Applied Data Warehouse

Data Warehouse Project: A Beginner's Handbook

Amit Jaiswar
Data Scientist

Intended Audience

This handbook is thoughtfully curated for individuals who are new to the world of data and data warehouses. With sufficient content and references, it ensures that by the end of this journey, you will have a fair understanding of how organizations implement data warehousing.

As you go through the handbook, you'll learn about important topics and get practical tips to apply your knowledge. By the end, you'll feel more confident and ready to succeed in your career.

Index

Chapter 1

Database Paradigms : OLTP and OLAP

In the context of data management, OLAP (Online Analytical Processing) and OLTP (Online Transaction Processing) are two fundamental concepts that play distinct roles in handling and analyzing data. Both OLAP and OLTP are crucial for businesses to manage their operations efficiently and gain valuable insights. However, they serve different purposes and are optimized for specific tasks.

Let's explore each one separately:

OLTP (Online Transaction Processing)
OLTP systems are designed for managing and processing the day-to-day transactional operations of an organization. These operations include tasks such as inserting, updating, and deleting records in a database.
Examples- order processing systems, banking systems, and point-of-sale systems.

Characteristics:

- Focus on real-time data processing.
- Normalized database structure to minimize redundancy and ensure data integrity.
- Handles a large number of short and quick transactions.
- Optimized for read and write operations.

OLAP (Online Analytical Processing)
OLAP systems are designed for complex queries and analysis of large volumes of historical and aggregated data. They support decision-making processes by providing multidimensional views of data.
Examples- data warehouses, business intelligence systems, and data mining applications.

Characteristics:

- Focus on complex queries and analytics.
- Typically involves a denormalized data structure for faster query performance.
- Aggregates and consolidates data for efficient reporting.
- Optimized for read-intensive operations.

In summary, OLTP systems are optimized for efficient transaction processing and maintaining data integrity in real-time, while OLAP systems are geared towards supporting complex analytical queries and decision-making processes by providing fast access to large volumes of historical data. These two paradigms complement each other in a typical enterprise environment, with OLTP systems capturing transactional data and OLAP systems analyzing this data to derive insights and inform strategic decisions.

When to Use Which?
Understanding the circumstances under which OLTP or OLAP databases are most appropriate is crucial for effective database management.

OLTP	**OLAP**
• When you need to process a high volume of transactions in real-time.	• When you need to perform complex analytical queries on large volumes of historical data.
• For applications where maintaining data integrity and consistency is critical.	• For generating reports, data mining, and decision support.
• For systems that require quick response times for individual transactions.	• When the focus is on analyzing trends, patterns, and relationships within the data rather than individual transactions.

Table 1.1: Comparison of OLTP and OLAP

Chapter 2

Data Buzzwords

In the recent world of data management and analytics, numerous buzzwords and keywords have emerged. These words represent the newest ideas and trends in storing and managing data. Let's look at some important data keywords like data lake, delta lake, data mart, data warehouse, lake house, and data mesh.

Figure 2.1: Data Keywords

Data Lake: A data lake is a centralized repository that stores vast amounts of structured, semi-structured, and unstructured data at any scale. Data lakes accommodate raw and unprocessed data, making it more flexible and accessible for data exploration and analytics. The data lake architecture allows organizations to capture diverse data types from various sources, providing a foundation for advanced analytics and data-driven insights.

Delta Lake: Delta Lake is an open-source storage layer that sits on top of a data lake, bringing ACID (Atomicity, Consistency, Isolation, Durability) transactions to big data workloads. It enhances the reliability and performance of data processing in data lakes, enabling real-time analytics and streamlining data pipelines.Additionally, Delta Lake ensures data consistency and data quality.

Data Mart: A data mart is a subset of a data warehouse, focusing on a specific business function, department, or user group. It is designed to serve the needs of a particular set of users, providing them with tailored data models and access to relevant information for their analytical requirements. Data marts simplify data accessibility and analysis for end-users, allowing them to make informed decisions based on their specific domain.

Data Warehouse: A data warehouse is a centralized and integrated repository that stores structured data from various sources, enabling business intelligence and data analytics. It serves as a single source of truth, where historical and current data is stored, transformed, and organized for querying and reporting. Data warehouses play a crucial role in business decision-making, offering valuable insights to stakeholders across the organization.

Data Lakehouse: A data lake house is a modern data architecture that combines the features of a data lake and a data warehouse. It typically uses a unified platform for storing both structured and unstructured data in its raw format, similar to a data lake. However, it also integrates elements of a data warehouse by organizing and structuring the data to facilitate easier querying and analysis. Additionally, it offers near real-time analytics and simplified data management.

Data Mesh: Data Mesh is a paradigm shift in data architecture that decentralizes data ownership and delivery. It advocates for domain-oriented decentralized data teams, where data producers and data consumers collaborate through self-serve data infrastructure. This approach reduces bottlenecks, improves data quality, and enhances the overall data ecosystem within an organization.

Data Governance: Data Governance is the framework that ensures data is managed, used, and protected optimally across the organization. It encompasses policies, processes, and practices that maintain data quality, security, compliance and usability. Effective data governance brings data trustworthiness, accountability, and strategic decision-making. It empowers organizations to harness the full potential of their data while ensuring ethical use, regulatory compliance, and the realization of data-driven goals.

In summary, the world of data warehousing is filled with buzzwords and innovative concepts that continually reshape how organizations handle and utilize their data. Therefore, by staying informed, businesses can maximize the power of data analytics to drive growth and achieve success in today’s dynamic landscape.

Chapter 3

Data Warehouse

A data warehouse is a centralized repository designed to store large volumes of structured and occasionally unstructured data from diverse sources, emphasizing a unified, historical view. It creates an environment, aligned with OLAP principles, that optimizes data for analytical insights, ensures efficient query performance, and facilitates user-friendly interfaces for exploring and extracting valuable information to support business intelligence and decision-making processes.

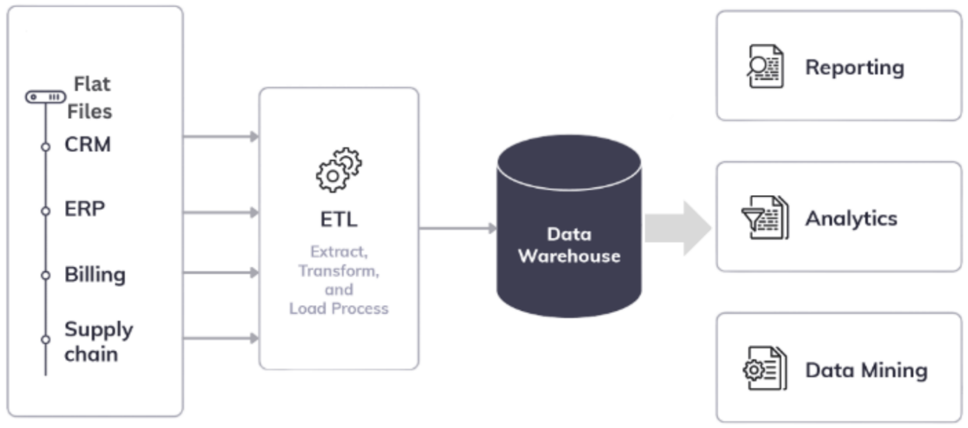


Figure 3.1: Data Warehouse

Need for Data Warehousing:

- **Data Integration:** Organizations often have data scattered across multiple systems and databases. A data warehouse integrates data from various sources, providing a single source of truth.
- **Historical Analysis:** Traditional databases focus on current data, whereas data warehouses store historical data. This enables users to analyze trends, track changes over time, and make informed decisions based on historical patterns.
- **Performance:** Data warehouses are optimized for query performance, making it easier and faster to retrieve and analyze large volumes of data compared to transactional databases.
- **Business Intelligence and Analytics:** Data warehouses are the foundation for business intelligence (BI) and analytics. They support the generation of reports, dashboards, and data visualizations that aid in decision-making.

Key characteristics of a data warehouse:

- **Subject Orientation:** Data in a data warehouse is organized around key business subjects or areas, such as sales, finance, or customer data.
- **Integration:** Data from different sources is integrated into a common format to ensure consistency and accuracy.
- **Time-variant:** Data warehouses store historical data, allowing users to analyze trends and changes over time.
- **Non-volatile:** Once data is loaded into the data warehouse, updates or deletions are infrequent. Instead, it retains a historical record of changes, with volatility varying based on the data nature and business requirements.

Let's see the practical need for data warehousing and how a data warehouse will help.

Consider a retail company that wants to analyze its sales performance. The company has data coming in from various sources, including point-of-sale systems, ERP, supply chain and customer relationship management (CRM) tools.

In the absence of a data warehouse, analyzing this scattered data would be time-consuming and challenging. However, with a data warehouse in place, the company can integrate data from all these sources into a centralized repository. This allows them to:

- Analyze sales trends across different regions and time periods.
- Identify top-selling products and customer preferences.
- Evaluate the effectiveness of marketing campaigns over time.
- Make informed decisions about inventory management and supply chain optimization.

Here, the data warehouse provides a comprehensive and historical perspective on sales data, allowing the company to extract valuable insights for strategic decision-making.

Learn more from here

- https://www.oreilly.com/library/view/data-warehouse/
- https://app.datacamp.com/learn/data-warehousing/
- https://www.youtube.com/dwh/

Chapter 4

Data Warehouse Architecture

Data warehouse architecture refers to the structure and components of a data warehousing system designed to efficiently collect, store, and manage large volumes of data from various sources for analysis and reporting purposes. A well-designed Data Warehouse architecture serves as the backbone for effective data analysis and decision-making.

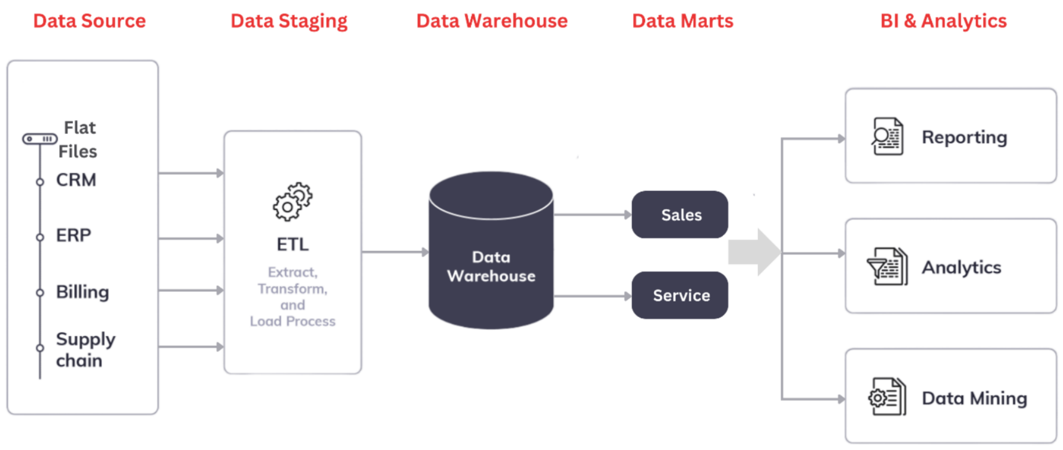


Figure 4.1: Data Warehouse Architecture

Here's a high-level overview of typical components and layers found in a data warehouse architecture:

1. Components of Warehouse:

1. **Source Systems:**
   The journey of data within a Data Warehouse begins with source systems, which can include various databases, applications, and external data feeds. These systems generate the raw data that will be processed and analyzed in the Data Warehouse.
2. **Data Processing:**
   The Extract, Transform, Load (ETL) process extracts data from source systems, transforming it into a suitable format for analysis. ETL tools play a pivotal role in this phase, ensuring the efficient and accurate movement of data.
3. **Staging Area:**
   The staging area serves as an intermediate storage space where raw data is temporarily held before undergoing further processing. This step allows for data

validation, cleansing, and transformation before it is loaded into the Data Warehouse.

4. **Data Warehouse Database:**
   At the core of Data Warehouse architecture lies the data warehouse database. This database is optimized for analytical queries and typically follows a dimensional model, incorporating tables like fact tables and dimension tables. Common database technologies include SQL Server, Oracle, and Snowflake.

5. **Data Mart:**
   Data marts are subsets of the overall Data Warehouse, designed to cater to specific business units or departments. They allow for a more focused and streamlined approach to data analysis, enhancing performance for targeted queries.

6. **Business Intelligence Layer:**
   The Business Intelligence layer is positioned on top of the Data Warehouse and provides tools and interfaces for end-users to interact with and analyze data. BI tools, such as Tableau, Power BI, or Looker, enable the creation of dashboards, reports, and visualizations.

By leveraging the components outlined above, organizations can construct a robust and effective data warehouse infrastructure capable of meeting their analytical and decision-making needs.

2. Introducing Medallion Storage Architecture

In recent years, the exponential growth of data has challenged traditional storage systems. As organizations collect petabytes of data, the demand for scalable, cost-effective, and high-performance storage solutions grows and this is where Medallion storage architecture comes into play.

A medallion architecture is a data design pattern used to logically organize data in a lake house, with the goal of incrementally and progressively improving the structure and quality of data as it flows through each layer of the architecture (from Bronze $\implies$ Silver $\implies$ Gold layer tables). Medallion architectures are sometimes also referred to as "multi-hop" architectures.

Bronze Layer

The Bronze layer serves as the initial landing ground for data streaming from external source systems. Here, data is stored in its raw, unaltered form, preserving its original structure and integrity. This layer maintains table structures mirroring those of the source systems, supplemented with metadata capturing vital information like load timestamps and process IDs. The primary focus of the Bronze layer is on facilitating quick Change Data Capture and maintaining a historical archive of source data. It ensures data lineage, auditability, and facilitates reprocessing without re-reading from the source systems, laying a robust foundation for subsequent data processing.

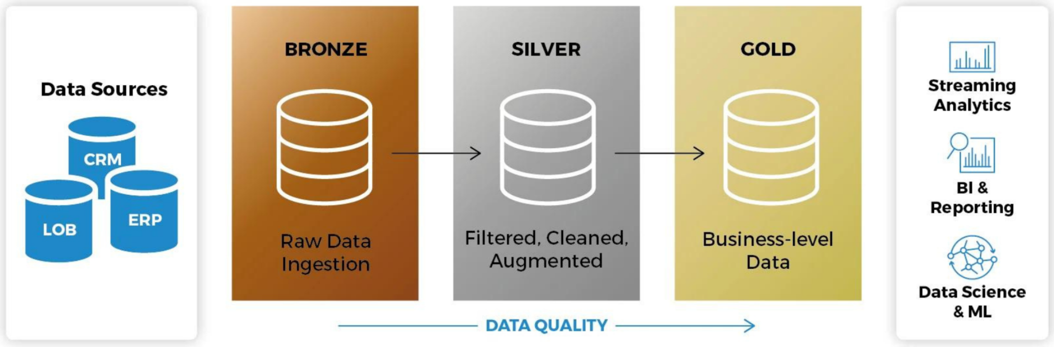


Figure 4.2: Medallion Architecture

Silver Layer
In the Silver layer, the raw data from the Bronze layer undergoes a transformational journey. Data is matched, merged, cleansed, and conformed to create an "Enterprise view" of key business entities and transactions. This layer harmonizes data from diverse sources, enabling an integrated view for self-service analytics, ad-hoc reporting, and advanced analytics, including Machine Learning (ML). While loading the Silver layer, emphasis is placed on speed and agility, with minimal transformations applied. The Silver layer acts as a springboard for departmental analysts, data engineers, and data scientists to undertake further analysis and projects, paving the way for informed decision-making.

Gold Layer
At the top of the lake house architecture resides the Gold layer, where data is curated into consumption-ready "project-specific" databases. This layer is dedicated to reporting and employs denormalized, read-optimized data models with fewer joins for enhanced performance. Here, final transformations and data quality rules are applied, culminating in the presentation layer of various projects such as Customer Analytics, Product Quality Analytics, and Sales Analytics. The Gold layer accommodates Kimball-style star schema-based data models or Inmon-style Data marts, providing a robust foundation for advanced analytics and decision support.

In summary, the adoption of modern Medallion architecture offers an effective approach to managing data storage. This strategy, when integrated with data warehouse architecture, not only enhances efficiency but also facilitates scalability, empowering organizations to adapt and grow in today's data-driven landscape.

Impact of Medallion Architecture on Modern Data Warehousing Practices
In modern data warehousing practices, similar concepts to the bronze/silver/gold layers may exist, although they might not always be referred to using the same terms. However, the underlying principles of refining and processing data progressively are commonly observed.

Here's how these concepts might manifest in modern data warehousing:

Staging Area: (Bronze Layer Equivalent):

- The staging area in a data warehouse is similar to the bronze layer, where raw data is initially loaded before any significant processing occurs.
- Raw data from various sources is ingested into the staging area without much transformation, maintaining its original form and structure.
- The staging area serves as a landing zone for incoming data, providing a temporary storage area before further processing.

Data Integration/Transformation Layer (Silver Layer Equivalent):

- After data is staged, it undergoes transformation and integration processes in this layer.
- Similar to the silver layer, data in this stage is cleaned, normalized, and transformed to ensure consistency and quality.
- Data integration tools and processes are utilized to merge and consolidate data from different sources, applying business rules and transformations as needed.
- This focuses on preparing the data for analysis by refining its quality and structure.

Data Warehouse/Analytical Layer (Gold Layer Equivalent):

- The final layer in modern data warehousing typically corresponds to the gold layer, where processed and refined data resides for analysis.
- Data in this layer is structured according to predefined schemas and is optimized for querying and analytical purposes.
- Aggregations, summarizations, and other optimizations may be applied to the data to improve query performance and support business intelligence activities.

- This layer serves as the foundation for reporting, dashboarding, and decision-making processes within the organization.

In summary, the terminology may vary, the concept of progressively refining and processing data from its raw form to a curated, structured state remains fundamental in modern data warehousing architectures. Each layer serves a specific purpose in the data pipeline, contributing to the overall effectiveness of data management and analytics.

3. Data Modeling in Data Warehousing

Data modeling refers to the process of designing and structuring data to meet the needs of an organization. It encompasses various methodologies, techniques, and approaches used to define how data is organized, stored, and accessed within a database or data warehouse environment. Although it is a highly subjective process, allowing for the customization of data warehousing solutions according to individual needs and preferences.

Let's discuss the most commonly used modeling practices:

1. Schema Modeling
2. Fact and Dimensional Modeling

Schema Modeling

Schema modeling refers to the architectural design or structure of the data warehouse. It defines how tables are organized, the relationships between tables, and the overall layout of the data warehouse. These models also determine the physical structuring of the data within the warehouse and how users and applications access it.

Common schema models include the Star schema, Snowflake schema, and Galaxy schema.

- **Star Schema:** In the star schema, a central fact table is connected to multiple-dimension tables. This design simplifies query complexity and enhances performance by providing a denormalized structure for efficient data retrieval.
- **Snowflake Schema:** The snowflake schema extends the star schema by normalizing dimension tables, reducing redundancy. While this offers benefits in terms of data integrity and storage efficiency, it may introduce additional joins in queries.
- **Galaxy Schema:** A galaxy schema is a hybrid of the star and snowflake schemas, allowing for both denormalized and normalized dimension tables. This model offers flexibility in balancing between query performance and data normalization.

Considerations:

- If your priority is normalization, the snowflake schema would be preferred.
- If your priority is optimized query performance with simplified data retrieval and analysis, the star schema would be preferred.
- Star schema is more commonly used in data warehousing environments compared to the snowflake schema because of its simplicity, better query performance, and widespread adoption in the industry.

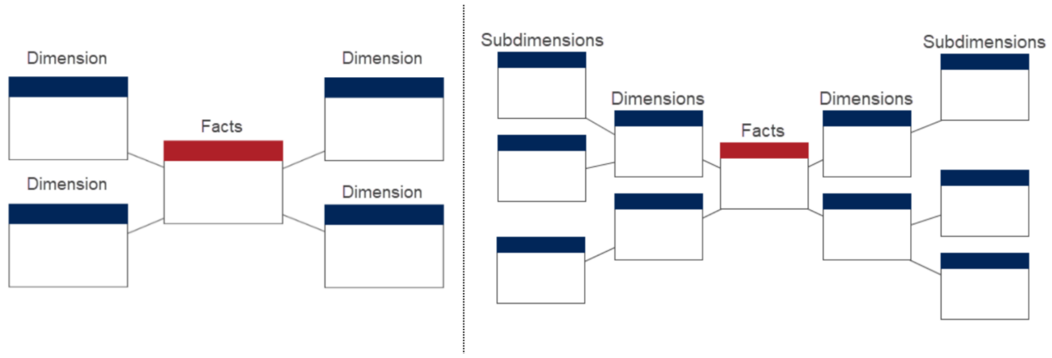


Figure 4.3: Star Schema Vs Snowflake Schema

Fact and Dimensional Modeling
Fact and dimensional modeling, on the other hand, are methodologies used within schema modeling to structure the data for analytical purposes. Fact modeling focuses on identifying and organizing the numerical data or metrics (facts) representing business events or transactions. Dimensional modeling involves organizing descriptive attributes related to the dimensions of the business (e.g., time, geography, product) to provide context for the facts.

The models mentioned above, such as the Star and Snowflake schema, are implementations of fact and dimensional modeling principles within the broader schema modeling framework. Hence, it can be inferred that they are inherently interconnected.

Considerations:

- The granularity of facts and dimensions should be carefully defined to strike a balance between detail and usability, here the grain represents the level of detail at which facts are recorded and dimensions are described.
- Fact and dimensional models should be flexible enough to accommodate changes

in business requirements and evolving analytical needs including the ability to add new dimensions or metrics without disrupting existing data structures.

How does dimensional modeling help?

- Simplified Analysis: Fact and dimensional modeling provide a simplified and intuitive data structure that makes it easier for end-users to query, analyze, and interpret data.
- Performance: By organizing data into fact and dimension tables, dimensional modeling optimizes query performance and enables fast, efficient analytical processing.
- Scalability: Fact and dimensional models are designed to scale with the growth of data and the evolving needs of the business, ensuring long-term viability and usability of the data warehouse.

In summary, schema modeling defines the overall architecture of the data warehouse, while fact and dimensional modeling are methodologies used within schema modeling to structure the data for analytical purposes. They work together to ensure that the data warehouse is optimized for querying and analysis.

4. Designing Data Warehouse:
Designing a data warehouse is a complex and crucial task for organizations aiming to leverage their data effectively for business insights and decision-making. A well-designed data warehouse lays the foundation for streamlined data management, efficient analytics, and actionable intelligence. However, when it comes to implementing a Data Warehouse, two common approaches are the Top-Down approach and the Bottom-Up approach.

Let's explore them:

Top-Down Approach : Inmon Model
In the Top-Down approach, the focus is on designing the overall architecture and structure of the Data Warehouse before dealing with specific data elements. It starts with a high-level view and gradually drills down into details.

Process Flow:

- Business Requirements Analysis: Understand the overall business requirements and goals.

- Data Warehouse Design: Design the architecture, data models, and framework.
- Data Extraction and Transformation: Implement the ETL processes for moving and transforming data.
- Loading Data: Populate the Data Warehouse with transformed data.
- Business Intelligence and Reporting: Develop tools and interfaces for end users to access and analyze data.

Characteristics:

- EnterpriseWide Perspective: This approach takes into account the entire organization's data and business needs.
- Comprehensive Planning: It involves extensive planning and design at the beginning of the project to establish an overarching framework.
- Centralized Control: The development process is centrally controlled, ensuring consistency and adherence to the defined architecture.

This implementation approach aligns with long-term organizational goals and ensures a comprehensive and integrated view of organizational data while providing centralized control to maintain consistency and standards. However, it can be time-consuming and resource-intensive, and flexibility may be limited when adapting to evolving business needs.

BottomUp Approach : Kimball Model
The BottomUp approach, in contrast, begins with individual departmental or business unit data marts and then integrates them into an enterprise-wide Data Warehouse. It starts with specific data elements and builds upwards.

Process Flow:

- Identify Business Unit Needs: Understand the specific data needs of individual departments.
- Develop Data Marts: Create smaller-scale data marts tailored to each business unit.
- Integrate Data Marts: Gradually integrate data marts into an enterprise-wide Data Warehouse.
- Expand and Enhance: Continue expanding the Data Warehouse based on additional business unit requirements.

Characteristics:

- Departmental Focus: Emphasizes the needs of specific departments or business units.
- Incremental Development: Data marts are developed and integrated one at a time, allowing for gradual expansion.
- Fast Deliveries: Faster delivery of results for specific business units.

This implementation approach offers flexibility and adaptability to changing business needs, facilitating faster delivery of results for specific business units. Additionally, it requires fewer resources at the outset. However, integration challenges may arise when combining individual data marts into an enterprise-wide solution, potentially leading to data redundancy if not well managed.

Choosing between the top-down and bottom-up approaches depends on organizational goals, resources, and the preferred balance between centralized control and departmental flexibility. Often, a hybrid approach that combines elements of both methods is employed to achieve the best of both methods.

Things to Consider While Designing a Data Warehouse Architecture

- **Scalability:** A robust Data Warehouse architecture should be scalable to accommodate growing data volumes and user demands. Additionally, it should scale in both horizontal and vertical directions based on specific project requirements.
- **Performance Optimization:** Implementing indexing, partitioning, and optimizing queries are crucial for maintaining optimal performance. Regular performance tuning ensures that the Data Warehouse remains responsive to analytical queries.
- **Metadata Management:** Efficient metadata management is vital for documenting data lineage, transformations, and business rules. This information facilitates data governance, auditing, and troubleshooting.
- **Data Governance:** Data Warehouse architecture must adhere to stringent security measures to safeguard sensitive information. It must include encryption, access controls, and audit trails, data quality, ownership, and accountability.

Chapter 5

Change Data Capture (CDC)

In today's data-driven world, businesses are generating huge amounts of data every day. To make informed decisions and gain valuable insights, it is crucial to keep track of data changes in real-time. This is where Change Data Capture becomes needful. CDC is a technique that enables organizations to capture and propagate data changes across various systems and applications.

What is Change Data Capture (CDC)?
Change Data Capture is a utility that is used to identify and capture the changes made to data in a database. It captures the modifications, additions, and deletions made to specific tables or rows, and then propagates those changes to other systems or downstream applications. By capturing and delivering real-time data changes, CDC helps maintain consistency and synchronicity among different databases or data stores.

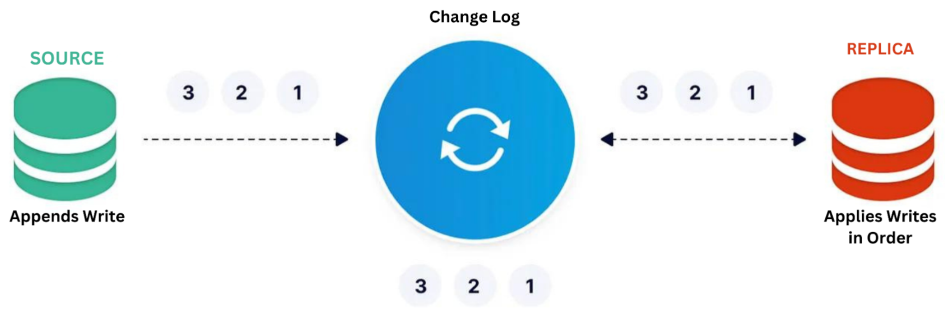


Figure 5.1: Change Data Capture

How Does CDC Work?
Change Data Capture (CDC) involves the following steps:

1. Identify Changes: Identify the data changes from the database (transaction) logs.
2. Extract Changes: Extract relevant data changes identified from the database logs.
3. Transform Changes: Extracted changes are transformed and formatted to match target systems or applications.
4. Propagate Changes: Transformed data changes are transmitted to target systems or applications using various methods.
5. Apply Changes: Target systems or applications receive and apply the propagated changes, ensuring data synchronization.

1. Methods to Capture Change Data

There are various methods of CDC; let’s explore each one of them.

Audit Column-Based CDC

By using the ‘Last_Update’ or ‘Last_Modified’ columns, one can create their own change data capture solutions. This approach only captures the rows that have been changed since the data was last extracted.

This method offers the benefits of being built with native application logic and not relying on external tools, but it has drawbacks including increased database overhead and additional scripts to **track deletes.**

Delta Table-Based CDC

This method involves comparing the data in two tables to identify any differences. Then, additional scripts can be used to apply the changes from the source table to the target table, providing an alternative way to capture and track data changes.

This approach offers the advantage of accurately tracking changed data using native SQL scripts, but it has the disadvantage of significantly increasing storage demand due to the need for copies of the data sources, lacks scalability for heavy transactional workloads, requires significant CPU resources for identifying differences, introduces latency, and cannot be performed in real-time.

Trigger-Based CDC

By utilizing database triggers and creating a custom change log in shadow tables, where triggers are activated before or after INSERT, UPDATE, or DELETE commands to track changes. While this method operates at the SQL level and is preferred by some users, it has the drawback of requiring triggers for each table in the source database, resulting in increased overhead and performance impact during ongoing operations. Furthermore, managing and maintaining these triggers as the application evolves can become a management burden.

However, a major drawback is that it significantly reduces database performance due to the requirement of multiple writes for every row insertion, update, or deletion.

Log-Based CDC

Log-based change data capture involves reading new database transactions, such as inserts, updates, and deletes, directly from the native transaction logs of source databases.

This approach allows for capturing changes without impacting the application or scanning operational tables, thereby reducing workload and preserving the performance of source systems.

This approach also offers advantages such as minimal impact on the production database, no need for additional queries per transaction, maintaining ACID reliability across multiple systems, and avoiding schema changes or extra tables in the production database.

Challenges in this approach include understanding the complex and undocumented logging format, adapting to changes in releases, managing metadata for change events, and a slight impact on performance due to log levels.

Each method of Change Data Capture (CDC) possesses its own set of advantages and disadvantages, and selecting the appropriate CDC method relies on considerations such as data volume, performance criteria, and the complexity of the data environment. Among these methods, Log-Based CDC is widely favored as one of the most commonly used approaches in today's world.

2. Managing CDC using Slowly Changing Dimension (SCD) Techniques:
SCD stands for Slowly Changing Dimension. It refers to a method used in data warehousing to manage changes to data over time. SCD techniques are employed when data in a database dimension needs to be updated, tracked, or historicized as it changes over time. There are different types of SCDs, commonly categorized as Type 1, Type 2, and Type 3, each serving different purposes in capturing and preserving historical data changes.

SCD Type 1: Overwriting Existing Data

- In SCD Type 1, the existing data is updated with new values. This approach overwrites the previous data and retains only the most recent values. SCD Type 1 is suitable when historical tracking is not required, and the focus is solely on the current state of the data. It is simple to implement, has low storage overhead, and is ideal for scenarios where maintaining historical information is not crucial.

SCD Type 2: Preserving Historical Versions

- SCD Type 2 involves creating new records to preserve the history of changes. Each change results in a new version of the dimension record, distinguished by a unique identifier or timestamp. This technique allows organizations to track historical changes and perform historical analysis and reporting. Although it requires increased storage and complex queries, SCD Type 2 is valuable when comprehensive historical tracking is needed.

SCD Type 3: Limited Historical Tracking

- SCD Type 3 captures changes by adding additional columns to the dimension table to store previous attribute values. Rather than creating new records, this technique retains limited historical information by maintaining both current and previous attribute values. SCD Type 3 strikes a balance between storage efficiency and historical tracking. It provides some historical context without incurring the storage requirements of SCD Type 2.

The choice of SCD technique depends on the specific requirements of the organization, data analysis needs, and storage considerations. While SCD Type 1 offers simplicity and low storage overhead, SCD Type 2 is suitable for comprehensive historical tracking, and SCD Type 3 balances storage efficiency and historical context.

3. Implementing SCD Techniques in Snowflake

Snowflake is a powerful data warehousing platform that supports the MERGE statement, which simplifies updating dimension records based on specific conditions. The MERGE statement combines INSERT, UPDATE, and DELETE operations into a single SQL statement, providing a streamlined approach to SCD1 updates.

3.1 Implementing (SCD1) in Snowflake: Slowly Changing Dimension Type 1

As mentioned earlier, SCD1 represents the most straightforward form of Slowly Changing Dimension. In this type, existing values are replaced with new ones, without retaining historical data.

Design the Base Table:

Prior to SCD1 implementation, it's essential to design your dimension table appropriately. The dimension tables must have -

1. Primary Key: A unique identifier for each record.
2. Dimension Attributes: Columns representing various attributes of the dimension, such as name, description, or category.
3. Timestamp Attributes: To track the validity period of each record.

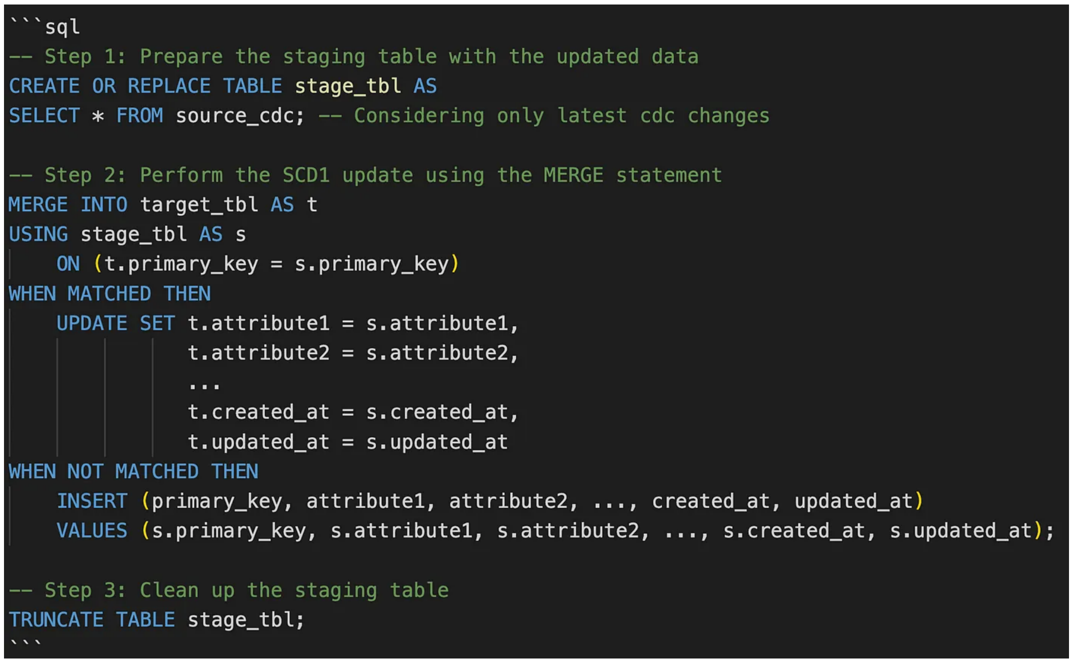
Implementing SCD1:

Step-1: Prepare Unified Stage Data

- Identify the records that need to be inserted or updated in the final table in the form of a stage table.

Step-2: Perform MERGE Operation

- Using the above staged data, execute the merge command to perform updates on existing records and insert them if they do not already exist.



~~~
```sql
-- Step 1: Prepare the staging table with the updated data
CREATE OR REPLACE TABLE stage_tbl AS
SELECT * FROM source_cdc; -- Considering only latest cdc changes

-- Step 2: Perform the SCD1 update using the MERGE statement
MERGE INTO target_tbl AS t
USING stage_tbl AS s
    ON (t.primary_key = s.primary_key)
WHEN MATCHED THEN
    UPDATE SET t.attribute1 = s.attribute1,
               t.attribute2 = s.attribute2,
               ...
               t.created_at = s.created_at,
               t.updated_at = s.updated_at
WHEN NOT MATCHED THEN
    INSERT (primary_key, attribute1, attribute2, ..., created_at, updated_at)
    VALUES (s.primary_key, s.attribute1, s.attribute2, ..., s.created_at, s.updated_at);

-- Step 3: Clean up the staging table
TRUNCATE TABLE stage_tbl;
```
~~~

Figure 5.2: SCD Type-1

3.2 Implementing (SCD2) in Snowflake: Slowly Changing Dimension Type 2
SCD Type 2 captures and manages current and historical data changes effectively. Instead of updating existing records, it creates new rows for each change in dimension attributes, maintaining multiple versions of records in the table where each version can be identified by a unique key and an effective date range.

Design the Base Table:
Prior to SCD2 implementation, it's essential to design your dimension table appropriately. The dimension tables must have -

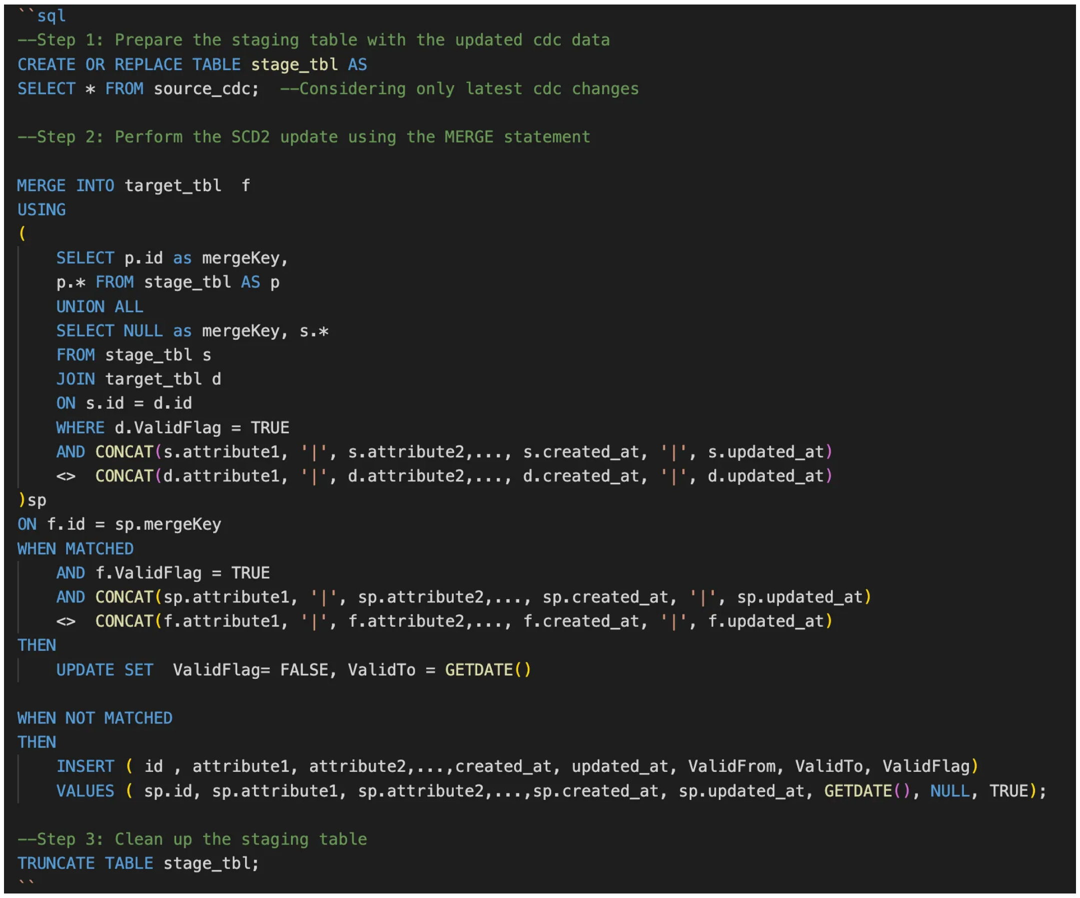
1. Primary Key: A unique identifier for each record.
2. Dimension Attributes: Columns representing various attributes of the dimension.
3. ValidFrom: The start date of the record's validity.
4. ValidTo: The end date of the record's validity.
5. ActiveFlag: A Boolean flag indicating if the record is the latest.

Implementing SCD2 in Snowflake:
The core idea here is to insert all records from the staging table into the final table, as

they represent the latest state for any records (identified by the primary key).

For this purpose, we maintain the **ValidFrom** = **Current_Timestamp()** and **ValidTo** = **NULL** along with an ActiveFlag set to 1, meaning an active record representing the latest state.

Conversely, we need to update the existing record in the final table that has been recently updated (present in the stage, identified by the primary key). Similarly, here we maintain **ValidTo** = **Current_Timestamp()** along with an **ActiveFlag** set to 0, meaning an inactive record representing the previous state.



```
``sql
--Step 1: Prepare the staging table with the updated cdc data
CREATE OR REPLACE TABLE stage_tbl AS
SELECT * FROM source_cdc;  --Considering only latest cdc changes

--Step 2: Perform the SCD2 update using the MERGE statement

MERGE INTO target_tbl  f
USING
(
    SELECT p.id as mergeKey,
    p.* FROM stage_tbl AS p
    UNION ALL
    SELECT NULL as mergeKey, s.*
    FROM stage_tbl s
    JOIN target_tbl d
    ON s.id = d.id
    WHERE d.ValidFlag = TRUE
    AND CONCAT(s.attribute1, '|', s.attribute2,..., s.created_at, '|', s.updated_at)
    <>  CONCAT(d.attribute1, '|', d.attribute2,..., d.created_at, '|', d.updated_at)
)sp
ON f.id = sp.mergeKey
WHEN MATCHED
    AND f.ValidFlag = TRUE
    AND CONCAT(sp.attribute1, '|', sp.attribute2,..., sp.created_at, '|', sp.updated_at)
    <>  CONCAT(f.attribute1, '|', f.attribute2,..., f.created_at, '|', f.updated_at)
THEN
    UPDATE SET  ValidFlag= FALSE, ValidTo = GETDATE()

WHEN NOT MATCHED
THEN
    INSERT ( id , attribute1, attribute2,...,created_at, updated_at, ValidFrom, ValidTo, ValidFlag)
    VALUES ( sp.id, sp.attribute1, sp.attribute2,...,sp.created_at, sp.updated_at, GETDATE(), NULL, TRUE);

--Step 3: Clean up the staging table
TRUNCATE TABLE stage_tbl;
``
```

Figure 5.3: SCD Type-2

To implement the solution, follow these steps:

Step-1: Prepare Unified Stage Data

1. Initially, we identify the records that require updating in the final table. These are the records that are still marked as active in the final table and are also present in the stage table. To accomplish this, we introduce a new column called merge key, which will hold the primary key value.
2. Next, we include all the records from the staging table and add the merge key column with null values for these records.
3. Subsequently, we perform a union of both data sets, resulting in a unified staging dataset.

Step-2: Perform MERGE Operation
Using the above stage data, execute the merge command with the appropriate action.

1. It performs the update if an update is present (where the merge key matches the primary key).
2. If performs the insert if it's a new record (latest version) where the merge key has null values.

In summary, both SCD1 and SCD2 are essential techniques for managing data changes in a data warehouse. Leveraging Snowflake's advanced capabilities makes their implementation a streamlined and efficient process.

Chapter 6

Let's Design a Data Warehouse

So far, we've covered several topics, starting with an exploration of modern data keywords. Then, we discussed the necessity of a data warehouse, examined its architectural components, and concluded with various data management techniques.

Now let us understand a use case and design a data warehouse using all the concepts that we have grasped so far.

Use Case: Data Management and Analytics at XYZ Enterprises

XYZ Enterprises faces a challenge with many disparate data sources, as sales data is recorded in MySQL, while employee and customer information is managed in a PostgreSQL system. Furthermore, additional marketing data is sourced from Google Analytics and the Kafka channel, as well as a few ad hoc data sources in the form of flat CSV files and spreadsheets. This fragmented data system causes problems for comprehensive analysis and decision-making.

The enterprise aims to overcome this hurdle by unifying its data by creating a Snowflake Data Warehouse, leveraging Amazon S3 as a Data Lake and Databricks to orchestrate the cron jobs. The goal is to employ SCD1 and SCD2 techniques for effective data management, develop a dedicated Data Mart for sales and marketing analysis, and implement PowerBI for user-friendly data visualization. Additionally, the goal is to establish a data governance tool for better metadata management and monitoring.

This initiative aims to streamline data processes, enhance analytical capabilities, and empower stakeholders with actionable insights.

Appendix

References

- https://www.techopedia.com/definition/24361/database-management-systems-dbms
- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-015-0025-0
- https://www.databricks.com/glossary/medallion-architecture
- https://docs.snowflake.com/en/user-guide/intro-key-concepts
- https://en.wikipedia.org/wiki/cdc
- https://www.striim.com/blog/change-data-capture-cdc/

- : Thanks : -

Follow Here