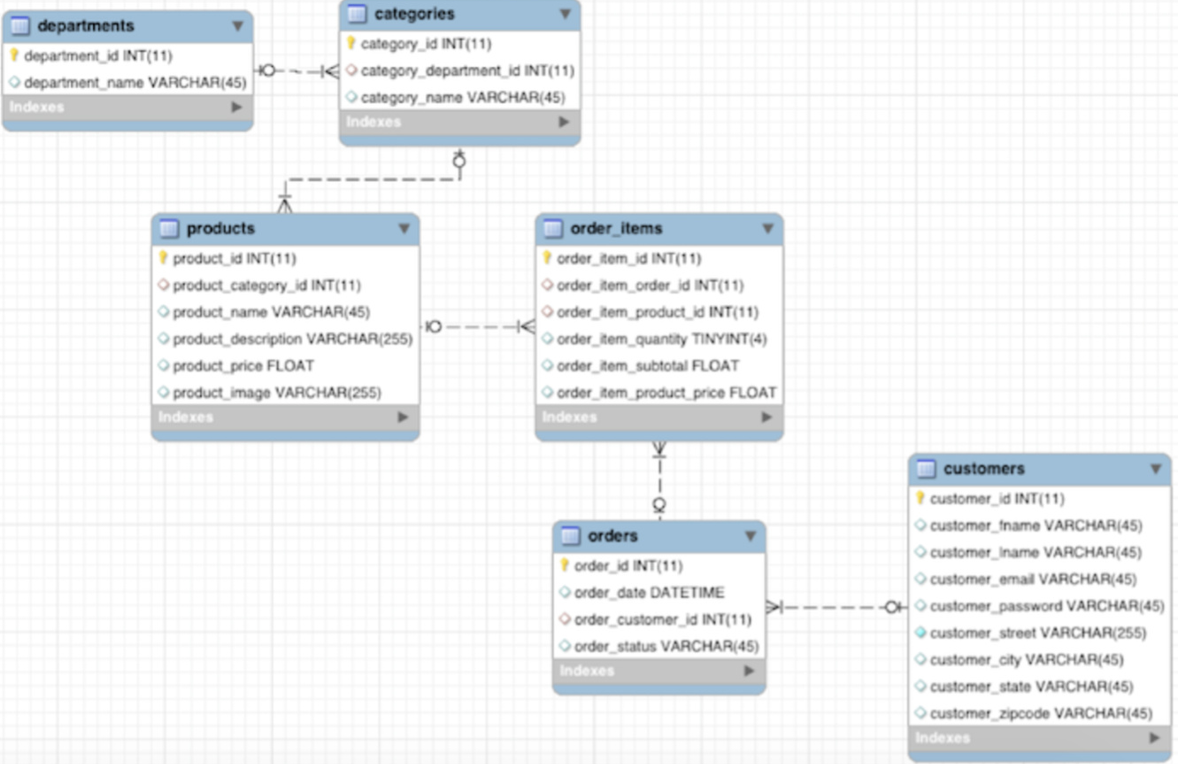
**Project 2 - Files to Database Loader Handout**

Overview

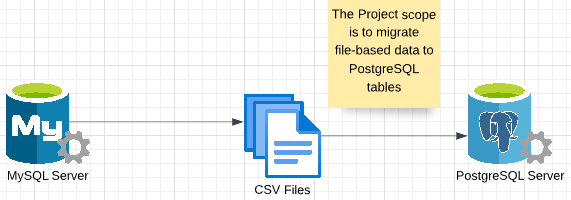
The objective of this project is to develop solutions based on the design provided. In this case, the source data was obtained in the form of files from a MySQL DB.

We require the data to be loaded into PostgreSQL, which is a common scenario in companies who want to change underlying database technologies, in which case the data from one DB needs to be migrated into another DB. The Project scope is to migrate file-based data to PostgreSQL tables

**Data Model Details**



Design



**Setup Instructions**

The previous project can be used as a reference for the following steps. Watch the relevant videos to take care of the setup.

1. Setup the Project Using VSCode
2. Make sure you have set up a virtual environment (creating venv, requirements.txt, etc.,) and installed dependencies for the project.
3. It is essential that you deploy the application with the core logic.
4. Be sure to drop the tables and recreate them using the scripts OR simply truncate the tables with the truncate command before running the scripts
5. Run the project after setting all the environment variables.

Validation Steps

* You should check whether the data in the tables have been populated by running queries.
* In postgres tables, we need to confirm that the schema structure (column name, data type, etc.) was accurately reflected from the CSV file. (**Hint**: Refer to schemas.json)
* Take the count of records in the CSV files and compare it to the number of records in the PostgreSQL tables. The count should match the numbers below.

1. select count(\*) from orders; --68883
2. select count(\*) from order\_items; --172198
3. select count(\*) from categories; --58
4. select count(\*) from customers; --12435
5. select count(\*) from departments; --6
6. select count(\*) from products; --1345

**Technologies Used**

* Programming Language – Python
* Pandas – For Converting CSV to Dataframe and then load the Dataframe into Postgres Database
* **Libraries:**
* !pip install **ipython-sql** - facilities to run sql queries
* !pip install **psycopg2-binary** - to connect to postgress database server
* !pip install **sqlalchemy** -is ORM and needs native database libraries (**psycopg2**)
* Pandas internally uses sql alchemy to connect to database
* **To update the env var**

Import os

os.environ.update ({'DB\_HOST':'LocalHost'})

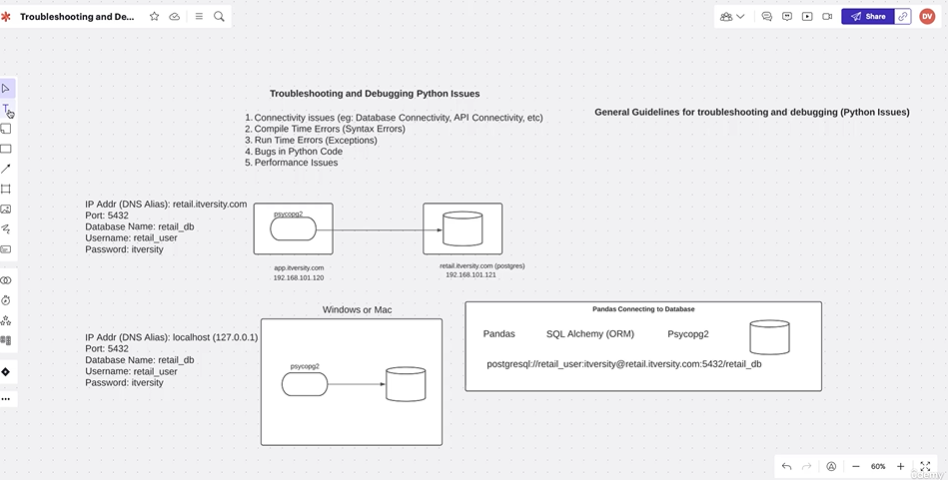
os.environ.get ('DB\_HOST')

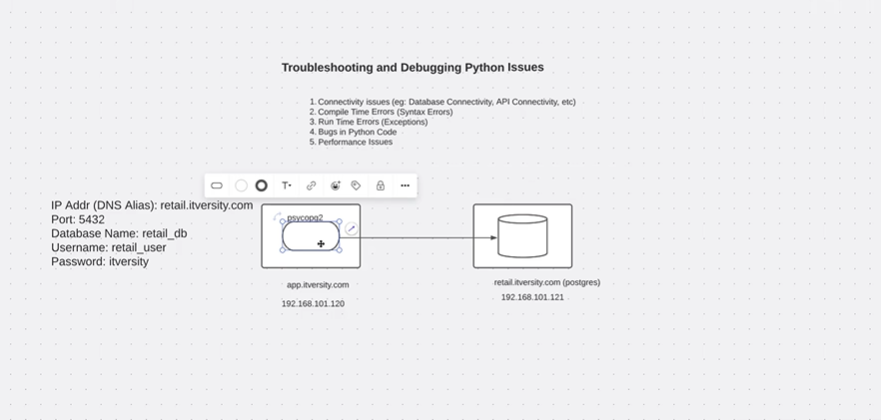
* df.to\_sql('orders',con=conn\_uri,if\_exists='append')
  1. if **replace** then drop the table and recreate table using the data in the **dataframe**
  2. if append means it will just append to the existing table
* **enumerate** – to get the index for each data frame .It is the python function to get the index while looping for each data frame
* **pip install -r .\requirements.txt -** run the file to install all the libraries
* **pip install python-dotenv --** to work with .env file

**Trobuleshooting and debugging python issues:**

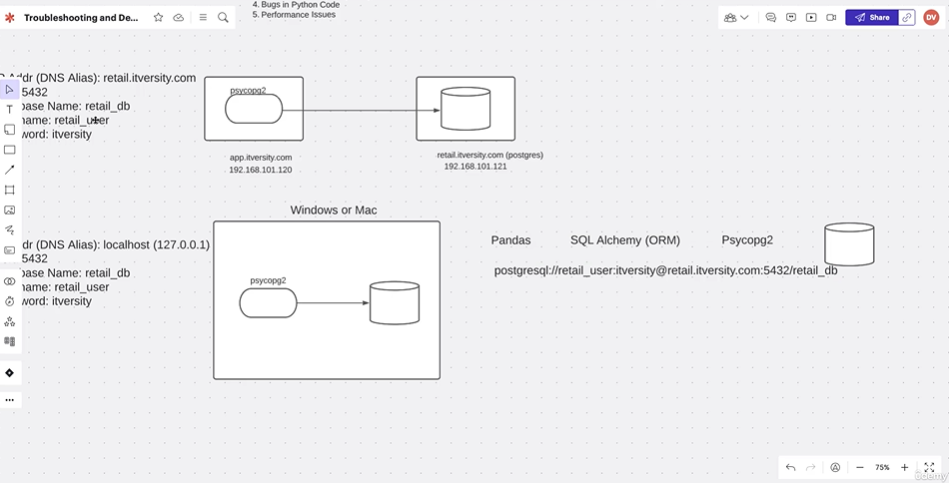
* Connectivity issues (database connectivity, API connectivity. Etc)
* compile time errors
* run time errors (exceptions)
* bugs in python code
* performance issues

**1. Troubleshooting and debugging python issues:**





* when comes to ip address you can use : **27:0:0:1** as default
* dns\_alias = pointing to **localhost**
* library **: psycopg2** to connect to database from our python application



* **Pandas** is the one of the popular **framework** with respect to **data** **engineer** or **data** **processing**
* To connect to database it uses another framework called **SQL Alchemy.**
* We can use **pandas** to connect to the **database**. However, **PANDAS** itself doesn't have ability to connect to the **database** internally. It uses something called as **SQL** **Alchemy**. SQL Alchemy is nothing but **ORM**.
* **SQL Alchemy** hides the complexity to connect to the target database using **low level API calls.**
* To connect to mssql database:
* pip install **pyodbc**
* import **pyodbc**
* # Replace these variables with your actual database connection details
* **db\_user** = 'your\_username'
* **db\_pass** = 'your\_password'
* **db\_host** = 'your\_host'
* **db\_port** = 'your\_port'
* **db\_name** = 'your\_database'
* # Constructing the MSSQL connection URI
* **db\_conn\_uri** = f'DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={db\_host},{db\_port};DATABASE={db\_name};UID={db\_user};PWD={db\_pass}'

**1. Establishing the connection**

* connection = pyodbc.connect(db\_conn\_uri)
* **To test the connection using telnet:**
* **telnet localhost 5432** -- to check the status
* If it is blank then it is listening to the port which we specify or else it will throw an error.
* **quit** – to come out of telnet
* **Telnet** is a tool and a **network** **protocol** that allows **users** to communicate with **remote** **computers** over a TCP/IP network
* Check weather **user** has a specific access to the **database.**
* Compile time error : is known as syntax error
* Run time error: one of the best way to handle is using exception handling
  1. It is a type of error, we are getting when you made mistake in the place of **int** we give **string**

Ex: try:

Except:

Exit ()

* Using **pass** is a way to have a syntactically correct placeholder that doesn't affect the **program flow,** while **exit** is a means to forcefully exit the program.

**Software development life cycle:**

* Requirements
* Design
* Development Team (unit testing is the first test done by development team and Bugs will handled by the dev team)
* Testing (QA Team)
* Production
* The way to set the **env variable** in **linux** by using **export** command in .**env** file
* **To connect to sql server:**
* To change the port in sql server:
* **Using SQL Server Configuration Manager:**
  + Open SQL Server Configuration Manager.
  + In the left pane, click on "SQL Server Services."
  + In the right pane, right-click on the SQL Server service that you want to restart (usually named "SQL Server (MSSQLSERVER)" or similar).
  + Choose "Restart."
* **Using SQL Server Management Studio (SSMS):**
  + Open SQL Server Management Studio.
  + Connect to the SQL Server instance.
  + Right-click on the server name in Object Explorer.
  + Select "Restart."
* **Using Windows Services:**
  + Open the "Services" application in Windows.
  + Find the SQL Server service in the list (it may be named "SQL Server (MSSQLSERVER)" or similar).
  + Right-click on the service and choose "Restart."

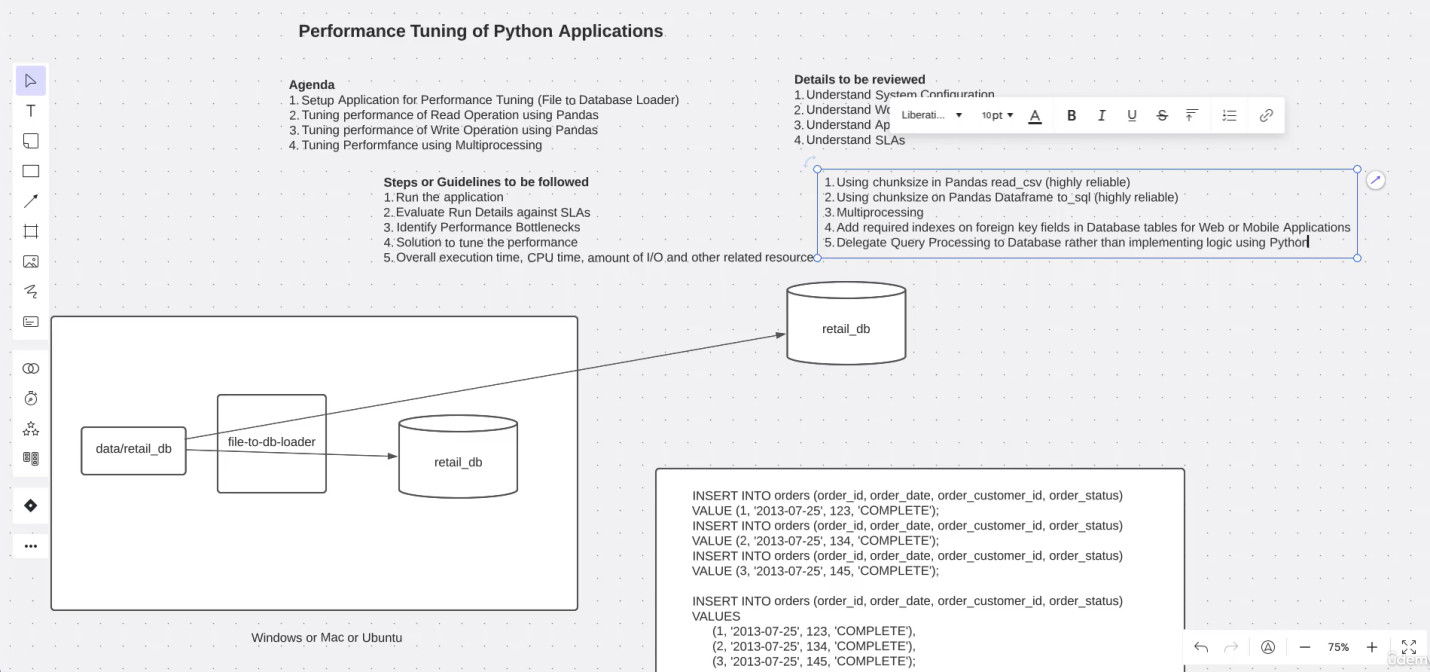
**To check in which port our sql server is running in sql server:**

* xp\_readerrorlog 0, 1, N'Server is listening on'
* SELECT HOST\_NAME() AS Hostname;
* we need driver called **pyodbc**

**Chunk** **size** - Once one chunk is processed, and then the chunk will be flushed out of the memory.

Then it will actually use memory for the next chunk, so and so forth.

**Performance Tuning in Python:**

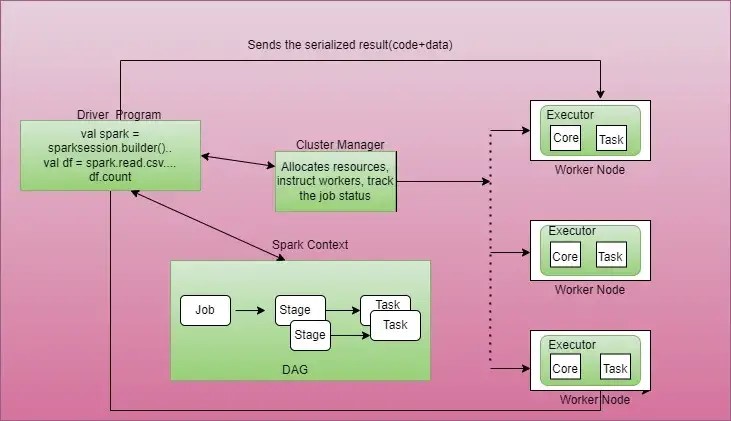


**Apache Spark:**

* To install the pyspark in vs. code studio
  1. pip install pyspark = to install the pyspark
  2. pip show pyspark = to show the version of the pyspark
  3. python -m pip –version = to check the pip version

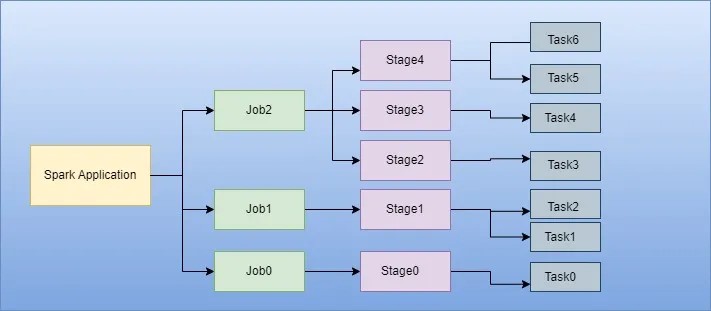
**Spark Jobs:**

To summarize, in Apache Spark, a job is created when an action is called on an RDD or DataFrame. When an action is called, Spark examines the DAG and schedules the necessary transformations and computations to be executed on the distributed data. This process creates a job, which is a collection of tasks that are sent to the worker nodes in the cluster for execution. Each task processes a subset of the data and produces intermediate results that are combined to produce the final result of the action.



## Actions in Spark that can trigger the creation of a job

1. [**Count** ()](https://sparkbyexamples.com/spark/spark-dataframe-count/): This action returns the number of elements in the RDD or DataFrame. When called, it triggers the computation of all the transformations leading up to the final count and creates a job to execute the computation.
2. [**Collect** ()](https://sparkbyexamples.com/spark/spark-dataframe-collect/): This action returns all the elements of the RDD or DataFrame as an array to the driver program. When collect () is called, it triggers the computation of all the transformations leading up to the final collection and creates a job to execute the computation.
3. **SaveAsTextFile** (): This action saves the contents of the RDD as a text file to a specified location. When called, it triggers the computation of all the transformations leading up to the final output and creates a job to execute the computation and write the output to the specified location.
4. [**Reduce** ()](https://sparkbyexamples.com/spark/spark-rdd-reduce-function-example/): This action applies a binary operator to the elements of the RDD and returns the result. When called, it triggers the computation of all the transformations leading up to the final reduce operation and creates a job to execute the computation.
5. [**Foreach** ()](https://sparkbyexamples.com/spark/spark-foreach-usage-with-examples/): This action applies a function to each element of the RDD or DataFrame, such as writing it to a database or sending it over the network. When called, it triggers the computation of all the transformations leading up to the final Foreach operation and creates a job to execute the computation.



# Spark Stage

In the context of Apache Spark, a stage is a unit of parallelism in a Spark job. It represents a set of tasks that can be executed together as part of a single job.

**Spark Executor** is a process that runs on a worker node in a Spark cluster and is responsible for executing tasks assigned to it by the Spark driver program.

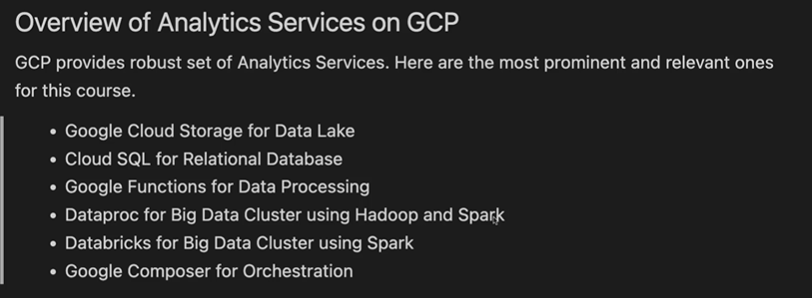
* Executes tasks on worker nodes as directed by the Driver.
* Multiple Executors run concurrently in a Spark application.
* Created when a Spark Context is created and runs until the application is terminated.
* Manages its own memory and executes individual tasks assigned by the Driver.
* Communicates with the Driver for task assignments and reports task status.

# Apache Spark Driver?

* Manages the overall execution of a Spark application.
* There is only one Driver per Spark application.
* Responsible for coordinating tasks, scheduling, and interacting with the Cluster Manager.
* Initiates Spark Context, which represents a connection to a Spark cluster.
* Monitors the execution progress and ensures fault tolerance.

**DAG** (Directed Acyclic Graph) in Spark/PySpark is a fundamental concept that plays a crucial role in the Spark execution model. The DAG is “directed” because the operations are executed in a specific order and “acyclic because there are no loops or cycles in the execution plan. This means that each stage depends on the completion of the previous stage, and each task within a stage can run independently of the other.

* The DAG plays a critical role in this process by providing a logical execution plan for the job.
* The DAG breaks the job down into a sequence of stages, where each stage represents a group of tasks that can be executed independently of each other. The tasks within each stage can be executed in parallel across the machines.
* The DAG allows Spark to perform various optimizations, such as pipelining, task reordering, and pruning unnecessary operations, to improve the efficiency of the job execution.
* By breaking down the job into smaller stages and tasks, Spark can execute them in parallel and distribute them across a cluster of machines for faster processing.
* Google Cloud
* Google cloud cli to interact with windows to Google cloud
* **Commands**= **gcloud** **init** = to initialize the cloud in sdk
* To give the project id i.e. existing one or new one
* **gsutil list** = to get the list of buckets in Google cloud



**Different types of relational databases:**

* **RDBMS :** is typically used for **transnational** based system
  1. Typically it is used for **order management system** or application

Ex: Amazon store

* 1. **Point of sale system** are using RDBMS

Ex: D Mart

* **Data Lake:**
  1. Typically it is used for storing all the data from different sources
* **Data Warehouse:** 
  1. It is typically used for reports and dashboards i.e. getting all the data from the data warehouse
* **NoSql:**
  1. Operational stores are used these NoSql database.
  2. Large scaled e-commerce websites are used NoSQL database for storage

Ex: Products details of e-commerce websites are stored in no-sql database

* 1. It is for chats, endorsements and recommendations
* **Graph based database:**

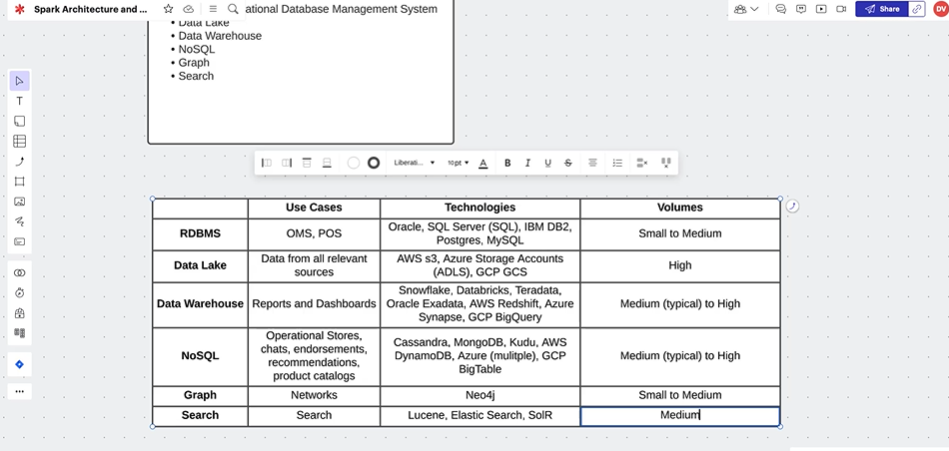
1. Facebook,linkedIn and Instagram are used graph based database to display **the friend networks**
2. Networks

* **Search based database:**

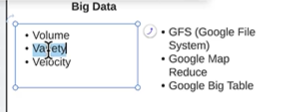
1. It is used to search for particular order or an product

**Ex**: elastic search

**Technologies for respective databases:**



**Big Data:**

****

* **If you take an example of Google search engine**

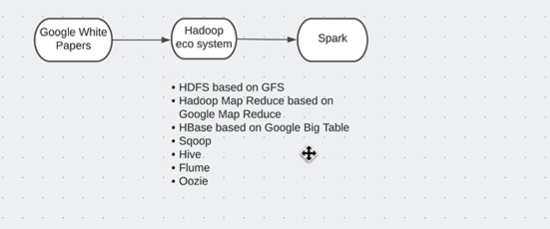
1. **Variety –** different types of data i.e. images,videos,text etc
2. **Velocity –** speed of the data flowing and needs to process the data
3. If you take the example of **Google search Engine**, the search engine is built based on the **data** that is crawled, the pages that are available in the Internet.
4. There are **Google Crawlers**, which will **crawl** all the pages that are available in the Internet.
5. The data is stored in **Google file system.** Now, they might have changed the way they are storing the data, but they used to store this data in a Google file system. So there is **huge volume** of data that is supposed to go into the **file system.**
6. That is where **volume** comes into picture. All the data that is out there as part of the pages in the internet has to be stored somewhere. It is not only supposed to be stored in Google file system, it is also supposed to process data in decent pace. That is where **velocity** comes into picture. Whatever **data** that is **crawled**, how to be stored in the **Google file system** as fast as it is possible. And also the data have to be processed as fast as possible. This data is supposed to be **indexed** and stored in **distributed indexing tables.** That is where **velocity** comes into picture. As Google Crawlers crawled the internet pages. There is no structure with respect to data. You have **images**; you have free flowing **text**, so and so forth. There is no structure as we used to see in RDBMS database tables. Hence **Variety** comes into picture as part of variety. You might have structured data, you might have semi-structured data, you might have unstructured data. Structured data means the data that is there as part of the tables.

Semi-Structured data means if you take the example of tweets and all the data is semi-structured, unstructured Means images, free flowing text doesn't have any structure.

That's where unstructured comes into picture. So when it comes to big data, the big data technologies are supposed to process all these types of Data to get us results when we try to search. So **volume**, **variety** and **velocity** are the key characteristics of the big data.

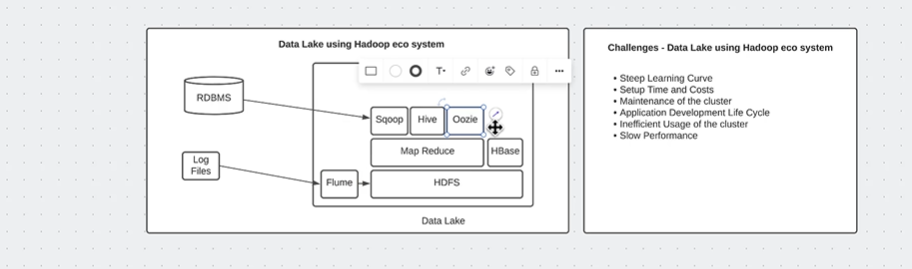
**Ecosystem of Big Data**

* Google white papers – hadoop eco system – **spark**



* Hadoop it doesn’t use memory as effectively
* It is slow and it doesn’t use cloud effectively

**Challenges:**



* The way different types of **databases** and **big data** **technologies** are related is based on **Data Lake.**
* Typically**, Data Lake** is built using big data technologies in **legacy systems.**
* The data lake used to be set up **on prem**, where a **huge cluster** used to be set up, and then there used to be **HDFS**, Hadoop, Map Reduce, HBase, Hive, Flume, Oozie, Sqoop, and etcetera. However, the trend has changed. Now it is more **cloud oriented.**
* If you want to build a data lake using Hadoop ecosystem on prem
* Let's assume the data lake is set up using the Hadoop ecosystem.
* Map Reduce is primarily used for **Batch Processing data**
* **HBASE** provides result as fast as possible

Ex: face book messenger earlier is used this technology

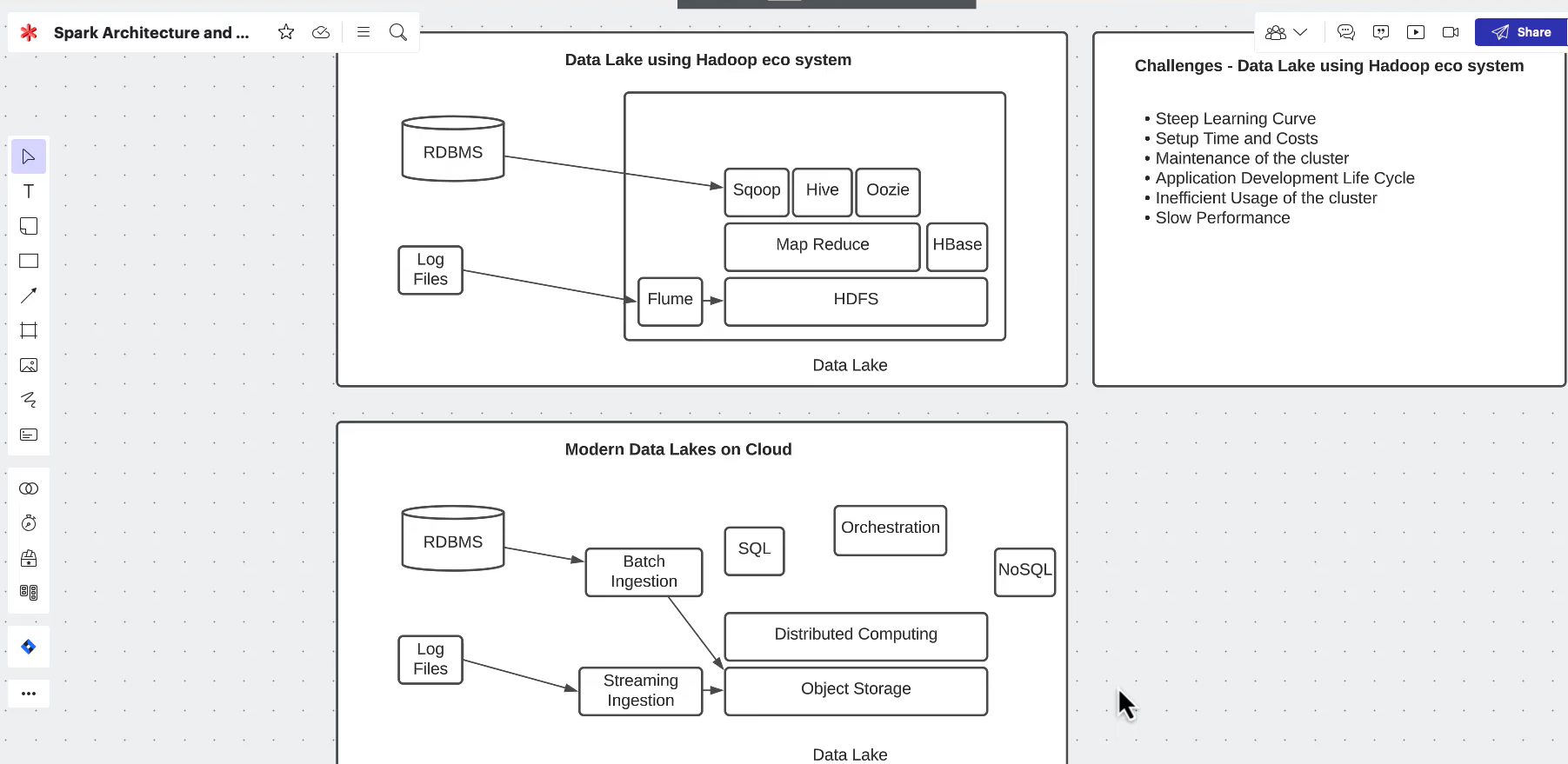
* Using **SCOOP**, you should be able to get the data from **RDBMS** **tables** into **Hive** tables.

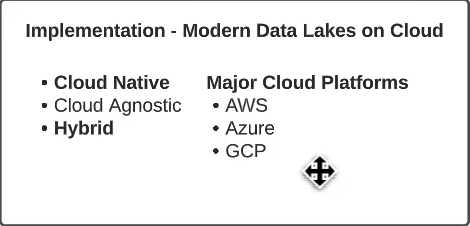
1. It is a command line tool
2. To build an **automation** for **scoops** **job** using the **shell scripts** or **python**
3. It uses Map Reduce to connect to the **source database**, get the data and store it into **HDFS** once data is stored in **HDFS**, If data is structured, people prefer to use **SQL** to process the data instead of using Java based Map Reduce.
4. **Hive** is nothing but **SQL based engine**. Hive is primarily to process the data within HDFS using SQL style syntax. It uses Map Reduce under the hood, but when it comes to development, we typically use the **SQL** style
5. **Log Files:** Keep in mind **Flume** doesn't use Map Reduce under the hood. It uses its own set of technologies or its own set of **frameworks** to read the log files. However, it connects to web server logs and it will make sure data is loaded to HDFS.
6. Now, if you wanted to get the **data** from **log files** into **HDFS**, you can actually use **Flume**
7. **Oozie** is nothing but orchestration tool using Oozie we will be able to orchestrate. Oozie is nothing but orchestration tool using Oozie we will be able to orchestrate. Scoop jobs and hive jobs so that they can run in orchestrated fashion. It is a XML based
8. HDFS is for **storage**, hive and map reduce are used for **compute**, scoop and flume are used for **ingestion**, oozie is used for **orchestration** and **HBase** is used for **NOSQL**

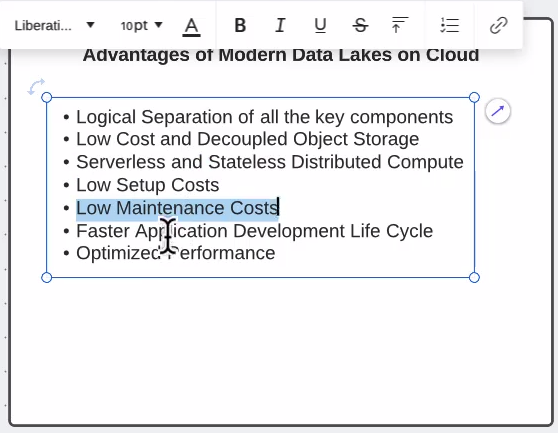
**Challenges:**

1. **Steep learning curve:** i.e. of different technologies with different languages
2. If none of the technologies solved our problem we need to use **map reduce**
3. It is totally **java** based or python script based
4. Set up time and costs
5. **Maintainenece** of the **cluster** (i.e. we need to replace the cluster if server is down)
6. Application development life cycle
7. Even with respect to **testing**, the testers need to perform both black box testing as well as white box testing.
8. **Inefficient** usage of the cluster.
9. The Map Reduce, which is a core component of Hadoop ecosystem to process the data, doesn't use **memory** and CPU efficiently. That’s why the workloads take quite a bit of time in processing, slow performance when it comes to processing the data

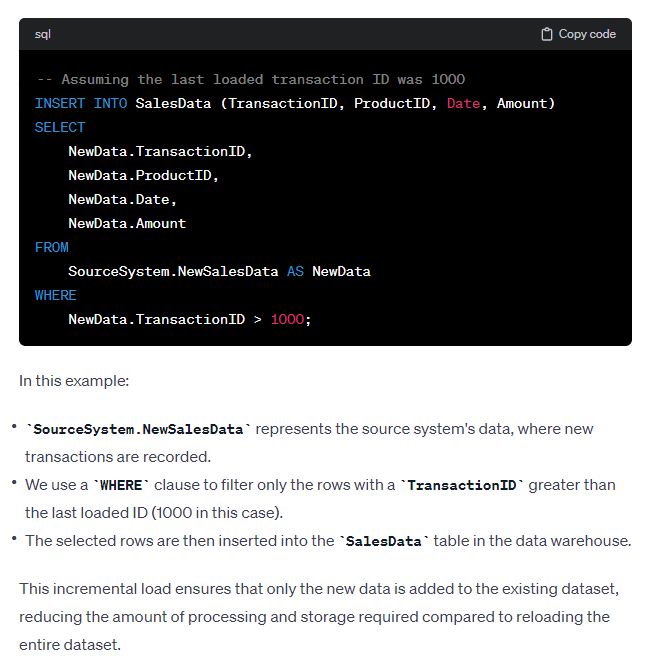
**Modern Data Lakes on cloud:**

****

****

****

1. **Full Load (Initial Load):**
   * Initially, you load the entire dataset into your data warehouse or database.
   * This process is time-consuming but necessary for the first load.
   * All rows from the source system are inserted into the **SalesData** table.
2. **Incremental Load:**
   * After the initial load, new transactions occur, and you want to update your data warehouse with only the new data since the last load.
   * Instead of reloading the entire **SalesData** table, you identify and load only the rows that have been added or modified since the last load.

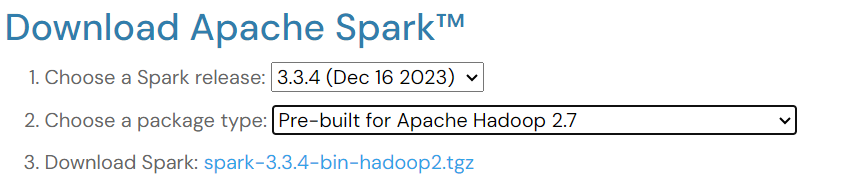
****

**Data Processing Libraries:**

1. **Python Pandas**- Pandas, by default uses only one **process** to process the data or one threads to process the data fashion.
2. **Python dask** - Dask, it will automatically use the full capacity of the system to process the data in distributed fashion.
3. **PySpark** - is a popular framework

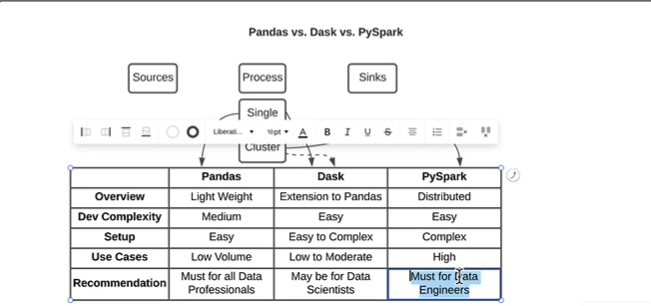
* **Set up pyspark in local environment**

1. Apache Spark Installation links: 1.
2. Download JDK: [https://www.oracle.com/in/java/techno...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbTNna3NjSkptd1BEQ3V6T08tTk03Z0MyMlFfUXxBQ3Jtc0tuRjdGTnBZMER3SjA4Sjh6MVdxUms0Y0JxVjllRHRTODdoUHpTREVDQzBCTWtoUjZPWVNOQTVWeW12bzUtWGdVX3ZvZXVJbXZLb0ttanJIR0VrM1J4SC1saF9BYjdZZDRKVGcyd2pFeU82UU1yaUpkbw&q=https%3A%2F%2Fwww.oracle.com%2Fin%2Fjava%2Ftechnologies%2Fdownloads%2F%23jdk19-windows&v=OmcSTQVkrvo) (pyspark supports jdk 8 and 11)
3. Download Python: [https://www.python.org/downloads/](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbmtueTJ1cXNHaG1NX3Z0SUxiOGNTLXVmOFpvQXxBQ3Jtc0tsb0pDVTVaTjZvazNucndyV2h6aTIxMldla2M2YkZxd2ZHYnpVRDg4LXY5dUtqdk12czdHR3R5ekl0TWVCVnZsM2lGXzlTMjZ3emdVTHE0ZWVWUFFBME1iVHM5ZWM0T3hNcXE4dTQ1SWZkNUtteHkzaw&q=https%3A%2F%2Fwww.python.org%2Fdownloads%2F&v=OmcSTQVkrvo)
4. Download Spark: [https://spark.apache.org/downloads.html](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbVExdUYzM1JoWXdiaEdJRGcxQ0hMT25sNjJRQXxBQ3Jtc0trbVh5WHJ3NkUzMHlxdEhjMi01Q3lQUFRxOVFpTEM4TFZXM016UGFCSlJLZ0FHUkx0N0ZxUWZvNjB5T2RCdUg1TXNuUWV5VUozVEJOQ2paR0YteF95YmVUTkx3ZmVKTTU3bWRKbHFGZ3VvX1lEVEF4Yw&q=https%3A%2F%2Fspark.apache.org%2Fdownloads.html&v=OmcSTQVkrvo) (
5. Winutils repo link: [https://github.com/steveloughran/winu...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbkhpTDVjSnoyLU9lQmJLRWVpR05GMk5JZE81QXxBQ3Jtc0tsT1k3Y1otaVhpcnVFMnNQTWt2OXEzT01EUTFpYzVvVDNsUWRRRDJQd1RZa3g5TzY1bjl4NUtLNGJrQ0lKMGQ3SjJGNzVZbjhIWno2QlZxN3pfZlhnb0JYMExCMXZ6UGw1b21SS0Y4dFB3aDBxNklVcw&q=https%3A%2F%2Fgithub.com%2Fsteveloughran%2Fwinutils&v=OmcSTQVkrvo)

****

**>** import pyspark

> print (pyspark.version)



* We need to set up the user variables and path

**Distributed Computing System:**

* Most popular one are hadoop and spark
* Cores and memory are important key terms = together called as cluster
* **Spark** has embedded in the **aws emr**
* **DataProc** is the popular in **GCP**
* **Data bricks** is available in all there cloud platforms
* **Snowflake** is a very popular, cloud based data warehouse.
* When it comes to **Spark**, it will be deployed on **multiple** **nodes** in a **cluster** it is called as a **spark cluster**.

**There are three categories of servers when it comes to Spark:**

1. **Masters -**are used to manage the **clusters .**It will just verify the worker nodes are up and running or not
2. **Drivers** - are used to submit the **spark** **application**
3. **Workers –**Workers are the ones which will actually take care of **processing the data** by running applications on them.
4. **Cluster manager** – external services for acquiring resources on the cluster

**These are four different cluster manager types that can be used to manage the clusters.**

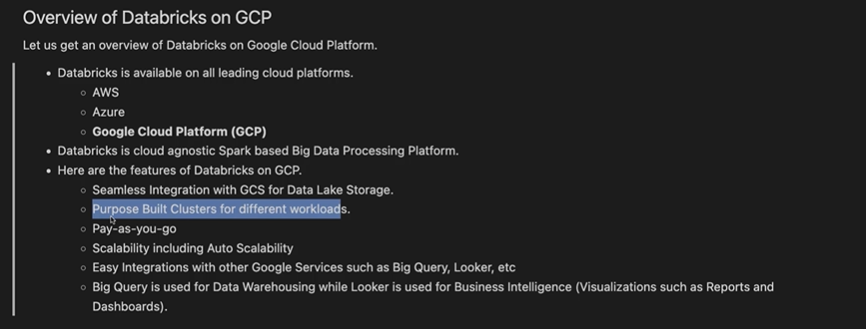
* Spark Native or Stand Alone - widely used as part of **Data bricks** platform.
* YARN – aws emr or Gcp
* Kubernetes – set up cluster in kubernetics pos
* ~~Mesos =not using now~~

**Components on pyspark:**

* Keep in mind that **executors** are nothing but **jvms**.
* JVM stands for Java Virtual Machine.
* All these are nothing but jvms on the servers where Spark runtime is set up, each JVM will have **slots**.
* A **slot** is nothing but unused capacity of a given node.
* Out of these 12 slots, four of them will be converted to tasks.
* They'll process the data.
* Once data is processed, the tasks will again become slots.

**Pyspark:**

* Provide the necessary permission through the **IAM** roles
* Create a new **workspace**
* Create a new **cluster**
* Once the **cluster** is create , we need to create an **notebook** to work on the created cluster



* To access the files in **dbfs** – we have a magic commands
* It is used to interact with file system and many other by eliminating the python code
* When it comes to **databricks** we have the file system , cluster and where we deploy the spark jobs using notebooks
* python –m venv **my\_env**
* python –version
* pip install databricks-cli
* pip list
* databricks --Version
* databricks --help

To Configure Databricks for GCP: -

* databricks configure --host <http://---.com> --token --profile Profile-Name
* databricks fs --profile Profile-name
* we need to create a token in the workspace
* databricks clusters list –profile name
* databricks clusters start --cluster-id number --profile Profile\_Name
* databricks clusters delete --cluster-id number --profile Profile\_Name = delete the cluster

To Configure Databricks in local: -

* databricks configure - to connect the databricks - type databricks host – upto .com ,user name – mail id and password of the login
* databricks fs ls - to get list of files
* git clone “path”
* databricks clusters list - to get list of clusters
* cat $HOME/.databrickscfg - to get the profile details
* databricks fs ls --profile DEFAULT - to list files in a directory
* databricks fs cp source dbfs:/target\_location --recursive –profile “name”
* databricks fs ls dbfs:/path --profile “name”

**Process the data:-**

* To use the **sql** we need to use the %sql and to use the python we need to change to python
* We need to load the data from local to distributed file system
* Next we need to create a temp view based on the file
* To select the file =

%sql

Select \* from TEXT. `Dbfs: /\_\_\_location\_\_\_` = here we need to use the tilde ``

* To Create a temp view syntax :

%sql

Create or Replace Temporary view view\_name (

) USING CSV

Options (path = ‘’, Sep = ‘’)

* To write the select data into the parquet format in dbfs

%sql

INSERT OVERWRITE DIRECTORY‘’

Using parquet

Select query

* Spark.read.text(`dbfs location`,wholetext = True).show(truncate = False)
* **Wholetext** = True means it will consider entire content as one record
* **Truncate =** False means entire content displayed as the output