TALLINN UNIVERSITY OF TECHNOLOGY

School of Information Technologies

Shahin Yusifli  214209IASM

# Impact of Data Storage Models on Business Intelligence Applications

Master's Thesis

Supervisor: Vladimir Viies, Ph.D.

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Shahin Yusifli  214209IASM

# Andmesalvestusmudelite mõju äriteabe rakendustele

Magistritöö

Juhendaja: Vladimir Viies, Ph.D.

Tallinn 2024

## Author’s declaration of originality

I therefore attest that this thesis is the work of my single authorship. There have been allusions made to all of the utilized resources, literary works, and other people's contributions. This thesis has not been submitted for review elsewhere.

Author: Shahin Yusifli 214209IASM.

## Acknowledgments

The author of this thesis would like to express his kind appreciation to his supervisor Vladimir Viies for their encouragement, responsiveness, guidance, and supervision throughout the whole thesis work.
I would like to extend my gratitude to the research teams behind Druid and Dremel for their open and direct evaluation of the software.
I also want to express my gratitude to my brother and family for their support throughout this journey.

## Abstract

In today's data-driven world, the influence of data applications is profound, impacting millions of lives daily. However, current implementations heavily rely on cloud storage services, which involve significant tradeoffs according to the CAP theorem. Exploring traditional disk storage approaches can unveil new possibilities, not limited to specific databases but encompassing real-time, batch data, master data, and metadata. This study focuses on utilizing disk storage for Business Intelligence (BI) applications, highlighting the critical importance of data preservation. In BI applications, the ability to compare and process both current and historical data is essential. Any loss of past data diminishes the value of real-time data, and vice versa. Therefore, ensuring fault tolerance is paramount. To address this challenge, we propose a decision-making framework leveraging RAID (Redundant Array of Independent Disks) data storage virtualization technology to enhance its positive impact. Our approach targets various categories of data and nodes within the application architecture, aligning with the architectures of prominent platforms like Druid, Dremel, and Snowflake, with a particular emphasis on Near Real-Time Data Vault. Our findings go beyond individual capabilities, offering a comprehensive guide for strategic decision-making in the dynamic landscape of modern BI applications. This framework serves as a compass for navigating the complexities and demands of data management in today's evolving environment

## List of abbreviations and terms

| | |
|---|---|
| BI | Business Intelligence |
| SSBI | Self-Service |
| DW | Data Warehouse |
| DB | Data Base |
| OLTP | Online Transaction Processing |
| OLAP | Online Analytical Processing |
| RAID | Redundant Array of Independent Disks |
| CPU | Central Processing Unit |
| DML | Data Manipulation Language |
| DDL | Data Definition Language |
| CAP | Consistency Availability Partition |
| CSRML4BI | Collaborative Systems Requirements Modelling Language for Business Intelligence |
| EDA | Event-Driven Architecture |
| SQL | Structured Query Language |
| SEM | Structural Equation Model |
| I/O | Input / Output |
| CMMI | Capability Maturity Model Integration |
| TQM | Total Quality Management |
| MDM | Master Data Management |
| RAP | Row Access Policy |
| ANSI | The American National Standards Institute |
| ACID | Atomicity Consistency Isolation Durability |

# Table of contents

# List of figures

## List of tables

## Introduction

Business intelligence applications play a crucial role in today's data landscape, enabling the representation of data insights in both quantity and quality. They significantly impact the decision-making processes of executives in large corporations and institutions. The Industry 4.0 revolution has brought about profound changes in descriptive data applications, necessitating the integration of diverse data sources with varying latency requirements. The diverse data handling processes involved in business intelligence applications make them valuable assets for organizations seeking insights into business objects and events. Various factors significantly influence descriptive data applications, with many frameworks focusing on formalizing requirements and interactions.

While non-technical factors can impact the interaction of business intelligence applications, technical limitations can affect the long-term vision of companies and organizations. In my previous research, I observed that storage models have a substantial impact on application performance and fault tolerance. My earlier work focused on geo-replicated storage models, highlighting their weaknesses under the CAP theorem. The CAP theorem formalizes a limited number of consistency models and replication policies applicable to geo-replicated storage models. Although many cloud vendors offer solutions that support enterprise profit-based products, radical improvements are necessary to gain a competitive advantage.

Therefore, I propose to analyze the detailed impact of disk virtualization techniques on storage. Business intelligence applications can address these demands by leveraging enhanced components from data applications, categorized into stages. Consequently, I aimed to formalize data RAID levels based on the stages of data application aligned with the requirements of descriptive data applications. Modern data application requirements emphasize real-time data analytics and self-service capabilities. Therefore, I chose to structure stages based on real-time data analytics OLAP databases like Druid and Snowflake, leveraging their advanced self-service features because they can provide valuable insights into RAID model selection, highlighting well-known limitations and use cases. Moreover, RAID models offer significant potential for performance enhancements compared to the stable consistency and policy offerings from cloud-based storage services of various vendors. Consequently, I aimed to introduce new rulesets within a comprehensive framework to optimize RAID model performance.

Rather than focusing on a high-level framework, I opted to develop a general opinion framework aimed at maximizing disk-based storage models to meet modern business intelligence (BI) requirements. This approach seeks to address the evolving needs of BI applications by providing a flexible and adaptable framework for disk-based storage optimization.

To achieve this, the following tasks will be undertaken:

1. Gather business intelligence requirements.
2. Analyze and gather nodes and data types in data warehouse stages based on gathered BI requirements.
3. Create rulesets to define the appropriate RAID level for each stage of the data warehouse, based on requirements for performance and fault tolerance.

These tasks will collectively contribute to the development of a robust framework that aligns disk-based storage models with the dynamic needs of modern BI applications, ensuring optimal performance and reliability.

## 1. Requirements of Business Intelligence Applications

Several types of enterprise applications can be employed as sources for descriptive data applications which require storing data as a single truth of source. In this way, we can eliminate duplicate data but it requires costly storage and storage management processes through the life cycle of data. 39 studies about industrial implementation of Business intelligence applications demonstrate that 56 percent of small and medium-scale corporations could benefit capabilities of Business intelligence applications in a long-term manner. Descriptive data applications help better decision-making processes for organizations to ensure business survival with counterintelligence. Also, it has significant benefits in the cost reduction process. Additionally, research studies [1][2][3] have demonstrated that putting Business Intelligence (BI) into practice has several benefits, including improved resource planning, cost savings, improved performance, increased productivity and efficiency, and support for business growth. Eventually, obtaining a competitive edge may be facilitated by these advantages [4]. Data-driven applications and market research play a crucial role in developing smart factories today. The cornerstone of effective marketing lies in market intelligence, which is essential for designing and implementing strategic marketing initiatives [6].

## 1.1 Quarterly Traditional Data Analysis

Quarterly business intelligence applications keep significant space in the data analytics industry which processes batch-loaded data from data warehouses or data marts in one time of day. These legacy systems have been used by many private sector and governmental organizations [7] because they are reliable in showing the most common business insights such as comprehension of organizational value between different business entities. For example, customer loyalty, fraud analyses, and purchase behaviors over different customer groups can be easily analyzed and categorized by legacy traditional Business intelligence applications. However, these data strategic data descriptions require a large amount of data which will be stored against complex ad-hoc and analytical queries. Regarding to implementation of a goal-oriented CSRML4BI model on descriptive data applications. It is observable to notice collaborative intercommunication and information awareness take significant spaces for Business intelligence applications as main requirements [8]. Additionally, the adaptability of quick business changes on legacy Business Intelligence applications through the data application lifecycle was a main requirement to gain competitive business advantage for big organizations [9].

## 1.2 Real-Time Data Analysis

The final framework emphasizes the importance of flexibility in both batch and real-time processing. It utilizes an event-driven architecture (EDA) that seamlessly integrates these two methods of data processing, as illustrated in Figure 1. Kafka, a robust framework for handling event streams, forms the core of our real-time capabilities. By leveraging Kafka to organize events into topics and partitions, the framework ensures message ordering and reliability through an append-only log structure. The EDA is specifically crafted to support analytics alongside event processing.

The data entity structure is governed by an Activity Schema that we developed. These entities form the backbone of our event-driven analytics platform, representing various aspects of our business and interconnected through activities. In terms of event data management, we follow a systematic process documented as a clear guideline for transforming raw event data into actionable insights. This process involves carefully selecting relevant actions, identifying appropriate data sources, and executing SQL queries with temporal joins.

The concept of hub and link tables is foundational to our architecture, providing the framework for efficiently tracking transactions and establishing relationships between business entities. We

utilize Snowpipe for real-time data loading, benefiting from its micro-batch ingestion capability to keep our analytics pipelines continuously updated with the latest data. Additionally, our architecture enables efficient management of semi-structured data through straightforward SQL operations.
Ultimately, our architecture is designed to empower analytics rather than merely manage data. By harnessing real-time capabilities, robust data modeling, and advanced analytics techniques, we derive meaningful insights that drive business decisions and foster innovation [10].

## 1.3 Self-Service Business Intelligence Applications

Self-service business intelligence must be a design pattern that offers individual data processing on different departments by interconnection with each other. Regarding to Structural Equation Model (SEM), we can see Self-Service Business intelligence tools easily adaptable for extending data-centric decision principles over the whole organization [11]. However, the mentioned descriptive data applications have certain requirements that are highly related to user abilities in work and design. Each department of an organization can easily access real-time or legacy data from understandable metadata and master data-enhanced data marts. In this, batch and stream data can be consumed to meet requirements for mass demand in organizations separately [11]. These requirements contain easy data accessibility and high data quality with offering addressing problems in data lineage sessions. To address these requirements, it is possible to decrease the workload on the data engineering and business intelligence departments dramatically because many departments can easily utilize their data sources. Unfortunately, it can be problematic for employees who lack knowledge about information about data and business objectives. To eliminate these pitfalls, we should provide recorded metadata and master data information under strong data governance principles. Additionally, attributes of this data governance can contain information about KPIs and created metrics in descriptive data reports with their accessible references.

## 1.4 Integration of Real-Time and Batch Data Analysis

Modern Business Intelligence (BI) platforms incorporate various solutions such as production reporting, end-user query and reporting tools, and OLTP (Online Transaction Processing) and OLAP (Online Analytical Processing) databases. These platforms handle both real-time and batch-stored data from OLTP databases for reporting and analysis purposes. However, achieving unified business insights poses challenges due to the diverse nature of data. Real-time data has different storage requirements compared to quarterly processed batch data, making it complex to establish a single unified truth of data. To address these challenges, adaptive systems have emerged that can integrate real-time data with batch loads effectively. The Near Real-Time Data Vault architecture is a standard framework used to manage and overcome integration challenges in such scenarios. Different database engines, like Druid, Snowflake, and Dremel, serve as notable references for handling real-time and batch data integration. These engines offer distinct nodes to address integration difficulties within BI applications. The selection of appropriate storage models for integrated data applications cannot solely rely on requirements and stored data types. The classification process for selecting these storage models will be elaborated upon in subsequent chapters, considering specific requirements and data characteristics.

## 1.5 Types of Data in Modern Business Intelligence Applications

Data shapes current technologies and capabilities of descriptive data applications. The current world of data is divided into many types and formats. Additionally, several emerging real-time and batch data sources are increased so dramatically even it can not predicted. These dramatic changes reflect the type of data storage and modeling of data to meet the requirements of predictive and descriptive data applications. If we take brief information about data which is divided into strictly batch and real-time data. Real-time data contains information about events occurring in the current time or timestamp. Most of the time real-time data consists of timestamps or events creating time, dimensions, and metrics. These timestamps attribute of data represents event creating time which is critical for real-time data analytics purposes. Also, it can be used for versioning data and determining whether data is historical or not. Regarding dimensions and metrics of real-time data, they should be in string and numeric data types respectively. However, batch data contains information about events or business objectives with additional technical and metadata attributes. These metadata columns can be inherited from source systems or ETL operations. But most of the time, they represent updated and inserted time. If we look at accepted data types of batch data, we can see that can be different for the stage of the data warehouse that is string or varchar for most common cases.

Data comes with processing requirements which should meet demands of different workloads. Descriptive data applications should process and aggregate individual rows for quantitative data analyses. Therefore online analytical processing and Online Transaction Processing databases differ in this point based on the processing of big amounts of data under technical and performance limitations. Priority of data in Online Transaction Processing databases is preventing data from different kinds of DML operations such as update, and delete anomalies, guaranteeing availability of data through low latency contained query results. It means operations will be row-based and we need to give access to each row more effectively instead of the whole columns. But in data for Business Intelligence applications instead of transactional ones, aggregated data analytics operations require to access data in column-based because we need to extract metrics from data. These metrics can route us to useful insights for business operations. Therefore, many enterprise data solutions should support complete drill-down aggregations rapidly and exploratory data analyses.

### 1.5.1 Metadata

Metadata plays a crucial role in descriptive data applications, aiding in the effective detection of insights and facilitating the integration of new data sources into data pipelines. Business intelligence (BI) applications often consume data from information marts established within data warehouses, emphasizing the need for efficient data pipelines to facilitate transactions to these marts.

| Type | Description | Examples |
|---|---|---|
| **Business Metadata** | Helps in understanding and using business data effectively | Business definitions, ontologies, taxonomies, physical names |
| **Maintenance Metadata** | Facilitates efficient management of data warehouse components | Record source metadata, table specifications, exception-handling rules |

| | | |
|---|---|---|
| **Technical Metadata** | Provides technical details about the data warehouse infrastructure | Source system information, data models, volumetrics metadata, data quality metadata |

Table 1. Metadata Characteristics

To address business problems and operational needs effectively, different metadata categorizations can be utilized which can be seen in Table 1. Business metadata encompasses several categories:

1. Business Definitions: Includes business column names that should be converted into human-readable formats for better understanding. Clear business descriptions help in identifying and retaining relevant data objects.
2. Ontologies and Taxonomies: Used in master data information marts to organize and categorize data effectively.
3. Physical Table and Column Names: Matched to relevant business names for clear business definition metadata.

In addition to business-related definitions, metadata for data warehouse maintenance and utilization is essential:

1. Record Source Metadata: Provides understandable text and sectional names for record sources.
2. Table Specifications: Includes size, purpose, list of columns, and constraints per attribute for efficient table management.
3. Exception-Handling Rules: Addresses potential errors and data quality concerns in data pipelines, reducing resource consumption for ELT (Extract, Load, Transform) and ETL (Extract, Transform, Load) processes.

Source system business definitions and business rules metadata offer insights into data patterns and definitions within source databases, benefiting BI applications. Technical metadata, focusing on the technical components of the data warehouse, includes:

1. Source System Information: Details about storage locations, staging area tables, and cloud-based data warehouses.

2. Data Models: Split into physical and logical models, encompassing table names, column names, data types, and constraints.
3. Volumetrics Metadata: Provides information on table size increase patterns and source database workloads, aiding in data warehouse planning.
4. Data Quality Metadata: Defines rules and metrics to measure data quality.

Apart from metadata, Process Execution information provided by data warehouse teams assists in understanding performance metrics and maintenance tasks. This data is often stored in independent information marts derived from ETL (Extract, Transform, Load) systems, enhancing operational insights and efficiency within data pipelines.

### 1.5.2 Master Data

Master data plays a critical role in current descriptive and predictive data applications, especially in self-service environments where awareness of various business entities is essential. It serves as the source of truth, although its representation may vary depending on the maturity level of a data warehouse or application. The Enterprise Information Lifecycle leverages master data to manage multiple data objects and their representations.

In essence, master data comprises entities related to people, processes, and technology, aiming to bridge the gap between business activities and the data collected from source systems to meet the requirements of Business Intelligence (BI) applications. Descriptive data applications rely on master data to ensure accurate and comprehensive data entities, often referred to as "golden records," aligning with essential business concepts.

While establishing and maintaining master data can be resource-intensive, it is instrumental in achieving specific goals such as improving data quality, streamlining information processing, reducing data redundancy, and facilitating data exchange. Detecting and correcting data errors through data harmonization processes, guided by Total Quality Management (TQM) principles and Six Sigma methodologies, ensures data accuracy and reliability.

| **Aspect** | **Description** | **Examples** |
|---|---|---|
| **Core Business Entities** | Critical for business operations and decision-making | Customers, products, suppliers |

| | | |
|---|---|---|
| **Data Quality Improvement** | Ensures accuracy, consistency, and reliability through data harmonization | Total Quality Management (TQM), Six Sigma methodologies |
| **Master Data Management** | Manages master data effectively and supports BI applications | CMMI, self-service applications, dedicated master data databases |

Table 2. Master Data Characteristics

Master Data Management (MDM) is key to utilizing and storing master data effectively, providing insights for self-service applications and facilitating data error corrections. MDM aligns with Capability Maturity Model Integration (CMMI), promoting good data flow, effective data control, and seamless implementation of Total Quality Management practices.

In operational and analytical data systems, master data management caters to diverse consumer requirements, adapting its representation based on the maturity level of the data application:

1. First Maturity Level: No dedicated Master Data Management; master data resides within databases or various data sources.
2. Second Maturity Level: Master data is represented in a separate information mart within a data warehouse for easy consumption by other data applications.
3. Third Maturity Level: Implementation of Master Data Management involves utilizing dedicated databases for master data collection and harmonization, ensuring consistency and accuracy across data applications.

Master Data Management is instrumental in supporting self-service BI applications and Total Quality Management initiatives where we can see real examples in Table 2. Serving as a repository for data models, entities, members, and attributes critical to organizational insights and operational efficiency.

## 1.6 Requirements of Modern Business Intelligence Applications for Storage Models

Requirements for modern descriptive data applications had been listed above such as integration of real-time and batch loads, information shareable, collaboration support, and offering information about data and business objects. However, it is not possible to limit data application in this way. The current market is so dynamic therefore our descriptive and predictive data applications can be affected. Any dramatic change occurring in data application needs unexpected pauses in data delivery to Business intelligence applications. For this reason, we should select a logical data model wisely to handle these changes effectively. Incremental business changes can be a good example of a quarterly challenge to big competitive corporations. Incremental business changes are the result of some change in company structure or they can be triggered by two or more organizations that acquire into one parent organization. We can keep track of changes in data pipelines. If there are some schema changes or data anomalies detected, workflow management tools can notify us. However, our descriptive real-time and traditional data applications need time to solve problems by employees. For this purpose, we need data isolation from source systems. Regarding functional requirements, we should meet the demands of data governance to address problems and support a collaborative work environment easily and data lineage is the most important part of data governance in the current acceptable data world [15].

### 1.6.1 Data Lineage

Data lineage was the main responsibility while maintaining and improving the data quality of the data application. Many reasons can lead to data anomaly in upstreams which are Business Intelligence applications for us. Missing SQL clauses or delivery changes in data contractors can reflect data in reports which can be notified through workflow orchestration or assertions from data tests. Data lineage service is required in every data application which requires extra metadata and technical columns alongside business columns. Physical data storage models do not have a big impact through adding and processing these additional attributes. It relies on the unique futures of used data platforms. For example, through Druid timestamp, headers and keys can be tracked which is suitable for real-time descriptive data applications. Also, in Snowflake, Information Schema allowed us to use different views to access executed queries and addressed schema objects. But if we need to categorize data lineage methods, we can use pattern-based and SQL parsing.

Pattern-based lineage services easily provide information about broken workflow logic via notification and we can easily address using a pattern-based lineage graph. Unfortunately, we do not have a chance to track the context of executed queries and addressed database objects. It is a lightweight way of addressing data lineage problems with low delay which most of the workflow orchestration and database management tools provide us. Also, it does not allow us to analyze previously executed operations which we can compare and we can easily detect broken or missing logic through pipelines.

Regarding the provided disadvantage of pattern-based lineage service, we can use SQL parsing to provide a lineage graph. The lineage graph should contain target and destination tables, and executed query contexts which should be extracted from the query. Additionally, we can provide error messages and execution time. SQL parsed-based lineage graphs make the anomaly detection process more accurate. If we look at Snowflake, Snowflake can provide executed query context at least 2 hours ago to Information Schema views which makes it impossible to use it for real-time analytics. We have market solutions but they can cost some money and it can lead to some legacy problems.  We can custom wrapper for this purpose or we can use other query execution tools such as dbt. But it is exactly predictive to Snowflake can be costly to provide SQL parsed data lineage service to real-time data analytics.

If we look through logical data models such as Data Vault 2.0 and Kimball, we can examine them with their unique features. As I mentioned, for the data lineage system, we need some technical and metadata columns alongside business attributes. These additional table attributes can be added or we can use the default property of logical models at first glance.

When we look Kimbal model, we can see all technical columns should be added to address specific data governance or continuous data flow promotion. We need to add a validality timestamp to each column which replaces natural keys with surrogate keys should be folwed. Additionally, we need to central metadata catalog solution to provide metadata columns to tables and pipelines. The kimbal model could keep its openness to modification because we can manipulate the model to get excellent results in analytical queries. Regarding the priority of performance, the paintability of the system decreased dramatically and we will need extra and resource-intensive extensions to address data lineage problems.

If we look Data Vault 2.0 model and its implementation, we can see several technical attributes and metadata columns provided default. For instance, replacing natural keys with hashed surrogate keys should be provided beforehand which validate and load date timestamp technical columns follow it. Regarding to type of table in the Data Vault model, we can use extra technical columns such as sequential keys in snapshot Point-in-Time tables or date columns in standard Point-In-Time tables. Additionally, all table types must contain source metadata attributes that point source of tables. The source of tables can be other tables or external stages. From these facts, it is easily understandable that data lineage services are easily fed by technical and metadata columns in this way.

### 1.6.2 Summary

To meet the needs of Business Intelligence (BI) applications, it's essential to enable self-service capabilities that empower users to make data-driven decisions independently. This extends the ability to leverage comprehensive data analytics by ensuring access to both real-time streaming data and historical batch data. Consideration of technical debt is vital when integrating diverse data sources into self-service BI applications. This involves managing any development shortcuts or compromises to ensure the long-term maintainability and scalability of the system. Integrating master and metadata into the BI reporting lifecycle is critical for providing context and understanding to data analysts and decision-makers. This integration reduces the time required to analyze data and makes insights more accessible and actionable. By integrating master and metadata within the data application or data warehouse, organizations can reduce the cost per insight. Staff can make informed decisions more quickly, which minimizes the resources needed to derive insights from data. Throughout the data lifecycle, various challenges can arise that affect data quality and integrity. Implementing data lineage services under established data governance rules helps address these issues, ensuring consistent data quality and governance. We can see all mentioned requirements as follows

1. Enable Collaboration:
    a. Enable easy sharing of data among teams.
    b. Utilize collaboration tools tailored for BI.
    c. Ensure comprehensive metadata and master data for clear problem understanding.
2. Integrate Historical and Real-time Data Streams for Comprehensive Analysis.
3. Establish Query Endpoints for Seamless Data Retrieval and Analysis.
4. Integration of Data Lineage to Ensure Data Quality and Compliance under Data Governance principles.

In summary, meeting the requirements for BI applications involves enabling self-service capabilities, managing technical debt, integrating master and metadata effectively, reducing the cost per insight, and implementing robust data quality and governance measures. By addressing these aspects, organizations can build BI solutions that empower users to make informed decisions based on high-quality data.

# 2 Tools and Components of Physical and Logical Data Models for Business Intelligence

The effective storage models for modern Business Intelligence (BI) applications are defined by their underlying design principles, which can vary depending on specific requirements outlined in earlier chapters that discuss trends in descriptive data applications. This chapter aims to provide a concise overview of the physical components of data storage models, covering workloads, networks, and data storage distribution techniques.

Furthermore, we will analyze nodes from Druid, Dremel, and Snowflake based on insights gleaned from their official industrial papers. Each node will undergo a comprehensive examination to identify its primary requirements, which will contribute to the decision-making framework presented in the final chapter of this thesis.

The sources cited in this article primarily originate from research teams associated with Druid and Snowflake, leading commercial analytical data platforms. The arrangement of nodes within each schema will largely follow the structure of the Dremel distributed data engine, which has been a significant reference in the domain of data-sharing architectures.

## 2.1 Physical Components of Storage Models in Modern Data Applications

Physical and logical data modeling require specific hardware components to efficiently manage the storage and processing of large-scale data. Configuring hardware for descriptive data applications is crucial for optimizing data warehouses, enabling efficient data consumption and long-term retention. Despite the prevalence of cloud-based data solutions, this study will concentrate on exploring on-premise data warehouse setups to assess their advantages compared to cloud-based data warehouse engines. Understanding the expected workload characteristics is paramount due to diverse requirements in data processing and storage.

### 2.1.1 Analyzing Fault Tolerance in Storage Models Using Different RAID Types

RAID is used to manage independent disks to provide more fault tolerance over performance trade-offs [16]. But firstly, I would like to highlight other relevant storage options such as SCSI and

Fiber Channels for data sharing and accessing desired data from remote servers [17]. Also, Network Attached Storage can be the case for transferring and keeping data over networks which has many real implementations in the current data industry. However, the easiness of RAID setup and big options for disk accessibility are the main causes for the selection of this data virtualization technology. It is possible to select budget-friendly options such as cloud storage services such as Network Attached Storage. It can provide parallel resource management and scaling as an initial advantage but it comes with some disadvantages. These disadvantages have been described under the CAP theorem which it is not possible to guarantee Consistency, Availability, and Partition at the same time. We can select the right consistency and replication model to address this problem but we just hide the problem and do not provide a real solution.

Simply data is stored in disks and we can group these disks under some array groups to achieve some results. These results are specific to the type of application. If we want to high data write and read operations, we need to use RAID-0. RAID-0 offers high I/O performance over multiple disks. It enhances data by stripping it into several blocks and these blocks extend over disks. It is the perfect solution for data marts and data that has historical relevance in other nodes. Another level of RAID is RAID-1 which we can use the advantage of data mirroring into unified disks. Written data is mirrored into two disks and two disks are represented in one interface disk. It provides enough good fault-tolerance and performance compared to RAID-0 but mirroring decreases storage capacity dramatically. All touched RAID categories aim to provide read and write operations at the same level. In some cases, we need to prioritize write operations which we will see in data handling nodes of stages of data application. For this purpose, we can use the advantages of RAID-5 where need to access 3 disks at least. Two disks must be employed for stripping as same as RAID-0. The last disk is used to store parity data which can be easily recovered. It provides high data write because we have multiple blocks in several disks that contain desired data. However, it is not easy to write data because it should be written on three disks at least. It can be an efficient solution for intermediate stages in data applications that meet historical data and data-sharing requirements of Business Intelligence applications. If stored data is in the gold record category, another RAID level can assist us. For instance, RAID-6 requires 4 disks and parity striping occurs a second time through a block of disks. In this way, it provides easy data recovery and we can easily replace lost data. Also, a minimum of four disks are required to conclude RAID-10 which is mixing striping and mirroring over several disk drivers. It provides high disk write and read operations alongside redundancy. It has some disadvantages which are inherited from RAID-1, it decreases the capacity of storage by half. It can be a good choice for nodes where data batch and stream data is stored and transferred into the Business Intelligence application without any intermediate layer. We have another mixing RAID level to achieve better data writes with a high level of fault tolerance. It is a mixing of the party and stripping together which RAID-5 and RAID-0 represent respectively.

.

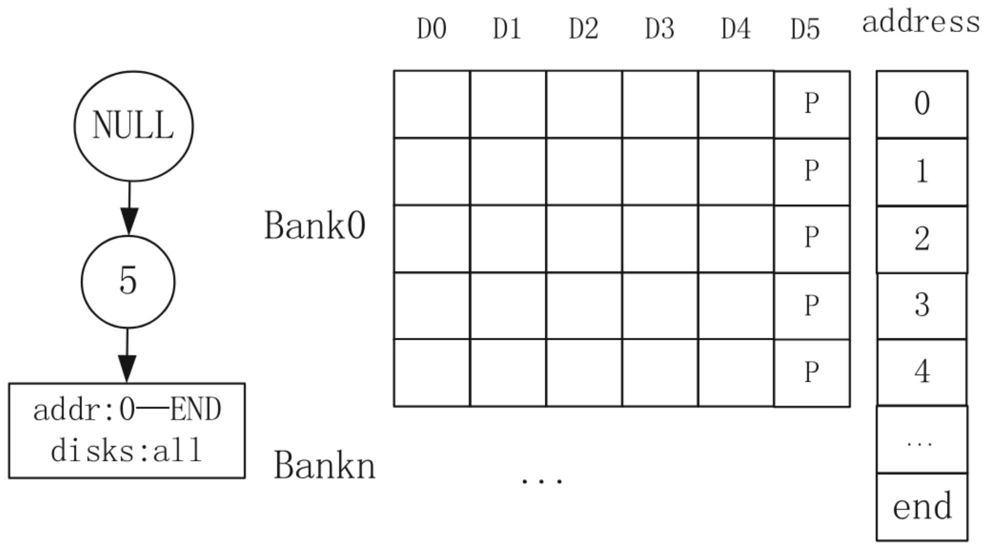


Figure 1 Data Striping in RAID-5: Hot and Cold Disk Considerations

Hot disks, in particular, are continually running at a greater speed and use more energy, whereas cool disks are always working at a lower speed and dissipate less energy. Furthermore, we use multispeed disks in the disk array to conserve energy. The goal is to balance the load between two disk zones: the hot disk zone, which contains popular files, and the cold disk zone, which contains unpopular files. The book describes how SEA allocates popular and unpopular files to these zones in a striping pattern to improve performance and energy efficiency. It also explains why a small percentage of disks are designated as hot disks, as well as the advantages of having a higher number of cool disks for energy savings and IntraRequest parallelism. Furthermore, it compares the performance and fault tolerance of the SEA0 and SEA5 algorithms, which are versions of SEA paired with RAID (Redundant Array of Independent Disks) structures, with SEA0 favoring speed and SEA5 providing fault tolerance which is demonstrated in Figure 1.

### 2.1.2 Analyses of Workload Characteristics for Storage Models

Workloads are closely tied to the maturity level of Business Intelligence (BI) applications. When a BI application derives insights from data using straightforward business processes, it typically involves extracting data from an Online Transactional Processing (OLTP) Database. In this scenario, individual rows of data are accessed to retrieve specific values that reflect the current state of business or operations. However, relying solely on transactional databases for analytical queries in data descriptive applications can lead to performance issues. To address complex data analytics challenges effectively, it's advisable not to directly query data from transactional databases. Instead, dedicated data analytics databases offer hardware configurations optimized for handling the latency between data loading and processing by BI applications. The selection of appropriate hardware is influenced by various types of data loads, including batch, stream, and near-real-time data loads, each differing in execution schedules. While workloads serve as initial criteria for hardware selection, additional requirements such as latency, consistency, data type support, response time, and predictability also play significant roles in determining the ideal hardware setup. These factors collectively shape the hardware infrastructure needed to support advanced BI and data analytics applications effectively.

One important aspect of data applications is latency. The lag or time interval between the generation of data and the process is called latency. For many descriptive and predictive data applications, this was a recognized drawback since each request necessitates a short wait between data serving and efficient query handling. Low latency is necessary for online transaction processing databases because users must be able to obtain pertinent data for everyday transaction processing. However, since batch loads are what create delays, we may take advantage of this in databases used for online analytical processing. The majority of batch loading happens late at night when no queries about analytical processing are being run. Our internal resources may manage data loading independently in this manner, allowing end users to take advantage of all available computing capacity. Data accessibility must be prioritized in today's distributed data applications in order to reduce the delay between data production and processing. First of all, multiple users using dispersed data infrastructure is suitable. When a user views and modifies a row of data, should other users be able to see the changes immediately? Or are consumers willing to put up with this kind of animosity? The answers to these questions indicate the kind of application that may be business-critical or mission-critical. We may possibly be able to solve these issues by computing expected and desired workloads under the Byzantine fault problem. To make sure that any changes to the data are reflected to all consumers, we may use linearizability, which is also referred to as atomic consistency, sequential consistency, and eventual consistency. Atomic consistency, which guarantees that all users see all changes, is useful for military and national security applications. Analogously, there exist two categories of methodologies for addressing business problems: eventual consistency and sequential consistency. Amazon, for instance, uses eventual consistency in most of its services. However, we should create a consistency model for the specific requirements of the descriptive data application. According to the CAP (Consistency Availability Partition) theorem, only two can be awarded from Consistency, Availability, and Partition. This selection procedure can be carried out in light of this finding. In three of them, satisfaction concerning the CAP theorem is not possible.

The crucial characteristic of databases is updateability, which pertains to the ability to modify data within a data application. As previously discussed, databases are categorized into analytical and transactional databases, each treating data differently. In Online Analytical Processing (OLAP) databases, data is typically considered immutable, meaning it is not changed once it is stored. However, logical data modeling concepts allow for changes to the state of data, effectively treating data warehouses as structured archives with hot access capabilities. State changes in data applications involve updating data while retaining the previous state, a process that can be tracked and documented using the selected logical data model. Updateability becomes particularly significant in business intelligence applications where historical comparisons are essential. Without data changes over time, there would be no historical data available for descriptive data applications, which rely on historical trends and comparisons for analysis and decision-making purposes. Therefore, the ability to update data while preserving its history is fundamental to the functionality and value of data warehouses and analytical databases in supporting business intelligence operations.

In contemporary data warehouses for data applications, updateable data types are becoming more and more significant. First and foremost, tabular data processing is the most practical method of handling data in business intelligence systems. Hierarchical data can be properly observed in this manner. Present-day data applications, however, may encounter unstructured and semi-structured data from several data sources. Flattening and parsing these non-conventional data loads is necessary, and it should be consistent with using traditional data techniques as well. Although SQL-based data platforms and ELT pipelines are the limitations of this categorization, No-SQL data solutions offer an opportunity to handle semi-structured data imports. Furthermore, unstructured data can be handled more effectively by dynamic query capabilities than by declarative SQL language.

As I mentioned earlier, data warehouses in data applications aren't just for storing data; they are responsible for efficiently processing and retrieving desired results quickly. This efficiency is achieved through the consistency, latency, and updateability of data. However, the focus is mainly on query response times, as user satisfaction depends on this. Users interact with data warehouses primarily through analytical or aggregational queries. Data warehouses fall under Online Analytical Processing (OLAP) databases, where we handle aggregation and processing queries instead of row-based single operations. Therefore, the need to scan and filter large amounts of data significantly impacts response times for users. Many databases can read large amounts of data from disks, process data, and perform calculations and aggregations in advance.

Lastly, it's important to address predictability in this paper, as databases need to anticipate the expected results and response times based on the query context and the data involved. This predictability can be achieved by leveraging metadata information about tables and their attributes. This metadata includes details about aggregations and the cardinality of columns needed for queries. Cached query results can also serve as predictive references for many workloads. However, every database platform should possess the capability to forecast the expected resource usage and response times required to retrieve desired results.

## 2.2 Optimizing Data Storage Efficiency with Logical Data Models

Logical data models evolve over time to meet the diverse requirements of consumer data applications. Online Transaction Processing (OLTP) systems typically demand normalized data models to ensure consistency and maintain acceptable latency levels. Emphasizing consistency helps eliminate anomalies in data transactions related to write, update, and delete operations. Implementing normal forms ensures a structured approach to mixing data types and organizing column dependencies:

1. All tables should have a primary key attribute.
2. Non-primary key attributes should depend on the primary key in a nontransitive manner.

Small organizations can benefit from integrating predictive and descriptive data applications. Modern Data Stacks like Snowflake support OLTP databases as staging layers within data warehouses or lakehouses. However, normalized data models may introduce high latency in select queries out-of-the-box, which is not suitable for Business Intelligence (BI) applications. To enhance query performance, transactional data often needs to be denormalized. Two common approaches to denormalizing transactional data are the Kimball model and the Data Vault model:

1. Kimball Model: Primarily focused on optimizing select query performance by denormalizing data. It emphasizes star schema or dimensional modeling, where data is organized into fact tables and dimension tables for efficient querying and reporting.
2. Data Vault Model: Emphasizes data governance and isolates data applications from business changes. It utilizes a hub-and-spoke architecture with raw data vaults and business vaults, enabling flexibility and scalability while ensuring data quality and traceability.

Both models serve distinct purposes:

1. Kimball Model: Ideal for BI applications requiring fast query performance and straightforward data access for reporting and analysis.
2. Data Vault Model: Suited for complex data environments requiring robust data governance, scalability, and flexibility to adapt to changing business needs without impacting existing data applications.

Choosing between these models depends on specific business requirements, data complexity, scalability needs, and the balance between query performance and data governance. Each model offers unique benefits and considerations, contributing to effective data management and analytics within modern data architectures that given in Figure 2 detailed.

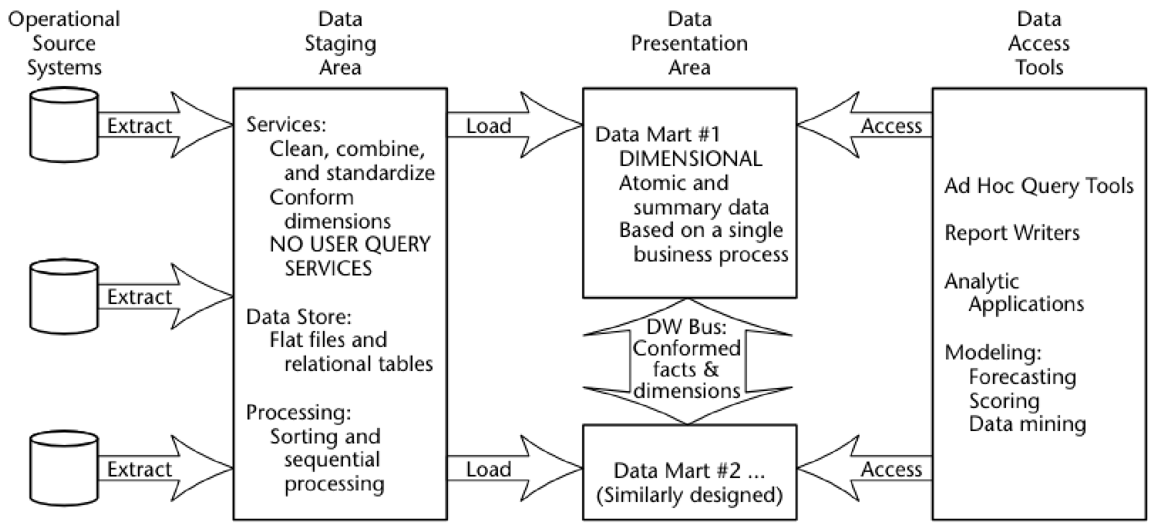

Operational Source Systems
Data Staging Area
Data Presentation Area
Data Access Tools
Extract
Extract
Extract
Services: Clean, combine, and standardize Conform dimensions NO USER QUERY SERVICES
Data Store: Flat files and relational tables
Processing: Sorting and sequential processing
Load
Load
Data Mart #1 DIMENSIONAL Atomic and summary data Based on a single business process
DW Bus: Conformed facts & dimensions
Data Mart #2 ... (Similarly designed)
Access
Access
Ad Hoc Query Tools
Report Writers
Analytic Applications
Modeling: Forecasting Scoring Data mining

Figure 2 Framework for a Logical Data Warehouse Model in Enterprise Data Applications [18]

In data applications, various upstream applications enable end users to interact, with a focus on both predictive and descriptive data applications. This thesis specifically emphasizes descriptive data applications to explore the impact of different data models.

Descriptive Business Intelligence applications encompass quarterly reporting and real-time or customer behavior analytics.

1. Quarterly Reporting: These applications consume data on a daily basis, utilizing batch data pipelines for data transformation and transfer.
2. Real-time Analytics: In contrast, real-time Business Intelligence applications rely on stream data pipelines to process data promptly, supporting log analytics, event analytics, and customer behavior analytics.

Quarterly Business Intelligence applications are well-suited for Modern Data Stacks, making data maintenance straightforward. Data for traditional BI applications is often sourced from various data contractors, with measures like masking and filtering applied to comply with Personal Identifiable Information (PII) requirements. This masked and filtered data is denormalized for ease of querying by BI reports, typically using data marts. Real-time data analytics demand immediate data availability without significant delays. These applications are effectively supported by Live Data Stacks, leveraging tools that process data with minimal or no delay. The architecture of Live Data Stack includes methods to enhance logical and physical data models for low-latency data processing. One prominent approach is the "set and forget" architecture, where data is retrieved from the source and processed without formal ingestion, contrasting with Modern Data Stack methods. While Live Data Stack technologies are still evolving, this thesis will explore their maturation and effectiveness in enabling real-time Business Intelligence applications.

In this thesis, data platforms will be analyzed to support both real-time and quarterly reporting use cases within physical data storage models. Specifically, the adaptability of Snowflake for self-service BI reporting will be evaluated. Additionally, the impact of logical data models, specifically Data Vault 2.0 and Kimball, on the performance and maintainability of Business Intelligence applications will be explored.

Several metrics will be used to assess the effectiveness of combining logical and physical data models:

1. Delay Tolerance: The system's ability to handle data processing at different intervals ranging from one day to one hour or even one minute will be examined. This metric is crucial given the diverse timing needs of real-time versus traditional descriptive data applications.
2. Adaptability to Data Pipelines: The storage architecture and logical models should be adaptable to both batch and stream data pipelines. This flexibility ensures that the system can support various data ingestion methods effectively.
3. Easy Maintenance and Modification: In a competitive market environment, data platforms must be easily maintained and modified to accommodate incremental business changes. This metric assesses how well the system can evolve with organizational shifts, such as departmental changes or company acquisitions.
4. Handling Out-of-Sequence or Late-Arriving Data: Business Intelligence applications may encounter data arriving out of sequence or delayed. The data architecture should handle such scenarios seamlessly to maintain data integrity and accuracy.
5. Data Lineage: Understanding and documenting data lineage is essential, particularly in upstream data applications. Lineage graphs help detect null clusters or outlier data, providing insights for downstream data processing and troubleshooting.

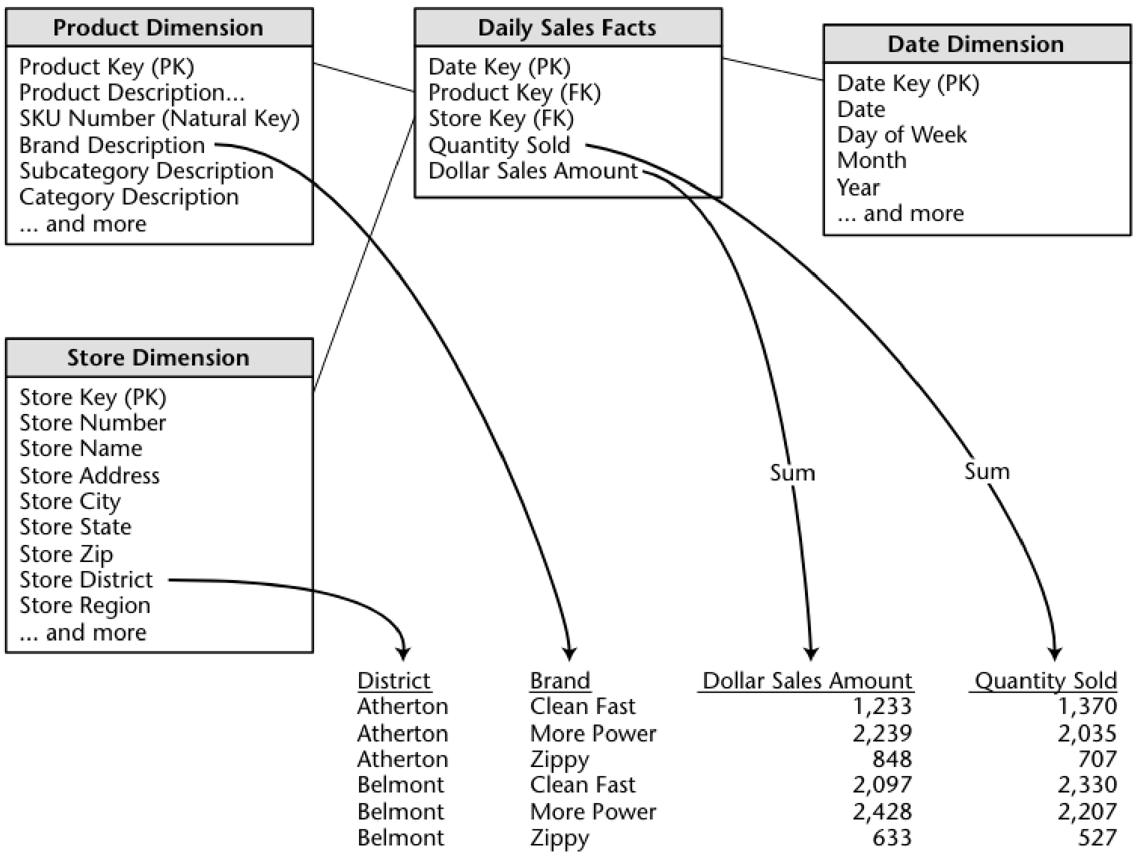
By evaluating these metrics, the thesis aims to provide insights into designing robust data architectures that support a range of data applications efficiently and effectively. This comprehensive analysis will contribute to improving the performance, reliability, and adaptability of Business Intelligence systems in today's dynamic business landscape.

### 2.2.1 Understanding Dimensional Modeling for Effective Data Representation

The Kimball model is widely adopted in traditional data applications across various industries due to its effectiveness in efficiently handling ad-hoc and business intelligence queries with minimal delays. One of its notable strengths lies in its ability to provide detailed event information with high granularity, emphasizing events occurring throughout different business processes.

In the Kimball model, events are organized and represented using Fact tables, which contain specific details about the events and their associations with Dimension tables that provide information about related stakeholders or entities. Fact tables serve to represent different types of events, with Transactional Fact tables being among the most common. Examples of events captured in Transactional Fact tables include sales transactions, orders, and requests that are anticipated to occur and conclude within specific time periods.

Alternatively, events that occur over time can be captured using Periodic and Accumulating Snapshot Fact tables. Periodic Snapshot Fact tables gather data over intervals of time (e.g., daily, monthly), allowing for the tracking of events over specific periods. Accumulating Snapshot Fact tables record snapshots of changing data, making them useful for monitoring how events evolve across phases or milestones.



| District | Brand | Dollar Sales Amount | Quantity Sold |
|---|---|---|---|
| Atherton | Clean Fast | 1,233 | 1,370 |
| Atherton | More Power | 2,239 | 2,035 |
| Atherton | Zippy | 848 | 707 |
| Belmont | Clean Fast | 2,097 | 2,330 |
| Belmont | More Power | 2,428 | 2,207 |
| Belmont | Zippy | 633 | 527 |

Figure 3. Insight Extraction from Multidimensional Kimball Model [18]

The Kimball model's flexibility allows for the representation of various event types using suitable Fact tables, tailored to specific data collection and analysis needs within the business environment. To enhance granularity and provide deeper insights, the Kimball model leverages refined Dimension tables. These Dimension tables can be categorized into different types:

1. Role Playing Dimension: Contains extended date data used to establish time hierarchies in BI applications.
2. Conformed Dimension: Shared across multiple Fact tables, providing consistent event detail representations.
3. Junk Dimension: Comprised of attributes representing states as flags, aimed at reducing the number of necessary joins in queries.

Overall, the Kimball model's approach to organizing events through Fact and Dimension tables offers a structured and effective framework for capturing, analyzing, and understanding diverse business events with varying levels of granularity and detail. This model is instrumental in

supporting robust business intelligence and analytics capabilities within organizations which they can extract insights from modeled data as seen in Figure 3.
Kimball's model can be extended intensively for each requirement. For example, we can represent event details with natural keys or we can replace natural keys with surrogate keys to enhance data governance. Also, we are free to select Snowflake or Star schema to address storage or query performance restrictions as well. Through data pipelines, we need extra configuration to address adding metadata columns as well. But for all disadvantages, the Kimball model has the noticeable performance to provide data or create data marts for predictive and descriptive data applications. It has conformed to best practices to address Change Data Capture, Slowly Changed Dimension functions, and Late Arriving Dimension kind of problems with just additional table attributes and small configuration in the continuous data flow.

### 2.2.2 Near Real-Time Data Vault Architecture

The Data Vault model is mainly used to ensure isolation between incremental business changes and represented data in data applications. The Data Vault model has different extensions but currently, Data Vault 2.0 is highly preferred regarding its extensions. Metadata and technical columns are also provided along with technical columns but regarding the type of table technical columns can be extended. In the Data Vault model, there are main tables that can be categorized into Hubs, Links, Satellites, and Reference tables. Additionally, we should have Query Assistant tables. Data, Bridge, and Poin-in-Time tables are the main components of Query Assistant tables which decrease required joins and enable access to data in the required period which is essential to Business intelligence applications and their used data marts.
Hub tables in the Data Vault model represent business objectives which should contain the natural key as the business key, hashed surrogate key, load date as technical columns, and source metadata column. If we would like to show the interaction of business objects, we should use Link tables. Link tables should contain hashed surrogate keys of interacted business objectives alongside load date and source attributes. There are some extensions of Link tables to show transactional events as a result of different business objectives. This extension is called Transactional Link tables which can support descriptive data applications with transactional hierarical data. Regarding business objectives and interaction between them, most descriptive and predictive data applications need the state of business objectives and the result of their interactions as well. Satellite tables are

utilized to show the state of business objectives which should contain hashed surrogate keys of Hub or Link tables, descriptive payload data alongside standard technical and metadata attributes. We can use different types of Satellite tables to address the unique requirements of business intelligence applications. We can utilize Effective Satellite tables to represent discrete payload data of Link tables between valid periods. The mentioned valid period was generated using additional start date and end date technical columns. For instance, if we need to represent multiple engines of one customer, we can use Multi-Active Satellite tables which contain multiple valid at the same time time. Additionally, we can address out-of-sequence data, we can use Extended Tracking Satellite tables which keep track of loaded records. These tables help us keep detailed payload information of business objectives and their relation results efficiently.

After keeping payload data and success isolation successfully, we need to optimize performance quieres because the Data Vault model needs to complex join operations to access a state of business objectives. For this purpose, we should use multiple, Query Assitant tables. The first and required table is the As of Date table which just keeps dates in certain order. The date attribute of the As of Date table is used to initiate Point-In-Time and Bridge tables. Point-in-time tables describe payload data from multiple satellites that split on the date column of the As of Date table. In this way, we decrease required joins to access a state of business objects and increase the performance of Business intelligence causing fetch heavy queries. Additionally, we can access current valid relations of business objectives and their results using Bridge tables. Bridge tables are a special version of Link tables that reduce required join operations. It uses descriptive payload data from Effective tables and contains current valid relationships between hub and link relations.

### 2.2.3 Data Vault Techniques to Support Self-Service Business Intelligence Requirements

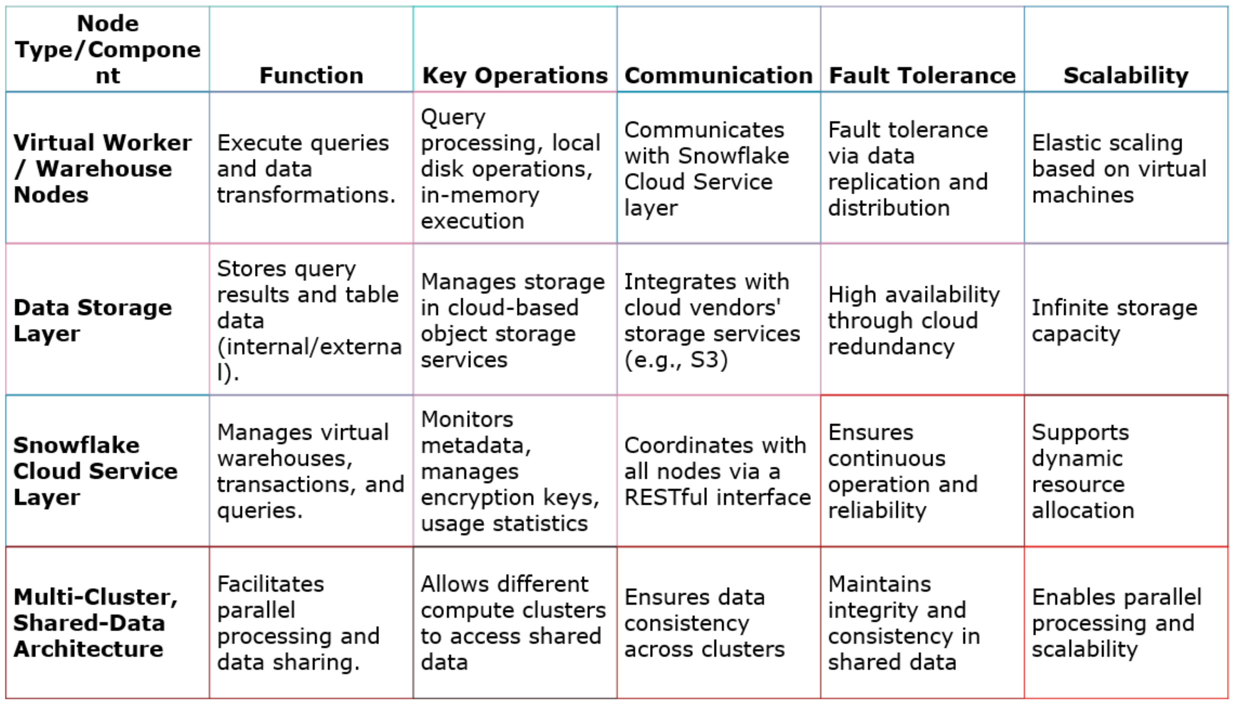
To provide correct information to upstream data applications such as predictive and descriptive data applications, we should handle maintenance tasks. These maintenance tasks address inserted data to the data warehouse without any conflict but it causes data anomalies in Business Intelligence reports. The most common problem for this is out-of-sequence loads, it can occur related to some delayed or dependency-based data load. Out-of-sequence data can lead to erroneous reporting, delays, and reloads but we can have an automation pattern

Batch files caused out-of-sequence loads mostly because file-based uploads can follow missing file distributions inside selected bucketing providers. We can not fully eliminate file-based batch operations because many data platforms provide continuous data flow in this way. While we get out-ot-sequence problems through this traditional approach. We need to roll back the previous data batch and get sequence order in the data warehouse or data mart. Following rollback operation downstream, quantive and qualitative business intelligence reports need to be rolled back or they will be subject to data lineage in the worst scenario. File-based incremental data ingestion and processing use date attribute which relies on the source because the source data is presented in the most relevant. Data in the source are categorized as transactional data which just show the current state of business objectives at first glance. However, all upstream applications aim to rely on a sequential order of data and come with continuous dataflow over data application but for some reason, this logic can be broken. Thus broken sequential order leads to business data integrity issues which can be seen as data anomaly or in technical debt upstream reports. To address this problem, we should use exact automation patterns that they handle through data upload. Through data pipelines, we should divide data pipelines into two main parts. The first part of pipelines is responsible for reading data from source which we can use maximal timestamp or batch partition approaches. In our case of out-of-sequence data load, we should pay attention to the second part of incremental data pipelines which is responsible for writing operations. To address out-of-sequence data, automatic tracking and correction patterns should be used. The advantages of Data In Data Vault 2.0, we can use a special kind of Satellite Table which this table keeps track of all loaded data of business objectives and the relation between these business objectives. These kinds of tables are called Extended Tracking Satellite tables which contain technical hashed surrogate keys from referenced Link, Hub, Satellite tables, load date, and source metadata column. We explain automatic tracking and correction patterns using the Extended Tracking Satellite table in real cases. We imagine we maintain a data warehouse of online payment services. Our ship of customers visits A city on Monday and Friday. He also visits hometown B on Tuesday. In logical order, the state of placement should be ordered A(Monday)->B(Tuesday)->A(Friday) but unfortunately batch file contains B replacement state faces abortion regarding dependency problem. Then we had A(Monday)->A(Friday) in currently, after we finish fixing the dependency-related problem we insert data about B. Finally, we faced this data A(Monday)->A(Friday)->B(Tuesday) inheritance. Also, we can imagine just one row shown for each location because our descriptive payload checking is done through the location column in Satellite tables then data about Monday and Friday will contain the same as a different column which generates a derived column based on descriptive payload data. Used logic in reading operation look A(Monday) and A(Friday) which they have the same hashdiff and it decides okay if ship stays on the same location and new data is coming which it has different hashdiff. In this case, it eliminates the record about A(Friday) and replaces it with B(Tuesday). Also, it replaces from date to Tuesday instead of Friday on data of A(Monday) because an update occurs on the data ingestion order. To address this problem, we need the table to save all load data and compare it for correction. Indeed, it has some similarities to the implementation of the reconciliation pattern which we use data from the third table. This third table in this case called the Extended Tracking Satellite table, is updated or inserted with data, and checking operations occur to keep timeline correctness. In technical depth, it can be accessed by using the Merge pattern through writing part of incremental ELT pipelines. Using the Merge pattern, we can go away from table locking which can occur regarding transaction policies of the utilized data platform.

Extended Tracking Schemas can be used to store out-of-sequence concerns and handle them in data applications. These schemas record anomaly and discrepancy-related information that can be used to evaluate and control technical debt in the data pipeline. Out-of-sequence issues can appear as inaccurate depictions of the status of business objectives in the context of business intelligence systems unless the timetable is adjusted. Applications for descriptive data can be used to show and evaluate this disparity, Descriptive data applications can employ correlation differences and anomaly detection to track down the source of an issue by utilizing technical debt upstream. This method makes it easier to comprehend problems with data integrity and makes it easier to take corrective action to guarantee the correctness and dependability of data throughout the business intelligence ecosystem.

### 2.3.1 Nodes for batch data handling over Dremel and Snowflake’s Data Cloud

With several complex data governance options, Snowflake is a cloud-based SaaS data warehousing solution that allows ELT transformation using both Python and SQL. Since Snowflake does not support usage of the offline version of the product, our connection to Snowflake's global self-isolated data mesh must be active.  Distributed computing improves it by enabling us to split the complexity of several queries and run them simultaneously. In addition to distributed computing using various cloud providers' virtual nodes, we may benefit from nearly infinite computational storage. The object storage of various bucketing systems improves virtual compute storage. More elastic computing data processing is achieved by storing a significant amount of structured and semi-structured data in object-based storage services. Furthermore, it's worth noting that Snowflake is a versatile system. It serves multiple users (multi-tenant), ensures secure transactions, boasts high scalability and elasticity, and supports SQL comprehensively. Additionally, it comes with built-in features tailored for semi-structured and schema-less data. If we want to make our data application more accessible to other inner or outer organizations, we can do so using the sharing functionalities of Snowflake because it uses shared nothing architecture at all. These advantages allow us to collect and store data from different sources such as Enterprise Resource Planning, Customer relationships, and other company internal and external applications. Also, we have a great chance to collect and process schemaless and event-based data which came from logs, sensors, the Internet of Things, and different growing volumes. Many relevant solutions in the industry rely on Hadoop-based solutions but Snowflake uses Spark-based solutions which create suitable ground execute operations in-memory based. As was previously noted, Snowflake is a tool that can be used to build data warehouses. Its goal is to divide processing and storage, which is in accordance with the idea that data warehouses may improve ELT operations. Since SQL should be used to create declarative or dynamic pipelines, Snowflake provides the SnowSQL SQL standard, which supports ANSI SQL and allows ACID transactions. SnowSQL can be used to build dynamic data pipelines that manage semistructured data in real time and guarantee continuous data flow. Also, we can generate schemas from these semistructured files using different Snowflake functions. Snowflake could guarantee availability and durability against accidental data loss because all data rely on cloud infrastructure. Additionally, data are distributed over regions which leads to ensuring data expending. Snowflake does not need to execute maintenance tasks because the Snowflake Computing Layer handles many parts. One of the maintaining tasks is comparing data on tables. Using the compared algorithm on Snowflake is property and it is not open source. But it may have many similarities with generic compromised algorithms which Druid uses as well. In this way, I can mention all data on the Snowflake network is encrypted and Role Based Access Control exists which all compute resources require on a creational step. Additionally, we have a chance to ensure fine-grained access control on the SQL level through DDL and DML operations.



| Node Type/Component | Function | Key Operations | Communication | Fault Tolerance | Scalability |
|---|---|---|---|---|---|
| **Virtual Worker / Warehouse Nodes** | Execute queries and data transformations. | Query processing, local disk operations, in-memory execution | Communicates with Snowflake Cloud Service layer | Fault tolerance via data replication and distribution | Elastic scaling based on virtual machines |
| **Data Storage Layer** | Stores query results and table data (internal/external). | Manages storage in cloud-based object storage services | Integrates with cloud vendors' storage services (e.g., S3) | High availability through cloud redundancy | Infinite storage capacity |
| **Snowflake Cloud Service Layer** | Manages virtual warehouses, transactions, and queries. | Monitors metadata, manages encryption keys, usage statistics | Coordinates with all nodes via a RESTful interface | Ensures continuous operation and reliability | Supports dynamic resource allocation |
| **Multi-Cluster, Shared-Data Architecture** | Facilitates parallel processing and data sharing. | Allows different compute clusters to access shared data | Ensures data consistency across clusters | Maintains integrity and consistency in shared data | Enables parallel processing and scalability |

Table 3. Main features of nodes for handling and serving batch load in Snowflake global mesh

We can select three main cloud vendors as infrastructure for our data warehouse or data application but the most well-suited is Amazon Web Services cloud vendor. Inside Snowflake, users can create accounts and these accounts can have databases that can use nodes in Table 3. It should be considered that the data of other users are isolated which means a single application serves multiple users over a global data mesh. For providing privacy and security Multi-tenancy has been adopted within Snowflake Row Access Policy (RAP). We can create different roles and roles can be assigned to different data sources such as tables, views, and materialized views. Also, it is guaranteed by Snowflake that the offered SaaS application will be deployed in the Virtual Machine solution of the selected provider. In this way, we can achieve high-level isolation of data and database objects which we can keep our data and properties invisible to others. Snowflake's other unique future is COTS (commercial off-the-shelf) software solutions in which you just need to pay executing time not for used data because all data are stored in object bucket storage with advanced data compressed algorithms. We can benefit massively from parallelism on Snowflake whose default provided consistency policy does not lock tables while query executions. Table point lookup queries are enhanced by SOS. A single or few different searches are returned by a point lookup query. It comes after clustering as a backup plan.

At first glance, Snowflake is an advocate of enterprise-ready data applications that can guarantee interoperability by providing high availability. Many large organizations and financial institutions leverage the unique features of Snowflake's service-oriented architecture. This approach allows competitive companies to develop independently scalable services and achieve high fault tolerance against undesired practices. Snowflake's architecture resembles different nodes in engines like Druid and Dremel (used by BigQuery). Each service within Snowflake communicates with others through a RESTful interface, enabling the implementation of best practices in microservice architecture within a finely-grained data warehouse platform. This architecture promotes modularity, scalability, and robust communication among components, enhancing the overall efficiency and reliability of the data warehouse ecosystem. If there is some need to talk about the services of Snowflake, I can start from the data storage layer of Snowflake which is mainly employed to store query results and table data. These table data can be external and internal which affects the capability of the data storage layer itself. Mentioned information about queries and table attributes mainly stored on object-based storage services of different cloud vendors. The second most important layer of Snowflake is the virtual warehouse which represents a distributed computing future. Virtual warehouses rely on different types of virtual machines of nodes based on utilized cloud vendors and the cloud environment provides elastic clusters for query executions. As previously mentioned all data layers are based on the Snowflake Cloud Service layer which is responsible for managing virtual warehouses, transactions, and quires. Additionally, it monitors and serves metadata information about database schemas, encryption keys, and usage statics about table attributes. These three service layers of Snowflake contain its main functional and competitive advantage against other SaaS-based cloud platforms.

#### 2.3.1.1 Virtual Worker or Warehouse Nodes

Virtual warehouse on Snowflake represents the work power of nodes that aim to execute written queries and we can determine the size of virtual warehouses in different sizes to meet our special requirements. This variety of virtual warehouses comes from used nodes or virtual machines. Regarding shared-noting architecture, commodity and scalability of hardware should be guaranteed and all executions take place in the same hardware with others. However, we can see every query processor executes queries on its local disks. Giving the ability to execute and derive results in memory enhances the performance of data operations because we can access data in local disks more efficiently. Additionally, each node in the spate local disk gives us a chance to store tables over nodes horizontally. Horizontal table storing is beneficial because each node can access the same part of data and we can keep different partitions of data in this way. Therefore, it should be mentioned that each separate node has the responsibility for storing rows of these tables in its local disk. These unique features of the shared-noting architecture are so suitable for logical modes one of which is a star schema. If we look start schema, we can see there are dimensions and fact tables to show event and event details that occur during a business process. When we look at shared-noting architecture, we can see little bandwidth can be used to join broadcast dimensions with portioned fact tables. This small bandwidth allows us to run expensive analytical queries with high performance. Most of the time, we need to share our data with other internal and external data consumers, and our data-sharing service benefits from this architecture. Shared-nothing architecture came with little contention for shared data sources and local disks where partitions of table information are stored. Unfortunately, we could not achieve pure shared-noting architecture at all. Before Snowflake, Spark and Hadoop-based solutions could not be part of pure-noting architecture because their approach has logical pitfalls. The first broken token in logic is each node has the same responsibility and runs on the same hardware. It means tightly coupling storage and compute resources inside nodes. Another not meted requirement is different priorities on workloads and hardware. We can consider, that different workloads or queries can be executed on our data platform which can be analytical, maintenance, or data load. All quires have their performance requirements. For example, data load queries need to write data from sources based on different data reading and writing patterns to provide continuous data flow through the data platform which requires high I/O operations alongside high processing capability. Most probably, stored and transformed data can be required by analytical operations which means low I/O operations with high processing capability. As we can see different quires require different performance requirements on the hardware side but most legacy shared-noting architecture can not meet this demand. All legacy aprachs rely on homogeneous hardware which nodes process and store data there. Therefore, homogeneous hardware can not meet the requirements of different focused workloads. Also, we should talk about another pitfall of legacy solutions,

This architecture incorporates several critical components aimed at optimizing resource management, scalability, and performance within a distributed data system. Here's an overview of the key elements and their functions:

1. Combining Query Processing and Data Shuffling on Nodes: By consolidating query processing and data shuffling on the same node, data travel is minimized, and proximity is maximized, leading to improved efficiency in processing tasks.
2. Impact of Membership Changes on System Performance: Significant changes in cluster membership, such as node additions or removals, may necessitate large-scale data redistribution to rebalance the cluster, potentially impacting system performance temporarily.
3. Challenges of Upgrades or Modifications in Shared-Nothing Architectures: In shared-nothing architectures where nodes share common hardware, upgrades or modifications can impact the entire system, requiring careful planning to minimize disruptions.
4. Utilization of Various Node Types in Cloud Environments: Leveraging different node types like EC2 instances in cloud environments enables elastic scaling, allowing resources to dynamically adjust based on workload demands, leading to improved availability and performance.
5. Separation of Storage and Computing Resources: By separating storage (e.g., S3) and computing (e.g., Snowflake shared-nothing engine), this architecture adopts a loosely coupled design that enhances flexibility and scalability.
6. Optimization of Compute Nodes: Compute nodes utilize high-speed SSDs for faster read/write operations and local drives for temporary data storage and caching, optimizing performance and minimizing network traffic.
7. Multi-Cluster, Shared-Data Architecture: This design enables different compute clusters to access shared data while ensuring data consistency and integrity. It supports parallel processing and scalability, facilitating efficient data processing across multiple clusters.

In summary, these components collectively contribute to a robust and scalable architecture that addresses the challenges of distributed data processing, enabling efficient resource utilization and enhanced performance in cloud-based environments.

### 2.3.2 Nodes for real-time data handling over Druid

Druid contains several nodes and each node type has its responsibility. Additionally, intercommunication between nodes is minimal and livability would not be affected by communication failures of different types of Druid nodes. Real-time, broker, coordinator, and historical nodes are the most used node types in data application infrastructure which are visualized in Figure 6. Druid is the implementation of concepts that must guarantee low latency, quick data exploration, and low latency. Druid is designed for real-time and batched data from different data sources while keeping access to data in state. Before Druid, we need to employ a stream processing platform to join and process data while event ingestion from real-time data sources or message buses. However, it provided many data-related operations such as Joins, and Look-ups to process data after ingestion. For processing efficiency, set and forget-based data processing applications can be used to reach and process data during data transmission between source and destination but they do not support data compression. Druid is designed to be utilized as a service layer for Kafka Flink Druid (KFD) architecture which is the presentation layer of a high throughput message bus. Therefore, it can be surely said that Druid can be the main source and query endpoint for the exploratory data analysis process. Another advantage of Druid is uses a concurrency control mechanism for decision-required scanable versions of segments and handles hieratical immutable files effectively.

There should be data sources for stored and represented data in Druid. These data sources provide us with timestamped events which are used for first-level query pruning and partitioned into segments. Kafka is the best option for handling and serving this kind of event in low latency for us. We can create open and direct communication using APIs of microservices to ingest data into Druid but we need to message bus to increase fault-tolerance and eliminate duplicates in case of failure of real-time nodes which can be noticed from Figure 4. After ingestion is completed successfully, we will have a time interval of data that can be used to attach files to version string by the end of their name aliases. Time interval and string versions allow Druid to eliminate old and unused data which can increase the performance of queries on recent data.

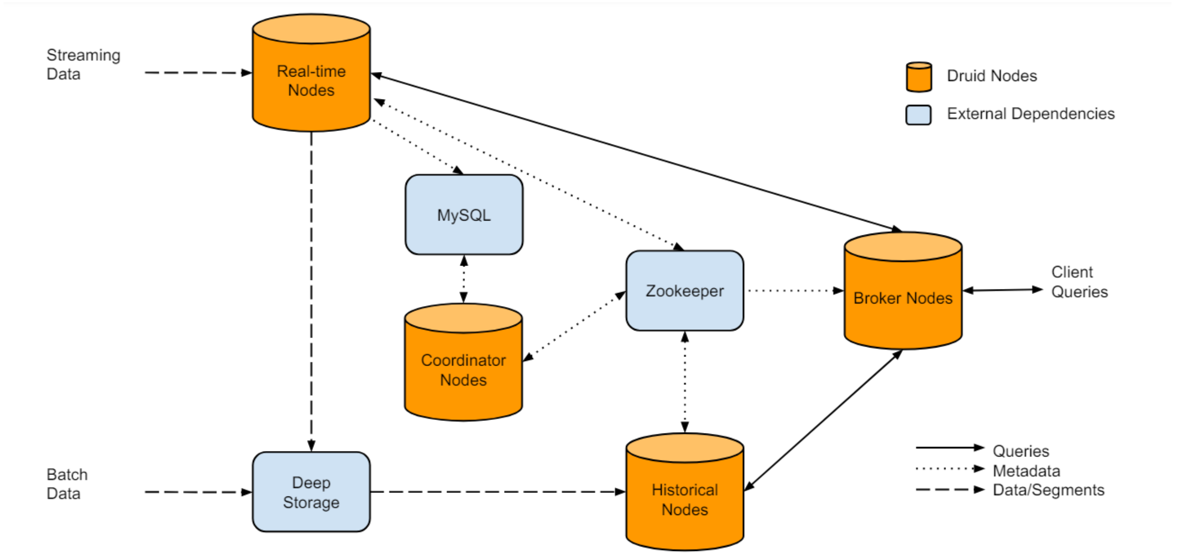


Figure 4. Visual Representation of Data Flow in a Druid Cluster Architecture [19]

Additionally, fault tolerance is another topic in Druid because different nodes are automatically yielded in specific periods. In this way, the availability of nodes for end-users and internal communication keeps them alive in most cases. Also, this coordination relation between internal communication between nodes guarantees a link between read and write operations. Therefore, Druid is called a self-contained database for different intervals of data sources. In this way, we can reach a high level of consistency and low latency when working with different DML operations on batch and real-time data. . Druid should be significantly paid attention to different nodes on main cases which is easily noticed in Table 4.

| **Node Type** | **Function** | **Operations** | **Fault Tolerance** | **Scalability** |
|---|---|---|---|---|
| **Real-Time Node** | Ingests and indexes real-time data. | Indexes and merges data, informs ZooKeeper. | Reloads indices via Kafka offsets, replicates data. | Handles up to 150,000 events/second. |

| | | | | |
|---|---|---|---|---|
| **Historical Node** | Manages and serves historical data. | Loads/drops segments, uses Local Cache. | Uses current state for fault handling. | Supports large data volumes via parallelization. |
| **Broker Node** | Merges query results, query endpoint for clients. | Executes queries, caches results. | Maintains previous cluster state. | Balances high query loads, supports caching. |
| **Coordinator Node** | Manages segment distribution and replication. | Manages segments, load balances, handles metadata. | Ensures segment distribution with redundancy. | Adapts to data access patterns. |

Table 4. Druid nodes on mission critical categories

For a better understanding work process in Druid, we should realize the data handling process in Druid. Druid requires a message bus for eliminating duplicate values in case of some unexpected fault and comparing events if needed. We can consider that Kafka is selected as a message bus and we have two data about clicking and advertisements which is common for most real-time cases. First of all, we must know all streams or data pipelines are logically related to each other. In the first step, data shuffling must be done to guarantee two different local events on the same topics which will be used in the following step. In the second step, we join events and can add flag columns to represent metrics. Following the second step, we can process and clean data in the third step. After joining and clean operations are completed successfully, we can deliver final data to Druid Real-Time nodes or Historical nodes over Deep Storage.

For a better understanding work process in Druid, we should realize the data handling process in Druid. Druid requires a message bus for eliminating duplicate values in case of some unexpected fault and comparing events if needed. We can consider that Kafka is selected as a message bus and we have two data about clicking and advertisements which is common for most real-time cases. First of all, we must know all streams or data pipelines are logically related to each other. In the first step, data shuffling must be done to guarantee two different local events on the same topics

which will be used in the following step. In the second step, we join events and can add flag columns to represent metrics. Following the second step, we can process and clean data in the third step. After joining and clean operations are completed successfully, we can deliver final data to Druid Real-Time nodes or Historical nodes over Deep Storage.

#### 2.3.2.1 Real-Time Node

Real-time nodes are the initial and main power functionality of Druid. Real-time nodes are responsible for interacting with real-time consumers and buffer applications to handle ingestion row data, and persist and merge processed data to batch-ready data sources. Through Druid internally, it is inlined to communicate with Broker, MySQL metadata database, and with Deep Storage cluster to traffic metadata, segments, and query data access. The data handling process through Real-Time nodes contains ingestion of row data, assigning index to row data in heap memory, and directing indexed data to disk regarding heap overflow problems. If the default or configured time window is finished persist data off-heap memory and write to Deep Storage. Indexing data allows us to process this data during the validation of the window period if we get a query access request from the Broker node. Indexing coming row data and creating immutable blocks of data were the first processes in Druid's real-time data handling process. The indexing process occurs on disk which allows Druid to fast processing but merging and storing immutable blocks of data take place on Disk. Alongside with indexing process, the handoff process through Druid is highly selective.

Through real-time data handling, our data will be accepted configured, or defaulted in the period. Regarding this period, our Real-Time nodes inform ZooKeeper which it serving data, and other services such as Broker can direct queries to process data. Also, as I mentioned we will have a window period time slot that follows in period for writing data from the in-memory index to disk which is ready for batch storing. The window period is important to increase the resistance of our data handling to process data loss from delays in event delivery. After the in period and the window period concluded successfully, Real-Time nodes inform ZooKeeper that segments or immutable blocks of data are available and queryable now which Real-Time nodes announce it will not serve data relying on a specific period. In case of data drop off required by some reason in Real-Time nodes, it can be done if Coordinator nodes can give the correct location of data in Real-Time nodes.

If we talk about data handling we should touch on availability and scalability at the same time. Foremost, the main data ingestion of Druid is Kafka applications. Therefore Druid adapts to increase durability against Kafka applications. For example, Real-Time nodes behave in Kafka applications as message buses which if there is some failure we can reload peristed index based on last commitment offset. Additionally, we can initialize multiple Real-Time nodes for replication of data. In this way, we can increase the performance of Real-Time nodes of Druid to 150000 events per second. Our data processing scalability relies on a Real-time node schedule that aims to merge locally persisted indexes.

#### 2.3.2.2 Historical Node

There is no direct communication between nodes inside the Druid database. For regulating segment loads and serving we need assist of Historical nodes. Historical nodes are powered by shared noting architecture which we can see in many Online Analytical Processing data platforms such as the Dremel engine of BigQuery. As I subsequently mentioned, all communication should be maintained through direct communication with the ZooKeeper. Historical nodes first announce to ZooKeeper about its online state and needed data serving. State and data announcements should lead to special instructions against segments. These instructions should be accepted and translated by ZooKeeper and they must contain information about where an immutable data block or segment is located, how it should be decomposed, and how it should be processed. Most of the time, these accepted instructions aim to drop or load immutable blocks of data or segments.

Through Druid, we can improve the performance data access process using Local Cache for this purpose. At first glance, Historical nodes communicate the result of the desired query in Local Cache if it is found, They inform the ZooKeeper. In case it is not found in the Local Cache ZooKeeper look mentioned location in the instructions. If the segment is available, the ZooKeeper announces data source is ready for queries. Additionally, Historical nodes are inclined to process segments in available cores to avoid some unexpected faults. Therefore, we should understand each segment has its upper limits to storing immutable data. Historical Nodes are the most important part of Druid because all data are stored and served there. For this reason, it was called as usual Druid production node.

We can look deeply at the consistency details of Historical nodes considering selected consistency policy and availability capabilities. Historical nodes rely on segments which segments are immutable data blocks based on time auto-detected time partitions. Immutable time blocks give us the possibility to use a read consistency policy which improves the performance of our data fetch operations. Read consistency policy allows us to use the benefits of parallelization which default aggregation and concurrent scan mechanisms can exist without blocking data sources or tables. Additionally, if we look availability of Historical nodes we can see it is rely on ZooKeeper because all communication takes place over ZooKeeper. However, this dependency is not at a high level because Historical nodes keep the current state of data and it compares this state with ZooKeeper which means we can use the last state of data if there is some fail happens.

#### 2.3.2.3 Broker Node

Druid is a real-time high-performance database that should accept SQL queries to retrieve results for us. To achieve this, Druid initializes Broker nodes. These nodes are responsible for merging partial results obtained from both real-time and historical nodes. They communicate with ZooKeeper to obtain information about the location of queryable segments. This communication with ZooKeeper enables executed queries to identify the root data sources necessary to return the desired results effectively. Most of the time same queries are executed at the same time for a long which means saving the results of queries in cache can increase result retrieval time in advance. We have different strategies to implement database disk buffering to change the results of subsequently executed queries. LRU-K is used in Druid to keep tracking recent K references to highly used database pages. LRU invalidation strategy mainly aims to assist local heap memory. Additionally, it simulates external Memcached stores which rely on distributed key-value sources. If we would like to understand workflow query execution and result delivery. First of all, the Broker gets an SQL query which leads to a search query result on chance if the result can not be found, it goes through Historical and Real-Time nodes. I should say Real-Time nodes rely on query results that can not be found in Cache because Real-Time data is mutable and it is open to change regarding new data from the message bus if some failure occurs. Real-time data can be permanently changed and caching results can be unreliable. Also, Caching inlined to add extra durability alongside query performance improvements. In case of all, big scale fail occurs on all nodes, some query results can be reachable to us. Same as per previous information about clusters, the livability of the Broker node does not fully rely on ZooKeeper because the Broker node saves the previous state of the cluster and segments which adds extra realized against accidents. As it was described, Broker nodes are utilized as query end-points for most database clients. To understand and clarify the pictures, we should understand the relation between the Broker and the ZooKeeper. Indeed, the Broker does not rely on ZooKeeper to enhance of fault-tolerance but it is significant for Broker nodes. Desired data can be in Historical and Real-Time nodes which communication is provided by ZooKeeper. Also, expected data can be split between Real-Time and Historical nodes, and data are merged by ZooKeeper for accepted queries of the Broker node. If we talk about querying immutable files, we should keep intervals of query and timelines. Broker nodes easily handle intervals using look-ups into timelines to get the most recent data using metadata columns and string versioning of segments. The result of this timeline should be considered for remapping results into Broker nodes in case of addressing results of queries in cache and Historical. In this way, we can enhance the concurrency control mechanism of Druid more effectively. Additionally, these kept metadata can be used for the drop-off of unneeded segments more quickly instead of creating communication with other nodes over the ZooKeeper node.

### 2.3.2.4 Coordinator Node

We have segments of immutable data blocks in Historical nodes and we can query them with Broker nodes. But Broker nodes need to get required and must eliminate segments from Historical Nodes to which Coordinator nodes take responsibility. Historical Nodes know how they can load, respond, and drop immutable data blocks but they need the location of these blocks which the Coordinator establishes this important communication. Additionally, all communication occurs over ZooKeeper. However, there is no information is given about data management and distribution of segments over Historical nodes. If we would like to know the main cluster behind it, we should point Coordinator cluster of Druid. The coordinator cluster is responsible for loading new data, replicating segments, dropping outdated ones, and moving data to the load balancer. We can manage immutable segments using multi-version control concurrency control swapping protocol. In this way, we can maintain stable desired views of data. Redundant backups can be easily get using multiple Coordinator nodes which would be controlled by a single node. This single Coordinator node over others should be selected in the leader election process. Additionally, all Coordinator nodes should use ZooKeeper to maintain and establish connections between other nodes. When Coordinator nodes interact with ZooKeeper it aims to compare the state of expected and state of actual connection details over other Historical nodes. Coordinator nodes should maintain two connections. The first connection should target to ZooKeeper to get ground for connection to other nodes. However, the second connection creates the same point with the internal MySQL database where the Coordinator cluster can access metadata information about segments in Historical nodes. This metadata information contains data about served segments and data governance. Served segments are mainly tagged after the window period is finished in Real-Time nodes which means immutable data blocks are ready to use against query contexts from the Broker node on Historical nodes. At another glance, metadata about data governance contains how segments have been created, how should they be destroyed, and how they must be replicated. Also, segment creation related metadata point operational and configuration parameters in the same way. There are some roles between Coordinator nodes in which we can load and drop data from Historical clusters. Additionally, Coordinator nodes can assigned to segments or immutable blocks of data to different Historical node tiers. These Historical node tiers differ regarding data access rate and we can manage different replicas of segments in these tiers using rules. These rules came from an internal MySQL metadata database which executed in exact periods against segments on ordinary replicas or different replicas in multiple tiers of Historical nodes. All mentioned rules contain first fit apply logic which executes against the nearest segment that is suitable regarding information that came from ZooKeeper. As you consider, the reaction is important in retrieving the desired result of the query, therefore, the same load distribution algorithm is used for this purpose. Through rooting queries on the right data real-time and batch data sources, load balancing should be considered. Also, we should keep this simple logic in mind, smaller segments can boost the performance of queues and load balancing using this simple logic. Through load balancing, query patterns take a point for replication and distribution and these operations occur regarding segment source, recency, and size. The ability of Broker nodes to behave the same as other nodes was talked about in previous chapters.

### 2.3.3 Compression of Immutable File Formation Methods for Data Applications

The storage format of Druid is based on columnar storage format which is highly selective in Online Analytical Processing databases and file formats. Data and column aggregation can be stored in this columnar which makes it easy to access data through aggregation results of data. The main representative of data in Druid is the data source which contains segments. These segments are partitioned timestamped events from real-time data consumers or messaging services such as

Kafka. If we need to use metadata properties of data sources or tables in Druid, we should keep in mind that version strings on name aliases of data sources will be considered. When we input a query to the Broker node, Druid executes a read operation that relies on the latest one based on the version string. Additionally, the concurrency control mechanism of Druid benefits from version string which partition data into segments, and these segments are replicated and distributed by Coordinator nodes. As previously mentioned, version strings are significant in indicating the freshness of data. Therefore this specialty in the file of Druid allows us to access newer versions of data conveniently. Alongside data types in Druid, we can see there are three main data type representations. The first one must be a timestamp which is used for string versioning of data based on specific 1 hour time intervals. The second common data type is the string which is convenient for representing dimensions. The last one is numeric, numeric data type is highly used for metrics which can be generated during ingestion or after ingestion process. To increase the performance of data operations in dimension columns, inverted indexes have been used by default. Also, a log-structured merge tree is utilized for data that have been newly added by Deep Storage or Real-Time nodes. Real-time nodes mainly store new data in heap-based JVM stores in key-value pairs which are highly optimized for writing operations. Hadoop-leveraged MapReduce jobs are responsible for the partition of data for immutable blocks or segments and we can see the same approach for Spark nodes. As previously mentioned, traditional sources of data to segments are real-time nodes but Deep Storage can be employed for accepting static files. It leverages static files and creates immutable blocks that can be served over Historical nodes. The assistance of Hadoop-based technologies is not limited to establishing new immutable blocks. Also, they can be used for adding metadata to the MySQL metadata database of Druid. Hadoop Batch Indexer is used for adding metadata information to the metadata database of Druid which is kept for future usage in versioning segments based on ingestion time intervals. Additionally, I can clarify for clarifying all process segments are created when they are stored in Deep Storage. As I mentioned, metadata is used for the versioning of segments, but it was unknown when versioning was defined exactly. Version strings of segments have been defined when the Batch process started in Deep Storage. With the assistance of the Hadoop Batch Indexer, metadata is written and it will be used in Unifying Views where Coordinator Nodes can serve them to other nodes.

Through Druid, we can experience several functionalities that can be seen on traditional databases. One of these functionalities is the compression process which makes data in a relatively small size.

It is known that a small immutable block of data or segments can bootstrap the performance of quires. Through establishing, segments and immutable blocks for real-time and batch data, we should keep in mind that HashMap is used by Druid. HashMap is employed to finalize to creation of immutable data blocks and provide incrementally populated segments. It is defined as a data ingestion process. HashMap should be used for real-time and batch ingestion operations to make ready data in our service. From the fault tolerance side, copies of one segment are stored in multiple nodes for handling unexpected disasters. Several copies can be selected in configuration files.
However, data should be transformed against many well-known generic compression algorithms. Encoding is the first step for this purpose which encodes string-based columns into integer arrays which is more suitable for generic compression algorithms

## 2.4 Optimized Query Handling for Logical and Physical Components of Storage Model

Transactions involve sequential orders of Data Manipulation Language (DML) and Data Definition Language (DDL) queries, either individually or together. Stored Procedures (SP) and User Defined Functions (UDFs) can also participate in transactions, particularly in Extract-Load-Transform (ELT) data transformations. Although nested queries and semistructured file formats are common in data processing, Snowflake transactions do not allow nested transformations for consistency improvements. Each transaction is associated with a specific session, and Snowflake prohibits the mixing of sessions within the same Snowflake account, even in multithreaded environments.
In SnowSQL implementations, transaction management typically falls into two main states. The first state involves explicit declaration and management of transactions using transaction management keys. In the second state, transactions are automatically handled by Snowflake due to the default support of AUTOCOMMIT. However, certain scenarios, such as mixing data manipulation with data definition queries, can pose challenges that transaction management keys may not fully address. Yet, this is considered a minor issue within data warehousing operations.
When dealing with multiple transactions, Snowflake offers a scoped transactions concept. For example, if one Stored Procedure (SP) calls another within a transaction, Snowflake creates autonomous sessions for each, thus avoiding true nested transactions. Transaction management clauses can be either explicit or implicit; for instance, within a session, the first block of queries might use explicit transaction statements, while the second block does not specify any transaction

management keywords. In this case, Snowflake defaults to AUTOCOMMIT, treating all implicitly managed transactions as part of a single transaction, eliminating the need to maintain separate queries.

There are various isolation policies and levels implemented in data warehouse platforms to ensure transactional integrity and consistency. If we take our focus through Snowflake, we can see there is just one isolation level offered. This isolation policy is called READ COMMITTED Isolation which creates ground for queries using data from the table in the last commitment state. READ COMMITTED Isolation should be the main selection over Data Warehouse solutions because Data Warehouses must face reading-heavy queries from upstream such as Business intelligence applications. It comes with some disadvantages alongside high performance on data manipulation query executions. One of the most noticeable disadvantages is queries consider the last commitment state when they start processing data. We have query A which executes Table X but query B changes data in Table X during the execution of query A. Query A did not process or query data in the table because it was already changed by another query. In incremental continuous data flow, there can be situations where idempotency is violated during incremental writing operations. For this purpose, selecting a merge pattern allows us to consolidate multiple case-based operations under a single transaction, ensuring better transactional integrity and consistency. Our data operations can require other data manipulation operations such as updating, deleting, and inserting and we would like to keep track of data operations to avoid some disadvantages of READ COMMITTED Isolation level. We can use resource looking which can resist parallel operations. As previously mentioned, selecting the Merge pattern is important to ensure incremental and idempotent writing operations. However, it's important to note that write operations are not restricted by table locking alone in Snowflake. You can insert data into both internal and external tables within Snowflake alongside other Data Manipulation Language (DML) operations, even if the targeted resources are subject to resource locking. Snowflake's architecture allows for efficient handling of concurrent operations on tables, ensuring data integrity and consistency across different types of operations.

Effects of Snowflake transaction and isolation policies can be easily seen in real examples. Many organizations power their data application with Data Vault 2.0. Through the Data Vault 2.0 model, we write data from sources to different tables to show business objectives states, and relations. Common Hub tables can serve multiple business objectives and should be updated consistently to reflect changes across these objectives. This ensures that updates or modifications impacting various business processes are accurately reflected in the centralized Hub tables. New data to this Hub table require parallel updates from stage tables to the representation of these objectives. If we would like to update the Hub table same time from two different stage tables, we may face duplicate values because Snowflake employees READ COMMITTED Isolation level which parallel transactions can not see each other. Additionally, data Insert operations can not be subject to table locking in Snowflake. Neither Isolation nor resource locking can eliminate the possibility of duplicate values through Insert operations. Therefore Merge pattern should be consumed which resource locking can put on hold until other queries are finished. These different approaches of isolation and resource locking on different data manipulation operations split into passive integration and concurrency loads in more advance. For this reason, we can mention Hub tables which do not have dependency regarding isolation differences can leave target tables with the same integrity level.

### 2.4.1 Search Optimization and Query Acceleration

Search Optimization Service is designed to improve the performance of queries that return one or a relatively tiny number of rows. Additionally, scanned columns of the table should contain at least more than 100000 unique values. We can add lookup and analytical queries which they selective point lookup tables containing supported predicates. These selected predicates have importance because the Search Optimization Service just addresses queries in which there are equality searches, search columns in VARIANT data types, search relies on geography functions, and lookup operations are enhanced by substring and regular expressions search. If we look at industry implementations of Search Optimization Services, we can see fast response required descriptive data applications such as Business Intelligence offers open ground for it. Additionally, predictive data applications require to access the small values of data from big data sets which require extensive set filters. If these filters are based on selective predicates, the Search Optimization Service can bootstrap the performance of these queries as well. Search optimization Services can improve the performance of quires that use substring, regular, and equality operations because these quires can return relatively small numbers of results and we just need to look at immutable data blocks for small-size desired results. Also, queries that use data from variants, semi-structured arrays, and objects can benefit from Search Optimization Services if they use selective predicates.

The same thing is acceptable for geography functions that run against GEOGRAPHY columns in big volume distinct value columns.

Search Optimization Services can improve the performance of search queries using special data structures. This data structure is called a search access path which is designed to use skip micro-partitions based on looking data and the Snowflake Copmute Service layer does not need to scan unnecessary micro-partitions as well. We can manually enable Search Optimization Service to load this data to the search access path. In case any data update occurs during the optimization process Snowflake automatically handles new updates and reflects to our search access path data structure. Therefore, stakeholders of Snowflake data warehouse do not need to manually execute maintenance tasks or lock resources during the optimization process. Additionally, the cost for these operations is automatically factored into other services provided by Snowflake. Regarding noticeable performance improvements, addressed data sources should meet some requirements. The initial requirement is that the specified columns of the table should not be primary cluster keys. Following this, query-touched columns need to very high cardinality level which counts to 100000 distinct values in each of the majority of involved columns. Lastly, we have restrictions on column data type granularity, column types can not be FLOAT and GEOMETRY. Additionally, we can see the same reactions over data types in sessions, there is no type casting accepted in table columns. But type casting in static values in filtering operations and type casting from int and float to string is acceptable for optimization by Search Optimization Service. Our views and secured views can benefit from the Search Optimization Service because views are created on SELECT statements if these statements contain accepted clauses and there is not any restring data type or casting operation, our views can benefit from it. As different types of views, our joint operations can benefit from this. Some filtering or selective predicated can be used to improve the performance of join operations which opens the door for Search Optimization service if addressed data types meet requirements too. Unfortunately, we have more restrictions for Search Optimization Service regarding to type of data sources and column definitions. Exactly, External and Dynamic tables cannot undergo alterations with the Search Optimization Service, unlike Materialized Views, which can indeed benefit from this service. Also, column definition that contains collate, column concatenation, and analytical operation are not suitable to be part of the optimization process as well. In some cases, we need to retrieve archive data or use the previous state of the table alongside with current state. It can be a costly query but we can not use a Search

Optimization Service with a Time Travel date because a Search Optimization Service can just work with current data.

Query Acceleration Service is one of the unique services of Snowflake to improve performance queries without changing virtual processing power. Query Acceleration Service aims to share processing operations over suitable available servers and it costs a relatively small amount compared to waiting for execution queries in normal customers. We can imagine we create a virtual processing resource or warehouse in Snowflake for data analysts. Normally, they should execute ad-hoc and low resource-intensive queries but they execute unpredictable complex queries instead. However, the allocated warehouse force is to wait for us to get the desired result because the size of the warehouse is small compared to large queues. We have two options to address this problem. The first option is to change our warehouse or virtual processing power manually but the data analyst can forget to change it after the execution of a complex query concluded successfully and continuing with a changed warehouse can cost some money. However, the second option offers to usage of Query Acceleration Service which we can extend the warehouse if needed with a shared workload over available servers.

Query Acceleration Service can be enabled to reduce the impact of outliers that offload portions to required sources. There are special types of queues which can benefit from this. If the query requires large scans and utilizes a selective filter to get desired results, we can use from acceleration service. The main advantage for us is reduced wait time over wall-clock time spent for scanning and filtering. Additionally, it opens the ground for execution in available nodes parallel. Therefore, Query Acceleration Service relies on the availability of virtual warehouse services. Before its enabling occurs, we can use certain system functions to detect virtual warehouses. The most important system function is ESTIMATE_QUERY_ACCELERATION for this purpose. In this way, we can get a virtual warehouse list and their options for each scalar. Scalar in Snowflake virtual warehouse leads to extra costs and can enhance performance in advance therefore it should be selected wisely to control costs

### 2.4.2 Fast response data views

In Snowflake, there are various types of views including standard views, secured views, and materialized views. These views can be leveraged for creating data marts, optimizing performance by pre-computing join operations, and providing secure access to query results. From a performance improvement standpoint, materialized views will be examined alongside other optimization services.

Snowflake provides Materialized Views to improve performance on particular data processing activities including range search, equality search, and sorting operations. When queries match particular conditions, they can be converted into table-based structures like as views. The prerequisites for materializing queries include ensuring that the intended return size is modest and that the query's underlying base database changes seldom. This stability guarantees that the materialized views are effective and reliable at optimizing query performance. After meeting requirements confirmed successfully, we can materialize the result of heavy operations such as semi-structured files and long-time needed aggregation operations and data processing through external tables and stages. Additionally, we do not need to run helper or maintenance tasks to up-to-date our Materialized Views because Snowflake takes this responsibly automatically and implicitly. When there is some change or new data insertion occurs, transparency Materialized Views is going to be updated automatically. There are other alternatives for Materialized Views because it has some disadvantages such as frequent data changes on data sources not allowed, relying on intensive data sources, and requiring additional storage opposite to caching. Regarding these pitfalls, we can benefit from Materialized Views utilizing them in data mesh infrastructure, improve the performance of same join operations, and provide a secure way of materializing queries.

Materialized views can be utilized to enhance the performance of satellite tables with parent keys by storing precomputed query results, which accelerates data retrieval significantly. It's important to note that materialized views are not compatible with window functions and are not suitable for point-in-time (PiT) analysis involving joins.

For point-in-time (PiT) analysis without joins and leveraging JoinFilter efficiently, Current Point-In-Time (CPIT) tables can be a preferable alternative over daily PiT data. CPIT tables optimize data access by directly accessing relevant properties from surrounding satellite tables. Conditional Multi Table Inserts can effectively manage Point-In-Time tables containing daily, weekly, and monthly data, ensuring the correctness and completeness of PiT datasets and enabling smooth data insertion based on predefined criteria.

Data Vault architecture is advantageous for shared business data models and multitenancy scenarios. It enables multitenancy by accommodating multiple tenants within a single, scalable data model. This approach ensures effective and secure data management while meeting diverse business requirements across various tenant companies..

## 3 Analysis and Findings

### 3.1 Experimental System Setup

Firstly, we need to complete the theoretical frameworks of an ideal data warehouse and data marts to align with the fundamental requirements of descriptive data applications. In the initial introduction chapter discussing Business Intelligence application requirements, along with the nodes and components of data applications, we can systematically categorize requirements based on these nodes per stage. Each specific requirement for effective modern descriptive data applications should correspond to stages, with each stage containing nodes. I observe that by configuring a data storage model for each stage, nodes can inherit this storage model. This configuration also allows for using these nodes in different architectures while maintaining the required storage model. Therefore, we must establish a common data application infrastructure for the data warehouse, where these nodes can be most commonly utilized, ensuring that the data application meets the requirements of modern Self-Service Business Intelligence (SSBI) applications.

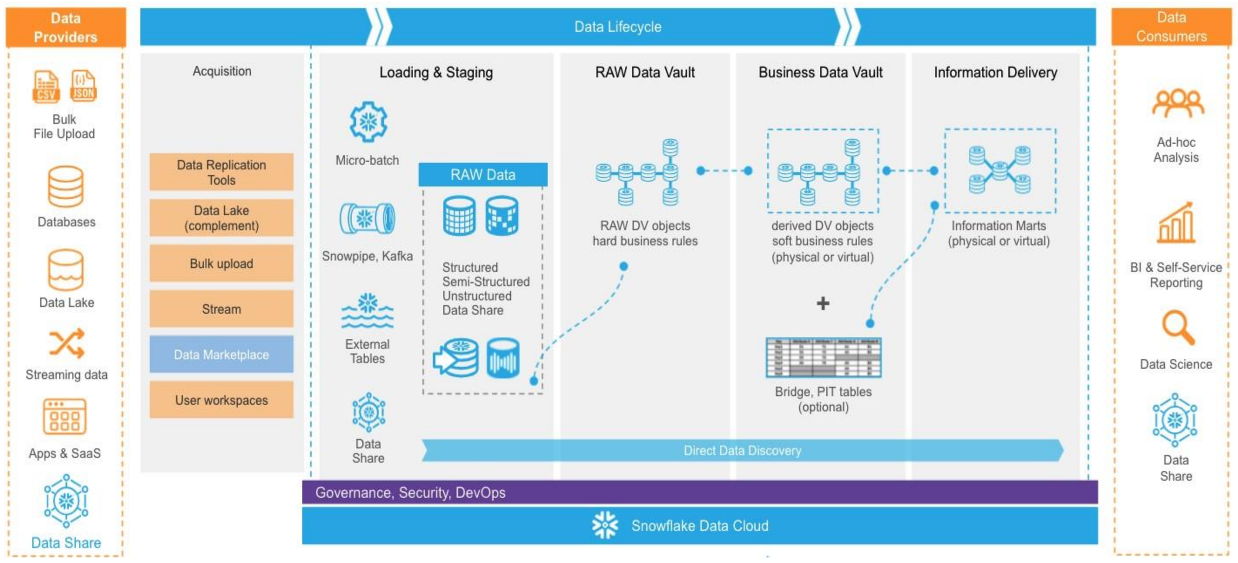


Figure 5. Near Real-Time Data Vault Architecture with Enhanced Batch and Real-Time Loading Capabilities [20]

I analyzed the dimensional Kimball and Data Vault 2.0 logical data models, which are significant for creating a Near Real-Time Data Infrastructure meeting the requirements of collaborative and unified data processing for batch and real-time loads. Data Vault 2.0 is utilized for managing data and generating hash keys based on descriptive data and business keys. The processed data with these hash keys is divided into hubs, links, and satellites to represent business objects, object relations, and descriptions of these business objects, respectively. Additionally, in the second chapter, we encounter different table types in Data Vault 2.0 that is described in Figure 5, including assistance tables designed to enhance query performance, meeting the requirements of ad-hoc queries and performance needs of Quarterly Business Intelligence applications.
In addition, the final stage of the data application involves information marts that contain easily queryable data organized in a dimensional design. For our initial data warehouse setup, we need nodes capable of handling both real-time data processing and batch data processing. Real-time nodes write data to memory initially, with data being flushed to disk blocks after a guaranteed minimum time period. We primarily differentiate data based on its real-time processing needs. Both read and write operations occur at the same level, and faults can arise due to the data handling load. The data is considered temporary, meeting the data integration requirements of Business Intelligence applications.

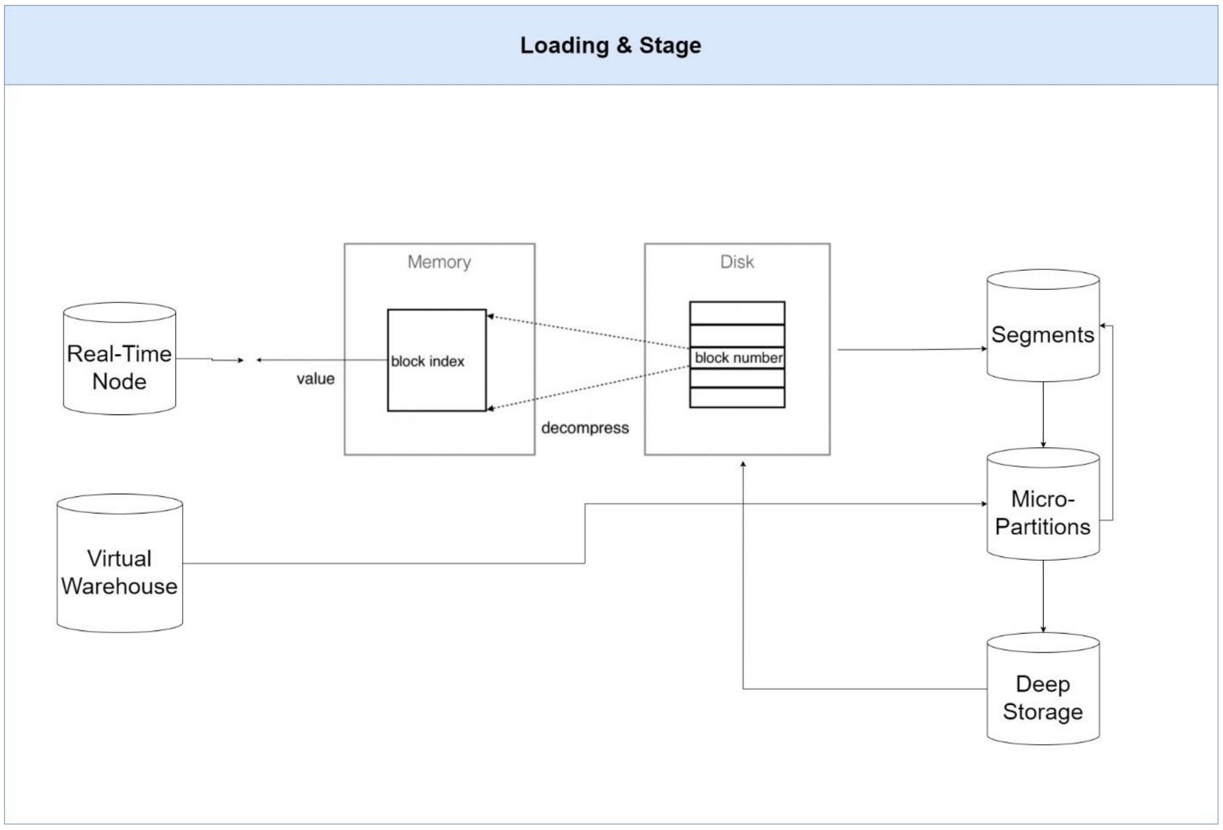


Figure 6. Data Handling and Disk Persistence Process in Loading and Staging

This stage of the data warehouse is highly technical, ensuring data integrity and providing metadata along with technical columns. Reading operations are minimal, with a predominant focus on write operations. Hash surrogate keys are generated from business keys and other descriptive payload data to meet the long-term requirements of the data warehouse where nodes has relation such as Figure 6. This approach allows us to fulfill the needs of descriptive data applications, which require adaptability to sudden business changes. The data stored in these tables is permanent and holds significant value for Data Lineage operations.

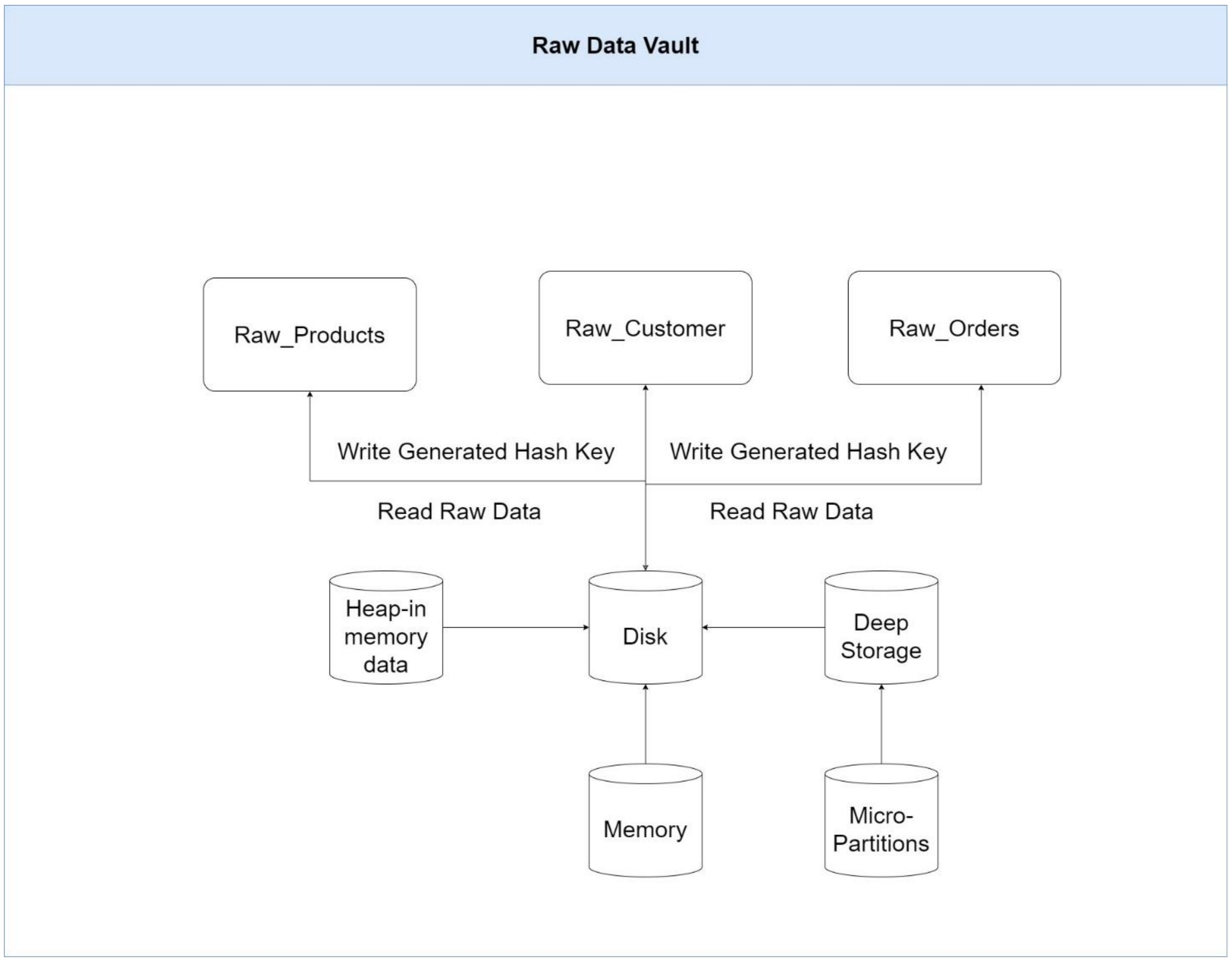


Figure 7. Reading Data and Writing Hash Surrogate Keys to Disk

The Business Data Vault is a critical stage of the data warehouse where read operations are in high demand. It must also be strictly fault-tolerant because it retains both historical and current data. The primary objective of this stage is to meet the performance requirements of Business Intelligence applications by facilitating efficient data querying, enhancing data shareability, and enabling consistent collaboration. All types of tables within this stage are designed for permanence, and read operations are given high priority to ensure optimal performance and accessibility which nodes order reflect it based on Figure 7.

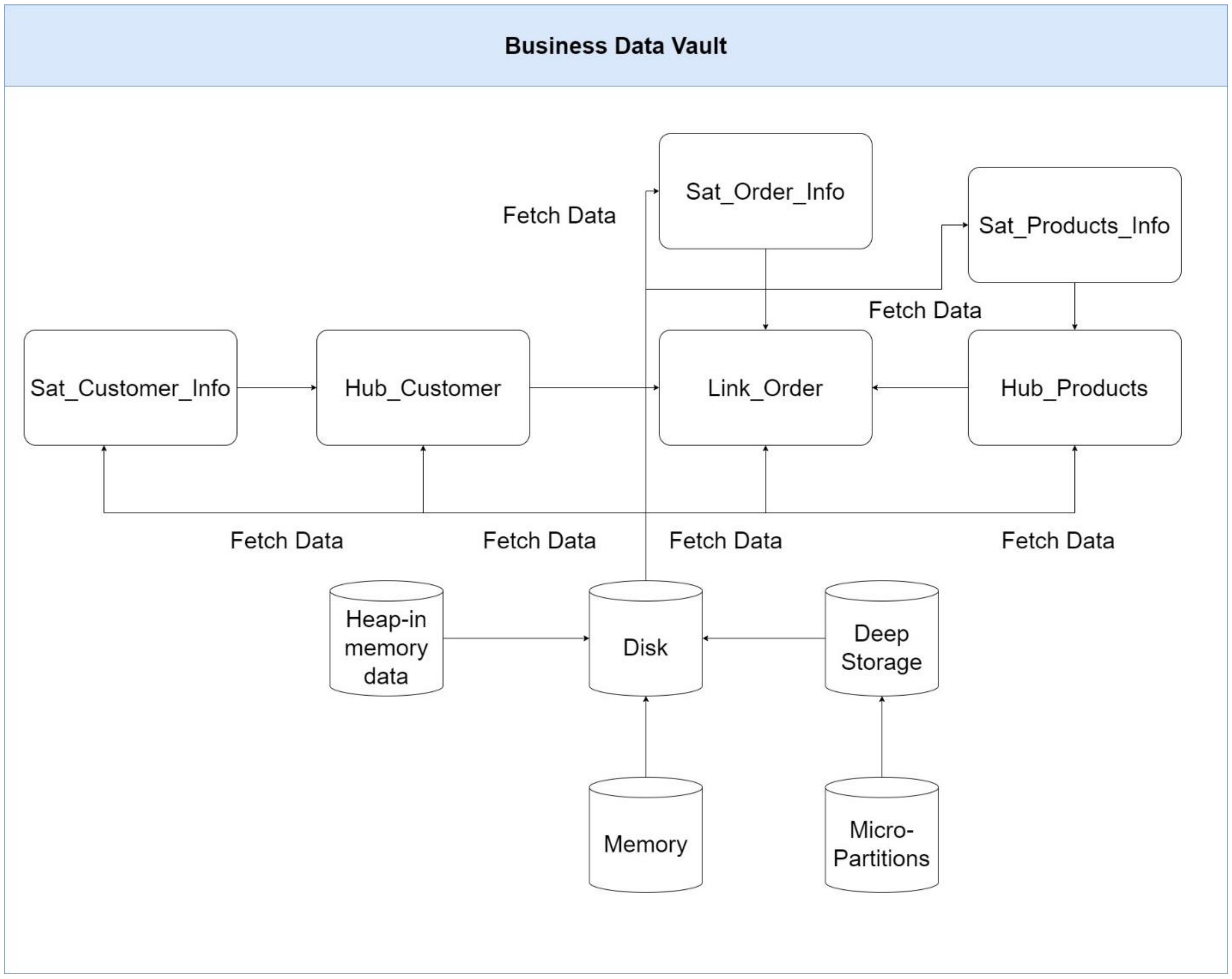


Figure 8. Data Reading Operations in Business Data Vault

To meet the performance and collaboration requirements of Business Intelligence applications, it's essential to incorporate an Information Delivery layer into the data warehouse. This layer is responsible for creating data marts tailored to different collaborative departments and business groups. Any errors or changes to the data within this layer do not impact the entire dataset since it serves as a representation layer that is described in Figure 8. The Information Delivery layer predominantly supports heavy read operations, as most operations involve querying data. The data in this layer is temporary and can be regenerated based on intermediate layers if needed, prioritizing performance over fault tolerance. Despite the importance of fault tolerance, the ability to recreate data mitigates potential issues. This layer serves as a query endpoint where most queries are executed, making it a crucial stage for collaboration, data sharing, and ad-hoc query execution within descriptive data applications.

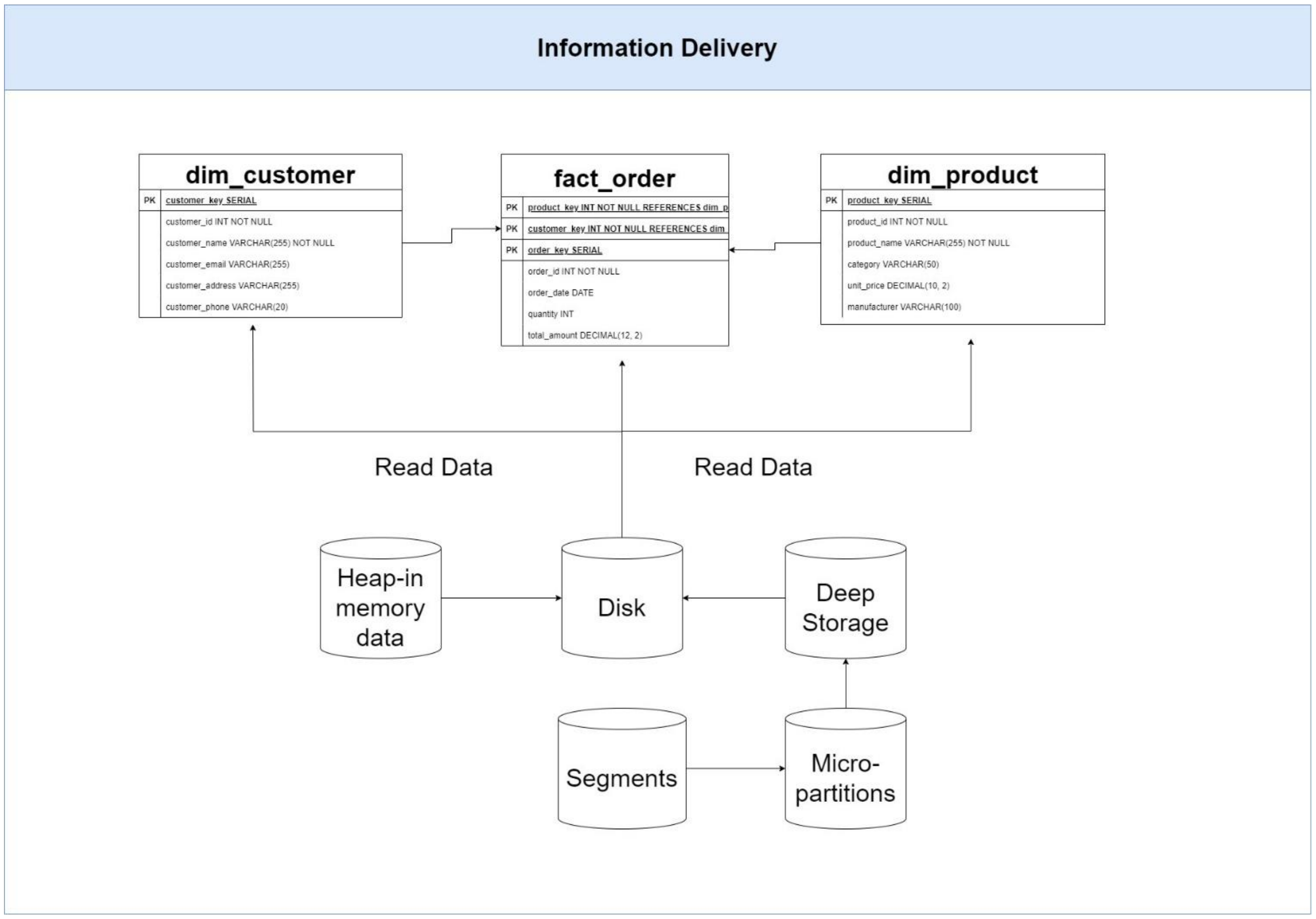

Information Delivery
dim_customer
PK
customer_key SERIAL
customer_id INT NOT NULL
customer_name VARCHAR(255) NOT NULL
customer_email VARCHAR(255)
customer_address VARCHAR(255)
customer_phone VARCHAR(20)
fact_order
PK
product_key INT NOT NULL REFERENCES dim_p
PK
customer_key INT NOT NULL REFERENCES dim
PK
order_key SERIAL
order_id INT NOT NULL
order_date DATE
quantity INT
total_amount DECIMAL(12, 2)
dim_product
PK
product_key SERIAL
product_id INT NOT NULL
product_name VARCHAR(255) NOT NULL
category VARCHAR(50)
unit_price DECIMAL(10, 2)
manufacturer VARCHAR(100)
Read Data
Read Data
Heap-in memory data
Disk
Deep Storage
Segments
Micro-partitions

Figure 9. Reading Operations from Disk to Dimensional Model in the Information Delivery Stage of a Data Warehouse

Master data and metadata are not typically stored within the data warehouse; instead, they are maintained in separate databases. The primary responsibility of information marts is to establish dynamic pipelines and provide additional information to the Data Lineage service. Additionally, the collaborative work environment demanded by Self-Service Business Intelligence applications relies on metadata and master data to reduce the time required per report. Most of the data in information marts is permanent, and read-based operations should be prioritized during the storage model selection process. All table types within these marts are designed to be permanent to ensure data integrity and accessibility which Figure 9 and Figure 10 supports statements.

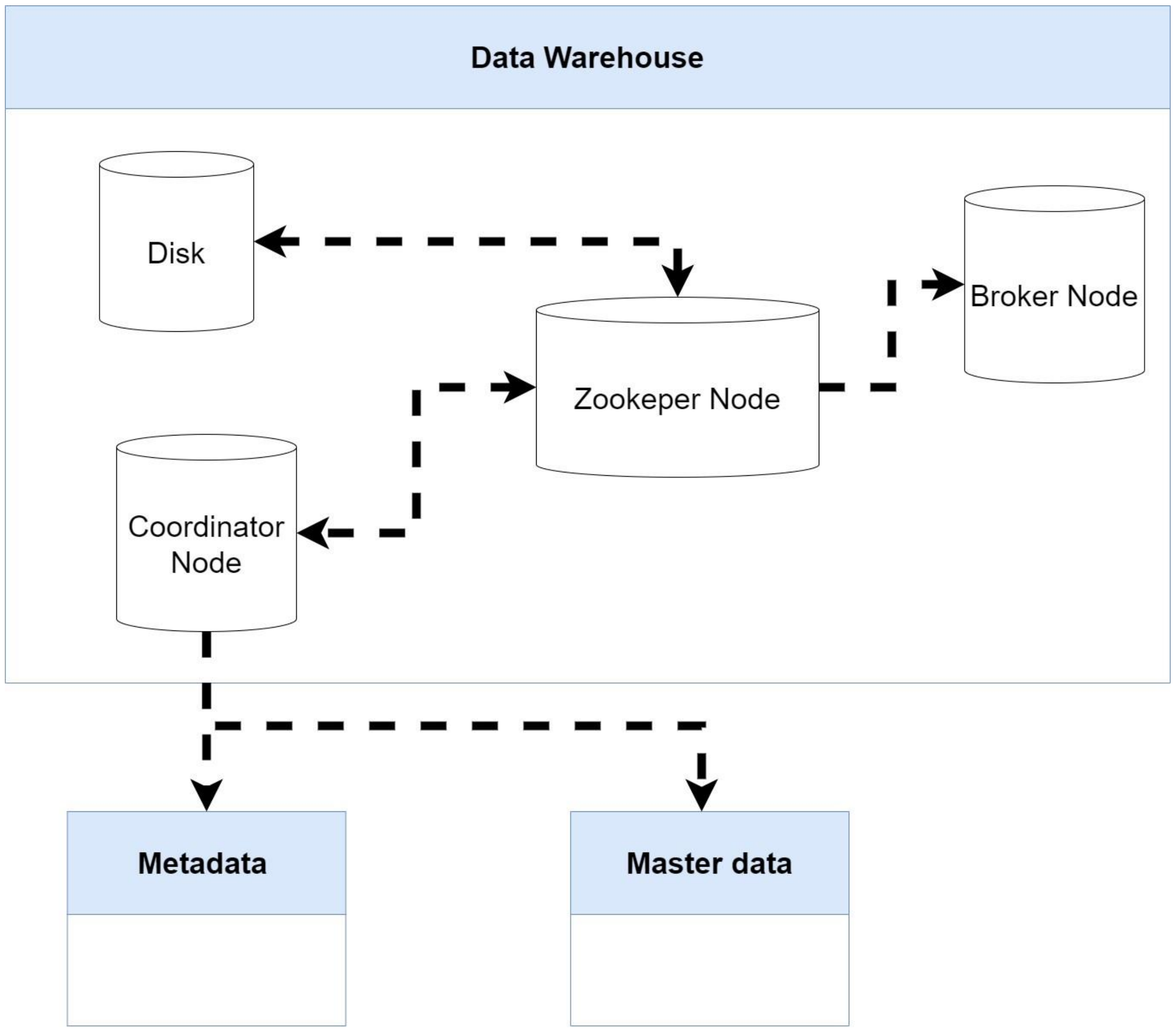


Figure 10. Master Data and Metadata Management in Data Warehouse Environments

## 3.2 Results

### 3.2.1 Ruleset for selection RAID levels in data application based on impact on Business Intelligence application

We have talked about the requirements of Business Intelligence which should provide coloration, unifying real-time and historical data. Also, query endpoints should be enhanced by metadata and master data management and this extra information about will be used in data lineage services. Regarding these requirements, we look through detailed components and methods which we can achieve by them. These components have been analyzed based on nodes of commercial data warehouse solutions such as Druid and Snowflake and distributed data OLTP database engine. In this way, we have a chance to form and categorize these nodes, file types, and metadata processors based on logical data modeling to achieve requirements. And it is time to analyze the impact of data storage models. As was discussed in the introduction, cloud storage-based geo-replicated storage models are highly used and it is the most continent way but its disadvantage is described under the CAP theorem which points to fault-tolerance and performance tradeoffs in mission-critical systems. For this purpose, we will analyze each stage of the data warehouse for high-performance Business Intelligence applications. This stage will contain nodes, data categories, and loads and a simple decision-making ruleset will be provided based on these inputs to increase the performance of the data storage model based on the priority of fault-tolerance and performance. Lastly, we will set extra rules for making these processes more energy-efficient.

The first stage of a data warehouse is for data loading and staging row data for a temporary time. Most of the time, its interacted stakeholder is object storage which data elimination process that occurs after data ingestion. Also, we can reload data from sources in case of some unexpected failure with idempotent metadata-based data pipelines. This stage contains Real-Time nodes and Deep Storage nodes to handle and unify real-time and batch data through transient tables. Data read and write operations occur frequently in high doses. Regarding these nodes, table type, and I/O activity, we should select RAID-0 which distributes data into multiple blocks of disks.

The second stage is the intermediate stage which has special importance for enterprise data warehouse architecture and our requirements taken from this kind of large-scale decision-making system. This stage is responsible for processing and historical data which contains many viable assets for the company. Data isolation occurs in this stage which hashed surrogate key generation happens. All table types should be permanent which is the source of metadata. Micro-partied and segments are the main types of data in this stage. I/O operations are not in high frequency compared to the Staging and Loading stages. Most important our system should be highly fault-tolerant because we keep every value data asset here. For these reasons, we can use RAID-1 data mirroring which is less costly but it has performance problems. To address these data problems we can use adding data parity methods in our disk storage. Intermediate Stages are divided into Raw Data Vault and Business Data Vault modes. Performance is a priority for the Business Data Vault Stage and RAID-5 can be selected for this stage. But performance is not a big case in Raw Data Vault as fault-tolerance therefore RAID-1 can be used for keeping tradeoff cost and performance alongside fault-tolerance.

The last stage of a data warehouse is the Information Delivery stage which is mainly used by predictive and descriptive data applications. Virtual warehouse nodes are the main part handling data under Broker nodes. Also, we do not need to strictly apply fault-tolerance because we can easily back up data from intermediate stages, especially from the Business Data Vault stage. For these reasons, we can use RAID-0.

The stages of the data warehouse concluded with succeed from the selection of RAID-based disk virtualization based on performance degradations but our data applications for efficient Business Applications with a data warehouse. Also, we should pay attention to Metadata and Master data databases. Most data ingestion and procession based on metadata and master data should be available for data pipelines and Business Intelligence applications. For this reason, we can select RAID-1 and RAID-5 based on a load of data.

### 3.2.2 Improve performance of data strips in real time based on current load changes in disks

In many cases through the ruleset for selecting RAID levels, data striping is selected based on regarding their high performance. Striping can be utilized in each stage of the data warehouse standalone or parting data into blocks of disks. Therefore, it is enabled to striping is selected in RAID-0 and RAID-5. In the initial scenario, we can consider less amount of disks but the number of disks can be increased dramatically to handle and present data in enterprise-level descriptive data applications. For this reason, the provided ruleset should be enhanced by advanced algorithms to increase the capacity of data application. Mirroring data into multiple disks under RAID-1 is a mass data operation in which we replicate data to make more fault tolerance which we just copy written data into at least two or multiple add numbered disks.

For this reason, striping data needs extra logic to write data into the most relevant disks. In modern, disk virtualization techniques, hot and cold disks are used to keep data in more logical parts. Hot disks are employed to store data that is acceded over data internal and external data queries through data applications. However, cold disks are used to store archive data which can be determined from the size of the data based on the selected template. To optimize data striping, we should accept we will have multiple disks and a hot-cold template should be used. Under these curcumins, a dual-engine-based high-performance storage system (DSH) algorithm can be used. This algorithm has great advances in operating CPU power more effectively. Also, we can see the same positive effect on disk energy usage when we write data without opening extra disks. The DSH algorithm determines the size of the strip which decreases the size of strips following this operation. After size adjustment, frequency must determined based on accepted data loads. Additionally, some metrics should be measured before and after implementation. The first metric is disk consumption time which all should be equal. Also, we do not get data anomalies in delete operations. To meet these operational requirements, the algorithm first looks at openly available disks. If the disk is open and meet contains enough space for data then it can be can employed. If there is no open disk, an algorithm looks at cold and hot disks respectively. Cold and hot disk selection took place for size strips based on the selected template.

We can see the DSH algorithm in the overall architecture in Figure 11. To optimize this process, the DSH communication framework transfers the task of generating verification data within the

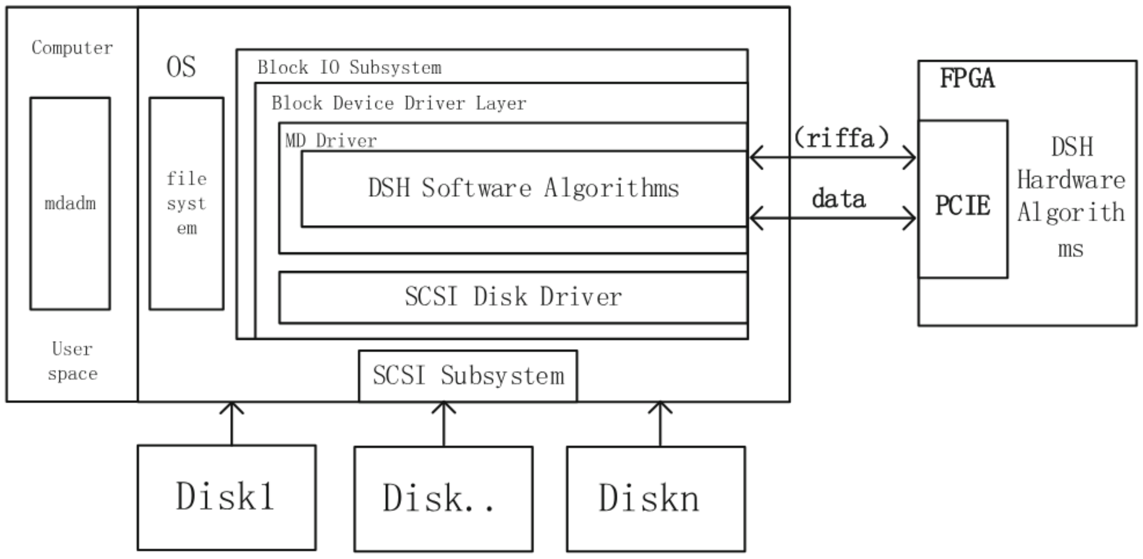


Figure 11 DSH Interaction Infrastructure [29]

DSH software algorithm to an FPGA. The data is communicated to the FPGA via PCIE and the Riffa architecture and then returned after processing. This approach effectively conserves significant CPU and GPU resources. The traditional software RAID setup starts with creating a software RAID in user space using the MDADM tool. This soft RAID creates a virtual hard disk (MD) in kernel space, which consolidates the disks forming the RAID. From the perspective of user space, there appears to be just one MD hard disk. Access to the individual disks occurs indirectly through read-and-write operations on the MD hard drive.

### 3.2.3 Final framework

The final design framework will integrate RAID selection and optimize data striping across hot and cold disks. Initially, we determine the appropriate RAID type based on the stage of the data application, aligning with the migration of requirements across logical and physical components. Next, the striping optimization process involves two key parts. First, we select the appropriate disks or allocate new ones.

```
function select_raid_configuration(stage):

    if stage == "Data Loading and Staging":

        return "RAID-0"

    else if stage == "Intermediate Stages (Raw Data Vault)":

        if "Raw Data Vault" in nodes:

            return "RAID-1"

    else if stage == "Intermediate Stages (Business Data Vault)":

        if "Business Data Vault" in nodes:

            return "RAID-5"

    else if stage == "Information Delivery Stage":

        if "Information Delivery" in nodes:

            return "RAID-0"

    else:

        return "RAID-0"
```

Second, if we have an existing disk then we select a cold or hot disk as destination. I provide the pseudocode of each operation and an overview of the process instead of adding comments.

This function select_raid_configuration determines the appropriate RAID configuration based on the stage of data processing. It checks the stage parameter to decide which RAID level to recommend. If the stage is "Data Loading and Staging", it suggests using RAID-0 for performance. For the "Intermediate Stages (Raw Data Vault)", it recommends RAID-1 if the "Raw Data Vault" nodes are present. Similarly, for the "Intermediate Stages (Business Data Vault)", it advises RAID-5 if the "Business Data Vault" nodes are detected. Finally, for the "Information Delivery Stage", it suggests RAID-0 if "Information Delivery" nodes are part of the setup. If none of these stages match or if no relevant nodes are detected, RAID-0 is recommended as a default.

This function calculates priority values for each disk based on the number of requested disks, disk free addresses, and total free addresses. It iteratively assigns priority values to disks starting from the highest priority down to accommodate the requested number of disks.

```
function calculate_first_priority(request_disks, disk_free_addresses,
all_free_addresses):

    N = total_number_of_disks

    average_priority = 100 / N

    priority_granularity = (disk_free_addresses / all_free_addresses) *
average_priority

    if request_disks <= N:

        current_priority = average_priority

        first_gradient_total_priority = 0

        for i from 1 to request_disks:

            ith_gradient_priority = current_priority - (i - 1) * priority_granularity

            first_gradient_total_priority += ith_gradient_priority

        Nth_gradient_priority = 100 - first_gradient_total_priority

        Nth_gradient_priority_number = request_disks

        for j from request_disks + 1 to N:

            jth_gradient_priority = ith_gradient_priority - priority_granularity

    return priority_values_for_each_disk
```

This function prioritizes disks for striping based on their status (open or cold) and their associated priority values. It first selects all currently open disks and then chooses additional cold disks based on their priority, ensuring that the maximum number of disks for striping is respected.

```
function calculate_second_priority(total_disks, open_disks, cold_disk_priorities):

    max_disks_striping = total_disks

    selected_disks = []

    for disk in open_disks:

        selected_disks.append(disk)

    sorted_cold_disks = sort_disks_by_priority(cold_disk_priorities,
descending=True)

    for disk in sorted_cold_disks:

        if len(selected_disks) < max_disks_striping:

            selected_disks.append(disk)

    return selected_disks
```

This particular opinion framework can be applied to green, mission-critical projects that involve extensive operations reliant on promises. Certain governmental and non-cloud applications can benefit from geo-replicated storage services. Moreover, major cloud vendors can leverage this option framework to improve their data warehouse solutions. By incorporating components from Druid and Snowflake, the final model is formalized, allowing this paper to address fault-tolerance issues and offer solutions.

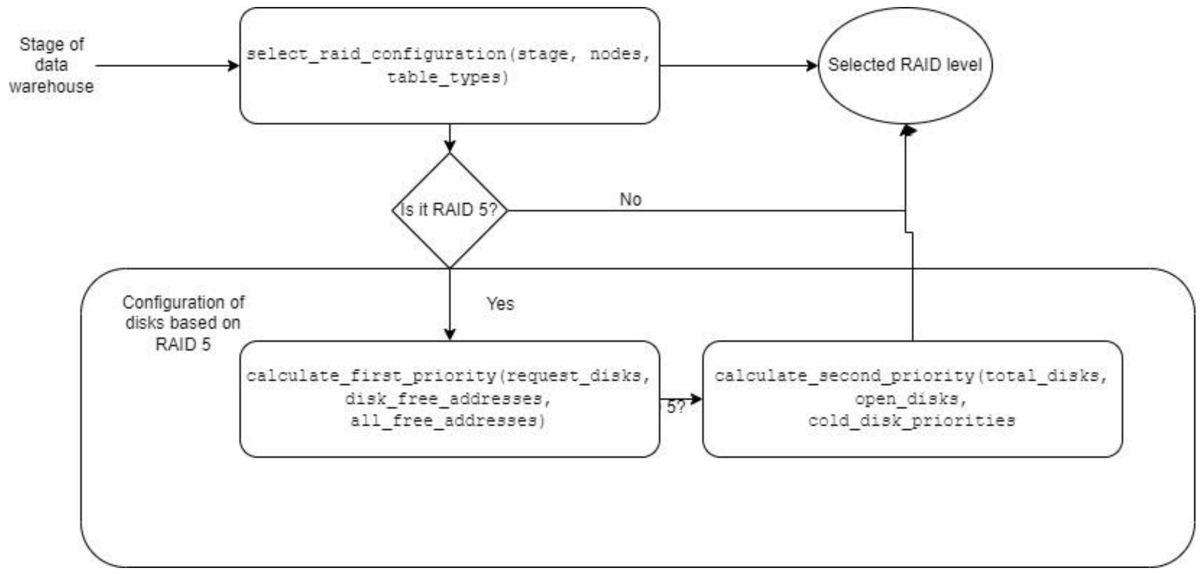

Stage of data warehouse
select_raid_configuration(stage, nodes, table_types)
Selected RAID level
Is it RAID 5?
No
Yes
Configuration of disks based on RAID 5
calculate_first_priority(request_disks, disk_free_addresses, all_free_addresses)
5?
calculate_second_priority(total_disks, open_disks, cold_disk_priorities

Figure 12. Architecture of selection RAID levels with input of stage of data warehouse.

In summary, the implemented decision-making process for selecting and configuring RAID levels in a data warehouse environment demonstrates a structured approach to optimizing storage solutions. The process begins by evaluating the stage of the data warehouse, along with nodes and table types, to determine the most appropriate RAID configuration. Specifically, the flowchart highlights a distinct pathway for RAID 5, a commonly used RAID level known for its balance of performance, redundancy, and storage efficiency as Figure 12 described.

For RAID 5 configurations, the methodology includes two key steps: calculating the first priority based on requested disks, disk free addresses, and all free addresses, followed by calculating the second priority involving total disks, open disks, and cold disk priorities. This iterative process ensures a comprehensive evaluation and optimal allocation of disk resources.

The feedback loop between the first and second priority calculations underscores the system's adaptability, allowing for refinements and adjustments based on evolving requirements and priorities. This iterative approach not only enhances the accuracy of disk configurations but also ensures that the data warehouse can maintain high performance and reliability.

Overall, the decision-making process outlined in this study provides a robust framework for RAID configuration, particularly emphasizing the intricate requirements of RAID 5. The approach ensures that data warehouses can efficiently manage storage resources while maintaining the necessary levels of data redundancy and access speed. This methodical and adaptive framework can be instrumental in advancing data storage strategies in complex, large-scale data warehousing environments.

# 4 Conclusions

In this thesis, I investigate the complex link between data storage types and the performance of Business Intelligence (BI) systems. My inquiry dug into the sophisticated needs of current business intelligence systems, which involve the seamless integration of real-time and historical data, as well as the capacity to offer self-service analytics.

I began by deconstructing the many phases of data application, methodically studying the components and procedures required to fulfill the expectations of business intelligence applications. Through this investigation, I discovered that storage models play an important role in influencing the performance and fault tolerance of these systems. My investigation took context to the world of RAID (Redundant Array of Independent Disks) technology, where I developed a set of criteria for selecting RAID levels based on the individual requirements of each step of data application. By matching RAID configurations to the needs of data loading, intermediate stages, and information transmission, I created the basis for improved speed and fault tolerance.

Furthermore, I investigated data striping, highlighting the need of optimizing data distribution across disks in real-time applications. I aimed to improve data application efficiency and capacity by proposing a complete framework for data striping optimization that included hot and cold disk methods.

My study resulted in the creation of a comprehensive framework that combines RAID selection and data striping optimization, providing a strategic approach to improving the performance of BI applications. I want to help enterprises to confidently and effectively traverse the difficulties of modern data management by employing innovative algorithms and approaches.

Looking ahead, our findings open the path for additional progress in the field of data storage optimization for BI applications. As technology advances, I believe that my framework will serve as a foundation for promoting innovation and efficiency in data-driven decision-making processes.

In conclusion, this thesis represents a significant contribution to the field of Business Intelligence, offering actionable insights and strategies for optimizing data storage models to meet the evolving demands of modern business environments.

## References

[1] Passlick, J Guhr, N Lebek, B. Breitner, M.H. “Encouraging the use of self-service business intelligence—An examination of employee-related influencing factors” *J. Decis. Syst*. 2020, 29, 1–26.

[2] Djerdjouri, “M. Data and Business Intelligence Systems for Competitive Advantage: Prospects, challenges, and real-world applications”. *Mercados y Negocios* 2020, 1, 5–18.

[3] Maune, A. “Competitive intelligence as a game changer for Africa’s competitiveness in the global economy” *J. Intell. Stud. Bus*, 2019, 9, 24–38

[4] Jalil.N, A.Prapinit, P.Melan, M.Mustaffa, A.B, “Adoption of Business Intelligence-technological, individual and supply chain efficiency”, *Proceedings of the International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI),* Taiyuan, China, 15–17 October 2021; pp. 1245–1267

[5] Dadkhah.M, Lagzian.M, Rahimnia.F, Kimiafar.K, “The potential of business intelligence tools for expert finding”, *J. Intell. Stud. Bus*, 2019, 9, 82–95

[6] Gebhardt.G.F, Farrelly.F.J, Conduit.J, “Market intelligence dissemination practices. J. Mark”, 2019, 83, 72–90

[7] Chang, E, “Advanced BI Technologies, Trust, Reputation, and Recommendation Systems”, *7th Business Intelligence Conference,* 2020, Sydney, Australia

[8] Miguel A. Teruel, Alejandro Mate´, Elena Navarro, Pascual Gonza´lez, Juan C. Trujillo, “The New Era of Business Intelligence Applications: Building from a Collaborative Point of View”, 2019

[9] Jayanthi Ranjan, “Business Justification with Business Intelligence”, 2009

[10] Distributed data processing for fourth-generation smart factories J Wickström – 2023

[11] Jens Passlicka, Nadine Guhra, Benedikt Lebekband Michael H. Breitne, “Encouraging The Use of Self-Service Business Intelligence an Examination of Employee Related Influencing Factors” , *JOURNAL OF DECISION SYSTEMS2020*, 2020, VOL. 29, NO. 1, 1–26

[12] Wang, J, Omar, A.H, Alotaibi.F.M, Daradkeh.Y.I, Althubiti.S.A., “Business intelligence ability to enhance organizational performance and performance evaluation capabilities by

improving data mining systems for competitive advantage", Information Processing and Management", 2022 Vol. 59 No. 6, 103075.

[13] ADAM HOSPODKA, "Overview and Analysis of Data Vault 2.0 - Flexible Data Warehousing Methodology", 2022

[14] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, "Dremel: Interactive Analysis of Web-Scale Datasets" Google, Inc.

[15] Soňa Karkošková, "Data Governance Model To Enhance Data Quality In Financial Institutions", 2022

[16] Patterson, David Gibson, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", *SIGMOD Conference*, 2024

[17] Michael Olschimke, Daniel Linstedt, "Building a Scalable Data Warehouse with Data Vault 2.0", 2015

[18] Ralph Kimball, Margy Ross, "The Data Warehouse Toolkit"

[19] Fangjin Yang, Eric Tschetter, Xavier Léauté, Nelson Ray, Gian Merlino, Deep Ganguli, "A Real-time Analytical Data Store", 2019

[20] Shahin Yusifli, "SCALABLE TRANSACTION MANAGEMENT AND GEO-REPLICATION IN CLOUD DATA STORAGE SERVICES" , 2021

[21] Ting-Yu Lin , Tseng-Yi Chen, "HSMR-RAID: Enabling a Low Overhead RAID-5 System over a Host-managed Shingled Magnetic Recording Disk Array", 2023

[22] Kanhaiya Sharma, Bindu G. Madhavi, Ankur Goyal, Deepak Parashar, Lalit Shrotriya, Anurag Gupta, "Real-Time Data Analysis, Significance, Architectures and Applications for Informed Decision-Making", 2023

[23] Zhibing Sha, Jiaojiao Wu, Jun Li, Balazs Gerofi, Zhigang Cai, "Proactive Stripe Reconstruction to Improve Cache Use Efficiency of SSD-Based RAID Systems", 2023

[24] Mailasan Jayakrishnan , Abdul Karim Mohamad, Mokhtar Mohd Yusof, "Assimilation of Business Intelligence (BI) and Big Data Analytics (BDA) Towards Establishing Organizational Strategic Performance Management Diagnostics Framework: A Case Study", 2021

[25] Fangjin Yang, Xavier Léauté, "The RADStack: Open Source Lambda Architecture for Interactive Analytics" , 2018

[26] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, Philipp Unterbrunner, “The Snowflake Elastic Data Warehouse”, 2015

[27] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, “Dremel: Interactive Analysis of Web-Scale Datasets”, 2018

[28] P A Rahman, G D'K Novikova Freyre Shavier, “Reliability model of disk arrays RAID-5 with data striping”, 2018

[29] Edward A. Lee, Soroush Bateni, Shaokai Lin, Marten Lohstroh, Christian Menard, “Quantifying and Generalizing the CAP Theorem”, 2021

[30] Vasthu Imaniar Ivanoti, Muhammad Royani, “DATA WAREHOUSE MODEL BASED ON KIMBALL METHODOLOGY TO SUPPORT DECISION MAKING IN ASSET MAINTENANCE”, 2021

[31] Paquita Putri Ramadhani, Setiawan Hadi, Rudi Rosadi, “Implementation of Data Warehouse in Making Business Intelligence Dashboard Development Using PostgreSQL Database and Kimball Lifecycle Method”, 2021

[32] Iuri D. Nogueira, Maram Romdhane, Jérôme Darmont, “Modeling Data Lake Metadata with a Data Vault”, 2018

[33] Yevgeni Kuznetcov, Maxim Fomin , Andrei Vinogradov, “Multidimensional Information Systems Metadata Repository Development with a Data Warehouse Structure Using "Data Vault" Methodology”, 2019

[34] A R, Anindita Sarkar Mondal, Madhupa Sanyal, Hrishav Bakul Barua, Samiran Chattopadhyay, Kartick Chandra Mondal, “Comparative Analysis of Object-Based Big Data Storage Systems on Architectures and Services”, 2023

[35] Alice Rey, “Bridging the Gap between Data Lakes and RDBMSs”, 2024

[36] Viktor Leis, Christian Dietrich, “Cloud-Native Database Systems and Unikernels: Reimagining OS Abstractions for Modern Hardware”, 2023

[37] Jinhong Li, Qiuping Wang, Shujie Han, Patrick P. C. Lee, “The Design and Implementation of a High-Performance Log-Structured RAID System for ZNS SSDs”, 2024

[38] Ramy Fakhoury, Anastasia Braginsky, Idit Keidar, Yoav Zuriel, “Nova: Safe Off-Heap Memory Allocation and Reclamation”, 2024

[39] Ronak Pansara, “Unraveling the Complexities of Data Governance with Strategies, Challenges, and Future Directions”, 2023

[40] Abdulaziz Aldoseri, Khalifa N. Al-Khalifa, Abdel Magid Hamouda, “Re-Thinking Data Strategy and Integration for Artificial Intelligence: Concepts, Opportunities, and Challenges”, 2023

[41] Marcus Eriksson, Bruce Ferwerda, “Towards a User Experience Framework for Business Intelligence”, 2021

[42] Dmytro Yaroshenko, Kent Graziano “Building a Snowflake Data Vault”, Data Vault Alliance

## Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Shahin Yusifli

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Impact of Data Storage Models on Business Intelligence Applications", supervised by Vladimir Viies
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

06.05.2024

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

## Appendix 2 – RAID selection algorithm

The function employs a series of nested if-else conditions to evaluate specific criteria and return corresponding values based on the results of these evaluations. Each condition is checked sequentially, and upon meeting a true condition, the function executes the return statement associated with that condition, thus concluding the function's execution.

```
function select_raid_configuration(stage):

    if stage == "Data Loading and Staging":

        return "RAID-0"

    else if stage == "Intermediate Stages (Raw Data Vault)":

        if "Raw Data Vault" in nodes:

            return "RAID-1"

    else if stage == "Intermediate Stages (Business Data Vault)":

        if "Business Data Vault" in nodes:

            return "RAID-5"

    else if stage == "Information Delivery Stage":

        if "Information Delivery" in nodes:

            return "RAID-0"

    else:

        return "RAID-0"
```

## Appendix 3 – First priority utilizing algorithm

The function evaluates an initial condition and, if true, initializes a loop to iterate over a collection, performing specific actions within the loop. Subsequently, it executes a second loop to iterate over another collection and, after completing both loops, the function returns the computed or accumulated result.

```
function calculate_first_priority(request_disks, disk_free_addresses,
all_free_addresses):

    N = total_number_of_disks

    average_priority = 100 / N

    priority_granularity = (disk_free_addresses / all_free_addresses) *
average_priority

    if request_disks <= N:

        current_priority = average_priority

        first_gradient_total_priority = 0

        for i from 1 to request_disks:

            ith_gradient_priority = current_priority - (i - 1) * priority_granularity

            first_gradient_total_priority += ith_gradient_priority

        Nth_gradient_priority = 100 - first_gradient_total_priority

        Nth_gradient_priority_number = request_disks

        for j from request_disks + 1 to N:

            jth_gradient_priority = ith_gradient_priority - priority_granularity

    return priority_values_for_each_disk
```

# Appendix 4 – Second priority utilizing algorithm

The function `calculate_second_priority` initializes a maximum number of disks for striping and an empty list to store selected disks. It first adds all open disks to the selected disks list, then sorts the remaining cold disks by priority in descending order and continues adding them to the selected disks list until the maximum number is reached, finally returning the list of selected disks.

```
function calculate_second_priority(total_disks, open_disks, cold_disk_priorities):

    max_disks_striping = total_disks

    selected_disks = []

    for disk in open_disks:

        selected_disks.append(disk)

    sorted_cold_disks = sort_disks_by_priority(cold_disk_priorities,
descending=True)

    for disk in sorted_cold_disks:

        if len(selected_disks) < max_disks_striping:

            selected_disks.append(disk)

    return selected_disks
```