**Enterprise Analytical Data Lake (EADL)**

1. Introduction

2. Logical Architecture

3. Source Table Data Extractor

4. Staging Data Processor

5. Core Data Processor

6. File/Table Configuration

7. Cleansing Techniques

8. Transformation Techniques

9. Performance Optimization Techniques

10. Deployment and Execution

11. Future Enhancements

12. Conclusion

1. **Introduction**

Czech Bank's credit Card Review team need an analytical platform to mine and analyse the bank data to classify/sort the type of customer and, approve or reject the credit card or loan application. As the credit card/loan review team plays a crucial role in maintaining the integrity and profitability of program, it requires access data platform to visualize and process the bank data.

**Technical Description:**

1. Designed and Implemented a Data Lake for credit card review team to perform the data analytical processing and visualization.
2. Data Lake did not use any cloud resources/services but strictly used on-prem bank resources and hardware.
3. Input files are in csv format. Output or core layer data files are stored in HDFS with parquet format and snappy compression. Partitions created for the business date.
4. External HIVE tables created for core layer files.

**2. Logical Architecture**

A diagram of a bee

Description automatically generated

**3. Source Table Data Extractor**

Python-based data extraction module has been implemented to extract the source data from SQL Server database. The extracted data been saved in CSV file format. Along with the csv data files, control file for each data files, get generated with the following information:

*(i) data file name*

*(ii) hash algorithm name*

*(iii) computed hash value*

*(iv) number of columns*

*(v) number of rows*

These files will be kept in a shared location for data consumers.

**4. Staging Data Processor**

Upon extracting the source data from the database, scheduler job will transfer the extracted data from the shared location to the edge node.

The staging data processor reads the data files as well as control files, and perform some standard data validation. The following validations have been performed by the staging data processor:

1. Existence of the data file.
2. Hash Value validation.

i.e., compute the hash value for the data file using the algorithm stated in the control file and compare the computed hash value with the control file hash value.

1. Validate the column count between the data file and the control file value.
2. Validate the row count between the data file and the control file value.

If all the validation is successful, staging data processor will move the data file from the edge node to the HDFS location. However, it does not modify the incoming file content instead it will just validate and transfer the file from the Linux/edge node to the HDFS cluster.

**5. Core Data Processor**

When the staging is successful, core data processing layer reads the staged data for further processing. The staged data has been mapped to the external table with string data type for individual columns. So there is no cleansing or transformation has been applied to the staged data. The core data processor will read the staged data with string data type columns for the further cleansing and transformations.

It uses high level API to read the staged data via external tables. After reading the staged data from the staging layer, core data processing layer perform the following tasks:

1. convert the individual columns data type from string to correct data type.
2. apply cleansing and clean the data.
3. apply transformation to transform the data.

As the solution is fully configurable, different cleansing and transformation can be applied to the tables. Eventually, clean data has been written to the core layer (HDFS) with the ORC file format and snappy compression. Furthermore, it will create new partition for the current as of date and add the partition detail to the core external high table. The core external table will have, correct data type for the data and serve the user queries.

**6. File/Table Configuration**

A screenshot of a computer screen

Description automatically generated

**7. Cleansing Techniques**

A screenshot of a computer

Description automatically generated

**8. Transformation Techniques**

A screenshot of a computer

Description automatically generated

**9. Performance Optimization Techniques**

1. Higher Level APIs have been used.
2. Partitions are made for each business date
3. Hadoop standard file format(orc) and compression technique (snappy) used
4. User defined functions and GroupByKey are not used as it affects the data processing speed.
5. Minimum logs are used to improve the performance.

**10. Deployment and Execution**

/home/itv007175/project\_src/staging\_data\_processor > /usr/bin/python3.6 validator\_main.py

spark-submit core\_data\_processor\_main.py 20230928

hadoop fs -ls /user/itv007175/datalake/core/\*/\*

hadoop fs -rm -r /user/itv007175/datalake/core/\*/\*

/user/itv007175/datalake/stage/disp/

/usr/bin/python3.6 validator\_main.py 20230927

/usr/bin/python3.6 validator\_main.py 20230928

11. Future Enhancements

12. Conclusion

Appendix:

Sample data file:

Sample control file:

Config file: