WOMEN WHO CODE

Building a Data Engineering Portfolio Project - Part 2

Anju Mercian

Data Engineering Consultant
www.anjumercian.me

August 2022

About Anju Mercian

- Graduated from Syracuse University, NY with a Master's degree in Computer Engineering
- Worked at Intel for 5 years as an Infrastructure Engineer
- Worked at VMware for one year as a Project Manager
- Worked at Intuitive surgical for one year as a Business System Analyst
- Worked at Omdena as a Platform Engineer
- Currently working as a Data Engineering Consultant @ Meta

Agenda

- Recap
- Data Quality
- Data Pipelines
- Data Partitioning
- Backfilling
- Apache Airflow
- Apache Airflow UI
- Q&A

Scope the Project and Gather Data

***GOAL**:* This project aims to build a data analytics table using the crime data, enriching the data with the housing dataset and the climate data by state for data analysis.

This is built for a data analytics table to help answer questions such as:

- How is the crime rate in a particular area based on weather?
  - Are there more crimes in summer or winter?
- Crime rate and property data
- Property rates and weather
  - Are property rates higher in colder climates or warmer climates?

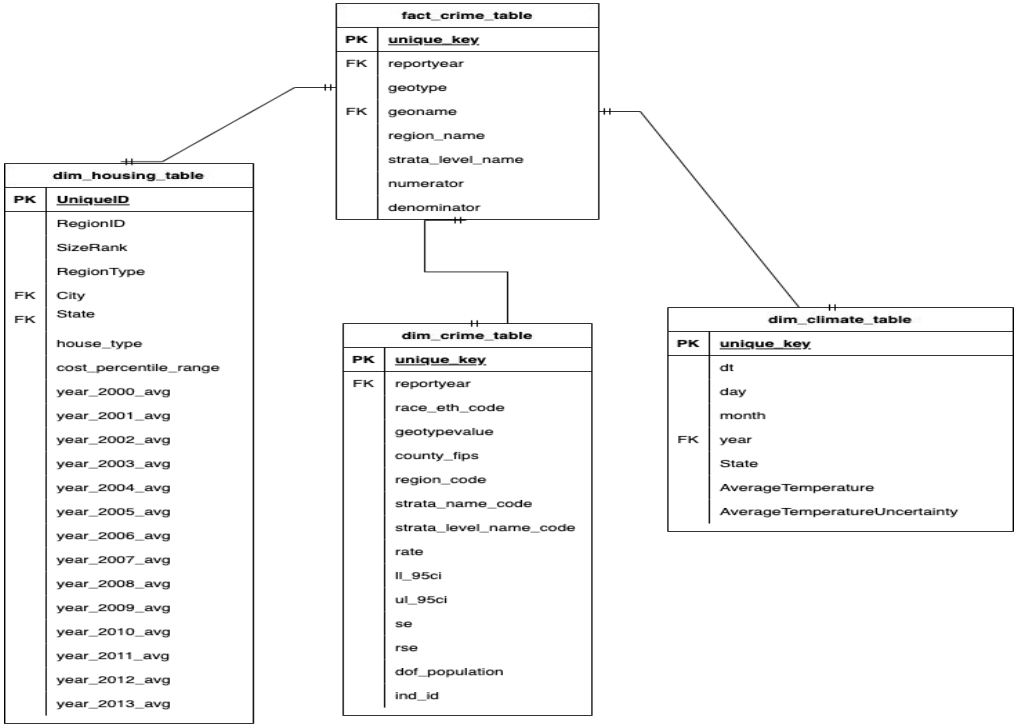
Explore and Access the Data

Data Sources:

- Crime Data: This data comes from the data.world, data about violent crimes. A data dictionary is included in the excel.
- Housing Data: This dataset came from Zillow.
- Climate Data: This data comes from data world.

***Note**: Try choosing non-Kaggle data sources because Kaggle is one of the most used data sources and is relatively clean.*

Define the Data Model



Source

Data Quality

“Data quality is the answer to the question “How is my data?” If your data helps you with business operations and decisions, then you can say that your data is of good quality.”
Data is generally considered high quality if it is “fit for [its] intended uses in operations, decision making and planning”.

- Data must be a certain size.
- Data must be accurate to some margin of error.
- Data must arrive within a given timeframe from the start of execution.
- Pipelines must run on a particular schedule.
- Data must not contain any sensitive information.

Data Quality Checks

- Primary Key distinct; No duplicates.
- Foreign Keys are not nulls.
- Rows are all intact when data is moved.
- Not too many nulls, nulls to be filled in.
- Data must not contain sensitive information.

Data Validation: Makes sure the data is present, correct and meaningful.

Data Quality is the measure of how well a dataset satisfies its intended use.

Resource

Data Pipelines

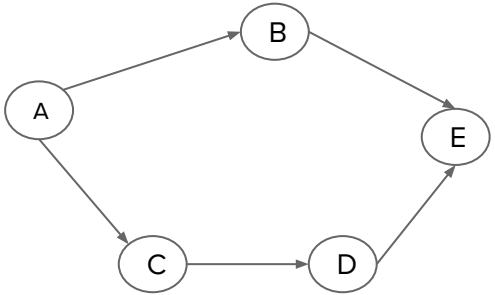
What is a data pipeline?

A series of steps in which data is processed.

- Scheduled triggers and external triggers.

Data Pipelines

- There can also be an ELT (Extract, Load and Transform).
- Data pipelines are well performed as DAG's.
- The conceptual framework of data pipelines will help to better organize and execute everyday DE tasks.
- DAG's are Directed Acyclic Graph: No cycles and are directed.



Data Partitioning

- Pipeline data partitioning is the process of isolating data to be analyzed by one or more attributes, such as time, logical type or data size.
- Data Partitioning leads to faster and more reliable pipelines.
- Three types:
  - Schedule Partitioning: Based on Schedule
  - Logical Partitioning: Based on Conceptual Data. Unrelated separate processing.
  - Size Partitioning: Based on data size. Airflow has limited memory.

Data Partitioning

- Pipelines designed to work with partitioned data fail more gracefully.
- Partitioning makes debugging and rerunning failed tasks much simpler.
- It also enables easier redos of work, reducing cost and time.
- If data is partitioned appropriately, your tasks will have fewer dependencies on each other.
- Time sensitive data partitioning by datetime stamp is ideal.

Data Lineage

The data lineage of a dataset describes the discrete steps involved in the creation, movement, and calculation of that dataset.

Why is data lineage important?

1. **Instilling Confidence:** Being able to describe the data lineage of a particular dataset or analysis will build confidence in data consumers (engineers, analysts, data scientists, etc.) that our data pipeline is creating meaningful results using the correct datasets. If the data lineage is unclear, it is less likely that the data consumers will trust or use the data.
2. **Defining Metrics:** Another major benefit of surfacing data lineage is that it allows everyone in the organization to agree on the definition of how a particular metric is calculated.
3. **Debugging:** Data lineage helps data engineers track down the root of errors when they occur. If each step of the data movement and transformation process is well described, it's easy to find problems when they occur.

Source: Udacity

Backfilling

- A DAG Run is an object representing an instantiation of the DAG in time. Any time the DAG is executed, a DAG Run is created and all tasks inside it are executed.
- A DAG with a start_date, possibly an end_date, and a schedule_interval defines a series of intervals which the scheduler turns into individual DAG Runs and executes. The scheduler, by default, will kick off a DAG Run for any data interval that has not been run since the last data interval (or has been cleared). This concept is called Catchup.
- There can be the case when you may want to run the DAG for a specified historical period. E.g., A data filling DAG is created with start_date 2019-11-21, but another user requires the output data from one month ago (i.e., 2019-10-21). This process is known as Backfill.
- You may want to backfill the data even in the cases when catchup is disabled.

Source

Apache Airflow

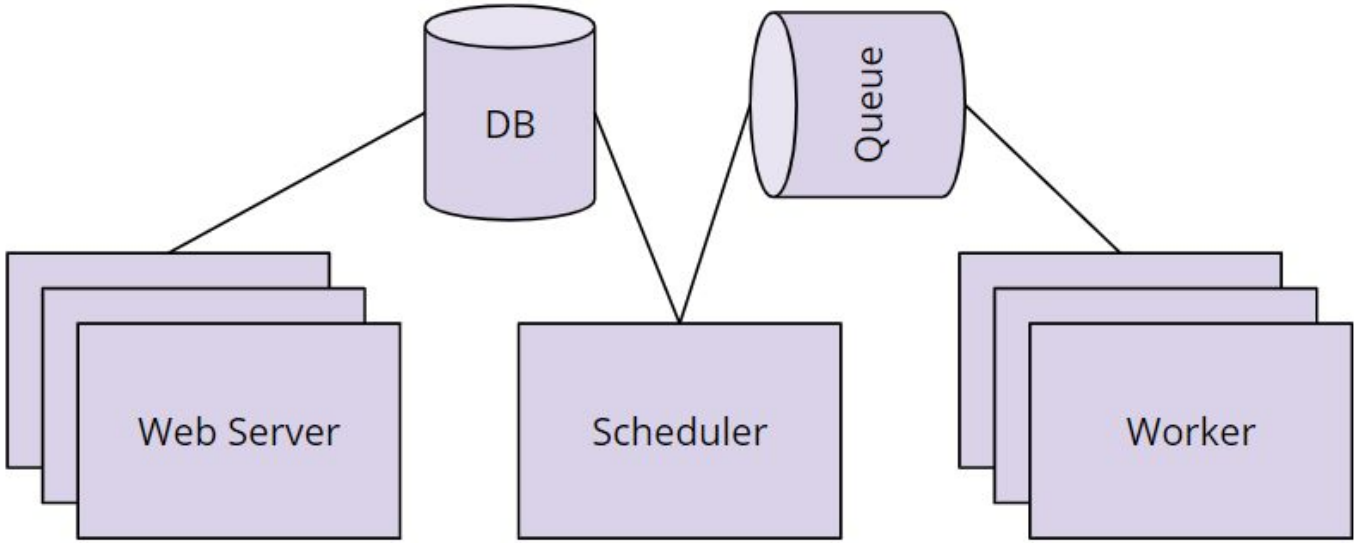
Apache Airflow

- Apache Airflow is an open source tool created by AirBnB in 2015.
- DAG based Schedulable data pipelines that can run in mission control environments.
- Written in Python
- Airflow is not a data streaming solution. It is more or less like a config file.
- Tasks do not move data from one to the other (though tasks can exchange metadata).
- Workflows are expected to be mostly static or slowly changing.
- Airflow workflows are expected to look similar from a run to the next. This allows for clarity around unit of work and continuity.

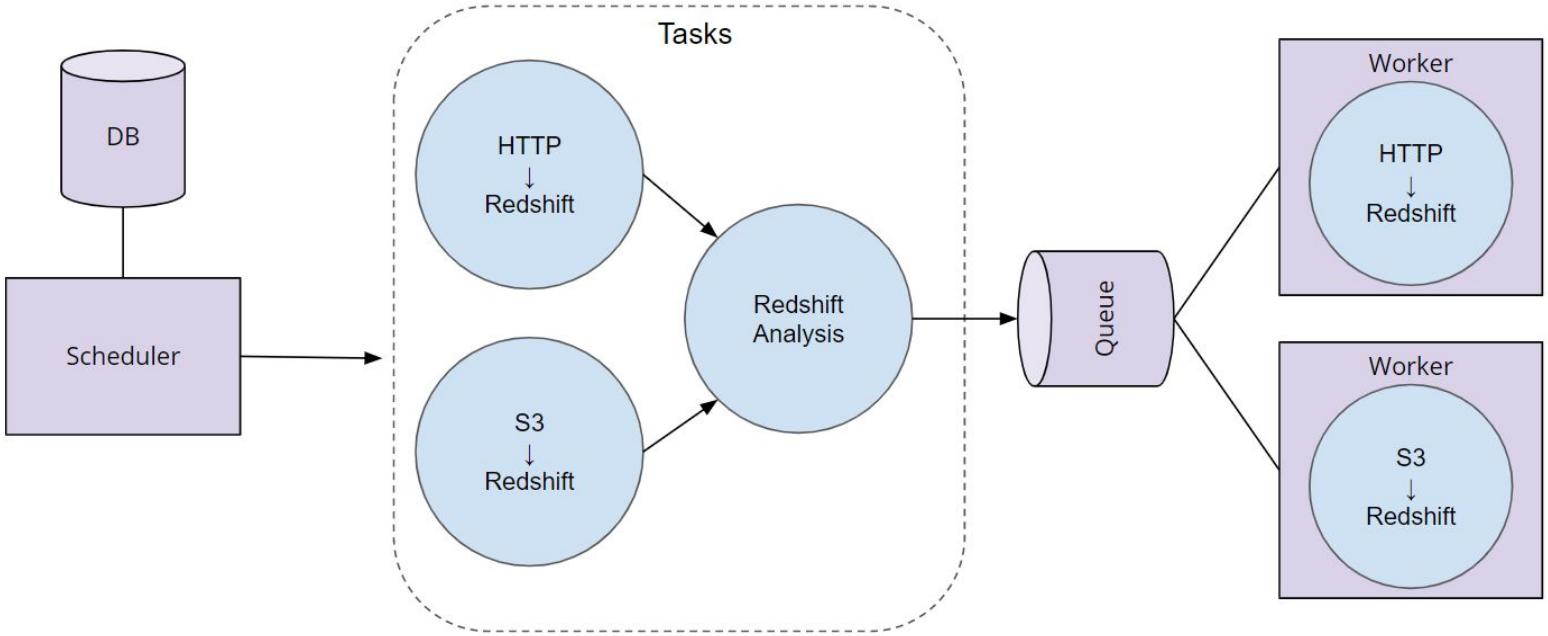
Apache Airflow (DAG)

Apache Airflow: “Airflow is a platform to programmatically author, schedule and monitor workflows. Use Airflow to author workflows as directed acyclic graphs (DAGs) of tasks. The Airflow scheduler executes your tasks on an array of workers while following the specified dependencies. Rich command line utilities make performing complex surgeries on DAGs a snap. The rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed. When workflows are defined as code, they become more maintainable, versionable, testable, and collaborative."

Components of Airflow



How Does Airflow Work?



Order of Operations

- The Airflow Scheduler starts DAGs based on time or external triggers.
- Once a DAG is started, the Scheduler looks at the steps within the DAG and determines which steps can run by looking at their dependencies.
- The Scheduler places runnable steps in the queue.
- Workers pick up those tasks and run them.
- Once the worker has finished running the step, the final status of the task is recorded, and additional tasks are placed by the scheduler until all tasks are complete.
- Once all tasks have been completed, the DAG is complete.

Airflow Operators

- Operators are the building blocks of Airflow DAGs. They contain the logic of how data is processed in a pipeline. Each task in a DAG is defined by instantiating an operator.
- There are many different types of operators available in Airflow. Some operators execute general code provided by the user, like a Python function, while other operators perform very specific actions, such as transferring data from one system to another.
- Operators are Python classes that encapsulate logic to do a unit of work.
- They are like wrappers.

Resource

Airflow Operators

- PythonOperator
- PostgresOperator
- RedshiftToS3Operator
- S3ToRedshiftOperator
- BashOperator
- SimpleHttpOperator
- Sensor
- etc

Connections

- Data pipelines interact with many data sources, Airflow DAG's have to connection with variables that change
- Connections can be accessed via airflow hooks

```
from airflow import DAG
from airflow.hooks.postgres_hook import PostgresHook
from airflow.operators.python_operator import PythonOperator

def load():
  # Create a PostgresHook option using the `demo` connection
  db_hook = PostgresHook('demo')
  df = db_hook.get_pandas_df('SELECT * FROM rides')
  print(f'Successfully used PostgresHook to return {len(df)} records')

load_task = PythonOperator(task_id='load', python_callable=hello_world, ...)
```

Source

Airflow Hooks

A Hook is a high-level interface to an external platform that lets you quickly and easily talk to them without having to write low-level code that hits their API or uses special libraries. They're also often the building blocks that Operators are built out of. They integrate with Connections to gather credentials, and many have a default conn_id; for example, the PostgresHook automatically looks for the Connection with a conn_id of postgres_default if you don't pass one in.

- HTTP Hook
- PostgresHook
- MYSQLHook
- SlackHook
- PrestoHook

Source

```
import datetime
import logging
from airflow import DAG
from airflow.contrib.hooks.aws_hook import AwsHook
from airflow.hooks.postgres_hook import PostgresHook
from airflow.operators.postgres_operator import PostgresOperator
from airflow.operators.python_operator import PythonOperator
import sql_statements
def load_data_to_redshift(*args, **kwargs):
    aws_hook = AwsHook("aws_credentials")
    credentials = aws_hook.get_credentials()
    redshift_hook = PostgresHook("redshift")
    redshift_hook.run(sql.COPY_ALL_TRIPS_SQL.format(credentials.access_key, credentials.secret_key))
dag = DAG(
    'lesson1.exercise6',
    start_date=datetime.datetime.now()
)
create_table = PostgresOperator(
    task_id="create_table",
    dag=dag,
    postgres_conn_id="redshift",
    sql=sql_statements.CREATE_TRIPS_TABLE_SQL
)

copy_task = PythonOperator(
    task_id='load_from_s3_to_redshift',
    dag=dag,
    python_callable=load_data_to_redshift
)

location_traffic_task = PostgresOperator(
    task_id="calculate_location_traffic",
    dag=dag,
    postgres_conn_id="redshift",
    sql=sql_statements.LOCATION_TRAFFIC_SQL
)

create_table >> copy_task
copy_task >> location_traffic_task
```

References

More Details

Apache Airflow

Apache Kafka

Apache Spark, Resources: Link

Sign Up Future Workshops in WWCode Python Track

Additional Sources

Q & A

WOMEN WHO CODE®

Backup

GitHub link

Agenda

- About Me
- Portfolio Projects
- Data Engineering (DE)
- Technical Skills
- Project Steps
- Spark
- Code walkthrough
- Q&A

What Is a Portfolio Project?

"A portfolio project or a capstone project is a well thought out and designed project that showcases your skills to a potential employer or the world."

Benefits of a Portfolio Project

- *Showcase your skills*
- *Learn new skills*
- *Grow it as a side project after you have a job*
- *Build personal brand*
- *Help with Data Design Interviews*

What Is Data Engineering?

“Data engineering is the complex task of making raw data usable to data scientists and groups within an organization. Data engineering encompasses numerous specialties of data science.

In addition to making data accessible, data engineers create raw data analyses to provide predictive models and show trends for the short- and long-term. Without data engineering, it would be impossible to make sense of the huge amounts of data that are available to businesses.”

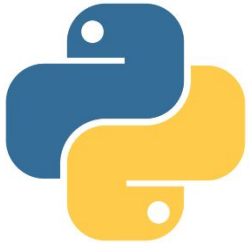
Source

Who Is a Data Engineer?

“Data engineers work in a variety of settings to build systems that collect, manage, and convert raw data into usable information for data scientists and business analysts to interpret. Their ultimate goal is to make data accessible so that organizations can use it to evaluate and optimize their performance.”

Source

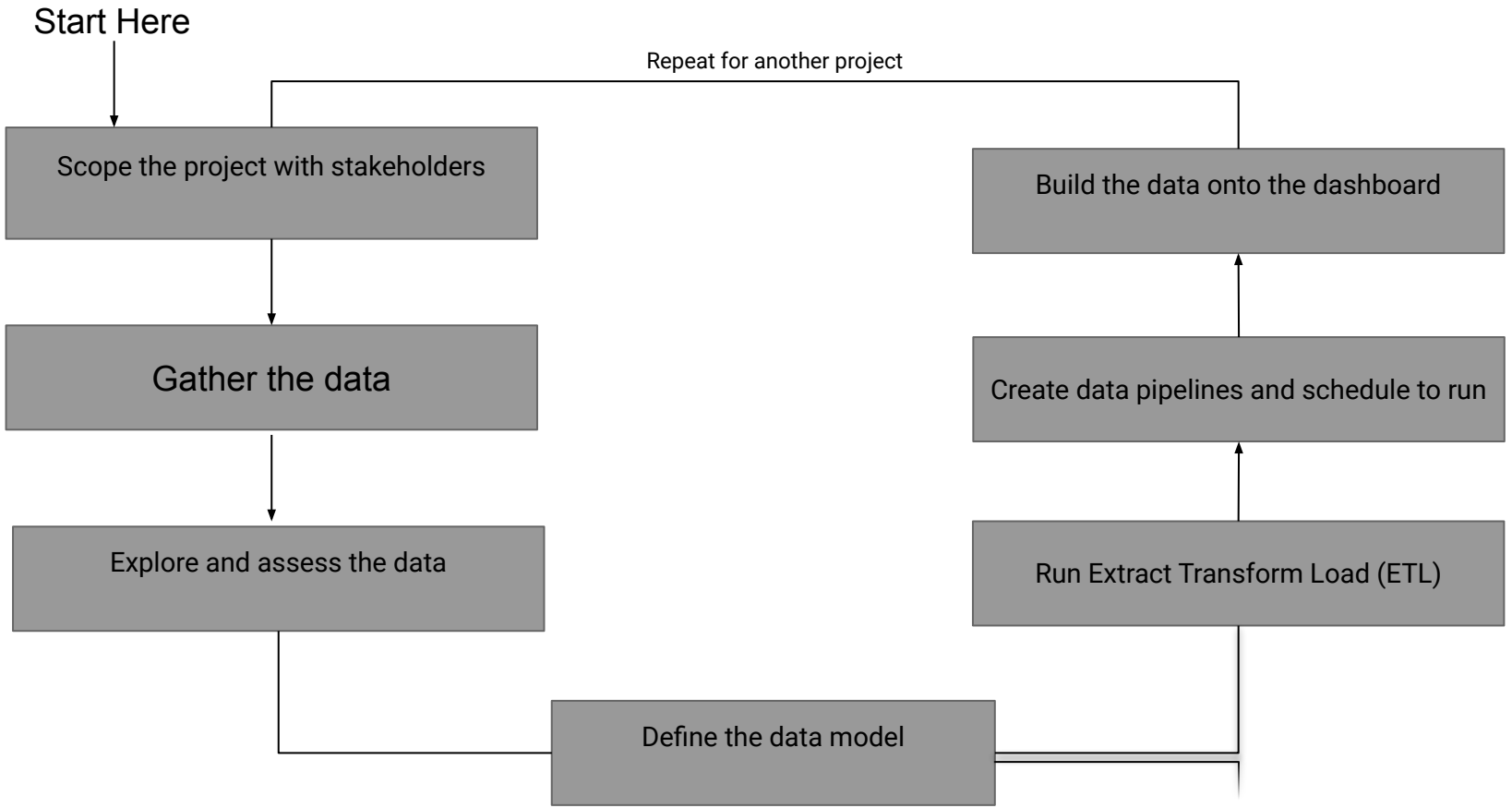
Technical Skills



EMR
RedShift
S3

Project Steps to Guide You

GitHub link


Start Here
Repeat for another project
Scope the project with stakeholders
Gather the data
Explore and assess the data
Define the data model
Run Extract Transform Load (ETL)
Create data pipelines and schedule to run
Build the data onto the dashboard

Scope the Project and Gather Data

***GOAL**:* This project aims to build a data analytics table using the crime data, enriching the data with the housing dataset and the climate data by state for data analysis.

This is built for a data analytics table to help answer questions such as:

- How is the crime rate in a particular area based on weather?
  - Are there more crimes in summer or winter?
- Crime rate and property data
- Property rates and weather
  - Are property rates higher in colder climates or warmer climates?

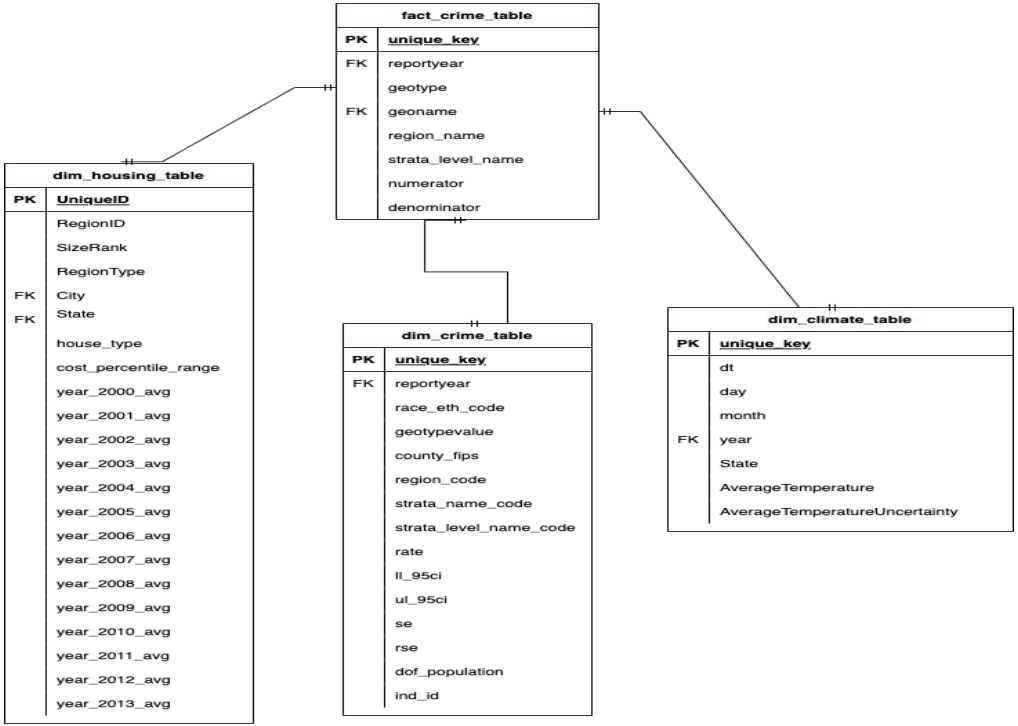
Explore and Access the Data

Data Sources:

- Crime Data: This data comes from the data.world, data about violent crimes. A data dictionary is included in the excel.
- Housing Data: This dataset came from Zillow.
- Climate Data: This data comes from data world.

***Note**: Try choosing non-Kaggle data sources because Kaggle is one of the most used data sources and is relatively clean.*

Define the Data Model



Source

Run ETL to Model the Data

Modularized into functions the cleaning steps and creation of the fact and dimension tables. Data quality checks happened at this stage. Created the data dictionary.

***Note**: Data dictionaries and modularizing the pipelines are very important and useful for portability of code.*

Deploy

Deploy on GitHub or a dashboard.

Apache Spark

Big Data

What is Big Data?

- When the data processing cannot be completed by a single machine. For example, if your system capacity is 8Gb and you are trying to process a file >8Gb it will be termed as big data.
- In order to process the data, data has to move in and out of memory. The process is called thrashing. If it takes too long for thrashing when using a single machine, then it is considered big data.

Spark is designed to optimize memory and minimize i/o using network.

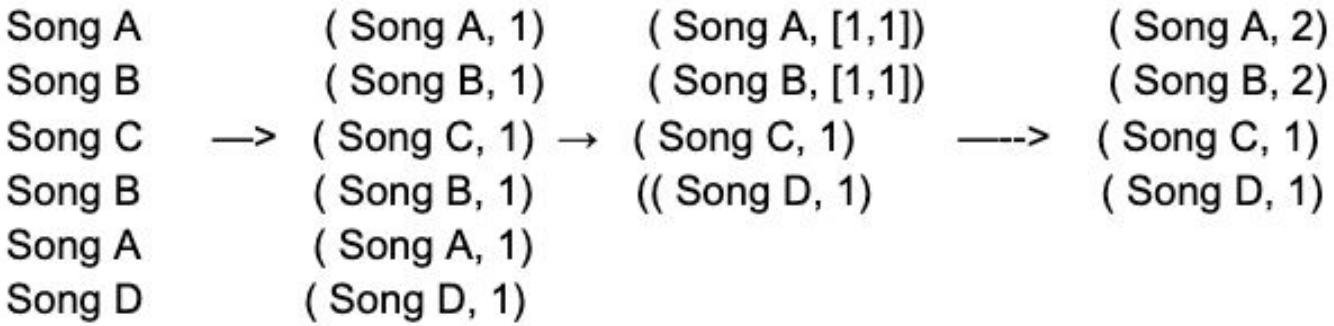
Numbers everyone should know

- **Memory** - short term storage - 100 ns
- **SSD** - long term storage - 16 micro second
- **Network** - 150ms
- **CPU** - 0.4ns
- Fastest processing is the CPU.
- Slowest processing is the network.

Apache Spark Introduction

- Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets. It can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two qualities are key to the worlds of big data and machine learning, which require the marshalling of massive computing power to crunch through large data stores.
- Founded by Matei Zaharia
- Started as a project in AMPLab at U.C. Berkeley in 2009 as a better way as compared to MapReduce.

MapReduce


Song A
Song B
Song C
Song B
Song A
Song D
—>
(Song A, 1)
(Song B, 1)
(Song C, 1)
(Song B, 1)
(Song A, 1)
(Song D, 1)
→
(Song A, [1,1])
(Song B, [1,1])
(Song C, 1)
((Song D, 1)
—-->
(Song A, 2)
(Song B, 2)
(Song C, 1)
(Song D, 1)

Apache Spark Introduction contd..

- Spark is currently one of the most popular tools for big data analytics. There are others like Hadoop, but Hadoop is an older technology.
- Spark is faster than Hadoop, and it is developer friendly.

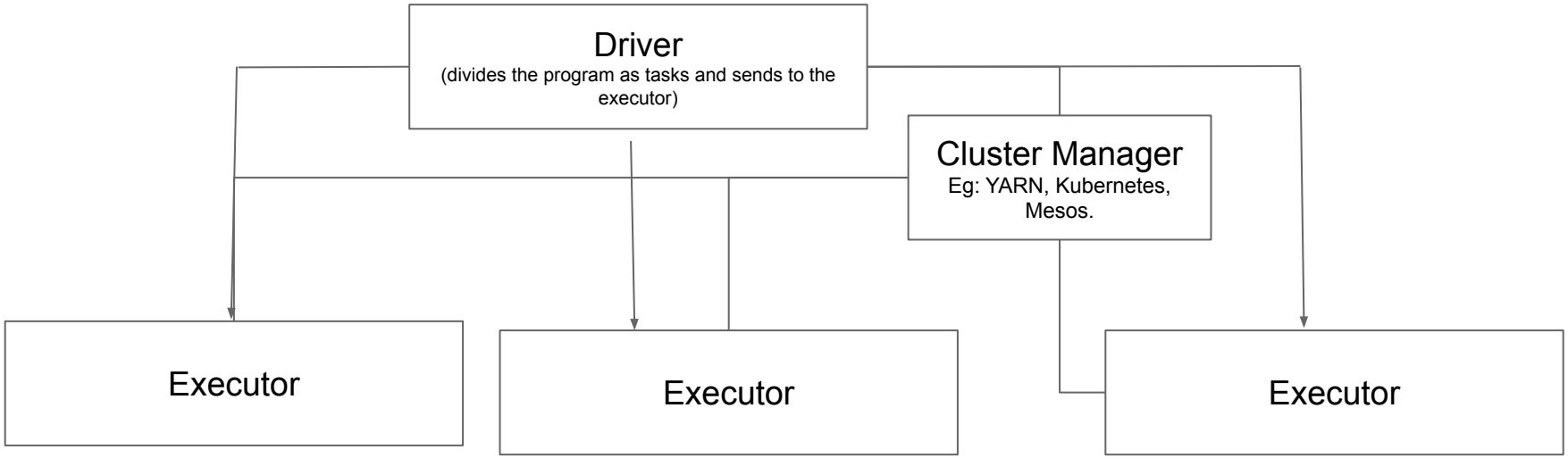
```
#from pyspark.sql import SparkSession

#spark = SparkSession.builder.\
#config("spark.jars.repositories", "https://repos.spark-packages.org/").\
#config("spark.jars.packages", "saurfang:spark-sas7bdat:2.0.0-s_2.11").\
#enableHiveSupport().getOrCreate()

#df_spark = spark.read.format('com.github.saurfang.sas.spark').load('../../data/18-83510-I94-Data-2016/i94_apr16_sub.sas7bdat')
```

```
#write to parquet
#df_spark.write.parquet("sas_data")
#df_spark=spark.read.parquet("sas_data")
```

Apache Architecture



- Can be run in two modes ‘local’ and ‘cluster’

Spark vs MapReduce

- The Hadoop ecosystem is a slightly older technology than the Spark ecosystem.
- Spark's in memory engine is faster processing time.
    - Hadoop MapReduce is slower than Spark because Hadoop writes data out to disk during intermediate steps. But many companies use Hadoop ecosystem. Spark can be used in conjunction with Hadoop.
- While Spark is great for iterative algorithms, there is not much of a performance boost over Hadoop MapReduce when doing simple counting. Migrating legacy code to Spark, especially on hundreds of nodes that are already in production, might not be worth the cost for the small performance boost.
- MapReduce has 2 stage execution Map and Reduce. Spark uses DAG has multiple stages making distributing easier.
- Spark is developer friendly.

Data Wrangling in Spark

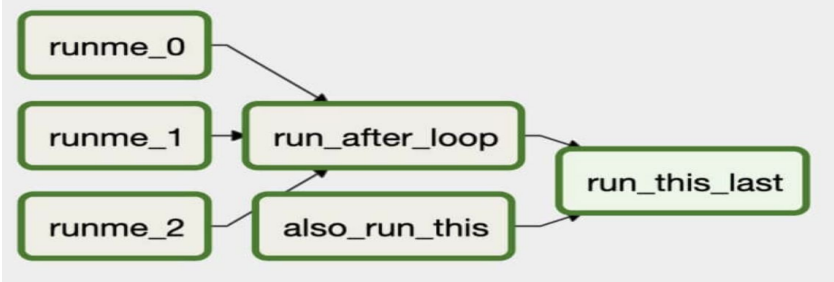
- Built with Scala and Java
- Uses functional programming
  - Perfect for distributed system
  - Uses Pure Functions: F(x) = x+5; x=3; F(3)=8
  - Python function:
    - w=-1
    - Def func(x):
      - Global x
      - Return x+w+5
- PySpark was built with functional programming in mind.

Functional vs. Procedural

- Functional is:
  - Uniform
  - No unintended side effects
  - i/p data not altered
  - All function must be pure
- Procedural is:
  - Not Uniform
  - i/p can be altered
  - Functions can have side effects

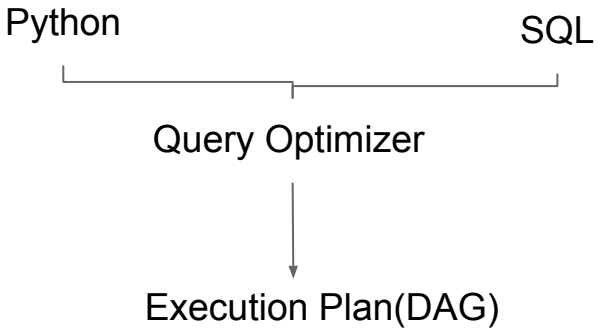
Spark Data Processing

- Uses Lazy Evaluation
- Directed Acyclic Graph (DAG) is built to help with processing



- Spark first copies the data in order not to change the original input.
- Spark uses both imperative(Python)[How - Result oriented] and declarative(SQL)[What - Result oriented]

Spark Data Processing



- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

Spark Commands

- **select()**: returns a new DataFrame with the selected columns
- **filter()**: filters rows using the given condition
- **where()**: is just an alias for filter()
- **groupBy()**: groups the DataFrame using the specified columns, so we can run aggregation on them
- **sort()**: returns a new DataFrame sorted by the specified column(s). By default the second parameter 'ascending' is True.
- **dropDuplicates():** returns a new DataFrame with unique rows based on all or just a subset of columns
- **withColumn()**: returns a new DataFrame by adding a column or replacing the existing column that has the same name. The first parameter is the name of the new column, the second is an expression of how to compute it.
- **Aggregate functions**
    - Spark SQL provides built-in methods for the most common aggregations such as count(), countDistinct(), avg(), max(), min(), etc. in the pyspark.sql.functions module.

Spark Commands contd..

- **User defined functions (UDF)**
  - In Spark SQL we can define our own functions with the udf method from the pyspark.sql.functions module. The default type of the returned variable for UDFs is string. If we would like to return an other type we need to explicitly do so by using the different types from the pyspark.sql.types module.
- **Window functions**
  - Window functions are a way of combining the values of ranges of rows in a DataFrame. When defining the window we can choose how to sort and group (with the partitionBy method) the rows and how wide of a window we'd like to use (described by rangeBetween or rowsBetween).

[Source]

Spark Use Cases

- Data Analytics
- Machine Learning
- Streaming
- Graph Analytics
- Apache Spark comes with the ability to run multiple workloads, including interactive queries, real-time analytics. One application can combine multiple workloads seamlessly.
- Spark is not a data storage system, and there are a number of tools besides Spark that can be used to process and analyze large datasets.
- Spark is used for efficient memory usage and distributed map-reduce processing.

Spark Limitations

Spark Streaming's latency is at least 500 milliseconds since it operates on micro-batches of records, instead of processing one record at a time.

Native streaming tools such as Storm, Apex, or Flink can push down this latency value and might be more suitable for low-latency applications. Flink and Apex can be used for batch computation as well.

Another limitation of Spark is its selection of machine learning algorithms. Currently, Spark only supports algorithms that scale linearly with the input data size. In general, deep learning is not available either, though there are many projects integrate Spark with Tensorflow and other deep learning tools.

Source: Udacity

Let's do a walkthrough of the code...

WOMEN WHO CODE®

Upcoming for Next Week

WOMEN WHO CODE®

Data Quality

4.2 Data Quality Checks

Explain the data quality checks you'll perform to ensure the pipeline ran as expected. These could include:

- Integrity constraints on the relational database on making sure the primary key is not duplicate.
- Unit tests for the scripts to ensure they are doing the right thing
- Source/Count checks to ensure completeness

Run Quality Checks

In [43]:
```
# Perform quality checks here
cur.execute("SELECT COUNT(*) FROM fact_immigration")
conn.commit()
if cur.rowcount < 1:
    print("No data found in table fact_immigration")

cur.execute("SELECT COUNT(*) FROM dim_airline")
conn.commit()
if cur.rowcount < 1:
    print("No data found in table dim_airline")
```

In [44]:
```
cur.execute("SELECT cicid, COUNT(cicid) FROM fact_immigration GROUP BY cicid HAVING COUNT(cicid) > 1")
conn.commit()
print(cur.rowcount)
```

```
0
```

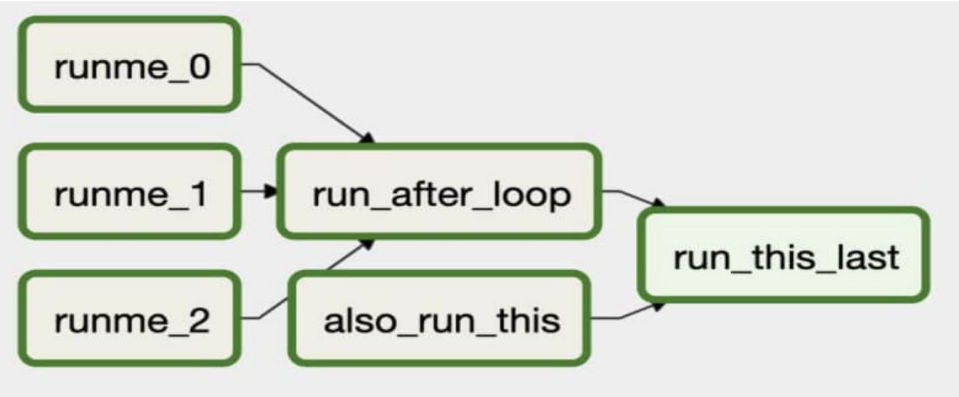
In [45]:
```
cur.execute("SELECT cicid, count(cicid) FROM fact_immigration f \
             INNER JOIN dim_demographics d ON f.state_code = d.state_code \
             GROUP BY 1")
conn.commit()
print(cur.rowcount) #printitng the rowcount.The ETL is working, checked with mergeing fact_immigration and dim_demographics table.
```

```
196
```

Apache Airflow (DAG)

Data Pipelines: A series of steps in which data is processed.

Directed Acyclic Graphs (DAGs): DAGs are a special subset of graphs in which the edges between nodes have a specific direction, and no cycles exist. When we say “no cycles exist”, this means the nodes cannot create a path back to themselves.



Cloud Deployment

Amazon - Redshift, EMR, S3

Microsoft Azure

GCP (Google Cloud Platform)

Closing

- Scope requirements and gather data
- Explore and access the data.
- Define the Data Model - Data Design
- Run ETL Pipelines
- Deploy to a dashboard or github.
- Upgrade using Pyspark, Cloud, Automated pipelines.
- Essential Data Quality