VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATA ENGINEERING (CO5240)

ASSIGNMENT: DATA LAKEHOUSE FOR SHIPPING DELIVERTY MANAGEMENT SYSTEM

Lecturers: **PhD.Phan Trong Nhan**
Group: **2010702 - Tra Trung Tin**
2370506 - Dang Lam Tung
2370700 - Nguyen Ngoc Gia Van

HO CHI MINH CITY, March 2024

CONTRIBUTION

Number	Student ID	Full Name	% Contribution
1	2010702	Tra Trung Tin	$33.\overline{3}\%$
2	2370506	Dang Lam Tung	$33.\overline{3}\%$
3	2370700	Nguyen Ngoc Gia Van	$33.\overline{3}\%$

Contents

1

Introduction

1.1 Overview of Delivery Shipping Management

Delivery shipping management involves the coordination and execution of the delivery process, from the point of order preparation to the successful delivery to the customer's address. It encompasses various stages and activities, including order preparation, scheduling of deliveries, selection of shipping methods, tracking of shipments, and customer interaction. In the context of your assignment, which focuses on Company X's delivery system, several key aspects of delivery shipping management are particularly relevant:

1. **Selection of Shipping Mode**:
   The assignment mentions two modes of delivery: single trip and round trip. Single trip involves delivering an order package directly to the customer's address, while round trip includes the additional step of returning support items to the store after delivery. It's essential to determine the most suitable shipping method based on factors such as the nature of the order, customer preferences, and cost-effectiveness.

2. **Coordination with Third-Party Shipping Services**:
   If Company X utilizes third-party shipping services, effective coordination is necessary to ensure smooth and timely delivery. This includes scheduling pickups, providing accurate shipping information, and monitoring the status of shipments. It's important to note that while third-party services may handle the physical delivery process, Company X remains responsible for managing the overall shipping operation and maintaining visibility into the process.

3. **Recording Shipping Information:**
   From the time an order package is prepared at the store to the completion of delivery, capturing and recording shipping information is crucial for tracking performance and ensuring accountability. This includes details such as order pickup times, delivery routes, delivery times, and any relevant notes or observations.

4. **Centralized Data Management:**
   To effectively track and analyze shipping performance, it's essential to centralize shipping-related data in a unified system. This ensures that all relevant stakeholders, including the CEO, have access to accurate and up-to-date information for monitoring performance and making informed decisions.

1.2 Data Lakehouse concept

Before delving into the Data Lakehouse, let's understand two basic concepts: Data Lake and Data Warehouse.

Data Warehouse

A data warehouse compiles raw data from various origins into a central storage and organizes it within a relational database framework. This method of data management primarily serves data analysis and business intelligence tasks, including enterprise reporting. The system use ETL (extract, transform, load) procedures to retrieve, modify, and transfer data to its intended destination. Nevertheless, its drawbacks include inefficiency and cost, particularly as the number of data sources and quantity of data grow over time.

Data Lake

Data lakes are commonly built on big data platforms such as Apache Hadoop. They are known for their low cost and storage flexibility as they lack the predefined schemas of traditional data warehouses. They also house different types of data, such as audio, video, and text. Since data producers largely generate unstructured data, this is an important distinction as this also enables more data science and artificial intelligence (AI) projects, which in turn drives more novel insights and better decision-making across an organization. However, data lakes are not without their own set of challenges. The size and complexity of data lakes can require more technical resources, such as data scientists and data engineers, to navigate the amount of data that it stores. Additionally, since data governance is implemented more downstream in these systems, data lakes tend to be more prone to more data silos, which can subsequently evolve into a data swamp. When this happens, the data lake can be unusable.

Data lakes and data warehouses are typically used in tandem. Data lakes act as a catch-all system for new data, and data warehouses apply downstream structure to specific data from this system. However, coordinating these systems to provide reliable data can be costly in both time and resources. Long processing times contribute to data staleness and additional layers of ETL introduce more risk to data quality.

Data Lakehouse

The data lakehouse optimizes for the flaws within data warehouses and data lakes to form a better data management system. It provides organizations with fast, low-cost storage for their enterprise data while also delivering enough flexibility to support both data analytics and machine learning workloads.

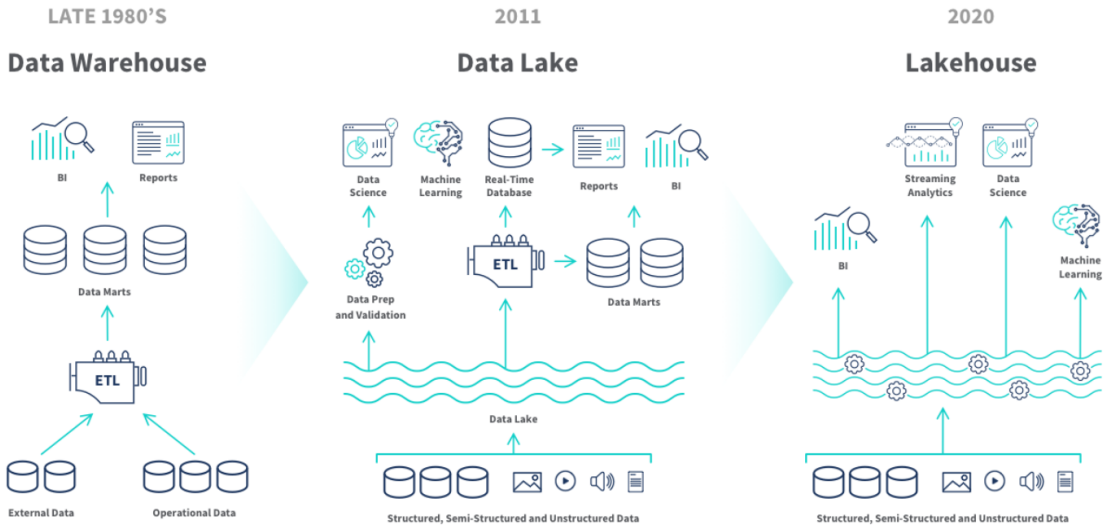

Figure: Data lakehouse - new technology

Key features of a data lakehouse

- **Flexible storage:** The Data Lakehouse allows organizations to store data in various formats. Data can be stored in the system similar to a Data Lake, helping to preserve the original characteristics of the data.
- **Data processing:** The Data Lakehouse combines data processing capabilities from the Data Warehouse, allowing for complex queries, data aggregation, and analysis in various ways.
- **Metadata management:** Metadata is meticulously managed within the Data Lakehouse to understand the structure, origin, and meaning of the data. This helps in searching, querying, and managing data more easily.
- **Integration with analytics tools:** Data Lakehouse typically integrates well with data analytics tools, enabling users to access and use data efficiently.
- **Support for ETL tasks (Extract, Transform, Load)**: The Data Lakehouse automates the extraction, transformation, and loading of data from various sources into the system, reducing the time and effort required.

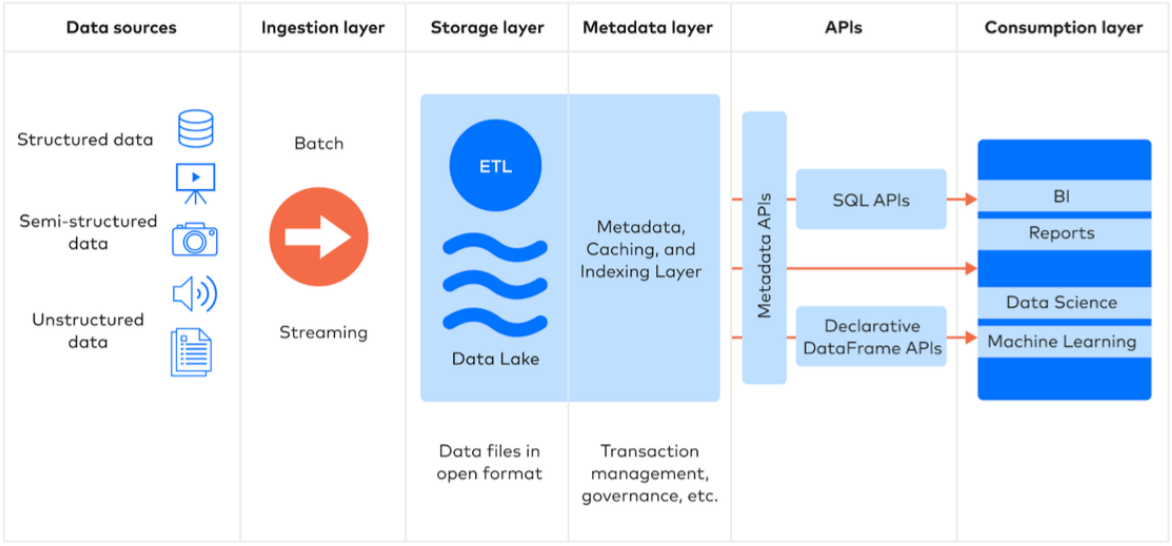

Figure: Data Lakehouse architecture

Data lakehouse architecture

A data lakehouse typically consists of five layers: ingestion layer, storage layer, metadata layer, API layer, and consumption layer. These make up the architectural pattern of data lakehouse.

- **Ingestion layer**
  This first layer gathers data from a range of different sources and transforms it into a format that can be stored and analyzed in a lakehouse. The ingestion layer can use protocols to connect with internal and external sources such as database management systems, NoSQL databases, social media, and others. As the name suggests, this layer is responsible for the ingestion of data.

- **Storage layer**
  In this layer, the structured, unstructured, and semi-structured data is stored in open-source file formats, such as such as Parquet or Optimized Row Columnar (ORC). The real benefit of a lakehouse is the system's ability to accept all data types at an affordable cost.

- **Metadata layer**
  The metadata layer is the foundation of the data lakehouse. It's a unified catalog that delivers metadata for every object in the lake storage, helping organize and provide information about the data in the system. This layer also gives user the opportunity to use management features such as ACID transactions, file caching, and indexing for faster query. Users can implement predefined schemas within this layer, which enable data governance and auditing capabilities.

- **API layer**
  A data lakehouse uses APIs, to increase task processing and conduct more advanced analytics. Specifically, this layer gives consumers and/or developers the opportunity to use a range of languages and libraries, such as TensorFlow, on an abstract level. The APIs are optimized for data asset consumption.

- **Data consumption layer**
  This final layer of the data lakehouse architecture hosts client apps and tools, meaning it has access to all metadata and data stored in the lake. Users across an organization can make use of the lakehouse and carry out analytical tasks such as business intelligence dashboards, data visualization, and other machine learning jobs.

1.3 Objectives of implementing a Lakehouse solution

Implementing a Lakehouse solution for tracking shipping information in business X serves several key objectives:

- **Unified Data Management**
  The primary objective of implementing a Lakehouse solution is to unify the management of shipping data. By consolidating data from various sources, including third-party shipping services and manual staff recordings, into a single platform, business X can establish a centralized repository for all shipping-related information.

- **Scalability and Flexibility**
  Another objective of implementing a Lakehouse solution is to provide scalability and flexibility to accommodate the growing volume and complexity of shipping data. By leveraging cloud-based infrastructure and distributed computing technologies, business X can scale its data processing and storage capabilities as needed, ensuring that the Lakehouse solution remains robust and adaptable to changing business requirements.

- **Data Quality and Consistency**
  Ensuring the quality and consistency of shipping data is a critical objective of implementing a Lakehouse solution. By incorporating data validation and cleansing processes into the data pipelines, business X can minimize errors and discrepancies in the shipping data, thereby improving the reliability and accuracy of analytical insights derived from the Lakehouse.

- **Advanced Analytics and Insights**
  Leveraging the capabilities of a Lakehouse architecture, business X aims to perform advanced analytics and derive actionable insights from the shipping data. This objective involves deploying machine learning models, predictive analytics, and data visualization techniques to identify trends, patterns, and performance metrics related to shipping operations. These insights empower decision-makers to optimize shipping processes, enhance customer satisfaction, and drive business growth.

- **Real-Time Data Accessibility**
  A Lakehouse solution enables real-time access to shipping data for all stakeholders within the organization, including the CEO. This objective ensures that decision-makers have access to up-to-date information on shipping performance, allowing for timely interventions and strategic planning.

2

Prepare and generate data source

2.1 Structured data

2.1.1 Components in the database system

In this assignment, our team is focusing on the shipping management system. Therefore, we will skip through the auxiliary data classes such as **product**, **item detail** class in this system and focus on the 5 main data classes: the **order** class, the **shipping** class, the **shipping provider** class, the **store** class, and the **staff** class.

- Order: This class represents an order placed by a customer. Attributes may include order_ID (PK), customer information (name, address, contact details), item details, price and created time, store_ID (FK).
- Shipping Class: This class represents the shipping process for an order. Attributes may include shipping_ID (PK), service_ID (FK), shipping status (completed, cancelled, processing), order_ID (FK), charged, and timestamps (completed_at, picked _up_at, boarded_at, accepted_at, cancelled_at), trip_type.
- Shipping Provider Class: This class represents the third-party shipping service used by the business. Attributes may include provider ID, provider name, service name.
- Store Class: This class represents a physical store location. Attributes may include store_ID (PK), store name, address, city.
- Staff Class: This class represents the information of staff members who work at the store. Attributes may include staff_ID (PK), staff name, email, store_ID (FK)

2.1.2 Relational database system design diagram

From the components required in the database of the Shipping management system, a relational database is designed based on the table descriptions below.

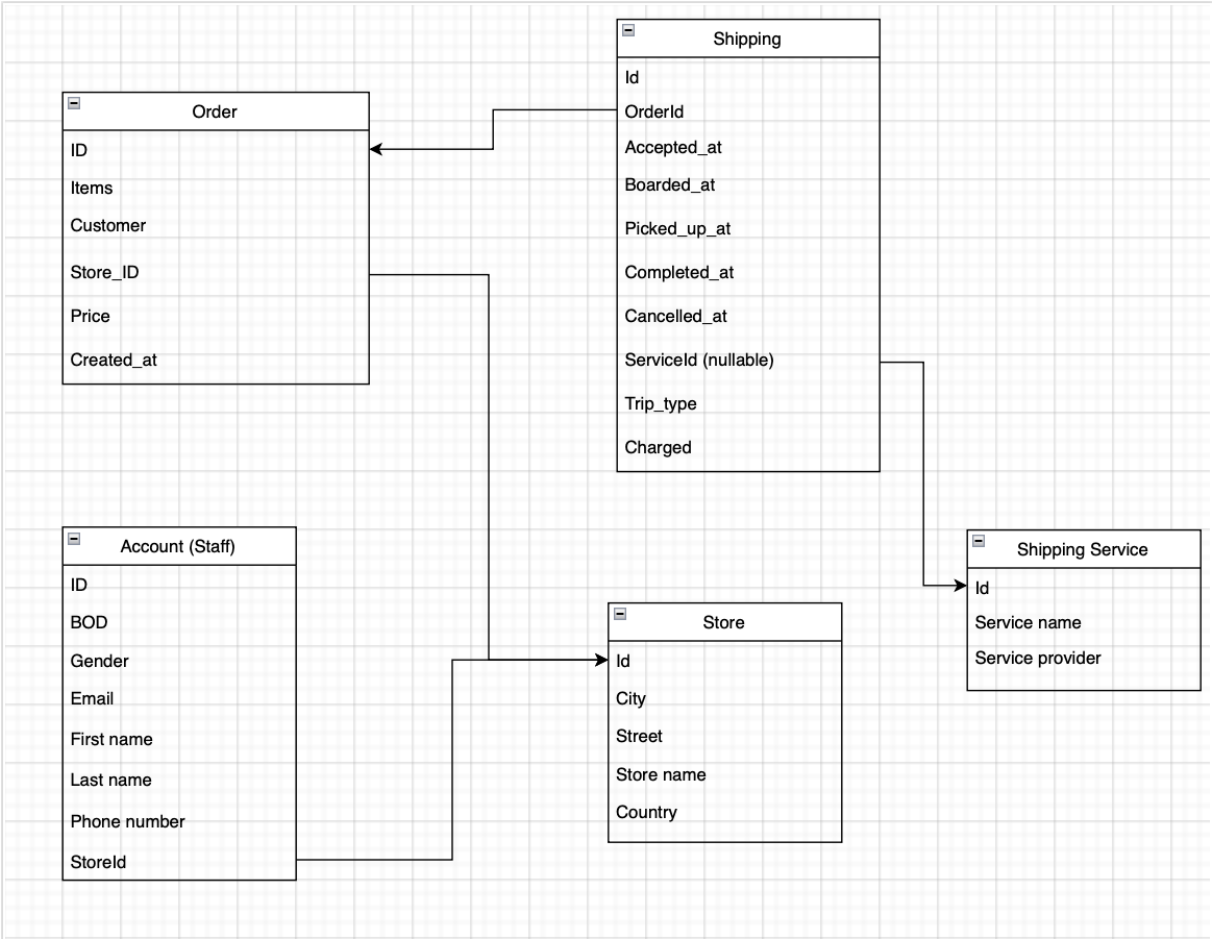

Figure: Relational database

For structured data management, our team has opted to utilize a relational database system, specifically PostgreSQL. This choice was made due to the structured nature of the data, which includes information such as order details for deliveries managed by third-party shipping services. The relational database schema is designed to efficiently store and manage this structured data, providing capabilities for querying, updating, and analyzing order information. With PostgreSQL, our system can ensure data integrity, consistency, and reliability for all transactions related to order processing and delivery tracking.

2.2 Unstructured data

In practice, the order information captured by staffs will be in the form of paper notes, typically in image format, either captured in image format or stored in digital formats such as Excel, S3 or administrative software like Jira or CMS systems, etc.

However the main objective of this assignment is to implement a data lakehouse solution,

so to simulate this process and generate fake data for testing purposes, our team has developed a Python source code that generates synthetic paper note data. This data is then converted into PNG format files, simulating actual paper note images. Subsequently, these generated images are uploaded and stored in the designated S3 bucket using the AWS SDK for Python (Boto3).

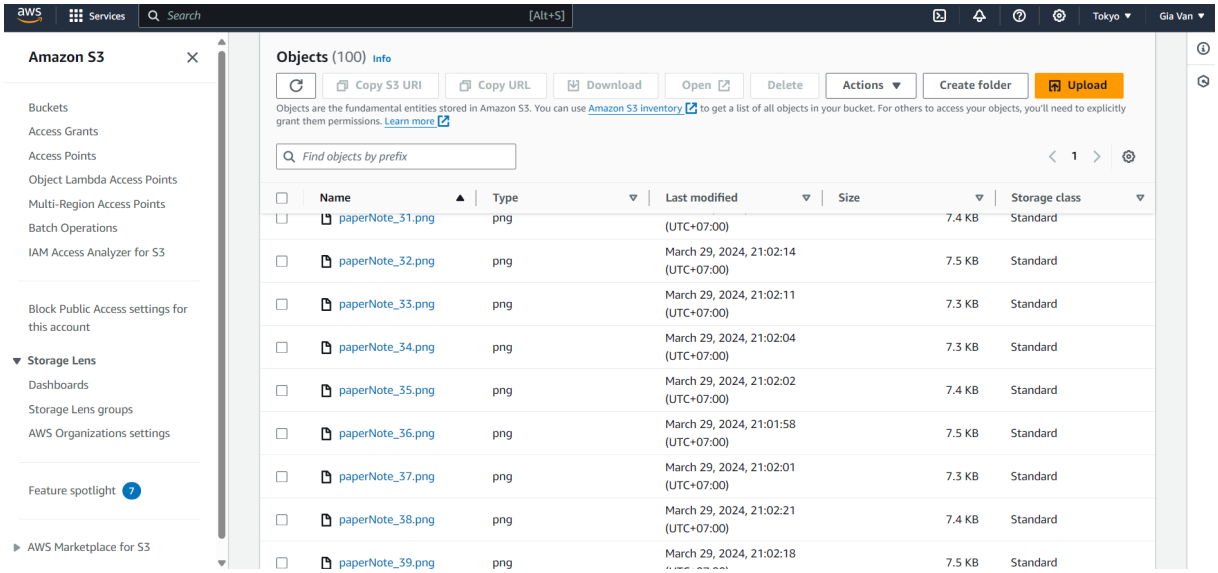

Figure: Paper notes store in S3 bucket

We have designed a RESTful API that serves as an interface for accessing the URL(s) of these paper note images on S3 bucket. The API dynamically generates and returns the URL(s) of the corresponding paper note image(s) stored in the S3 bucket.

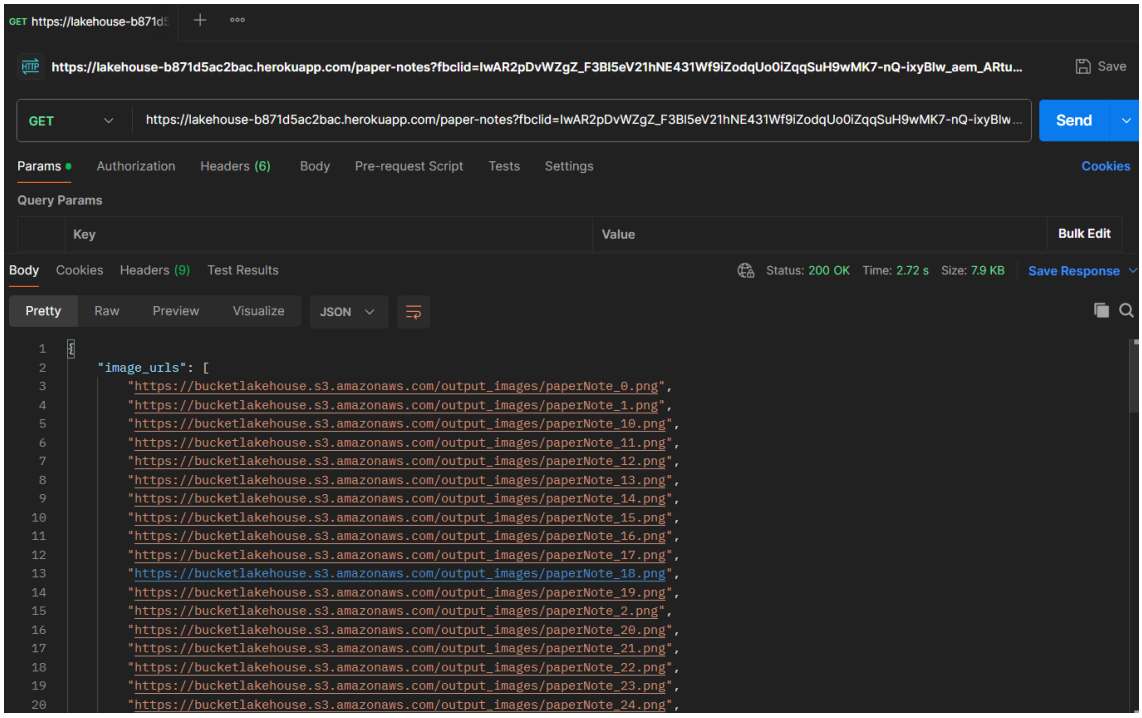

Figure: API Get all paper notes called from Postman

2.3 Generate data

Our team has implemented a Python script that generates fake paper notes data for a hypothetical scenario, such as customer orders with associated datetime details, customer details, order details, and randomly assigned staff details. This is source code to generate paper notes.

This source code implements a function to generate images from fake data generated above. All images in this folder will be uploaded to the s3 bucket as unstructured data prepared for this assignment.

Finally, we implement source code RESTful API to get all url of images stored in S3.

3

Technology

3.1 Data Lakehouse structure and technologies

Data lakehouses leverage a novel, open system design that adapts familiar data structures and management capabilities from data warehouses to the cost-effective storage typically associated with data lakes. This integration consolidates these functionalities into a unified system, streamlining data accessibility for teams. Consequently, data teams can work more efficiently, leveraging data without the need to navigate multiple systems. Furthermore, data lakehouses ensure that teams have access to the most comprehensive and current data for their data science, machine learning, and business analytics endeavors.

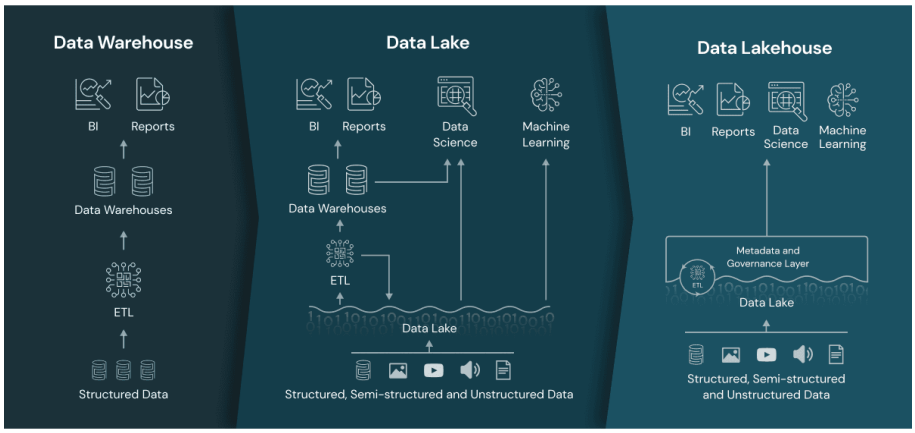

Figure 3.1: Data lakehouse vs data lake and data warehouse

Some key features of data lakehouse are:

Metadata Layers for Data Lakes: Metadata layers, such as the open-source Delta Lake, are positioned atop open file formats like Parquet files. They track the files constituting different table versions, offering robust management features including ACID-compliant transactions. These layers support additional functionalities typical in data lakehouses, such as streaming I/O (eliminating the necessity for message buses like Kafka), historical data retrieval, schema management, and data validation.

New Query Engine Designs: Traditional data lakes, stored on low-cost object stores, have historically suffered from sluggish accessibility. However, modern query engine designs now facilitate high-performance SQL analysis, caching frequently accessed data in RAM/SSDs (possibly transcoding it into more efficient formats), optimizing data layout to cluster co-accessed data, employing auxiliary data structures like statistics and indexes,

and executing vectorized operations on contemporary CPUs. These enhancements have brought data lakehouses' performance on large datasets to a level that competes with established data warehouses, as evidenced by TPC-DS benchmarks.

Optimized Access for Data Science and Machine Learning Tools: The open data formats used by data lakehouses, such as Parquet, greatly simplify data access for data scientists and machine learning engineers. These professionals can utilize popular tools in the data science and machine learning ecosystem, including pandas, TensorFlow, PyTorch, and others, which are already compatible with Parquet and ORC formats. Additionally, Spark DataFrames offer declarative interfaces for these open formats, enabling further input/output optimization.

Furthermore, features like audit history and time travel, inherent to data lakehouses, enhance reproducibility in machine learning projects. For further details on the technological advances driving the shift to data lakehouses, refer to the CIDR paper "Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics" and another academic paper titled "Delta Lake: High-Performance ACID Table Storage over Cloud Object Stores."

Data lakehouse is a new concept, hence there are many technologies can be used for create a data lakehouse. However, the core principle must be follow, that is: Metadata layers, such as the open-source Delta Lake, are positioned atop open file formats

3.2 Unstructured data processing - OCR

3.2.1 Optical Character Recognition

Optical Character Recognition (OCR) is the process that converts an image of text into a machine-readable text format. For example, if you scan a form or a receipt, your computer saves the scan as an image file. You cannot use a text editor to edit, search, or count the words in the image file. However, you can use OCR to convert the image into a text document with its contents stored as text data.

In this assignment, we have to process the papernotes that managers of each shop record the shipping conducted by the employees of the shops meanwhile the structured data is stored in the warehouse part of the data lakehouse. We created a mock version of the possible papernotes.

Papernotes of shops will be save on a server which represent the company's papernotes management system. Papernote will be fetch by the OCR app to push into the central data lakehouse. The papernotes will be pushed into the S3 object of the lakehouse as standard procedure of a datalake structure. Meanwhile the OCR app will process and push the processed OCR data into the warehouse.

Date time details :
Accepted at : 2024-02-19 12:15 PM
Completed at : 2024-02-19 03:15 PM
Boarded at : 2024-02-19 04:59 PM
Picked up at : 2024-02-19 06:27 PM

Customer details :
Name : Michael Perez
Address : 2166 Christine Keys
Phone : (574)451-9277
Email : andrewlittle@example.org

Order details :
Name : eat hope
Price : $82.95
Trip type : Round

Staff details :
Name : Emily Smith
Phone : 777-888-9999
Email : emily.smith@example.com
StaffID : SD26
Store : Fresh Fare

Figure: Example paper note

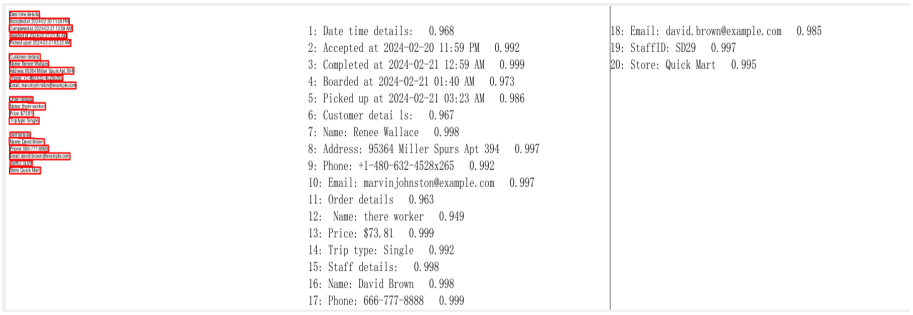

Figure 3.2: OCR output

We choose the PaddleOCR package, which contain the whole OCR pipeline for easier detection of the data. After extracted the data in the form of text string from the OCR, we use FuzzyMatching to fix any wrong character (if any).

Data after processed by the OCR pipline will be add to the ETL pipeline to push into the warehouse part. The structure of the papernotes are the same as the database for easier process data.

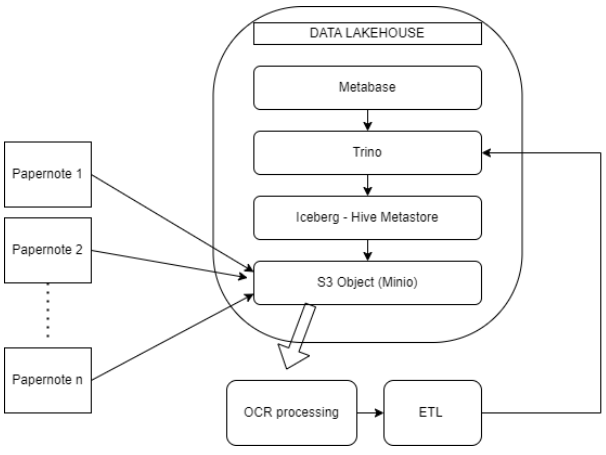

Figure 3.3: Dataflow of OCR component

This dataflow demonstrated the advantage of using a data lakehouse: We can use both **structured** and **unstructured** data types in a same storage layer: a S3 object, then we can query both structured and unstructured data by using single query engine.

3.3 Implementation

3.3.1 Data lakehouse technologies

These are the technologies we used in this assignment, the reasoning for the choice of each tech are discussed in each separated section below.

Figure 3.4: Our lakehouse technologies

3.3.1.1 Storage

For the storage layer, **Minio** has been selected, an open-source, S3-compliant object storage solution. This choice offers reduced operational costs compared to traditional data warehouses, allowing for the storage of various data types and formats. Importantly, since it is decoupled from our query engine we can scale it independently, if we need additional storage but not more processing power.

Key characteristics of MinIO encompass:

Object Storage Capability: MinIO offers a straightforward and scalable method for storing vast quantities of unstructured data in the form of objects within a flat namespace. Each object is uniquely identified by a key and can be accessed through HTTP/HTTPS protocols.

S3 Compatibility: MinIO seamlessly supports the Amazon S3 API, facilitating the utilization of applications and tools designed for S3 to seamlessly operate with MinIO as a substitute for S3 storage.

High Performance Optimization: MinIO is optimized for superior performance, leveraging contemporary hardware technologies such as multi-core processors and high-speed networking to achieve expedited data access with low latency and high throughput.

Distributed Architecture: MinIO can be deployed in a distributed manner, facilitating the dispersion of data across multiple nodes. This distributed architecture enhances performance and resilience against faults.

Implementation of Erasure Coding: MinIO incorporates erasure coding, a data protection technique distributing data and parity information across numerous drives or nodes to mitigate the risk of data loss.

Data Lifecycle Management: MinIO enables the establishment of rules governing the lifecycle of objects, including automated data transitions across storage tiers or expiration after a defined duration.

Encryption Capabilities: MinIO provides encryption mechanisms for both in-flight and at-rest data to ensure security during data transmission and storage.

Event Handling and Webhooks: MinIO supports event notification, empowering users to configure actions or triggers in response to specific events like object creation or deletion.

MinIO finds widespread utility across diverse applications, including backup and archival, content distribution, data lakes, and as a foundational storage component for cloud-native applications. Its open-source nature and compatibility with S3 contribute to its popularity and effectiveness in delivering scalable and economical object storage solutions.

Moreover, let's delve into some foundational components of MinIO:

Blob Representation: BLOB, an acronym for "Binary Large Object," assumes a pivotal role within the object storage ecosystem. MinIO leverages BLOBs as the primary method for storing a wide array of data types. BLOBs can accommodate various forms of unstructured data such as documents, images, audio files, videos, etc., owing to their flexibility in handling diverse data formats, sizes, and structures. Users can efficiently upload and access BLOBs based on established permissions, which include associated metadata facilitating streamlined data retrieval and manipulation processes.

Bucket Organization: MinIO organizes BLOB storage within conceptual entities known as "buckets." A bucket serves as a container encompassing related objects, policies, and configurations, akin to a volume in traditional file systems. Buckets offer a structured and efficient approach to managing BLOBs within the MinIO environment.

The seamless integration of MinIO with the S3 API enhances the utilization of BLOBs and buckets, fostering interoperability with S3-compatible services. This integration enables smooth migration and deployment of MinIO-based applications across various cloud platforms, ensuring consistent data storage and retrieval experiences across diverse environments.

Furthermore, the compatibility of MinIO with the Amazon S3 API stands out as a prominent feature. By aligning with the RESTful Object Storage Service provided by AWS,

MinIO presents a viable S3-compatible object storage solution.

The advantages of S3 compatibility are manifold, including the provision of a unified API compatible with any S3-compatible service, thereby facilitating application portability. MinIO enables code reuse, simplifying the migration process from AWS S3 and ensuring seamless compatibility.

S3 compatibility also promotes multi-cloud deployment strategies, facilitating integration with different cloud service providers such as Amazon, Google, Azure, and Digital Ocean. MinIO's flexible deployment options make it a preferred choice, endorsed by numerous software providers.

3.3.1.2 Table Formats

Table formats are instrumental in organizing data files and bringing database-like features to the data lake, a key distinction between a Data Lake and a Lakehouse. Apache Iceberg has been chosen for this project due to several advantages over Hudi and Delta tables. Iceberg offers faster insert and update operations, requires less storage, and facilitates easy partition and schema evolution across multiple query engines.

Apache Iceberg is an open table format designed for huge, petabyte-scale tables. The function of a table format is to determine how you manage, organise and track all of the files that make up a table. It is an abstraction layer between your physical data files (written in Parquet or ORC etc.) and how they are structured to form a table.

The project was originally developed at Netflix to solve long-standing issues with their usage of huge, petabyte-scale tables. It was open-sourced in 2018 as an Apache Incubator project and graduated from the incubator on the 19th of May 2020

Apache Iceberg keeps its records in object storage — unlike Apache Hive. Iceberg enables SQL behavior to be leveraged by multiple engines and it is designed for huge tables. In production, where a single table can contain tens of petabytes of data, this matters greatly. Even multi-petabyte tables can be read from a single node, without needing a distributed SQL engine to sift through table metadata.

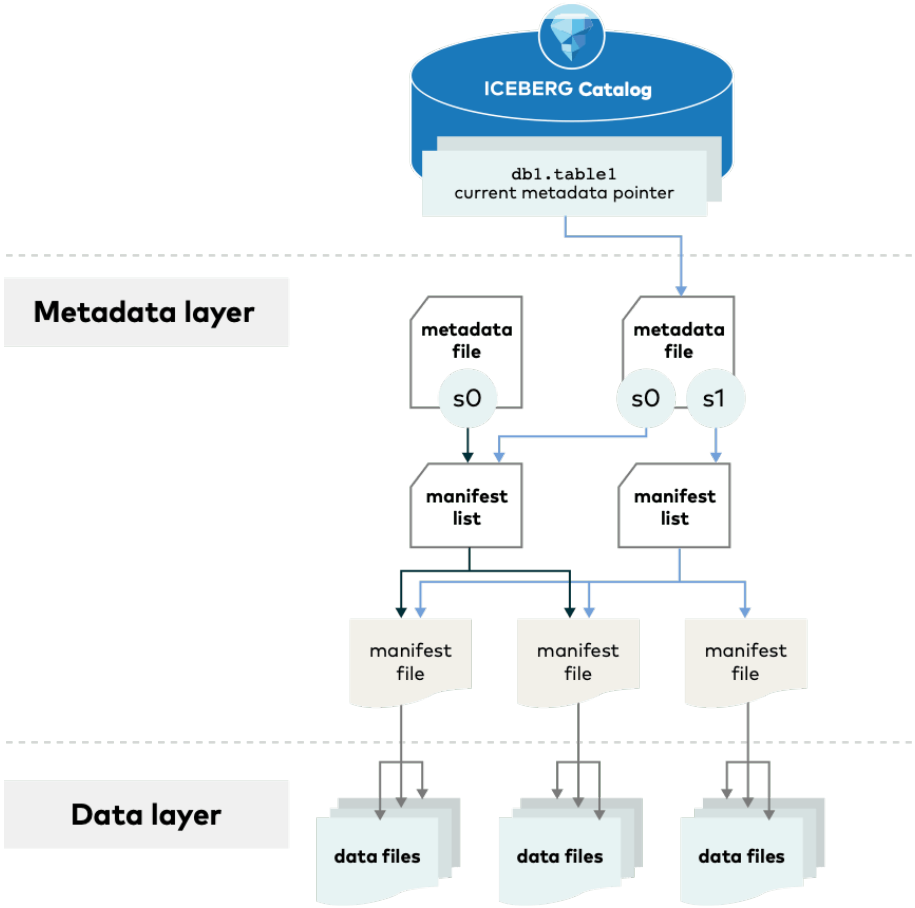

Figure 3.5: Iceberg data structure

Iceberg has been designed from the ground up to be used in the cloud and a key consideration was solving various data consistency and performance issues that Hive suffers from when used with data located in S3. Hive keeps track of data at the "folder" level (i.e. not the file level) and thus needs to perform file list operations when working with data in a table. This can lead to performance problems when many of these operations need to be executed. There is also the potential for data to appear missing when file list operations are performed on an eventually consistent object store like S3. Iceberg avoids this by keeping track of a complete list of all files within a table using a persistent tree structure. Changes to a table use an atomic object/file level commit to update the path to a new metadata file containing the locations to all the individual data files.

Another advantage of Iceberg tracking individual files rather than folders is that expensive list operations are no longer needed and this leads to performance improvements when performing operations like querying the data in the table. The table state is stored in a number of different metadata files which are represented in the diagram above, and described in more detail below:

Snapshot metadata file: contains metadata about the table like the table schema, the partition specification as well as a path to the manifest list.

Manifest list: contains an entry for each manifest file associated with the snapshot. Each entry includes a path to the manifest file and some metadata about the file, including partition stats and data file counts. These stats can be used to avoid reading manifests that aren't required for an operation.

Manifest file: contains a list of paths to related data files. Each entry for a data file includes some metadata about the file, such as per-column upper and lower bounds which can be used to prune files during query planning.

Data file: the physical data file, written in formats like Parquet, ORC, Avro etc. decorative separator

Benefits of using the table format Using the snapshot pattern means that Iceberg can guarantee isolated reads and writes. Readers will always see a consistent version of the data (i.e. no 'dirty reads') without the need to lock the table. Writers work in isolation, not affecting the live table and will perform a metadata swap only when the write is complete, making the changes in one atomic commit. Use of snapshots also enables time-travel operations as users can perform various operations on different versions of the table by specifying the snapshot to use.

There are huge performance benefits to using Iceberg as well. Instead of listing O(n) partitions in a table during job planning, Iceberg performs an O(1) RPC to read the snapshot. The file pruning and predicate pushdown can also be distributed to jobs so the Hive metastore is no longer a bottleneck. This also removes the barriers to using finer-grained partitioning. The file pruning available due to the statistics stored for each data file also speeds up query planning significantly.

Iceberg has an unwritten rule, to be invisible when being used in the Big Data stack. This philosophy comes from the SQL table space, where we never think of what is underneath the SQL tables. As any practitioner knows, this is simply not the case when working with Hadoop and Hive-like tables.

Iceberg keeps it simple in two ways. First, avoid unpleasant surprises when changes are made to tables. For example, a change should never bring back data that was deleted and removed. Second, Iceberg reduces context switching as what is underneath the table doesn't matter — what matters is the work to be done.

3.3.1.3 Schema Metastore

The Hive Metastore is utilized to store schemas, partitions, and other metadata of created tables in a central repository. In this project, MariaDB has been chosen as the backend relational database for the Hive Metastore. Although Apache Iceberg also supports a JDBC Metastore, which only works with Iceberg and restricts the creation of views, this option was not chosen to provide flexibility regarding table formats.

3.3.1.4 Query Engine

Trino serves as the query engine, enabling the creation and querying of tables based on flat files via Iceberg. It supports query federation, allowing for the joining of data from multiple sources. Trino is deployed in a Docker container and operates as a massively parallel processing database query engine, scaling vertically instead of increasing the processing power of a single node. Its connector-based architecture translates SQL statements into API calls, returning collections of rows in columnar format (pages) for processing by worker nodes based on the query or execution plan.

The deployment of Trino as a separate computational service, decoupled from storage, allows for easy scalability independent of data storage.

Trino is a SQL engine. More specifically, Trino is an open-source distributed SQL query engine for adhoc and batch ETL queries against multiple types of data sources. Not to mention it can manage a whole host of both standard and semi-structured data types like JSON, Arrays, and Maps.

3.3.1.5 BI Tool

Metabase has been added and pre-configured as a BI tool for direct interaction with the Lakehouse. Metabase serves as a robust open-source solution for data visualization and exploration, offering users the ability to seamlessly connect to various data sources, construct interactive dashboards, and derive insights with efficiency. Designed with accessibility in mind, Metabase caters to individuals across different proficiency levels, providing a user-friendly platform for data interaction.

Key features of Metabase include:

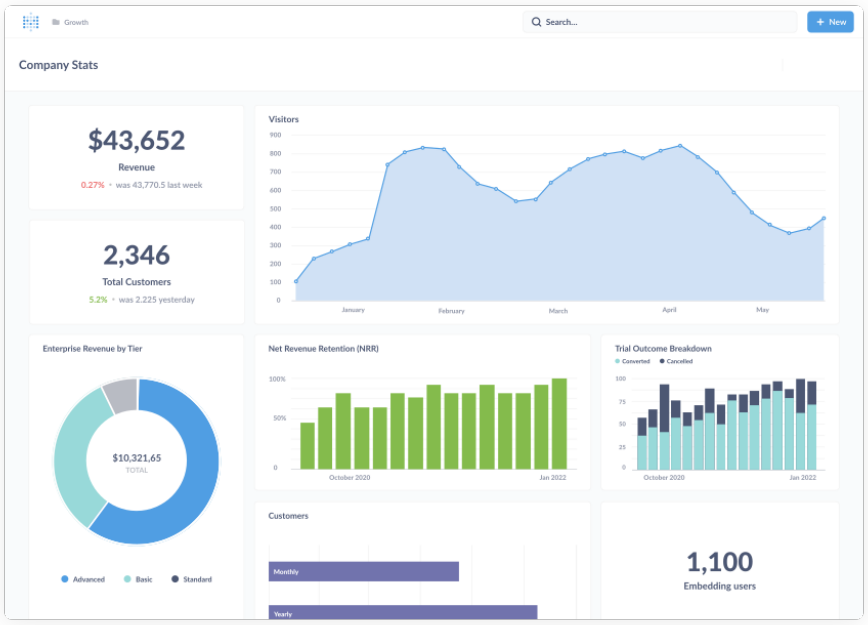

Figure 3.6: Example Metabase Dashboard

Simple Query Building: Metabase simplifies the process of generating SQL-like queries through an intuitive interface, eliminating the need for complex coding and enabling non-technical users to engage with their data visually.

Interactive Dashboard Creation: Users can leverage Metabase to construct dynamic and interactive dashboards, incorporating a variety of visualization elements such as charts, graphs, tables, and customizable filters to explore data from diverse perspectives.

Support for Multiple Data Sources: Metabase offers compatibility with an extensive array of data sources, encompassing databases like MySQL, PostgreSQL, and MongoDB,

as well as cloud services such as Amazon Redshift and Google BigQuery, ensuring seamless connectivity with existing data repositories.

Facilitated Data Exploration: Metabase facilitates in-depth exploration of data by enabling users to drill down, filter, group, and aggregate information, facilitating the rapid extraction of meaningful insights and identification of patterns and trends.

Intuitive Interface: Featuring a user-friendly interface, Metabase streamlines navigation and visualization creation, minimizing the learning curve for new users and enhancing overall usability.

Collaborative Data Sharing: Metabase facilitates the sharing of dashboards and reports among colleagues and stakeholders, fostering collaboration in data analysis endeavors and augmenting its utility as a collaborative tool.

3.3.2 Implementation

Link to source code can be found here.

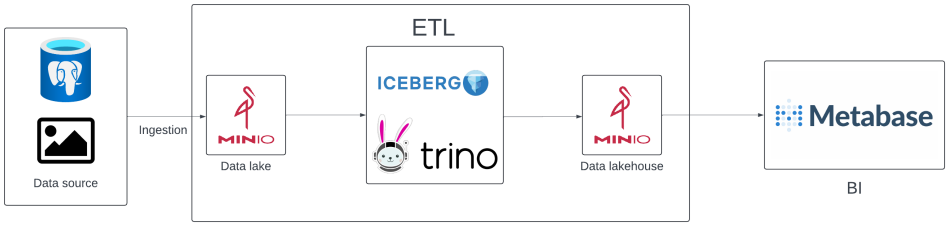

Figure 3.7: Data pipeline

3.3.2.1 Ingestion data

The data source comprises structured data stored in a PostgreSQL server and images of paper notes stored in an AWS S3 bucket. Each day, employees ingest this data from the source and store it in a Minio bucket, acting as a data lake. The structured data is stored in the Parquet format. We implement incremental ingestion to only add new data from the source to the data lake. In Minio, a folder is created for each table in PostgreSQL, with data partitioned by the current year, month, and day to facilitate access. Additionally, a folder named ”PaperNote” is created to store paper note images, also partitioned by the current year, month, and day.

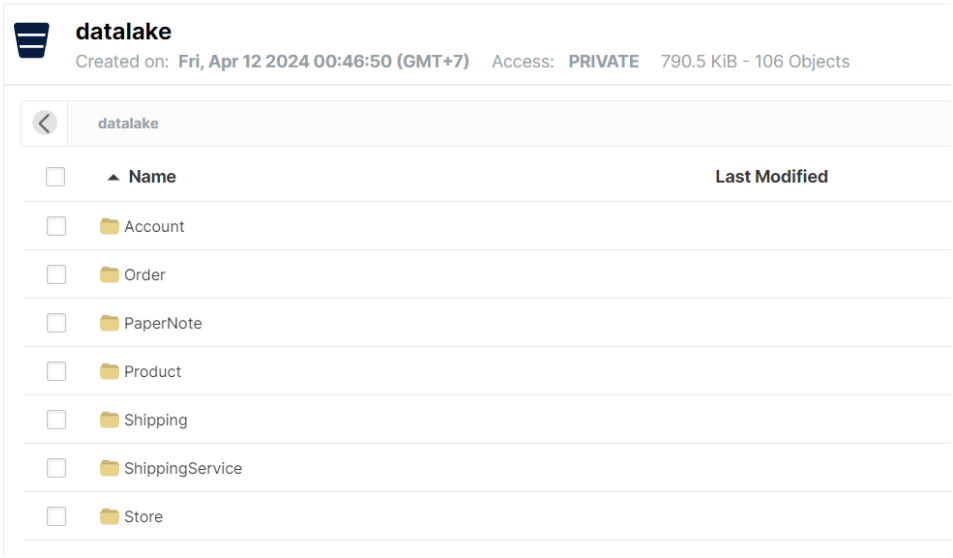

Figure 3.8: Minio bucket

3.3.2.2 ETL

- Extract: We load tables from the Minio bucket and store them into a Pandas DataFrame.
- Transform: We merge tables that we need and select features for BI. For paper notes, we detect the content of the image and add this data into a new table if necessary
- Load: We load the new table into the lakehouse using Trino and Iceberg. We create a folder named "Report" to store all new tables and create a separate folder for each specific report.

When we use Iceberg to load data to lakehouse, it will create for us a metadata layer.

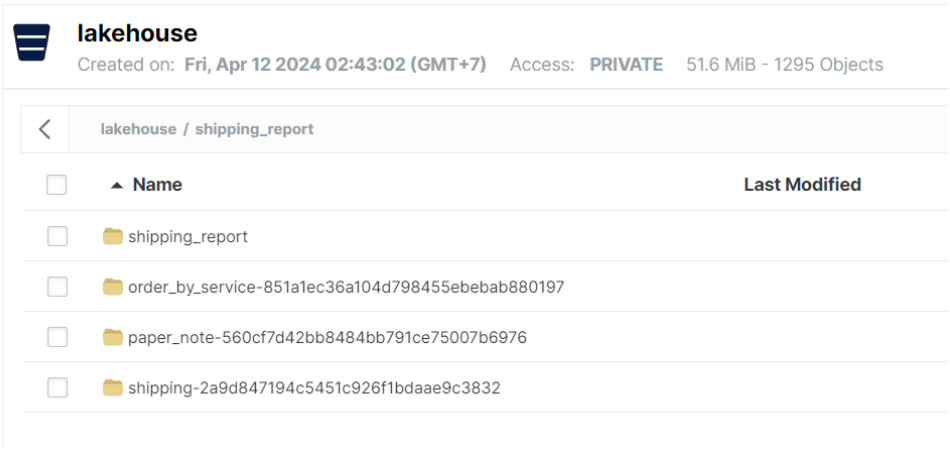

Figure 3.9: Folder in data lakehouse

4

BI and Report

Three tables we create in the previous strep is use for BI and report about performance of shipping. We use Metabase as our BI tools. It provides a lot of type of chart and WebUI for user to easy to interact. It also provides many features such as querying data, generating chart without coding, customizing charts, and more.

4.1 Cancel by store

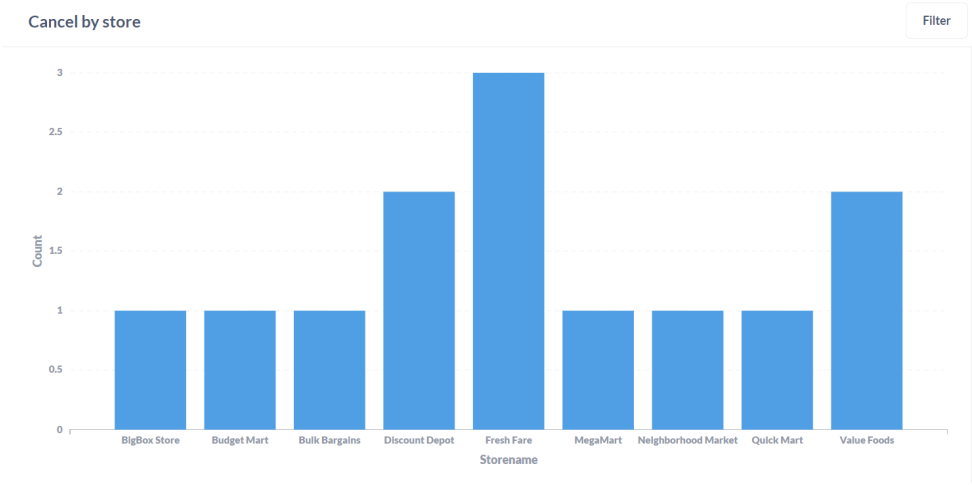

Figure 4.1: Cancelled by store

As a member of the Board of Directors, it's essential to investigate the high cancellation rate at the "Fresh Fare" store depicted in the image. Potential factors to explore include the quality of orders and the duration of shipping, among others.

4.2 Round trip and Single trip

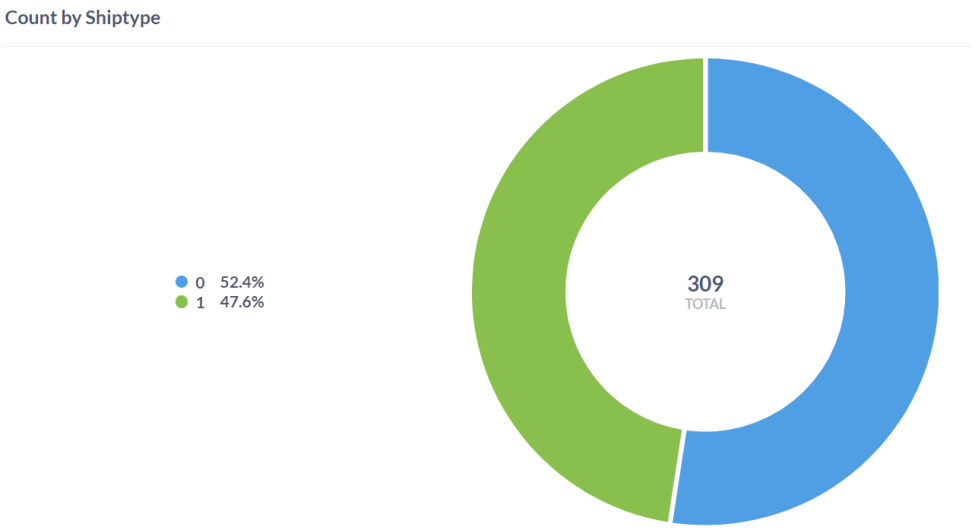

Figure 4.2: Round trip and Single trip

It show the number of round trip shipping and single trip shipping. 1 is denotes for round trip and 0 is denote for single trip.

4.3 Mean of duration of shipping service and staff

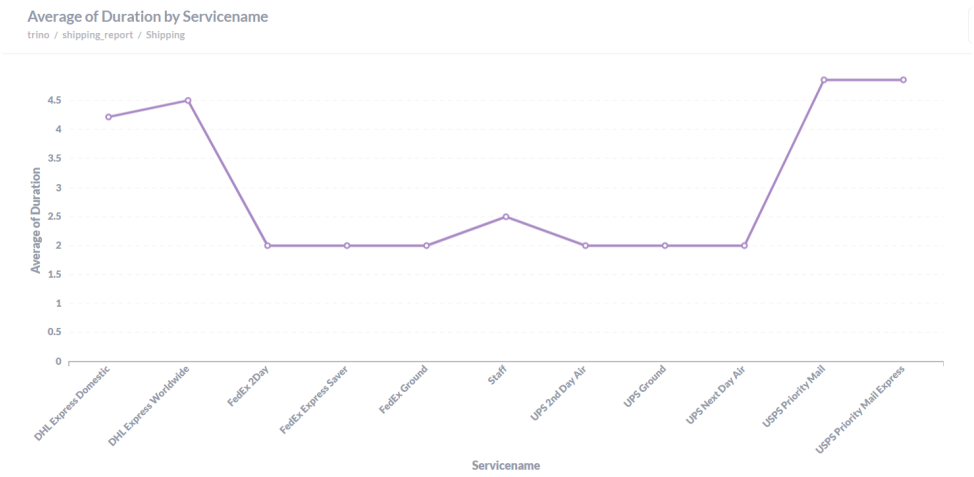

Figure 4.3: The number of delivery staff for each store

According to the chart, the delivery duration by our staff is shorter compared to most other services. We can optimize our staff resources to lower shipping costs and reduce delivery times. Additionally, we may consider addressing or discontinuing partnerships with services that consistently have longer delivery durations to enhance customer satisfaction.

4.4 The number of delivery staff for each store

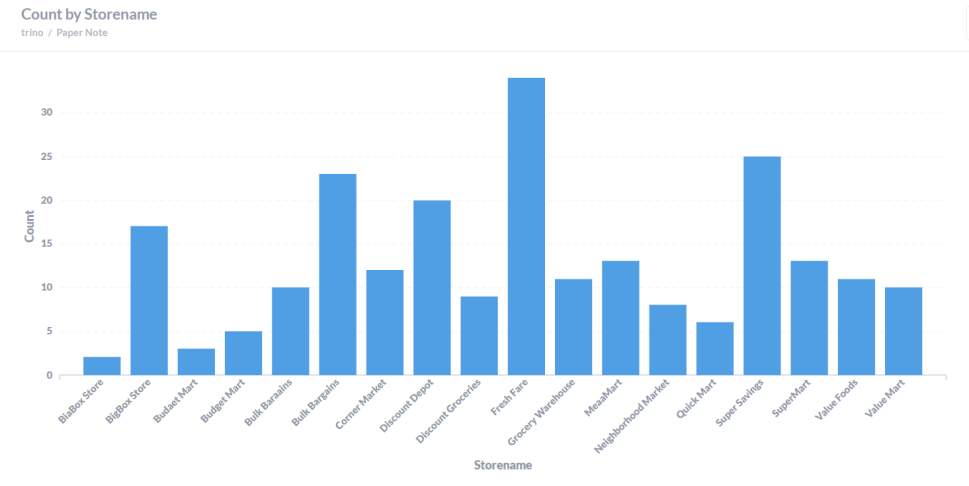

Figure 4.4: The number of delivery staff for each store

The chart indicates that "Fresh Fare" store has the highest number of delivery staff. This could suggest an imbalance in staffing levels at this location, prompting us to consider adjusting the number of staff assigned to delivery duties accordingly.

4.5 Income by date

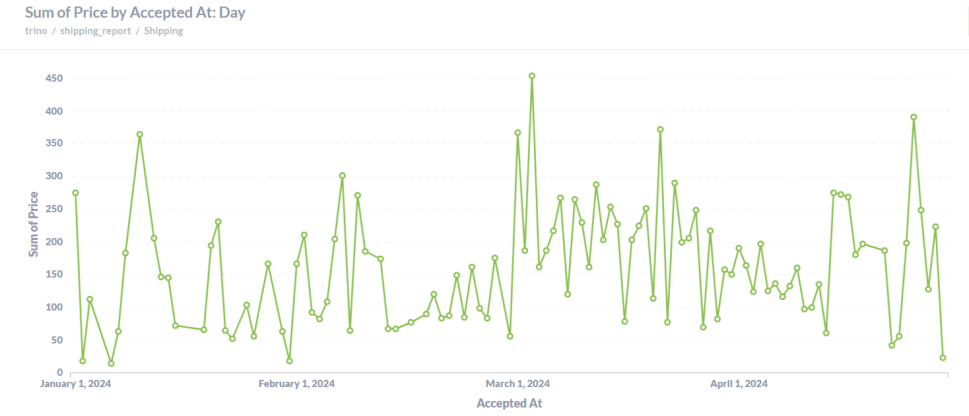

Figure 4.5: Income by date

The char shows income of all store by date.

4.6 Dashboarding

We can group multiple charts into the same location to facilitate the Board of Directors' tracking of shipping performance. Metabase also assists in dashboard creation. For

instance, we can utilize the above charts to design a straightforward dashboard.

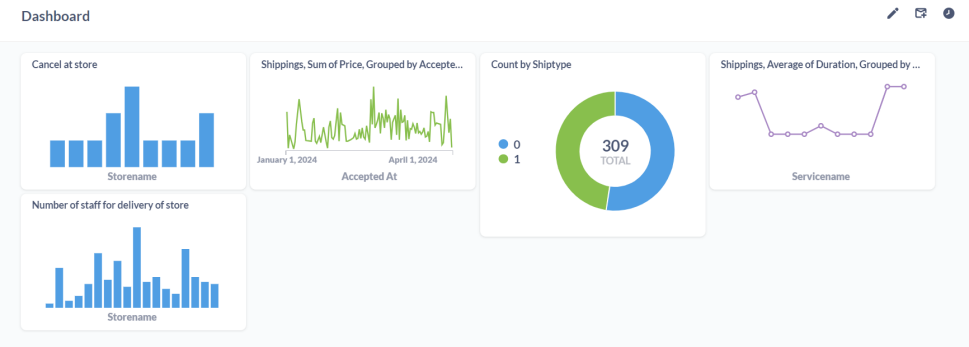

Figure 4.6: Dashboard

5

Conclusion

In conclusion, the emergence of data lakehouse represents a significant evolution in the field of data management and analytics. By combining the best attributes of data lakes and data warehouses, organizations can now achieve a unified platform that offers scalability, flexibility, and performance without sacrificing governance and reliability.

In this assignment, we presented the concept of a data lakehouse and how it addresses existing issues in data warehouses or data lakes. In a real-world transportation management problem, our team also proposed and implemented a data lakehouse architecture, data pipeline, and applied some BI techniques on the data lakehouse.

In addition to the features outlined in the report, our team identified some areas for improvement: using frameworks like PySpark for big data processing instead of Pandas to increase processing speed, and not implementing machine learning models for forecasting due to the fake data used in the assignment.

REFERENCES

[1] IBM. "What is a data lakehouse ?" (), [Online]. Available: https://www.ibm.com/topics/data-lakehouse (visited on 03/04/2024).

[2] Qlik. "What is a data lakehouse ?" (), [Online]. Available: https://www.qlik.com/us/data-lake/data-lakehouse (visited on 03/03/2024).

[3] Arvix. "Pp-ocr: A practical ultra lightweight ocr system." (2020), [Online]. Available: https://arxiv.org/abs/2009.09941 (visited on 03/02/2024).

[4] K. Pijanowski. "Building a data lakehouse using apache iceberg and minio." (), [Online]. Available: https://blog.min.io/building-a-data-lakehouse-using-apache-iceberg-and-minio/ (visited on 03/18/2024).

[5] HackerNoon. "Your definitive guide to lakehouse architecture with iceberg and minio." (), [Online]. Available: https://hackernoon.com/your-definitive-guide-to-lakehouse-architecture-with-iceberg-and-minio (visited on 03/14/2024).

[6] SettleDataGuy. "What is trino and why is it great at processing big data." (), [Online]. Available: https://www.theseattledataguy.com/what-is-trino-and-why-is-it-great-at-processing-big-data/ (visited on 03/15/2024).

[7] B. R. "Unleashing the power of minio: Your gateway to scalable object storage." (), [Online]. Available: https://medium.com/@rivera5656/unleashing-the-power-of-minio-your-gateway-to-scalable-object-storage-538fd18a059d (visited on 03/14/2024).