Contents

1.Architecture

1.1. What are different data validation approaches?

Every organization will have its own set of rules for storing and maintaining data. Setting basic data validation rules will assist your company in maintaining organized standards that will make working with data more efficient. The data that is moved to the Trusted Data Zone shall be typically validated to ensure data quality. Based on inputs from one Company, the following data validations are to be implemented for data ingested into the Trusted Zone:

As an example: MSISDN(ID), time_stamp, cell_id consider identifying the record duplicate, Field validations based on schema like:

-datatype & format validations,

-date & format validation

-Phone Number validations – Ex: Integer validation, Alphanumeric.

-IP (IPv4 / IPv6)

-Email Name validations,

-Referential integrity (like Subscriber or Prepaid / Post-paid type lookups).

-Range check:

Check data exists within a certain range. For example price of data

I believe that the business

In my experience, businesses whose data is entered by the user. Data validation must be done carefully.

1.2. When designing a data pipeline, what should you consider?

Data pipeline processes typically have the following stages in common: Extraction of data from one or more data sources.

1-Data extraction

2-Data ingestion

3-Data transformation stage (optional)

4-loding into destination facility

5- A mechanism for scheduling or triggering jobs to run

6- Monitoring the entire workflow

7- Maintenance and optimization as required to keep the pipeline up and running smoothly

The data pipeline needs to be monitored once it is in production to ensure data integrity. Some key monitoring considerations include:

1- Latency (time it takes for data packets to flow through the pipeline)
2- Throughput demand(The volume of data passing through the pipeline over time)
3- Warning errors and failures (network overloading, failures at the source...)
4- Utilization rate
5- system for logging events and alerting administrators when failures occur

When designing a data pipeline, we need to find **bottlenecks** and **synchronization between stages**. Extending this notion to all stages of the pipeline implies that all stages should take the same amount of time to process a packet. This means there are no bottlenecks, and we can say that the entire pipeline has been load-balanced. Consider also that due to **time** and **cost** considerations, pipelines are rarely perfectly load balanced.

Solution:

1- Parallelized (replicate it on multiple CPUs, cores, or threads,)
2- I/O buffers can help synchronization stages
3- Holding area for data between processing stage
4- Buffers regulate the flow of data (improve throughput)

In general, we have two types of pipeline data. Batching and streaming for each of which must be considered.

Batching data pipelines:

Batch processes typically operate periodically on a fixed schedule – ranging from hours to weeks apart. They can also be initiated based on triggers, such as when the data accumulating at the source reaches a certain size.

Streaming data pipelines:

Streaming data pipelines are designed to ingest packets of information, such as individual credit card transactions or social media activities, one-by-one in rapid succession. Minimal latency is important.

The use case differences between batch and stream processing come down to a tradeoff between **accuracy** and **latency** requirements. With batch processing, for example, data can be cleaned. And thus you can get higher quality output, but this comes at the cost of increased latency. If you require low latency, your tolerance for faults likely has to increase.

In some cases, we can have a combination of these two, which is necessary in designing items. For example, **Lambda architectures** combine batch and streaming data pipeline methods. Historical data is delivered in batches to the batch layer, and real-time data is streamed to a speed layer. These two layers are then integrated in the serving layer.

- Lambda can be used in cases where access to earlier data is required but speed is also important.

- A downside to this approach is the complexity involved in the design.

Choose a Lambda architecture when: you are aiming for **accuracy** and **speed**

Features of modern pipeline tolls:

When we need to choose tools for data pipeline, the following are required.

1-Automation

2-Easy of used

3- A drag-and-drop interface

4-Transformation support

5- Security

2. Programming

2.1. Consider an array of n integers, which is analogous to a continuous stream of integers.

A. Write an algorithm to find K largest elements from a given stream of numbers.

B. What is the time complexity of the algorithm?

1-Initially, Build a Min Heap of the first k elements (min is root)

2-Then, we will compare each element, after the kth element (array[k] to array [n-1]), compare it with min (root)

3- In this step, if the element is greater than the min then make it root and call Min Heap, else ignore it.

4-Finally, Min heap has k largest elements, and the root of the Min Heap is the kth largest element.

At the first step: O (k*log (k)) and then O ((n-k)*log (k)), finally Time Complexity: O (k*log (k) + (n-k)*log (k))

2.2. Given a paragraph of text, count the number of the words in that paragraph using Hadoop Map Reduce.

```
Map (Key=Line, value=Content):
    For each word W in value:
        Emit intermediate (w, 1)
```

Before it starts running your reduced program, Map programming does some sort of processing, distributed processing, figures out in this space of intermediate key values pairs that were created by map, which ones have the same set of keys, so it actually does a grouping operation(shuffling).

for example=> K1=>[V1,V2,V3,V4] , K2=>[V1,V2,V3,V4,V5,V6],..,K5=>[V1,V2]

It groups all of these and creates those lists. So now your key one will have value four.

After that, it starts calling reduced functions and it will instantiate five different invocations of function reduce.(because my example has 5 key)

```
Reduce (Key, Values):
  //key: a word and values: an iterator
     For each v in intermediate values:
        Result+=v
  Emit (Key, result)
```

The result is the number of times that specific word, this word, passed to me by key, was repeated in the input data. And now we can say emit, again, pass it back to the framework. Of course, emit here is pseudo-code.

Result=> (K1, 4), (K2, 6)… (K5, 2)

2.3. What do the following lines of code do? Why do we use Redis? What is the problem with this snippet?

This code indicates that it is first searching for a key in Redis if it finds one that returns the result. But if it does not exist, it sets it in Radis and only sets a time for it when it is set. For example, after 60 minutes, the key will be deleted. (It is used to cache data. (Run Redis on localhost: 6379))

I think the problem is in line 10 because it has to check the ID with a query and can use EXISTS.

3. Practical task

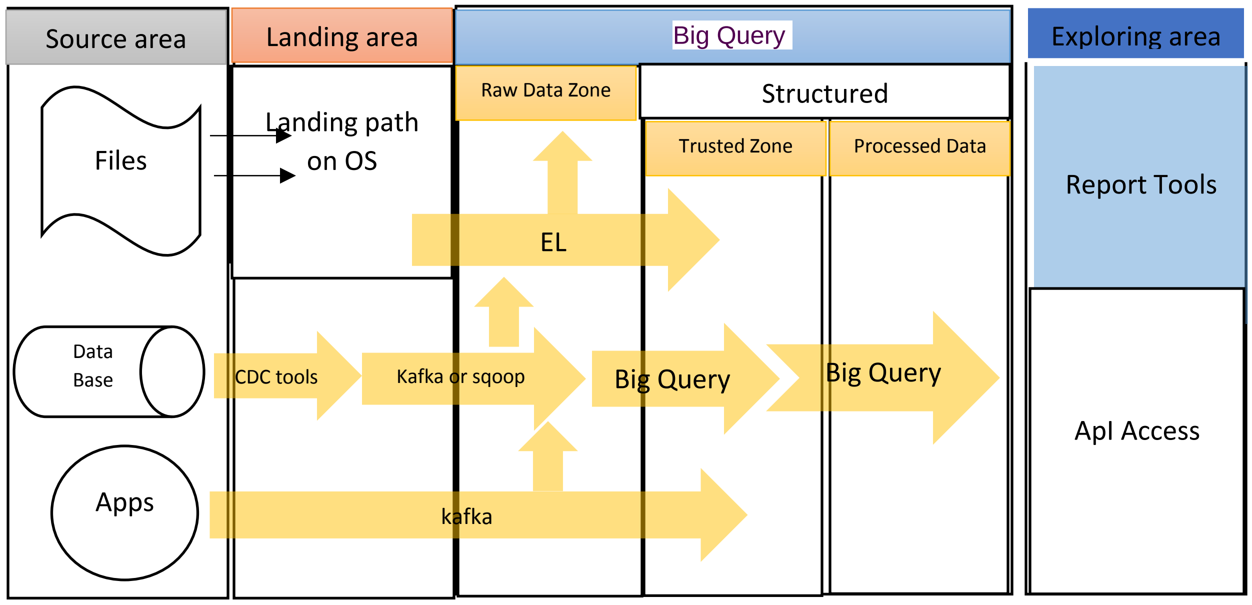
3.1. Approach to a Data Lake

One of the key principles is the choice of an approach to meet the requirements of ingesting, processing & consuming large volumes of data and cater to both near-real-time & batch mode operations. The obvious choices for this are – a traditional Data Warehouse, Data Marts or Data Lakes.

Data Lakes – data lakes are implemented with the principle of allowing any and all data in the enterprise to be ingested, explored, analysed and reported with minimal lag. The principle here is, the processing is driven by the use cases and to make data available to end-users and use cases as soon as available. This is achieved by:

- analysing use cases and creating specific ingestion & analysis jobs (specific to near real-time or batch use cases),
- allow structured, semi-structure & unstructured data be ingested, processed and analysed,
- enable exploratory data analysis & consumption,
- allow self-service data consumption.

The figure below illustrates the overall structure of my design including the various data zones implemented in the Data Lake. Further, the data flows between the various zones is also shown along with data landing area and external apps feeding data into the Data Lake. The following sections describe the data zones & data flow in more detail.



3.2. Data Zones

Each zone is a logic area used for coarse-grained access control & organization of data tables within a data lake. Further, the tables in a zone can be identified using tags.

Directory Structure for the Zones

```
<Project Name>
  └ Raw Data Zone (Dataset)
  └ Trusted Data Zone (Dataset)
  └ Processed Data Zone (Dataset)
```

Each Data zone shall have a dedicated directory created under the "Data Lake (Project name)" directory which in turn is created under the Big Query storage mount point created for the Data Lake.

Typical directory structure for zones

The Datasets named – "Raw Data Zone", "Trusted Data Zone", "Processed Data Zone" are created. Further, Tables are to be created in the "Raw Data Zone" dataset as described in the section below.

For Trusted & Processed Data zones, tables under the zone directories (datasets) are to be created corresponding to the specific ingestion or wrangling jobs. The table names shall be the same as the job names for which the directory was created.

3.3. Raw Data Zone

As the name suggests this zone is used to maintain copies / replicas of the raw data that is ingested into the Data Lake. The primary purpose of this zone is to retain raw data for a specified period, so that data maybe reprocessed on any use ace include data science tasks, re ingestion of data with new structure, mistake or error in ingestion to trusted zone or referred for auditing or debug purposes.

Raw Data Zone Directory Structure

```
Subject Area
  └ Date <YYYYMMDD>
```

Date, this is the date of collection or processing of the raw data from the source, typically named in the YYYYMMDD format. This is the final directory under which a successfully collected file is stored.

3.4. Trusted Zone

RAW data ingested first into RAW data zone and after processing and normalization with Big query processors and custom processors ingested into Data Lake and lands into this trusted zone.

Trusted Data Zone Directory Structure

```
Subject Area
  └ Date <YYYYMMDD>
```

The data that is moved to the Trusted Data Zone shall be typically validated to ensure data qualit.

3.5. Processed Data Zone

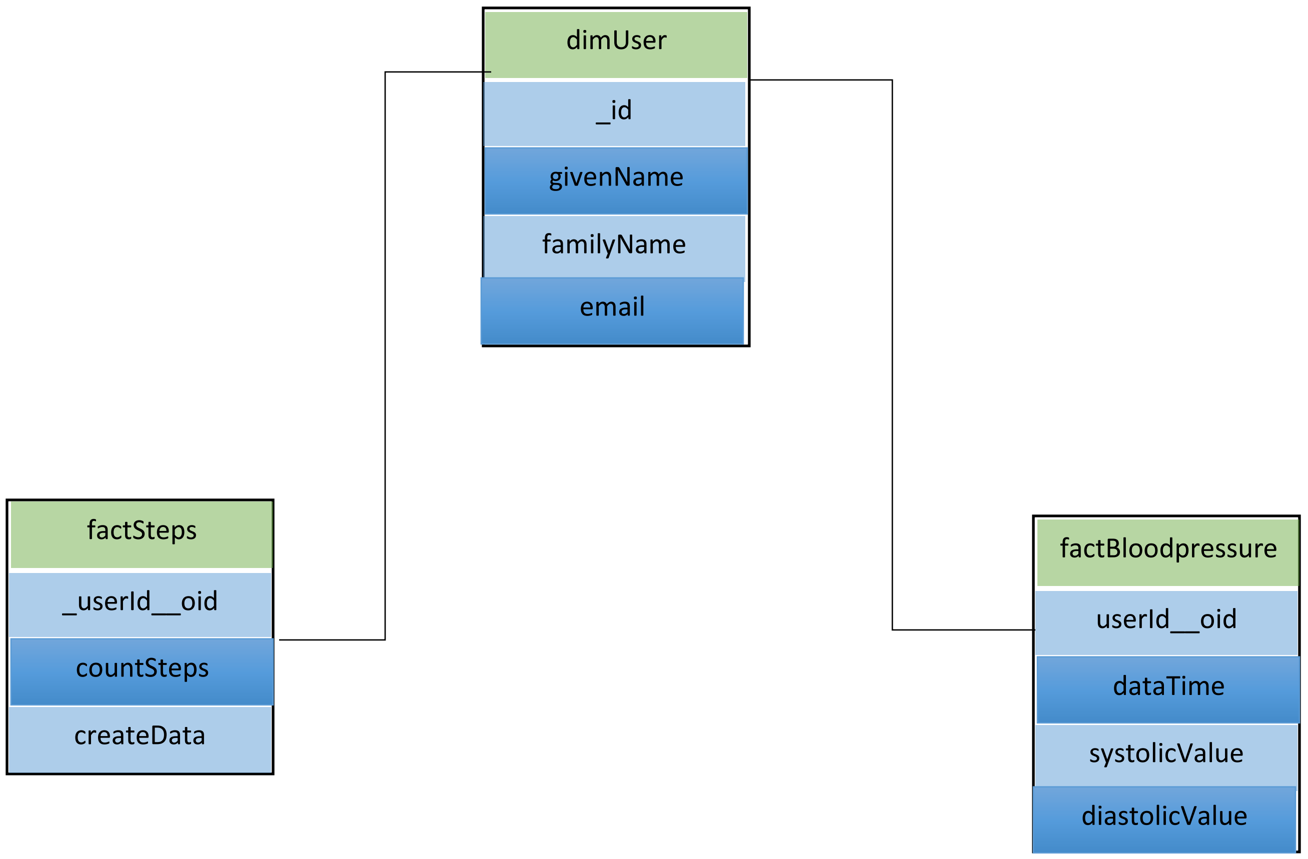
All the output of data wrangling & analytics tasks primarily stored is this zone. This includes summaries, features or aggregates generated from input data in the trusted zone and data generated by various analytics models & algorithms run on the Data Lake.

3.6. Fact & Dimension Modelling

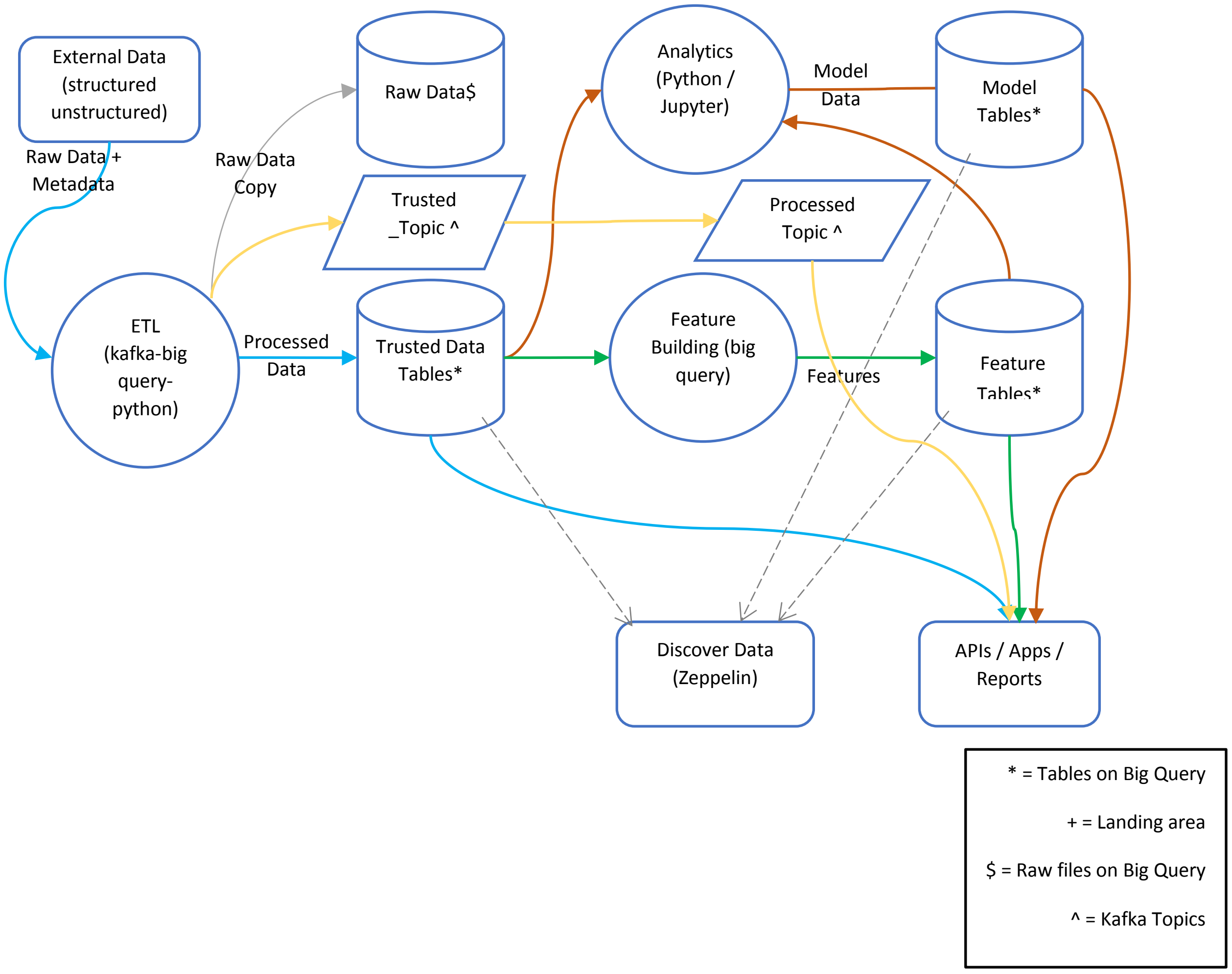
Data Lake using star schema for data of data marts in Big Query tables. STAR Schema, the centre of the star can have one fact table and several associated dimension tables. The fact table is at the centre which contains keys to every dimension table.

Mart Creation & Management, Marts are logical entities, hence they are primarily organized using tags added to tables during ingestion or wrangling / analytics jobs. Fine-grained access control maybe added to the table to limit access to specified users or user groups. These users or user groups may belong to a specific BU or Department. An appropriate naming convention for the table names may also be considered to enable end-users to discover & consume data from specific Marts.

The figure below illustrates the structure of my design including the various marts implemented in the Data Lake.(The design is based on the requested task)(can use to SCD type 2 for dimension user)


dimUser
_id
givenName
familyName
email
factSteps
_userId__oid
countSteps
createData
factBloodpressure
userId__oid
dataTime
systolicValue
diastolicValue

3.7. Data Ingestion Flow



Data Ingestion into the Data Lake happens over an ETL processes running within the Apache Air Flow system.

Data Flow, during Data Ingestion, the ETL job running within Air Flow(Python-Big Query-Kafka) fetches data from the external sources, transforms & loads into Big Query tables within the Data Lake. This data is represented as “Trusted Data Tables” in figure above. The raw data fetched from the sources is itself copied into “Raw Data” area which is a Big Query location in the Data Lake. The data in the “Trusted and Processed Data Tables” can then be fetched by the consuming apps, APIs & for reporting as shown above. This end-to-end flow is represented as the blue line in figure above.

To build the ETl in figure above. Based on previous conversations in the email, it has been assumed that the data is in the Row zone. At this stage, according to the needs of my task, the data is extracting and loading in the Trust zone after the transformation.

3.8. EDL Architecture

The data flows are described as seven primary flows / use cases:

1. Data Ingestion,
2. Data Storage and Data Zones,
3. Data Wrangling (Feature Building) through business rules and logics,
4. Analytics through modelling,
5. Metadata – Schema & Lineage and data catalogue,
6. Reporting / dashboards,
7. Infrastructure & Deployment tools,

1. Data Ingestion

Data Lake uses python code for data ingestion flow design and deployment and Kafka as streaming storage for CDC from RDBMS, NoSQL, and message storage for social media sources.

Big query: the primary tool used for the ETL operations in the Data Lake. The big query runs the ETL jobs configured to fetch, transform & load the data from external sources.

Airflow plays the role of the primary job scheduler.

Apache Kafka: Kafka is a unified platform for handling the real-time data feeds. Spark Streaming or big query streaming read data from a topic, processes it, and writes processed data to a new topic or ingests it to Data Lake, where it becomes available for downstream applications and other processing parts like mart and report.

2. Data Storage and Data Zones

Data Lake uses Cloud storage as a storage engine and on top of clued storage uses Cloud big table for table design and warehousing.

Data Lake uses a layered storage policy and in cloud storage, data will store in different locations based on the type of processing, Data Lake have 3 data zone:

- Raw Zone: It is used to maintain copies / replicas of the raw data that is ingested into the EDL.
- Trusted Zone: The data ingested into Data Lake lands into this zone. Further, certain processed data tables generated (like transaction table, lookup tables as part of the ETL jobs are also stored in the Trusted Zone.
- Processed Zone: All the output of Data Wrangling/Summarized results & Analytics tasks are primarily stored into this zone. This includes summaries, features or aggregates generated from input

data in the Trusted Zone and data generated by various Analytics models & algorithms run on the Date lake.

3. Data Wrangling (Feature Building)

- **Apache Spark and Big query**: Apache Spark and Big query are used in the Data Lake as the analytics engine for processing data & running feature building jobs. Further, the Livy API wrappers are used by various components for connecting external components to Spark and submit jobs to cluster. Each Livy session per user starts a new spark context, all user specific operations will be scoped in that context lifecycle. Spark Thrift Server also being used for connecting Spark SQL connections to cluster.

- **Big query**: Big query is the primary way by which data is processed to generate features (summaries, aggregates) for marts and analytics.

- **Apache Zeppelin**: Zeppelin is deployed as a tool for developing queries, joins & data visualization on Data Lake tables and to export limited data. Zeppelin is also part of sandbox.

- **Git**: Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

4. Analytics

- **Jupyter Hub**: this is a platform for creating standalone analytics sandboxes. Each sandbox uses Jupyter & its notebooks as the primary tool to build, trial, train & finalize a model using a Python analytics library.

5. Data Governance and Metadata (Schema & Lineage)

- **Apache Atlas**: Atlas provides the UI needed to search & drilldown into the metadata within the Data Lake.

6. Reporting & Dashboard

- **Tableau**: Tableau is the enterprise wide reporting & dashboarding

7. Infrastructure & Deployment tools

- **Docker & Docker Compose**: all components that are developed by your team for use in the Data Lake deployment are built as microservice and delivered as Docker containers. Further, where complex containers or container bundles are to be built, these are built using the Docker Compose tool to build & deploy these.

- **Kubernetes**: Kubernetes is the preferred tool for orchestration of containers,

3.9. Pseudonymization

Cloud DLP offers a branch of options to help you hide and remove the sensitive value in your data. one technique that leverages encryption to de-identify and some cases reidentify those sensitive values. Pseudonymization is a powerful full de-identification technic that uses encryption keys to replace sensitive data values whit cryptographically generated tokens.

Three types of cryptography methods:

1-Deterministic encryption using AES-SIV

-detected data is replaced with encrypted value and prepended with optional surrogate annotation

-supports most input type

2-Format preserving encryption (FPE):

-replace the value of an encrypted string

-same length (same character set as the original values)

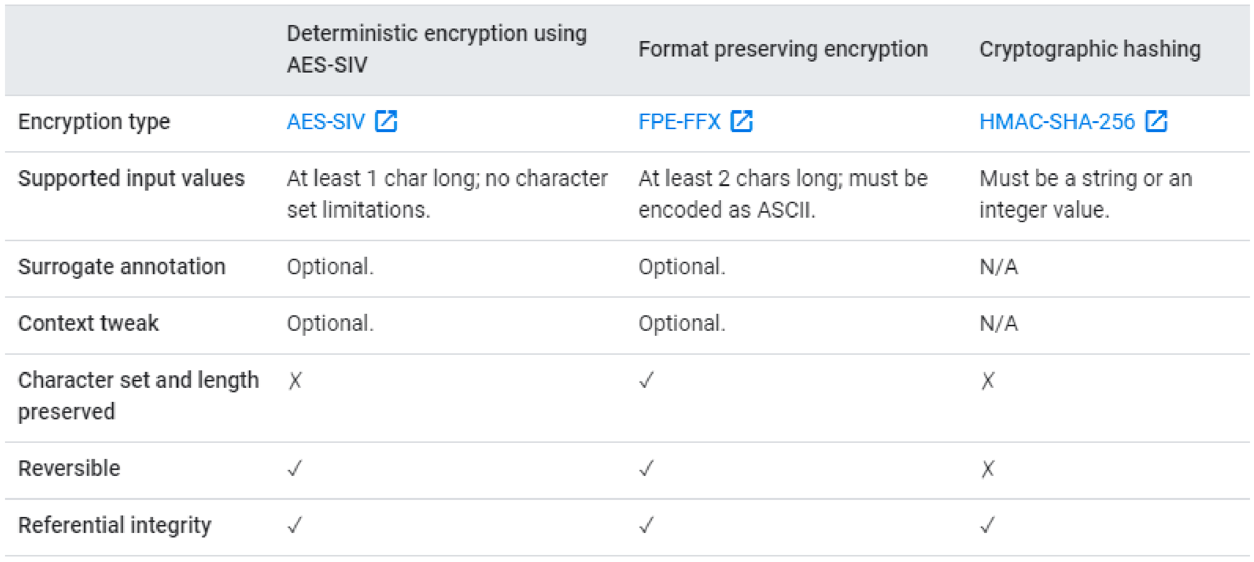
3-Cryptographic hashing

-replace sensitive data whit a hash value

-uses a one-way token (it can't be reversed)

-the perfect solution if you want each unique value to be transformed into a corresponding unique hash

The pseudonymization methods are summarized in the below table:



	Deterministic encryption using AES-SIV	Format preserving encryption	Cryptographic hashing
Encryption type	AES-SIV	FPE-FFX	HMAC-SHA-256
Supported input values	At least 1 char long; no character set limitations.	At least 2 chars long; must be encoded as ASCII.	Must be a string or an integer value.
Surrogate annotation	Optional.	Optional.	N/A
Context tweak	Optional.	Optional.	N/A
Character set and length preserved	✗	✓	✗
Reversible	✓	✓	✗
Referential integrity	✓	✓	✓

In projects we have worked on before:

Data Masking for sensitive data is provided through policies configured in Ranger and tools(Spark, Trino, Hive, HDFS). Masking or access control may be based on applying policies directly using the table or column names or by using tags to tables and columns.

According to the above, choosing which method is more appropriate؟

If we need to be a reversible process in Data Lake, it is better to don't use the third method. Between the first and the second method, the second method same length (same character set as the original values), and in some cases, its use may be better, for example, when information is to be given to the users. But the first method is the method that Google Cloud suggests. 'We recommend deterministic encryption using AES-SIV among all the reversible cryptographic methods that Cloud DLP supports because it provides the highest level of security."

As a result, according to less my knowledge of the needs of your business, I have done the first method.

Requires basic configuration as described here.

The following steps are required:

1- Enable the Cloud DLP and Cloud KMS APIs.
2- Create service account
3- Create a key ring and a key
4- Create a key
5- Create a base64-encoded AES key
6- Wrap the AES key using the Cloud KMS key: send the following request to the Cloud KMS API
7- When we save data in Raw zone data lake we can use:

```
ECLARE KMS_RESOURCE_NAME STRING;
DECLARE FIRST_LEVEL_KEYSET BYTES;
SET KMS_RESOURCE_NAME =  "gcp-
kms://projects/PROJECT_ID/locations/LOCATION_NAME/keyRings/KEY_RING_ID/cry
ptoKeys/KEY_ID";
# Set the first level keyset equal to the output from the previous step
SET FIRST_LEVEL_KEYSET =
b"\012\044\000\031\261\001\270\114\176\037\055\330......";
INSERT INTO trustedZone.user
SELECT "example_plaintext",
AEAD.ENCRYPT(KEYS.KEYSET_CHAIN(KMS_RESOURCE_NAME, FIRST_LEVEL_KEYSET),
"example_plaintext", "additional_data");
```

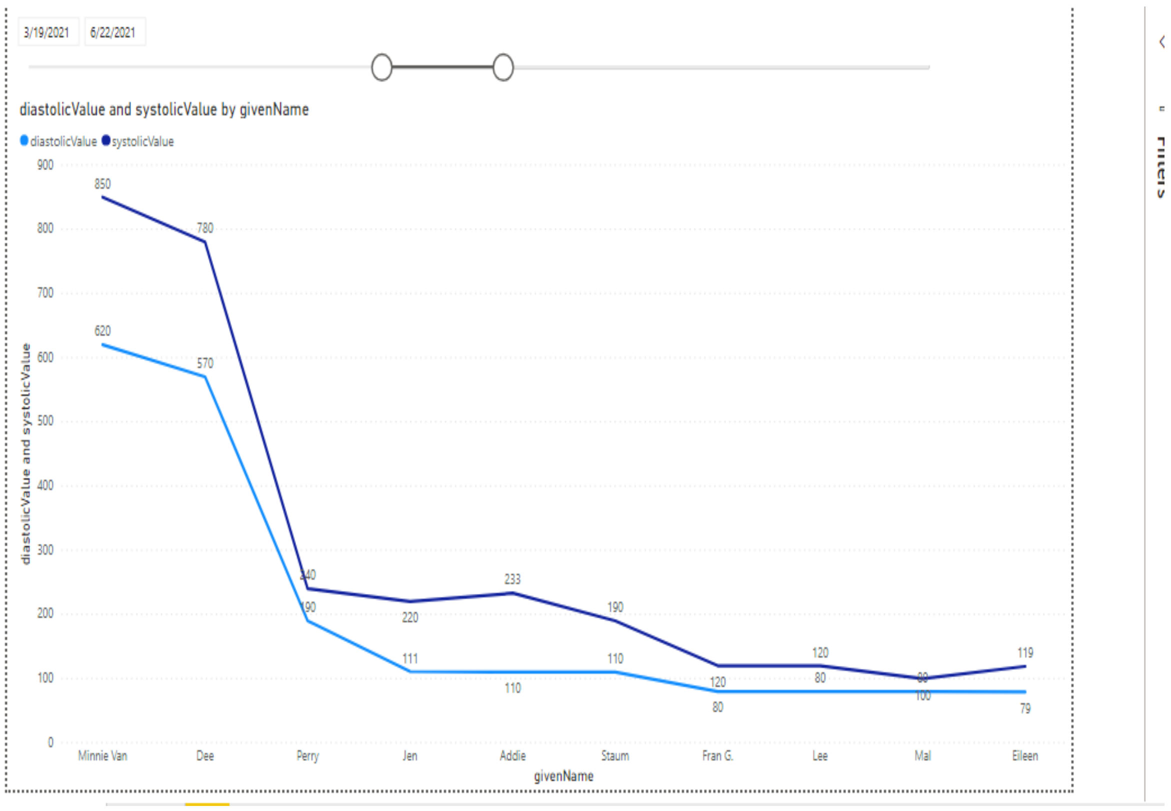
Now that the column is encrypted, we can decrypt the column with the same keyset chain using SQL decrypt functions.

```
DECLARE KMS_RESOURCE_NAME STRING;
   DECLARE FIRST_LEVEL_KEYSET BYTES;
   SET KMS_RESOURCE_NAME =  "gcp-
kms://projects/PROJECT_ID/locations/LOCATION_NAME/keyRings/KEY_RING_ID/crypto
Keys/KEY_ID";
   SET FIRST_LEVEL_KEYSET =
b"\012\044\000\031\261\001\270\114\176\037\055\330......";
```

```
   SELECT AEAD.DECRYPT_STRING(KEYS.KEYSET_CHAIN(KMS_RESOURCE_NAME,
FIRST_LEVEL_KEYSET), encrypted_data, "additional_data"),
   CAST(AEAD.DECRYPT_BYTES(KEYS.KEYSET_CHAIN(KMS_RESOURCE_NAME,
FIRST_LEVEL_KEYSET), encrypted_data, CAST("additional_data" AS BYTES)) AS
STRING)
   FROM DATASET_NAME.TABLE_NAME;
```

3.10. Report

The report was created using Power BI software and is located in the Power BI folder


3/19/2021
6/22/2021
diastolicValue and systolicValue by givenName
● diastolicValue ● systolicValue
diastolicValue and systolicValue
Minnie Van
Dee
Perry
Jen
Addie
Staum
Fran G.
Lee
Mal
Eileen
givenName
Filters