An Automatic Approach for Mapping Relational Database Schema into NoSQL Database Schema

Fenil Patel (202370451)

This dissertation was submitted in part fulfilment of requirements for the degree of MSc Advanced Computer Science

DEPT. OF COMPUTER AND INFORMATION SCIENCES

August 2023

DECLARATION

This dissertation is submitted in part fulfilment of the requirements for the degree of MSc of the University of Strathclyde.

I declare that this dissertation embodies the results of my own work and that it has been composed by myself. Following normal academic conventions, I have made due acknowledgement to the work of others.

I declare that I have sought, and received, ethics approval via the Departmental Ethics Committee as appropriate to my research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to provide copies of the dissertation, at cost, to those who may in the future request a copy of the dissertation for private study or research.

I give permission to the University of Strathclyde, Department of Computer and Information Sciences, to place a copy of the dissertation in a publicly available archive.

(please tick) Yes [✓] No [ ]

I declare that the word count for this dissertation (excluding title page, declaration, abstract, acknowledgements, table of contents, list of illustrations, references and appendices is approx. 14676.

I confirm that I wish this to be assessed as a Type 1 2 (3) 4 5

Dissertation (please circle)

Signature: Fenil Rameshchandra Patel

Date: 13/08/2023

ABSTRACT

As organizations face unprecedented volumes of diverse and unstructured data, NoSQL Document-Store databases have emerged as a compelling alternative to traditional relational databases. These databases offer enhanced flexibility, scalability, and performance when handling semi-structured and complex data. To harness the full potential of these NoSQL solutions, a seamless and automated conversion process from Relational Database Management Systems (RDBMS) to NoSQL Document-Store databases becomes essential.

This dissertation presents an innovative and automated procedure, deployed on the Snowflake data platform, designed to convert RDBMS schemas into document-oriented NoSQL representations in JSON format, specifically tailored for NoSQL Document-Store databases. The procedure targets the widely recognized TPC-H benchmark, a standard performance evaluation metric for decision support systems.

The research commences with a comprehensive analysis of RDBMS and NoSQL Document-Store database data models, identifying key disparities and challenges in schema mapping. Leveraging this understanding, a novel approach is developed, with a primary focus on automating data transformation for a seamless migration process.

The automated procedure comprises the following essential components:

- Schema Analysis: The procedure thoroughly analyzes the TPC-H benchmark schema in the RDBMS, capturing crucial metadata such as table structures, relationships, and constraints.
- Data Transformation: Leveraging the capabilities of the Snowflake data platform, the procedure automatically converts the tabular data from the TPC-H benchmark into JSON documents, effectively representing the schema in a document-oriented NoSQL format.
- TPC-H Benchmark Mapping: The converted JSON data is further optimized to suit the specific requirements of the TPC-H benchmark, ensuring accurate and consistent performance evaluations.

The efficacy of the automated procedure is rigorously evaluated through extensive testing using diverse TPC-H benchmark scenarios and datasets. The evaluation encompasses critical performance metrics such as data integrity, efficiency, and scalability, enabling a comparative analysis against manual conversion methods.

The outcomes of this research offer significant contributions to the database management landscape, providing organizations with a reliable and efficient solution to transition from RDBMS to NoSQL Document-Store databases. The automated procedure's adaptability and scalability lay the groundwork for future research, enabling seamless schema conversions for other industry-standard benchmarks and various NoSQL Document-Store database environments.

In conclusion, the developed automated procedure empowers enterprises to embrace the full potential of NoSQL Document-Store databases, facilitating data agility and scalability, and streamlining the migration process from RDBMS to NoSQL Document-Store databases on the Snowflake data platform, specifically tailored for TPC-H benchmark scenarios.

Acknowledgements

I would like to express my sincere gratitude to Prof. Abdulsalam Kalaji, my supervisor, for his indispensable guidance and unwavering encouragement during the course of this dissertation. Additionally, I extend my thanks to my friends and family for their steadfast support, which has been a driving force behind my progress.

Table of Contents:

Table of Figures: 7

1. Introduction: 8

2. Background: 10

2.1 Literature Review: 11

2.2 Extended Literature Review: 14

2.2.1 Integration into the Research: 14

3. Methodology: 16

3.1 Illustrative Model from TPC-H Benchmark: Foundational Basis for the Study: 16

3.2 Choosing Snowflake Data Platform for ELT: Facilitating Schema Mapping Decision: 18

3.3 Creating Tables and Loading Data into Tables in Snowflake Environment: 20

3.4 Optimized Model from Comprehensive Literature Review: 21

3.5 Constraints of RDBMS Tables as Foundation for Schema Mapping and Snowflake's Role in Converting Structured Data to JSON Format: 23

3.6 Handling Cardinality for Composite Keys in Schema Mapping: 24

3.6.1 Relevance in the Methodology: 26

3.7 Practical Application: Enabling Automatic Schema Mapping from RDBMS to NoSQL Document Store Database Format: 26

3.7.1 Snowflake Procedure for Creating Collections for Independent Tables: 27

3.7.2 Snowflake Procedure for Creating Collections with Referencing for Tables with Relationships: 30

3.7.3 Snowflake Procedure for Creating Collections with Embedded Documents for Tables with Relationships having smaller-sized data: 34

3.7.4 Conclusion: 38

4. Analysis: 40

4.1 Data Description: 40

4.2 Time Complexity Analysis of Automated Schema Mapping Procedures in Snowflake: 41

4.2.1 Time complexity for SCHEMA_AUTOMATE_INDEPENDENT procedure: 41

4.2.2 Time complexity for SCHEMA_AUTOMATE_REFERENCE procedure: 42

4.2.3 Time complexity for SCHEMA_AUTOMATE_EMBEDDED procedure: 42

4.2.4 Assessing Time Complexity of Database Queries and Stored Procedures: Challenges and Factors: 43

4.3 Performance Evaluation of Automated Schema Mapping Procedures in Snowflake Data Platform for Varying TPC-H Scale Factors: 43

4.3.1 Experimental Setup: 44

4.3.2 Results and Findings: 44

4.4 Data Migration and Analysis: 46

4.4.1 Migration Procedure: 47
4.4.2 Data Extraction and Staging: 48
4.4.3 SnowSQL Extraction and Local Storage: 48
4.4.4 Import to MongoDB Compass: 49
4.4.5 Validation and Performance Assessment of Automated Schema Mapping: A Journey from Snowflake to MongoDB Compass: 50
4.5 Addressing Schema Mapping Challenges: A Comprehensive Study and Automated Approach for Relational to NoSQL Database Conversion: 50
5. Discussion: 54
5.1 Limitations: 54
5.2 Potential: 54
6. Conclusion and Future Work: 56
6.1 Conclusion: 56
6.2 Future Work: 56
7. References 58

Table of Figures:

Figure 1 - A section of TPC-H Benchmark database model. 18
Figure 2 - A section of Snowflake Database objects. 20
Figure 3 - A section of creation of snowflake database objects. 21
Figure 4 - A section of populating the data into tables. 21
Figure 5 - The optimised model 22
Figure 6 - Query result for creating customer collection. 24
Figure 7 - Query result for creating orders collection. 25
Figure 8 - Results obtained after executing all 3 procedures while showing all the collection created present in PUBLIC Schema. 41
Figure 9 - Performance Metrics for Schema Mapping for TPC-H Schema of Scale Factor 1 45
Figure 10 - Performance Metrics for Schema Mapping for TPC-H Schema of Scale Factor 10 45
Figure 11 - Performance Metrics for Schema Mapping for TPC-H Schema of Scale Factor 100 46
Figure 12 - System Diagram 47
Figure 13 - Imported data to local machine storage system. 49
Figure 14 - Document Store Data in MongoDB Compass 50

1. Introduction:

Relational databases have been the bedrock of enterprise database management for several decades. However, with the exponential growth of data and the need for greater scalability and flexibility, traditional relational databases are encountering limitations in handling complex and ever-expanding datasets. As businesses strive for enhanced data management, the constraints posed by relational databases on scalability, budget, and agility have prompted organizations to explore alternative solutions [1]. In response to the surge in big data and the demand for adaptable and scalable data storage options, NoSQL databases have gained popularity as a viable alternative [2]. NoSQL databases, characterized by their schema-less or flexible structure, are specifically designed to manage unstructured and semi-structured data.

In light of these challenges and opportunities, migrating from Relational to NoSQL databases has emerged as a potential solution for businesses seeking improved data handling capabilities. However, this migration process is intricate and challenging due to the fundamental differences in the schema design of relational and NoSQL databases. The initial step involves identifying the appropriate data structures and their relationships in the relational schema and subsequently mapping them to a suitable NoSQL database schema. This task is complicated because NoSQL databases organize data differently, relying on collections, documents, or key-value pairs, as opposed to tables and predefined relationships used in relational databases. Particularly with large and complex databases featuring intricate interconnections and relationships, this process becomes exceptionally demanding.

This dissertation delves into the research context of migrating relational databases to NoSQL projects, addressing the challenges faced by organizations in this endeavor. The primary focus is to provide an efficient and automated approach to tackle the schema mapping problem, ensuring a smooth and accurate transition from relational to NoSQL databases.

Research Questions:

- To examine the constraints and shortcomings of conventional manual techniques for converting relational database schemas into NoSQL equivalents.
- To conceive and implement an automated mapping strategy capable of effectively surmounting the identified limitations and challenges.
- To meticulously assess the precision, efficiency, and scalability of the proposed automated technique, thus evaluating its overall performance and efficacy.
- To undertake a comprehensive comparison between the outcomes of the proposed strategy and those derived from existing manual mapping methods.

Summary of Achievements:

Throughout the course of this research, the study delved into the multifaceted challenges posed by the migration of relational databases to NoSQL environments. An in-depth comprehension of the structural disparities between these two database types led to the formulation of an innovative automated mapping strategy. This strategy, designed to transcend the constraints of conventional manual approaches, streamlines the migration process. The ensuing methodology equips organizations with the means to attain unparalleled scalability, flexibility, and data management capabilities during their migration endeavors.

The achievements of this dissertation encompass:

- A comprehensive analysis spotlighting the core differentiators between relational and NoSQL database schemas, elucidating their influence on the migration process.
- The identification and comprehensive discussion of complexities encountered during the migration of extensive and interconnected databases, thereby enhancing understanding of migration dynamics.
- The creation of an automated mapping strategy adeptly tackling the limitations and challenges ingrained in manual mapping methods. This strategy seamlessly translates relational database schemas into precise and well-structured NoSQL equivalents.
- A meticulous analysis gauging the performance, accuracy, efficiency, and scalability of the proposed automated technique, yielding insights into its practical viability.
- A thorough comparison that contrasts the results of the proposed strategy with those obtained from traditional manual mapping methods, providing organizations with the insights needed to make well-informed migration choices.

The findings and contributions of this research provide valuable insights and practical solutions for businesses seeking to capitalize on the benefits of NoSQL databases while navigating the complexities of database migration. By automating the schema mapping process, this work aims to empower organizations to embrace cutting-edge data management technologies and unlock new possibilities for enhanced business agility and growth.

2. Background:

As the volume of data continues to soar, the popularity of big data processing has surged, prompting businesses and researchers to explore effective data management solutions. Among the widely adopted methods for real-world data management are relational databases, which, despite their prevalent use, face challenges in handling the ever-expanding data volumes efficiently. In response, researchers have proposed denormalization techniques to enhance relational database performance and maintain data consistency during updates. Additionally, NoSQL databases have emerged as a promising alternative, catering to diverse data types while offering high reliability and scalability [3].

Mapping a relational database schema to a NoSQL database schema is a complex task, necessitating careful consideration of various factors such as data models, data access patterns, and performance requirements. Two primary methods for converting relational database schemas to NoSQL database schemas are schema transformation and data migration. Data migration involves transferring data from the relational database to the NoSQL database without altering its structure, while schema transformation involves adapting the relational database schema to align with the NoSQL database model.

The schema transformation process involves modifying the existing relational database schema to fit the NoSQL database schema, a laborious undertaking due to the substantial differences in schema structures. However, this method offers potential benefits such as improved performance, scalability, and flexibility in the NoSQL environment. On the other hand, data migration is a comparatively easier approach, as it requires minimal changes to the database structure during the migration process. Nonetheless, data migration might not yield the same level of performance as schema transformation, as the relational database schema may not be optimally suited for the NoSQL database.

Some of the techniques like Mongify are available for mapping relational database schemas to NoSQL database schemas, among which automated tools stand out. These tools analyze the relational database schema and generate a corresponding NoSQL database schema, streamlining the mapping process and saving time and effort. However, it is imperative to ensure that the resulting NoSQL database schema aligns with the application's requirements and specifications.

To generate the necessary background knowledge for formulating sound research questions and designing an effective research process, extensive literature review is the primary source of information. The research literature provides valuable insights into the challenges associated with mapping relational database schemas to NoSQL database schemas, as well as various methods, best practices, and standard data sets employed in the domain. This stage of the research focuses on defining the research problem, seeking solutions to the complexities of schema mapping, and understanding the implications of different approaches on database performance, scalability, and flexibility.

By drawing from the wealth of research literature, this study aims to contribute to the field by proposing a comprehensive and efficient methodology for mapping relational database schemas to NoSQL database schemas, considering the specific requirements and constraints of real-world applications. Through a well-informed research process, this investigation endeavors to tackle the complexities of schema mapping and enable organizations to make informed decisions in their pursuit of optimal data management solutions.

2.1 Literature Review:

It is difficult to convert RDBMS schema to NoSQL database schema because it requires a thorough comprehension of both the relational and NoSQL data models.

In recent years, there has been a growing interest in automatic approaches for this task. For instance, S. Hamouda et al. (2017) discusses the challenges of migrating from a relational database to NoSQL and proposes an approach using an Entity-Relationship (ER) model to design a document-oriented data schema. This approach ensures compatibility for all schema elements and handles complex relationships effectively. The findings demonstrate successful schema migration and highlight the importance of normalization and denormalization for data optimization in the NoSQL environment. Overall, the paper offers valuable insights for a structured and compatible migration strategy [4].

While the proposed method ensures compatibility for all schema elements and effectively handles complex relationships during the migration process. It emphasizes the importance of normalization and denormalization techniques for data optimization in the NoSQL environment. Offering valuable insights, the paper might consider a fully developed prototype implementation, which limits the ability to evaluate the practicality of the migration method in real-world scenarios. Future work should involve building a prototype to validate and refine the proposed approach for relational database to NoSQL migration.

Furthermore, A.El Alami et al. (2016) discusses the challenges of migrating from a relational database to NoSQL and emphasizes the importance of considering keys for data integrity. The authors provide insights into key design and implementation, along with the architecture of a prototype system for migration. The findings highlight the significance of key considerations in ensuring data integrity in NoSQL databases and offer practical guidance for a successful migration process [2].

Although the research paper emphasizes the importance of taking keys into account during the migration process from a relational database management system (RDBMS) to a NoSQL schema. It underscores the significance of key design and implementation for ensuring data integrity and efficient querying in the NoSQL environment. Additionally, the paper offers insights into the architecture of a prototype system that facilitates the migration. However, it does not extensively evaluate or provide a comparative analysis, which restricts a comprehensive understanding of the approach's impact and effectiveness. Moreover, the generalizability of the proposed approach to different NoSQL database systems may not be well-established. Overall, the paper provides valuable considerations for key design and implementation during the migration, but further evaluation and comparisons would enhance its practical applicability.

Also, Stanescu et al. (2016) deals with the difficulties of automatically converting SQL schemas to NoSQL schemas, with a focus on migrating from MySQL to MongoDB. The paper introduces an algorithmic approach and a mapping methodology to automate the schema conversion process. By providing a systematic method for mapping tables, columns, relationships, and constraints from SQL to NoSQL, the authors aim to simplify and expedite the migration process. The findings of the research highlight the effectiveness of the proposed automatic mapping approach and discuss its practical applicability for different types of databases. The paper concludes with recommendations for using the approach in practice and suggests potential areas for future work and improvements in the algorithm [5].

Whereas the research paper presents an algorithmic approach and mapping methodology for automatically converting SQL schemas to NoSQL schemas. This systematic method offers a structured process for developers seeking to migrate their databases from SQL to NoSQL, potentially saving time and effort. However, the limitations of the approach become evident in its testing on small-sized databases, raising concerns about its scalability and applicability to larger, more complex systems. The lack of testing on complex databases leaves uncertainties about its performance and potential limitations in real-world scenarios. For a comprehensive evaluation, further research is needed to validate the proposed method on a wider range of database sizes and complexities, ensuring its robustness in practical applications.

Besides, A. A. Imam et al. (2017) addresses the challenges faced with traditional cardinality notations in data modeling and proposes new approaches tailored for NoSQL document-store databases. Existing notations lack standardization and expressive capabilities to represent complex relationships between entities, leading to ambiguity and imprecision. The paper introduces custom cardinality notations that offer a more intuitive and accurate way to capture relationships within large and interconnected datasets. Additionally, the authors explore denormalization techniques to simplify schema design by embedding related data within a single document, enhancing query performance. The findings suggest that these new approaches can lead to improved performance when querying data and simplified data management in NoSQL document-store databases, promoting data agility and faster development iterations [6].

Although the research paper proposes novel cardinality notations and styles for data modeling, aiming to overcome limitations in traditional representations. By providing standardized and expressive notations, the paper enhances clarity and communication in data modeling, making it relevant and applicable across various domains. However, some aspects of data modeling may require customization, and challenges may arise when dealing with one-to-many relationships in NoSQL document stores. The experimental results could vary based on data growth, necessitating ongoing evaluation and adjustments. While the paper addresses a significant challenge, its scope may be limited to cardinality notations without considering broader database design challenges or modeling techniques.

Moreover, A. A. Imam et al. (2018) faces the obstacles of inadequate modeling techniques and guidelines for big data storage settings in NoSQL document-store databases. With NoSQL databases offering schema-free flexibility, the responsibility of designing and enforcing schemas falls on developers, making comprehensive modeling guidelines crucial for data availability and consistency. To tackle this problem, the authors employ an exploratory investigation, expert consultations, and empirical insights, including a heuristic evaluation of five NoSQL databases. The findings lead to the proposal of new modeling guidelines specifically tailored for NoSQL document-store databases, addressing the challenges of big data and schema-free environments. These guidelines aim to be practical and intuitive, providing a valuable instrument for knowledge transfer from academia to industry. By enhancing data modeling practices, the authors contribute to ensuring high data availability, consistency, and performance in the context of big data and NoSQL document-store databases [7].

Nevertheless, This research paper presents a novel approach introducing new cardinality notations and styles for data modeling, addressing limitations in traditional representations. By proposing standardized and expressive notations, the paper aims to enhance clarity and communication in data modeling, making it relevant and applicable across various domains. The guidelines offer practical solutions to model complex relationships in large datasets, benefiting both academic research and industry applications. However, the evolving landscape of big data and NoSQL technologies may

require continuous updates to the guidelines to remain effective. Additionally, further evaluation is needed to validate the practicality of the guidelines in diverse real-world scenarios. The paper does not provide explicit details of proprietary recommendations and their limitations, which could limit direct comparisons with other approaches.

In the work of B. Namdeo et al. (2021), a model for migrating from Relational to NoSQL databases is proposed [8] [9]. The model involves a two-step process. Initially, a snapshot of the data in the relational database is captured, creating a complete copy of the data at a specific time using tools like Microsoft SQL Server Integration Services (SSIS) [9] [10]. Following this, a new NoSQL database is established, and a continuous data streaming mechanism is implemented to keep the NoSQL database synchronized with real-time updates from the relational database. This synchronization ensures immediate reflection of any changes made to the relational database in the NoSQL counterpart [11]. This method offers a seamless migration with minimal downtime, using the snapshot as the starting point and real-time streaming to maintain the currency of the new database. Additionally, this approach promotes data consistency and accuracy throughout the migration process [8].

Moreover, R. Čerešňák et al. (2021) encounters the barriers of efficiently transforming a schema from a relational database to a non-relational (NoSQL) database, focusing on MongoDB. The migration from relational to NoSQL databases has become crucial to handle modern data challenges, but it presents complexities due to the absence of transmission standards. To tackle this, the authors proposed a set of mapping rules and a methodology for schema transformation. They successfully applied these rules to a case study, creating 16 documents in MongoDB from 14 relational tables. The findings demonstrated the effectiveness of the proposed approach in handling different types of relationships and cardinalities. However, during the reverse mapping process, they faced challenges with undefined values and compatibility issues. The research highlights the need for future work to improve verification of undefined values and develop an enhanced mapper to address compatibility concerns [12].

Nonetheless, The research paper proposes mapping rules and a methodology for comprehensive schema transformation from a relational database to a NoSQL database, with a focus on MongoDB. The mapping rules cover various types of relationships, enabling effective migration of different data models. The practical application of the methodology, based on ETL principles, allows for efficient data transformation during the migration process. The successful case study validation confirms the applicability and correctness of the mapping rules. However, the paper lacks detailed insights into the extent and nature of compatibility issues faced during reverse mapping, where undefined values in non-relational databases pose challenges. The reliance on human input and limited scalability assessment are also areas that require further consideration for real-world implementations.

While reading through the extensive array of research papers in the literature, a profound insight into the design principles of schema mapping from traditional RDBMS to NoSQL Document Store databases became evident. The wealth of knowledge presented in these papers shed light on various key considerations, such as handling complex relationships, applying normalization and denormalization techniques, and proposing custom cardinality notations for enhanced data modeling. Amidst these insights, one particularly promising revelation emerged - the convenience and effectiveness of automating the schema mapping process using Extract, Transform, Load (ETL) tools. The application of ETL tools offers a structured and systematic approach, streamlining the translation of database schemas and ensuring efficient data transformation during migration. As a result, this profound understanding emphasizes the immense value of leveraging ETL tools to facilitate seamless and accurate transitions from RDBMS to NoSQL Document Store databases.

2.2 Extended Literature Review:

The extended literature review undertaken for this dissertation played a pivotal role in shaping the methodology for mapping relational database schemas to NoSQL database schemas. Two prominent research papers have significantly contributed to the understanding of the challenges and opportunities inherent in this process. Below, these papers are examined in detail to illustrate their impact on the research direction:

1. A. A. Imam et al. (2018) explore the intriguing realm of NoSQL data modeling guidelines. Their paper revolves around the challenges of designing effective NoSQL database models, offering a comprehensive guideline framework. Notably, they exemplify their proposed guidelines through a model inspired by a traditional RDBMS Entity-Relationship (ER) diagram [7].
   However, a critical observation surfaces when dealing with large datasets in embedded one-to-many and many-to-many relationships. According to their experiments, performance degradation becomes apparent in such scenarios [6]. This finding impelled us to scrutinize the implications of embedding in the context of schema mapping, especially when facing substantial data volumes.
2. Expanding our exploration, A. A. Imam et al. (2017) dive into the intricacies of NoSQL data modeling once more. This time, they bring attention to cardinality notations and relationship styles. By introducing innovative cardinality notations tailored to NoSQL schemas, they tackle the challenge of accurately expressing relationships. More interestingly, they introduce the concept of relationship modeling styles, categorizing diverse approaches to relationship representation [6] .
   A significant takeaway emerges from their experimentation—applying embedding in both one-to-many and many-to-many relationships can lead to performance degradation, particularly when dealing with larger datasets. This observation raised our awareness of the intricacies associated with referencing and embedding choices in schema mapping.

2.2.1 Integration into the Research:

These research papers have been instrumental in shaping the trajectory of this dissertation's research methodology. Their insights informed critical decisions in balancing embedding and referencing strategies. Recognizing the potential performance challenges associated with embedding, particularly in scenarios involving substantial datasets, this work leaned toward a referencing approach. This approach ensures the stability of system performance as data scales.

Furthermore, the gaps and challenges highlighted in these papers provided fertile ground for innovation. Specifically, the absence of detailed consideration for composite keys in one-to-many relationships prompted the development of a novel approach. The solution involved the use of arrays of objects to represent composite keys, addressing a critical aspect of schema mapping that was previously underexplored.

In summary, the insights from these papers have greatly enriched the creation of an automated methodology for mapping relational database schemas to NoSQL schemas. By melding these

scholarly perspectives with practical innovations, this work achieves a harmonious blend of theoretical foundations and empirical rigor.

3. Methodology:

This section presents a synthesized set of guidelines drawn from empirical research, structured into seven subsections. The first subsection introduces an illustrative model derived from the TPC-H Benchmark, forming the foundational framework for the study. The subsequent subsection outlines the rationale behind choosing the Snowflake Data Platform for the ELT process, facilitating schema mapping. Transitioning to the third subsection, the process of data creation and loading into the Snowflake environment is detailed. The fourth subsection highlights the optimized model derived from an extensive literature review. In the fifth subsection, the pivotal role of RDBMS table constraints as a basis for schema mapping is discussed, along with Snowflake's role in converting structured data into semi-structured JSON format. Additionally, the sixth subsection addresses the often-overlooked consideration of cardinality for composite keys, along with the approach taken to address this challenge. Lastly, the seventh subsection demonstrates the practical implementation of the proposed guidelines, illustrating the automatic schema mapping process from RDBMS to NoSQL JSON format. This section presents an organized and efficient approach to the migration process, aligning with the overarching research objectives.

3.1 Illustrative Model from TPC-H Benchmark: Foundational Basis for the Study:

In this study, the illustrative model extracted from the TPC-H Benchmark with a scale factor of 1 served as a representative sample of a real-world data scenario specifically tailored for the Online Analytical Processing (OLAP) database. The TPC-H Benchmark is widely recognized and adopted as a benchmark for OLAP databases and decision support systems. It is designed to evaluate the performance and scalability of databases in complex analytical and reporting tasks commonly encountered in OLAP environments [13].

The choice of the TPC-H Benchmark with a scale factor of 1 ensured that our study focused on the challenges and requirements typically faced in OLAP databases, where large volumes of data are analyzed for complex queries and aggregations. This choice allowed for the simulation and evaluation of the performance of proposed schema mapping guidelines and the automatic mapping approach within a context relevant to OLAP applications.

The OLAP-specific nature of the TPC-H Benchmark allowed us to target the specific needs of data modeling and schema transformation for OLAP databases, which often involve multidimensional data and complex analytical queries. Consequently, the insights and recommendations derived from this study hold particular significance for organizations that rely on OLAP databases for their analytical and business intelligence needs [13].

Based on leveraging the TPC-H Benchmark as an illustrative model with a scale factor of 1 specifically benchmarked for OLAP databases, this study conducted a targeted investigation into the design principles of schema mapping from traditional RDBMS to NoSQL document store databases within the framework of OLAP data scenarios.

The model shown in Fig. 1 follows the Entity Relationship Diagram (ERD) notations and symbols proposed by and which are the most popular relational database modeling technique in both the academia and industry.

The given model in Fig. 1 describes a set of predefined entities and their relationships, representing a data model for a fictional business scenario. The main entities in the TPC-H Benchmark include the following:

- Customer: Represents individual customers or organizations, each having a unique identifier, demographic information, and a relationship with Orders.
- Orders: Represents purchase orders made by customers, each associated with a unique identifier, order date, and a foreign key referencing the Customer entity.
- Lineitem: Represents individual line items within an order, each containing information such as the quantity, extended price, and discount applied. Lineitem is linked to the Orders entity through a foreign key.
- Part: Represents individual parts available for sale, each having a unique identifier, manufacturer information, and a relationship with PartSupp.
- Supplier: Represents suppliers providing parts to the business, each having a unique identifier, address details, and a relationship with PartSupp.
- Nation: Represents nations or countries with attributes such as name and region. This entity is related to Supplier and Customer entities to represent their respective geographical locations.
- Region: Represents regions or continents with attributes like name. It is linked to the Nation entity to group countries into specific regions.
- PartSupp: Represents the relationship between Part and Supplier entities, indicating the availability and pricing of parts supplied by specific suppliers. This entity contains attributes such as supply cost and availability status.

The relationships among these entities are vital for generating complex analytical queries. For instance, the Customer entity is associated with the Orders entity through a one-to-many relationship, as each customer can place multiple orders. Similarly, the Orders entity is linked to the Lineitem entity through a one-to-many relationship, as each order can contain multiple line items. Also, the Nation entity is linked to the Customer and Supplier entity through a one-to-many relationship, as each Nation can contain multiple customers and suppliers. The Part entity is related to the PartSupp entity in a one-to-many relationship, as each part can be supplied by multiple suppliers, and each supplier can provide multiple parts. Moreover, the PartSupp entity acts as a bridge table that connects the Part and Supplier entities in a many-to-many relationship, indicating the various parts supplied by specific suppliers and their corresponding availability status.

While the number/formula below each table name in given Fig 1. represents the cardinality (number of rows) of the table.

Based on Fig. 1, This illustrative model serves as the foundation for mapping the RDBMS schema into a NoSQL Document Store Database, a process that will be detailed later while describing the automated schema mapping.

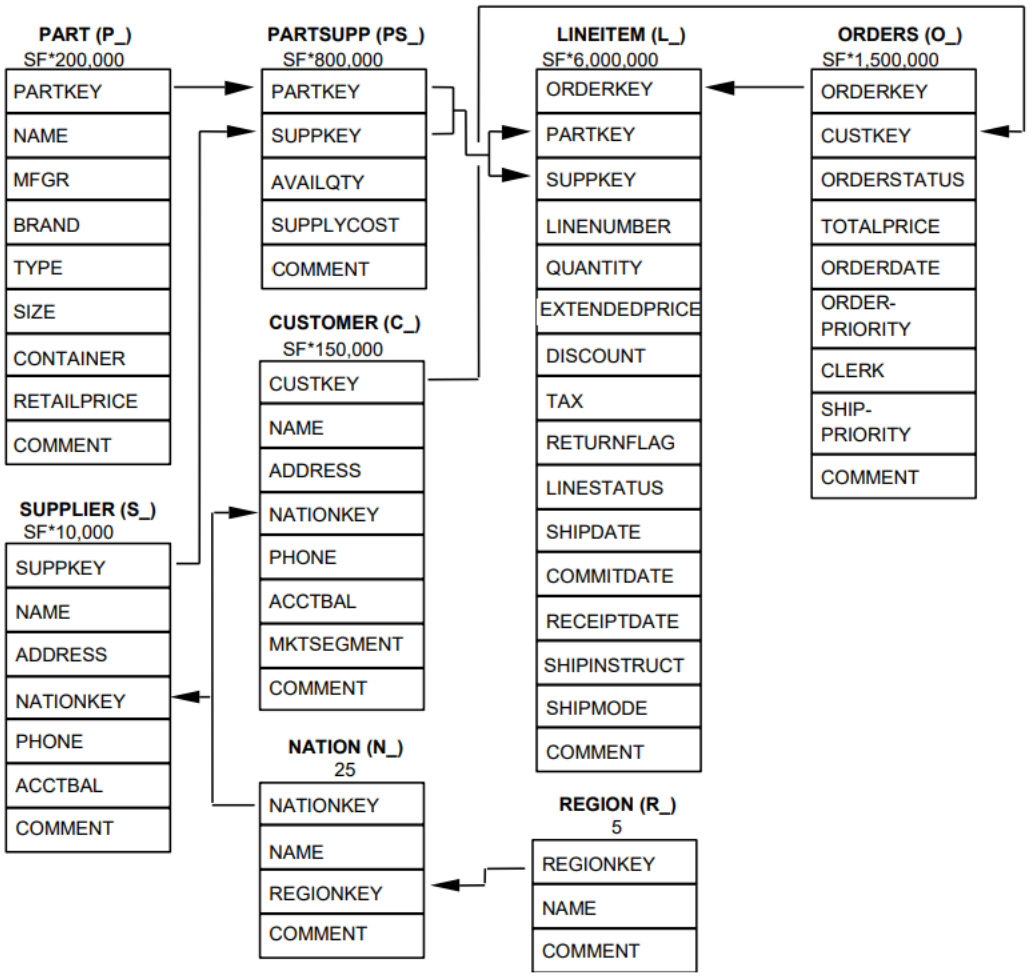


Figure 1 - A section of TPC-H Benchmark database model.

3.2 Choosing Snowflake Data Platform for ELT: Facilitating Schema Mapping Decision:

The selection of the Snowflake Data Platform for automating the schema mapping of a relational database into NoSQL JSON format was based on several crucial factors. Snowflake, a cloud-based data warehousing platform, is renowned for its scalability, performance, and user-friendly interface. These attributes make it an ideal choice to handle the substantial data volume involved in schema mapping and manage the intricate relationships inherent in relational databases.

A key consideration was Snowflake's adeptness in handling semi-structured and unstructured data, making it a perfect fit for NoSQL JSON format. Its support for flexible data models and native JSON capabilities proved advantageous in transforming structured relational data into JSON documents during the schema mapping process.

Furthermore, Snowflake's data sharing capabilities played a significant role in automating the schema mapping. Its seamless data sharing across various databases and systems facilitated smooth data transfer from the relational database to the NoSQL environment, streamlining the migration process.

Within our benchmark, data extraction will be conducted from the Snowflake Sample Data database. This database encompasses schemas for TPCH_SF1, TPCH_SF100, and TPCH_SF1000, as depicted in Fig. 2. The Snowflake Sample Data database serves as a valuable data source for our schema mapping endeavor. It is noteworthy, however, that this database is designed for data viewing purposes and does not grant access to table definitions.

To proceed with our schema mapping, we will leverage the data available in the Snowflake Sample Data database and create corresponding tables in our new database schema. Ensuring precision and alignment with the original TPC-H Benchmark, meticulous scrutiny of the table creation particulars as outlined in the TPC-H Benchmark documentation [13] will be undertaken. This documentation offers comprehensive information about the schema structure, table definitions, and relationships required to replicate the TPC-H Benchmark in our new environment.

With reference to the TPC-H Benchmark documentation, the construction of requisite tables in our new database and schema will be executed, meticulously adhering to benchmark specifications. This meticulous approach ensures that our schema mapping accurately represents the original data model, enabling a reliable comparison of the performance and efficiency of the database systems in handling the TPC-H workload.

Snowflake's ELT capabilities also contributed to the decision, as the ELT approach allowed data extraction, loading, and transformation within the platform itself. This simplified the transformation process, supported by Snowflake's native functions and SQL extensions for efficient schema mapping.

Snowflake's performance optimization features, enabled by its storage and compute separation, ensured quick and efficient execution of the schema mapping process, even with large datasets.

The platform's user-friendly interface and SQL-based query language empowered the development team to design and execute complex schema mapping queries without the need for specialized expertise.

To validate Snowflake's performance and capabilities, the schema mapping process was tested using a scale factor of 1 for the illustrative model from the TPC-H Benchmark.

Based on above factors, Snowflake emerged as the preferred choice for automating schema mapping due to its scalability, performance, native JSON support, data sharing capabilities, and developer-friendly features.

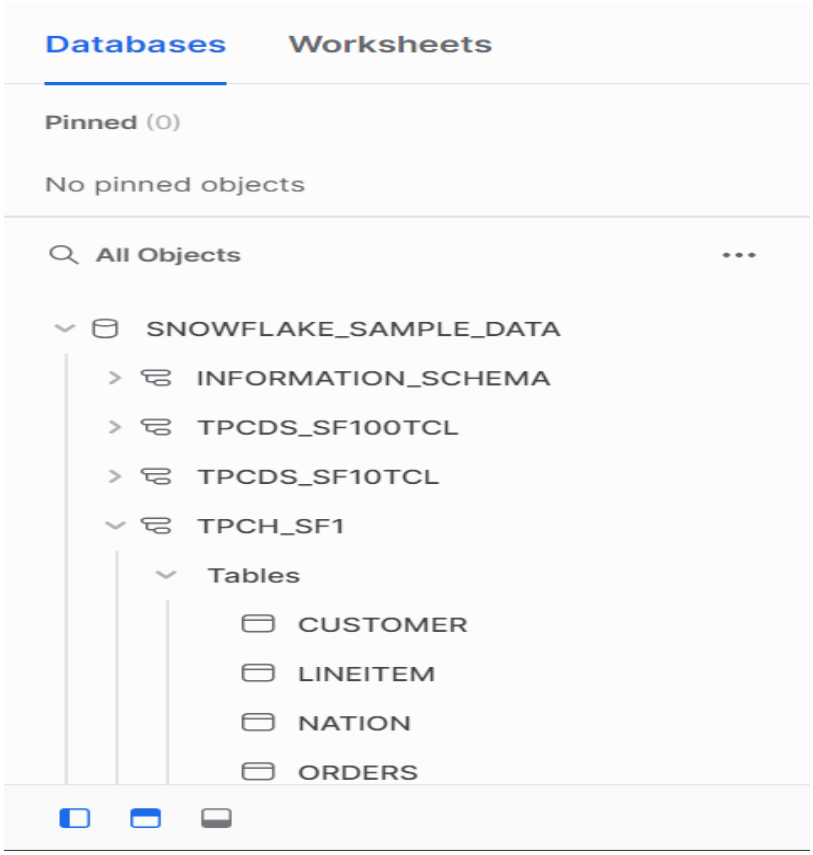


Figure 2 - A section of Snowflake Database objects.

3.3 Creating Tables and Loading Data into Tables in Snowflake Environment:

In Section 3, the process of crafting and loading data into the Snowflake environment is outlined. To ensure precision and adherence to the TPC-H Benchmark requirements, reference is made to the TPC-H Benchmark documentation [13], to construct the required tables with their respective constraints. The inclusion of primary key and foreign key constraints is essential for our automation of schema mapping, as this will be further elaborated in the following section.

Leveraging Snowflake's Data Sharing capability, data extraction from the Snowflake Sample Data database is streamlined. The focal point is the TPCH_SF1 schema containing pertinent benchmark data. This extracted data is effectively populated within the newly established tables in the Snowflake environment.

By meticulously creating and loading the data, adhering to the specified constraints, and utilizing Snowflake's data sharing capabilities, a robust foundation for schema mapping automation is established. The combination of accurate data representation and seamless data extraction ensures that our benchmark is based on real-world scenarios, enabling a comprehensive evaluation of the performance and efficiency of the database systems in handling the TPC-H Benchmark workload.

In addition to the previous information, below Figure 3, a new database named DATABASE_TPCH and a new schema named SCHEMA_TPCH within the Snowflake environment is established. Within the SCHEMA_TPCH, the imperative tables are crafted and subsequently populated with data sourced

from the SNOWFLAKE_SAMPLE_DATA database. Specifically, the TPCH_SF1 schema from this source (depicted in Figure 4) is employed.

Throughout the process of automating the mapping of relational database schema to NoSQL JSON format for the NoSQL document store database, all the collections and data representations will be stored in the PUBLIC schema of the DATABASE_TPCH database. This arrangement ensures a well-organized and centralized approach to schema mapping, making it convenient to manage and access the collections during the migration process. By structuring the data within the PUBLIC schema, we can effectively streamline the schema mapping automation, further enhancing the efficiency and clarity of the data migration.

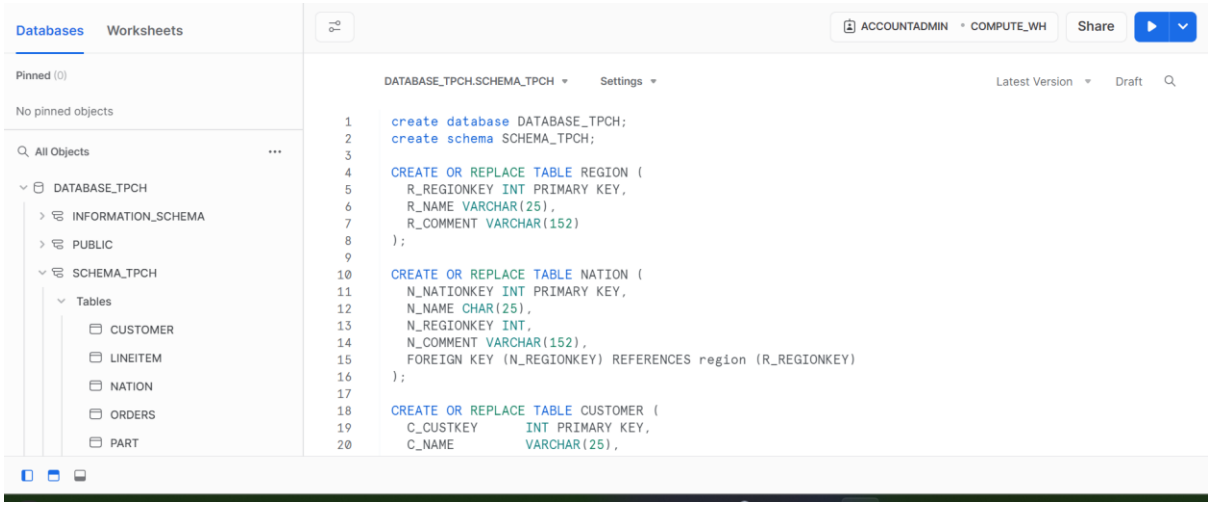


Figure 3 - A section of creation of snowflake database objects.

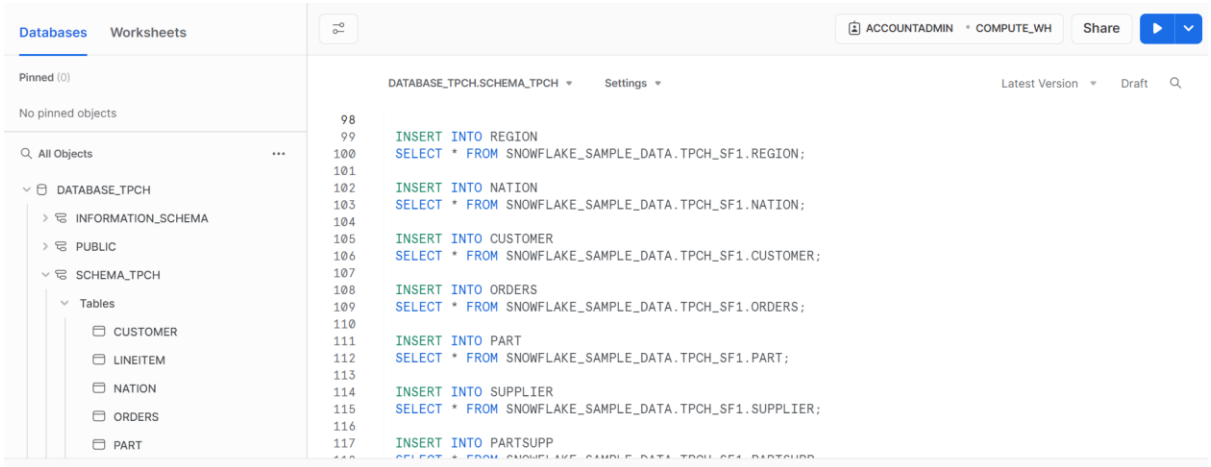


Figure 4 - A section of populating the data into tables.

3.4 Optimized Model from Comprehensive Literature Review:

In Section 4, an illustration of the refined model depicted in Figure 5 is presented. This model is an outcome of an exhaustive review of pertinent literature and the assimilation of guidelines from

multiple research papers. In particular, the principles put forth in the works of [4], [6] and [7] are integrated.

By incorporating insights from these papers, our optimized model reflects best practices and innovative approaches to address challenges related to schema mapping from relational databases to NoSQL JSON format. This model serves as a robust and efficient foundation for our automated schema mapping process, providing a clear and structured method for transforming the data into a NoSQL environment while ensuring compatibility and data integrity.

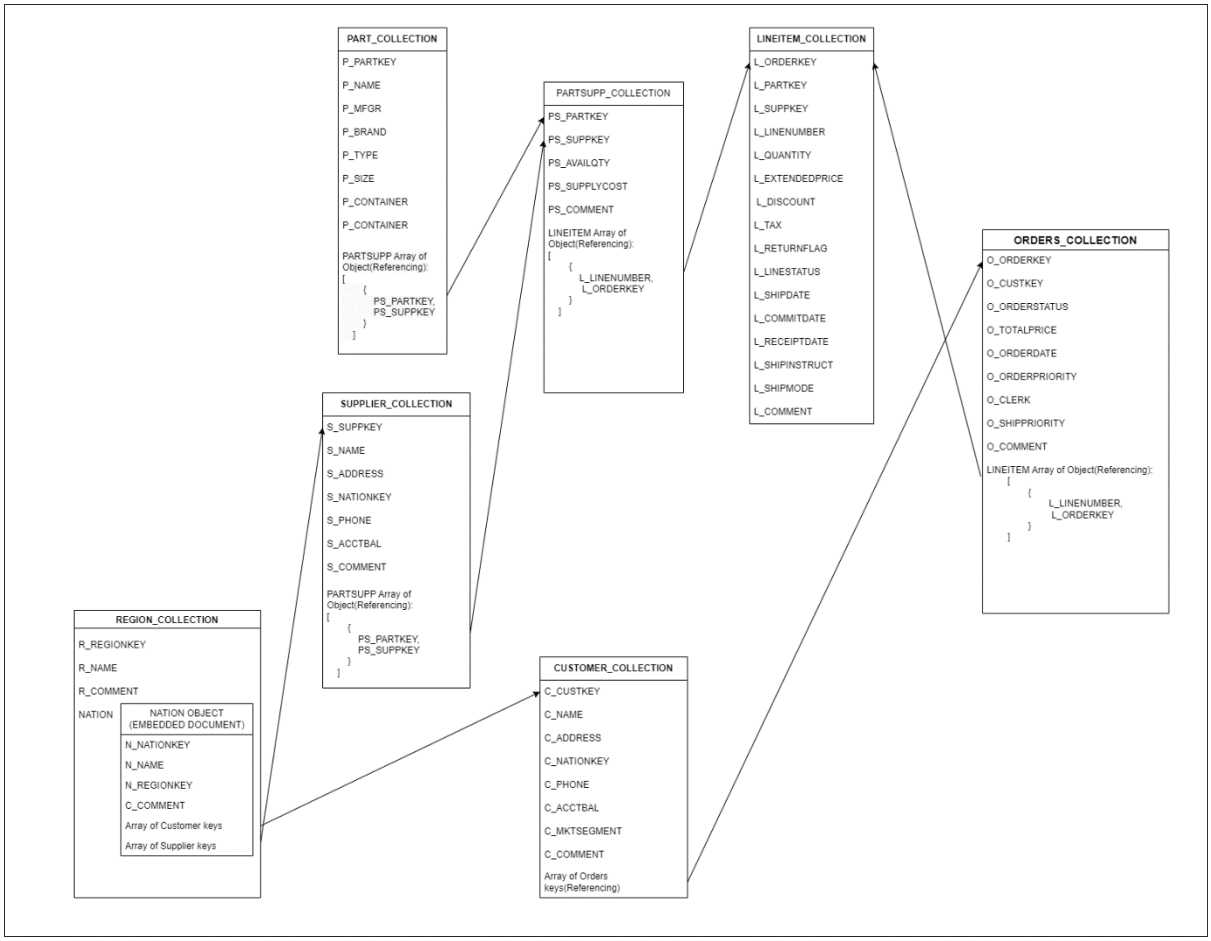


Figure 5 - The optimised model

In contrast to the model presented in Figure 5, where most of the collections for the NoSQL document store database are separate collections, the research papers by S. Hamouda et al. (2017), A. A. Imam et al. (2017), A. A. Imam et al. (2018); advocate considering one-to-many and many-to-many relationships from RDBMS tables as embedded documents. However, in my approach, I opted for referencing instead of embedding after carefully reviewing these papers. My decision was influenced by the experimental section of the paper by A. A. Imam et al. (2017), which demonstrated that the performance degrades when records in tables contain 5000 or more records. Additionally, MongoDB's white paper, highlights the use of referencing between documents in different collections when the combined document size would exceed MongoDB's 16MB document limit,

especially for modeling many-to-one relationships [1]. Referencing can also address various challenges, making it a suitable choice for modeling many-to-many relationships [1].

Therefore, while the mentioned papers propose embedding for certain relationships, I considered referencing to accommodate performance and size limitations, ensuring that our schema mapping process effectively handles complex relationships and optimizes performance in the NoSQL environment.

3.5 Constraints of RDBMS Tables as Foundation for Schema Mapping and Snowflake's Role in Converting Structured Data to JSON Format:

In Section 5, an exploration is undertaken regarding how the constraints associated with RDBMS tables are fundamental to our automated schema mapping approach. Throughout this automation process, substantial attention is given to the metadata of the tables, with a specific focus on the primary key and foreign key information. This metadata plays a pivotal role in determining whether referencing or embedding is the appropriate strategy for documents in other collections. Extracting this critical metadata from Snowflake involves executing the ensuing queries:

1. SELECT TABLE_NAME
   FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
   WHERE CONSTRAINT_TYPE = 'PRIMARY KEY'
   AND TABLE_SCHEMA = 'SCHEMA_TPCH';
2. SHOW PRIMARY KEYS IN SCHEMA DATABASE_TPCH.SCHEMA_TPCH;
3. SHOW IMPORTED KEYS IN SCHEMA DATABASE_TPCH.SCHEMA_TPCH;

The results obtained from these queries provide us with valuable insights into the primary keys and foreign keys of the tables. Additionally, if a primary key or foreign key of a table contains one or more columns, we can also determine its order based on the key sequence column. Iterating through this metadata information has allowed us to establish a robust foundation for our schema mapping automation, ensuring efficient conversion of relational database schema into NoSQL database schema.

Leveraging Snowflake's data platform capabilities for handling semi-structured data, the transformation of structured tabular data into JSON format – compatible with MongoDB's NoSQL database – becomes achievable.

Let us illustrate the usefulness of primary and foreign key constraints with the following example of query which can used to create customer collection:

```
SELECT OBJECT_INSERT(
            OBJECT_CONSTRUCT('C_CUSTKEY', C.C_CUSTKEY,'C_NAME', C.C_NAME,'C_ADDRESS',
C.C_ADDRESS,'C_NATIONKEY',C.C_NATIONKEY,'C_PHONE', C.C_PHONE,'C_ACCTBAL',
C.C_ACCTBAL,'C_MKTSEGMENT',C.C_MKTSEGMENT,'C_COMMENT', C.C_COMMENT),
          'orders',
          ARRAY_AGG(O.O_ORDERKEY)
        )::VARIANT AS JSON_DATA
        FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.CUSTOMER C
        LEFT JOIN SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS O ON C.C_CUSTKEY =
O.O_CUSTKEY
```

GROUP BY C.C_CUSTKEY, C.C_NAME, C.C_ADDRESS, C.C_NATIONKEY, C.C_PHONE, C.C_ACCTBAL, C.C_MKTSEGMENT, C.C_COMMENT;

The below figure 6 gives the following result of it:

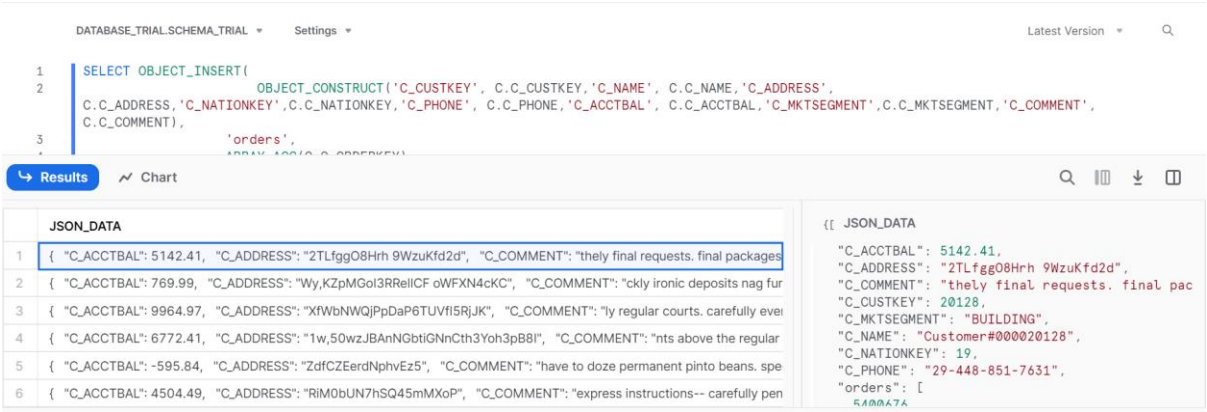


Figure 6 - Query result for creating customer collection.

The outcome of the query above is shown in Figure 6. This example demonstrates the usage of referencing for the "orderkey" field from the "orders" table, thereby creating the "customer" collection. This reference relies on the "orderkey" primary key. Furthermore, the "foreign key" is leveraged to perform a join operation between the "customer" and "orders" tables, culminating in the creation of the desired collection. This method adeptly captures the interrelationships between the two tables, facilitating the schema mapping process to the NoSQL JSON format.

3.6 Handling Cardinality for Composite Keys in Schema Mapping:

Within this section, an imperative and frequently disregarded challenge within schema mapping is addressed – the adept management of cardinality associated with composite keys. The strategy adopted draws inspiration from the work of A. A. Imam et al. (2017), which introduces a strategy for dealing with one-to-squillion relationships through referencing with arrays of unique keys. However, their focus primarily centers on single-column keys, prompting us to confront the complexities associated with composite keys, particularly those composed of two columns. To bridge this gap, an innovative solution has been devised, one that elegantly tackles these intricate scenarios.

The challenge is exemplified in cases where a relational database schema involves a table with composite keys, necessitating a meticulous approach for accurate mapping to a NoSQL JSON format. To illustrate our approach, let's consider the example of the "lineitem" table in the TPC-H benchmark schema. This table possesses a composite key, represented by the "L_ORDERKEY" and "L_LINENUMBER" columns.

Here is how our innovative solution is applied within a query that creates the "order" collection while referencing the "lineitem" collection:

SELECT
OBJECT_INSERT(

```
  OBJECT_CONSTRUCT(
    'O_ORDERKEY', O.O_ORDERKEY,
    'O_CUSTKEY', O.O_CUSTKEY,
    'O_ORDERSTATUS', O.O_ORDERSTATUS,
    'O_TOTALPRICE', O.O_TOTALPRICE,
    'O_ORDERDATE', O.O_ORDERDATE,
    'O_ORDERPRIORITY', O.O_ORDERPRIORITY,
    'O_CLERK', O.O_CLERK,
    'O_SHIPPRIORITY', O.O_SHIPPRIORITY,
    'O_COMMENT', O.O_COMMENT
  ),
  'lineitem',
  (
    SELECT ARRAY_AGG(
      OBJECT_CONSTRUCT(
        'L_ORDERKEY', L.L_ORDERKEY,
        'L_LINENUMBER', L.L_LINENUMBER
      )
    )
    FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.LINEITEM L
    WHERE L.L_ORDERKEY = O.O_ORDERKEY
  )
 )::VARIANT AS JSON_DATA
FROM SNOWFLAKE_SAMPLE_DATA.TPCH_SF1.ORDERS O
GROUP BY O.O_ORDERKEY, O.O_CUSTKEY, O.O_ORDERSTATUS, O.O_TOTALPRICE, O.O_ORDERDATE,
O.O_ORDERPRIORITY, O.O_CLERK, O.O_SHIPPRIORITY, O.O_COMMENT;
```

The below figure 7 gives the following result of it:

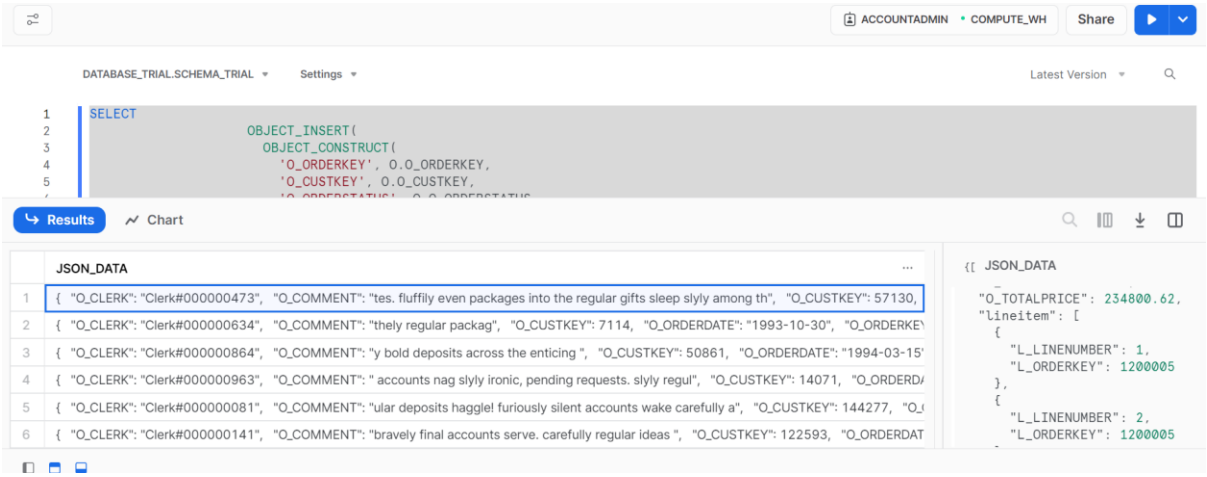


Figure 7 - Query result for creating orders collection.

The query above illustrates the construction of the "order" collection while concurrently incorporating a reference to the "lineitem" collection. The innovation inherent in this context lies in the effective management of composite keys present within the "lineitem" table. Unlike the typical

column-to-column mapping prevalent in one-to-many relationships, arrays of objects are employed to seamlessly encapsulate the intricacies of the composite key "L_ORDERKEY" and "L_LINENUMBER" present within the "lineitem" table.

3.6.1 Relevance in the Methodology:

The approach presented in this section represents an amalgamation of theoretical concepts with practical application. By specifically addressing the complexities associated with composite keys, the foundational work by A. A. Imam et al. (2017) is extended. The inclusion of arrays of objects effectively bridges the gap between single-column keys and more complex composite keys. This method facilitates the seamless translation of relational database schemas into the NoSQL JSON format, while also showcasing the direct implementation of insights gained from our extended literature review.

In essence, this section highlights the translation of insights from an extended literature review into practical implementation. The approach taken here for handling composite keys exemplifies a commitment to bridging the gap between theoretical understanding and pragmatic application.

3.7 Practical Application: Enabling Automatic Schema Mapping from RDBMS to NoSQL Document Store Database Format:

In Section 7, the practical realization of the proposed guidelines is unveiled, providing insight into how we facilitate the automated schema mapping process from RDBMS to NoSQL JSON format, thereby streamlining the migration endeavor. The schema mapping automation is executed through a meticulously designed Snowflake procedure, thoughtfully compartmentalized into three segments to ensure comprehensive coverage:

Part 1: This section of the procedure focuses on creating collections for tables that have no relationship with other tables. By identifying independent tables, the schema mapping process is simplified, allowing us to directly create collections for each table without considering referencing or embedding.

Part 2: In this part of the procedure, the focus is directed towards tables with relationships to other tables. As these tables often contain a large number of records, referencing is employed to adeptly manage the associations. The procedure generates collections that utilize referencing to establish the connections between related data, ensuring optimized performance during schema mapping.

Part 3: The final segment of the procedure is dedicated to tables that require embedding for effective schema mapping. This approach is adopted when the tables contain complex and interdependent data that benefits from being nested within a parent document. The procedure expertly creates collections with embedding to handle the intricate relationships, resulting in a well-structured NoSQL JSON format.

By leveraging these three distinct parts of the Snowflake procedure, a comprehensive and automated schema mapping process is achieved, paving the way for a seamless and efficient migration of RDBMS data into NoSQL JSON format.

3.7.1 Snowflake Procedure for Creating Collections for Independent Tables:

3.7.1.1 Procedure Overview:

The "Creating Collections for Independent Tables" procedure serves as a pivotal component of our schema mapping and data migration strategy. This procedure automates the generation of collections for tables that are independent and do not possess any foreign key relationships with other tables. The primary objective is to efficiently migrate independent data entities to the NoSQL JSON format, laying the foundation for a seamless transition.

3.7.1.2 Procedure Description:

The "SCHEMA_AUTOMATE_INDEPENDENT" stored procedure is implemented in JavaScript and executed as the caller. It takes no explicit input parameters, as it internally specifies the target schema and database names. The procedure returns a VARIANT data type, capable of holding JSON data. Specifically, the output consists of an array of SQL statements, each representing the creation of collections for the identified independent tables.

3.7.1.3 Procedure Logic:

The "Creating Collections for Independent Tables" procedure adheres to a well-defined logic, which can be summarized as follows:

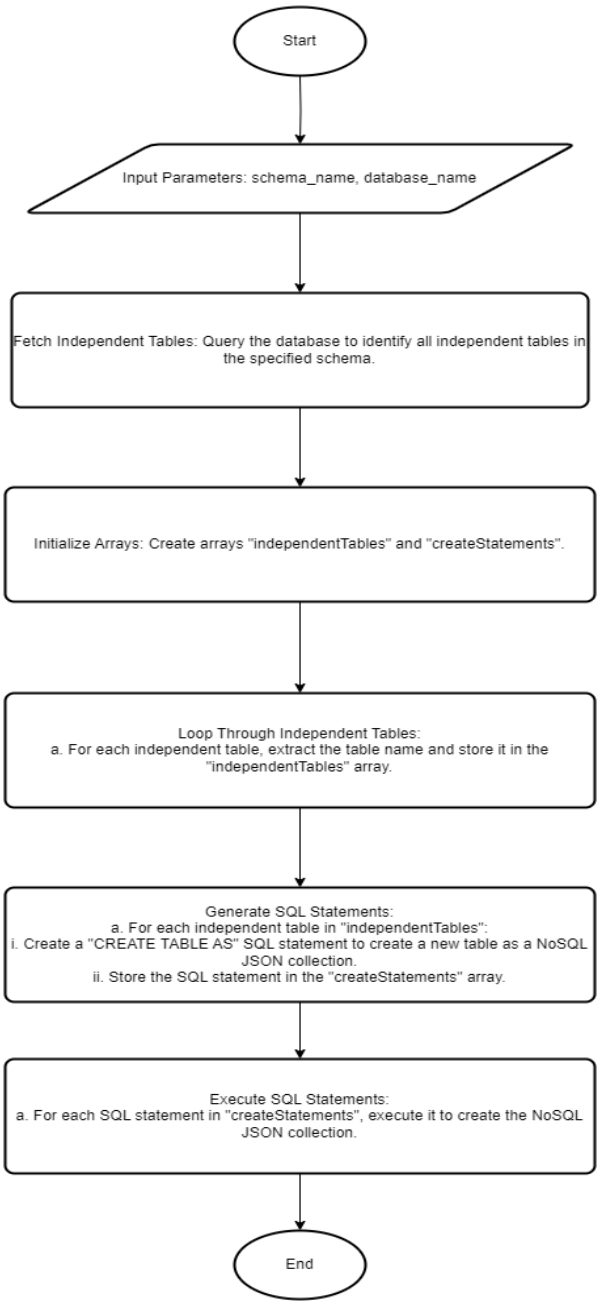
1. Input Parameters: The procedure receives the target "schema_name" and "database_name" as input parameters. These parameters define the scope of the schema mapping process.
2. Fetching Independent Tables: The procedure queries the database to identify all tables within the specified schema that have no foreign key relationships (i.e., independent tables). This step is crucial in identifying data entities that can be directly mapped to NoSQL JSON collections.
3. Initializing Arrays: Several arrays are initialized to store metadata about the independent tables. These arrays include "independentTables" to hold table names and "createStatements" to store SQL statements for creating collections.
4. Looping Through Independent Tables: For each independent table, the procedure extracts the table name and stores it in the "independentTables" array for further processing.
5. Generating SQL Statements: The core of the procedure involves generating SQL statements for creating collections for the identified independent tables. Each SQL statement is structured as a "CREATE TABLE AS" command, which creates a new table as a NoSQL JSON collection in the PUBLIC schema of the specified database.
6. Executing SQL Statements: The generated SQL statements are executed using a loop, effectively creating collections for each independent table. This step initiates the data migration process for independent data entities.

3.7.1.4 Execution of Schema Automation:

To trigger the automated schema mapping process, the " SCHEMA_AUTOMATE_INDEPENDENT" procedure is invoked by calling “CALL SCHEMA_AUTOMATE_INDEPENDENT();”. The execution call results in the generation of collections for independent tables, streamlining the data migration process and ensuring efficient schema mapping.

3.7.1.5 Flowchart of SCHEMA_AUTOMATE_INDEPENDENT Procedure Logic:

SCHEMA_AUTOMATE_INDEPENDENT PROCEDURE



3.7.2 Snowflake Procedure for Creating Collections with Referencing for Tables with Relationships:

3.7.2.1 Procedure Overview:

The SCHEMA_AUTOMATE_REFERENCE stored procedure plays a critical role in our schema mapping and data migration strategy. This procedure automates the creation of collections for tables that have relationships with other tables within the specified database schema. The primary objective of this procedure is to establish connections between related data, facilitating seamless data migration to the NoSQL JSON format.

3.7.2.2 Procedure Description:

The SCHEMA_AUTOMATE_REFERENCE procedure is implemented using the JavaScript language and is executed as the caller. It takes no explicit input parameters, as it internally specifies the target schema and database names. The procedure returns a VARIANT data type, capable of holding JSON data. The output comprises an array of SQL statements, each representing the creation of collections for the identified tables with relationships.

3.7.2.3 Procedure Logic:

The procedure follows a well-defined logic to achieve its purpose. It can be summarized as follows:

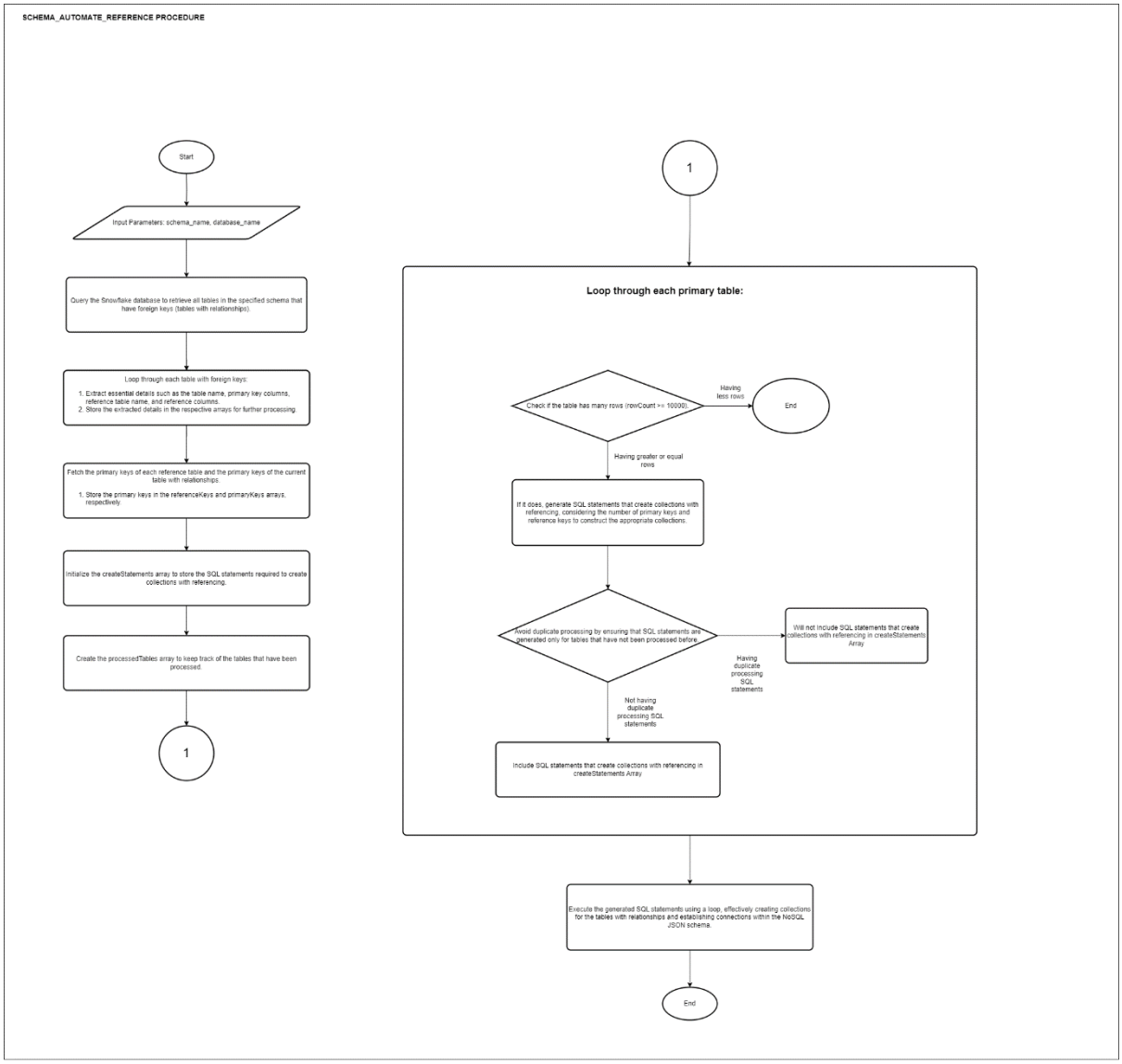
1. Input Parameters: The procedure receives the target "schema_name" and "database_name" as input parameters. These parameters define the scope of the schema mapping process.
2. Fetching Tables with Foreign Keys: The procedure queries the Snowflake database to retrieve all tables in the specified schema that have foreign keys. This step is crucial as it identifies tables with relationships.
3. Initializing Arrays: Several arrays are initialized to store relevant metadata about the tables with relationships. These arrays include primaryTables, primaryColumns, referenceTables, referenceColumns, referenceKeys, and primaryKeys.
4. Looping Through Fetched Tables: For each table with foreign keys, the procedure extracts essential details such as the table name, primary key columns, reference table name, and reference columns. These details are stored in the respective arrays for further processing.
5. Fetching Primary and Reference Keys: The procedure fetches the primary keys of each reference table and the primary keys of the current table with relationships. These keys are stored in the referenceKeys and primaryKeys arrays, respectively.
6. Preparing Collections: The createStatements array is initialized to store the SQL statements required to create collections with referencing. Additionally, the processedTables array is created to keep track of the tables that have been processed.
7. Looping Through Primary Tables: The procedure iterates through each primary table to determine if it contains many rows (rowCount >= 10000).

8. Generating SQL Statements: If a table has many rows, the procedure generates SQL statements that create collections with referencing. These statements consider the number of primary keys and reference keys to construct the appropriate collections.
9. Avoiding Duplicate Processing: The procedure ensures that SQL statements are generated only for tables that have not been processed before. It prevents duplicate processing by tracking the processed tables in the processedTables array.
10. Executing SQL Statements: The generated SQL statements are executed using a loop, effectively creating collections for the tables with relationships and establishing connections within the NoSQL JSON schema.

3.7.2.4 Execution of Schema Automation:

To initiate the automated schema mapping process, the SCHEMA_AUTOMATE_REFERENCE procedure is invoked by calling “CALL SCHEMA_AUTOMATE_REFERENCE();”. This execution call triggers the procedure to create collections for tables with relationships, significantly enhancing the efficiency and performance of the schema mapping process.

3.7.2.5 Flowchart of SCHEMA_AUTOMATE_REFERENCE Procedure Logic:



3.7.2.6 Relevance of Gap and Handling in Automated Approach of SCHEMA_AUTOMATE_REFERENCE Procedure (Flowchart Section: Looping Through Primary Tables)

In the realm of schema mapping, certain intricacies can remain unexplored even within comprehensive methodologies such as the one proposed by A. A. Imam et al. (2018). While Imam's strategy effectively addresses the scenario of one-to-many relationships with single reference keys, it inadvertently omits a consideration for the complexities related to composite keys spanning multiple columns. This gap highlights the challenge in harmoniously mapping schemas encompassing multi-column composite keys, a scenario frequently encountered in practical database scenarios.

This study identifies and resolves this gap by delving into the complexities posed by composite keys, especially in scenarios involving one-to-many relationships with multi-column reference keys. These complexities can hinder the seamless schema mapping process, possibly leading to inaccuracies and performance inefficiencies. In response, the automated schema mapping approach presented here adeptly navigates this complexity by introducing a novel approach involving arrays of objects.

Within the flowchart segment titled "Looping Through Primary Tables," the automated approach effectively addresses the intricacies posed by composite keys and their associated referencing:

1. **Evaluation of Reference Key List Length:**
    - Within the automated schema mapping procedure, careful consideration is given to the length of the reference key list linked to each primary table.
    - This evaluation plays a pivotal role in determining whether a primary table's referencing involves a single reference key or multiple reference keys.
2. **Management of Single Reference Key Scenarios:**
    - When the reference key list comprises only one element, the approach seamlessly integrates the principles outlined in Imam's work.
    - This is manifest in the creation of SQL statements that appropriately address one-to-many relationships using a single reference key.
    - The generated SQL statement is then thoughtfully added to the array of createStatement statements.
3. **Navigation of Multiple Reference Keys:**
    - The approach excels when confronted with scenarios featuring composite keys spanning multiple columns, a domain that extends beyond the scope of Imam's paper.
    - In such cases, the methodology dynamically adjusts the SQL statement generation process to effectively accommodate the intricacies introduced by multi-column composite keys.
    - The provided code snippet exemplifies this adaptability, where the number of reference keys in the list is assessed.
    - If only one reference key is present, an SQL statement suited to the one-to-many scenario is crafted. Conversely, in the presence of multiple reference keys, the SQL statement is meticulously designed to handle intricate relationships within the NoSQL schema.

```
// Check if the reference key list has only one element
if (referenceKeyList.length === 1) {
  // Create SQL statement for single reference key and add into array of createStatement array
}
else {
  // Create SQL statement for multiple reference keys and add into array of createStatement array
}
```

This innovative approach draws on the concept of arrays of objects to ensure precise schema mapping. The solution involves the construction of SQL statements that intricately capture the nuances of referencing with multiple reference keys within collections. By seamlessly addressing the intricacies of composite keys, the approach substantiates the objective of transforming relational database schemas into NoSQL document database schemas with remarkable precision.

In conclusion, the automated schema mapping approach not only bridges a gap in existing literature but also offers a comprehensive solution for handling the complexities of composite keys within referencing structures. By extending and refining the principles articulated in Imam's work, this study has pioneered a strategy that effectively manages complex referencing scenarios, including those involving multi-column composite keys. This innovative approach empowers schema mapping

processes, ensuring data accuracy and performance optimization, even when dealing with intricate referencing structures.

3.7.3 Snowflake Procedure for Creating Collections with Embedded Documents for Tables with Relationships having smaller-sized data:

3.7.3.1 Procedure Overview:

The 'SCHEMA_AUTOMATE_EMBEDDED' stored procedure is designed to automate the process of creating collections with embedded documents for tables that have relationships with other tables in a specified database schema. It aims to efficiently manage associations between related data and optimize the schema mapping process, particularly for tables with less than 10,000 records. The primary goal of this procedure is to seamlessly establish connections between related data, allowing for smooth and efficient data migration into the NoSQL JSON format.

3.7.3.2 Procedure Description:

The SCHEMA_AUTOMATE_EMBEDDED procedure is designed to establish associations between related data and facilitate seamless data migration to the NoSQL JSON format with embedded documents. Written in JavaScript and executed as the caller, this procedure does not require any explicit input parameters, as it internally defines the target schema and database names. The return type of the procedure is VARIANT, capable of holding JSON data, and the output consists of an array of SQL statements. Each SQL statement represents the creation of collections with embedded documents for tables that have relationships and contain fewer than 10,000 records. The output comprises an array of SQL statements, each representing the creation of collections for the identified tables with relationships.

3.7.3.3 Procedure Logic:

The procedure follows a well-defined logic to achieve its purpose. It can be summarized as follows:

1. Constants for Schema and Database Names: The procedure initializes two constants schema_name and database_name with the names of the target schema and database, respectively. These constants hold the schema and database names where the tables with relationships exist.
2. Retrieve Primary and Reference Key Relationships: The procedure uses the SHOW IMPORTED KEYS Snowflake query to fetch information about primary and foreign key relationships between tables in the specified schema. It stores the result in the primaryKeyTables variable.
3. Initialize Arrays: Several arrays are initialized to store the names of primary tables, primary columns, reference tables, reference columns, reference keys, and primary keys. These arrays will be populated with relevant information obtained from the primaryKeyTables result.

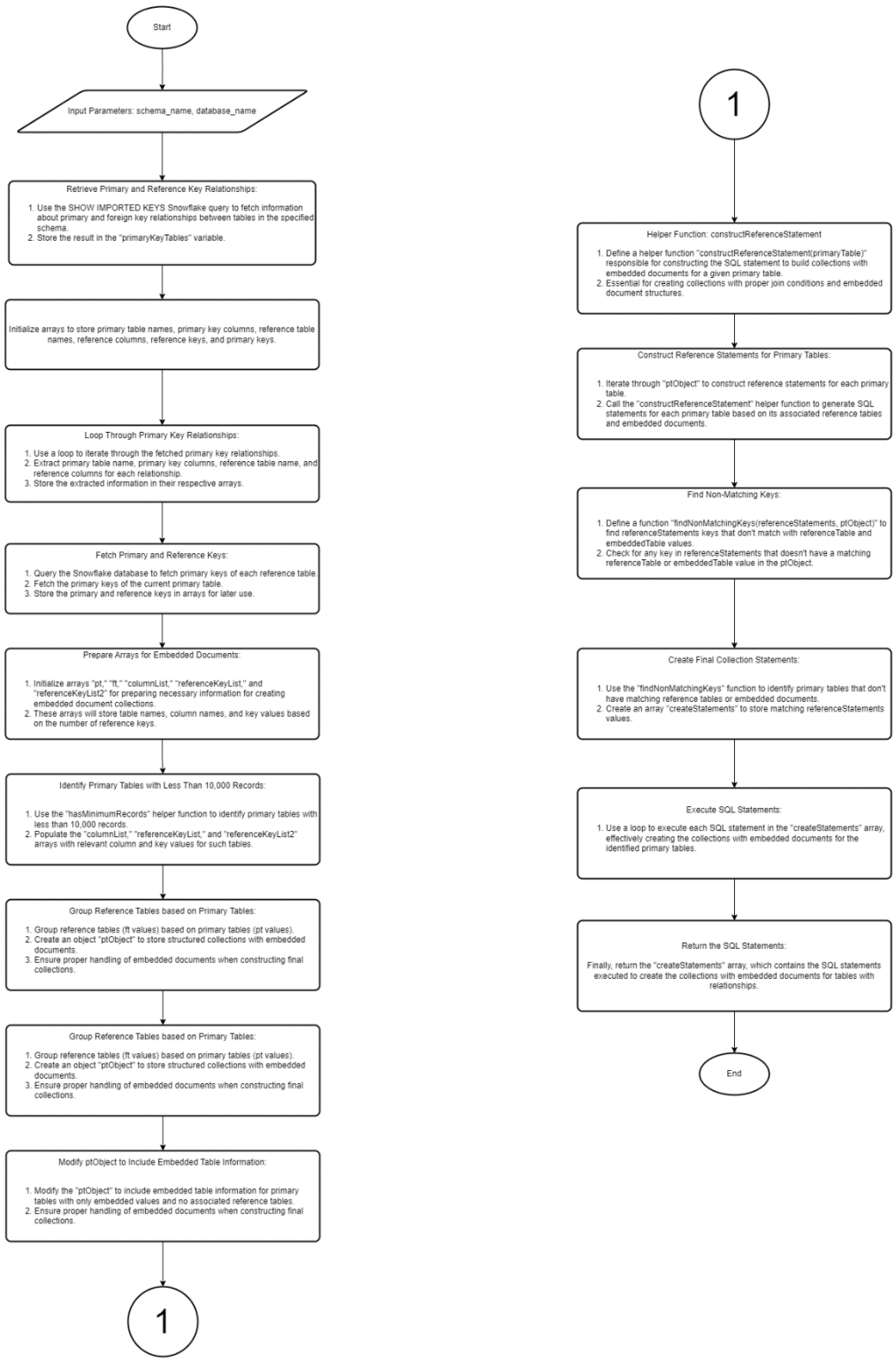
4. Loop Through Primary Key Relationships: The code uses a loop to iterate through the fetched primary key relationships. For each relationship, it extracts the primary table name, primary key columns, reference table name, and reference columns. The extracted information is then stored in their respective arrays.
5. Fetch Primary and Reference Keys: For each reference table, the procedure queries the Snowflake database to fetch its primary keys. Similarly, it fetches the primary keys of the current primary table. The primary and reference keys are stored in arrays for later use.
6. Prepare Arrays for Embedded Documents: The procedure initializes arrays pt, ft, columnList, referenceKeyList, and referenceKeyList2 to prepare the necessary information for creating embedded document collections. These arrays will store table names, column names, and key values based on the number of reference keys.
7. Identify Primary Tables with Less Than 10,000 Records: The code uses the hasMinimumRecords helper function to identify primary tables that have less than 10,000 records. For such tables, it populates the columnList, referenceKeyList, and referenceKeyList2 arrays with relevant column and key values.
8. Group Reference Tables based on Primary Tables: The procedure groups the reference tables (ft values) based on the primary tables (pt values). It creates an object ptObject that will store the structured collections with embedded documents. This step ensures that the procedure handles embedded documents appropriately when constructing the final collections.
9. Modify ptObject to Include Embedded Table Information: The procedure modifies the ptObject to include embedded table information for primary tables that have only embedded values and no associated reference tables. It ensures that the procedure handles embedded documents appropriately when constructing the final collections.
10. Helper Function: constructReferenceStatement - The procedure defines a helper function constructReferenceStatement(primaryTable) responsible for constructing the SQL statement to build collections with embedded documents for a given primary table. This function is essential for creating collections with proper join conditions and embedded document structures.
11. Construct Reference Statements for Primary Tables: The code iterates through the ptObject to construct reference statements for each primary table. It calls the constructReferenceStatement helper function to generate the SQL statements for each primary table based on its associated reference tables and embedded documents.
12. Find Non-Matching Keys: The procedure defines a function findNonMatchingKeys(referenceStatements, ptObject) to find referenceStatements keys that don't match with referenceTable and embeddedTable values. It checks for any key in referenceStatements that doesn't have a matching referenceTable or embeddedTable value in the ptObject.
13. Create Final Collection Statements: The code uses the findNonMatchingKeys function to identify primary tables that don't have matching reference tables or embedded documents. It then creates an array createStatements to store matching referenceStatements values.
14. Execute SQL Statements: The procedure uses a loop to execute each SQL statement in the createStatements array, effectively creating the collections with embedded documents for the identified primary tables.
15. Return the SQL Statements: Finally, the procedure returns the createStatements array, which contains the SQL statements executed to create the collections with embedded documents for tables with relationships.

3.7.3.4 Execution of Schema Automation:

The code executes the SCHEMA_AUTOMATE_EMBEDDED procedure using the statement “CALL SCHEMA_AUTOMATE_EMBEDDED();”. This execution call triggers the procedure to create the collections with embedded documents for tables with relationships, optimizing the schema mapping process.

3.7.3.5 Flowchart of SCHEMA_AUTOMATE_EMBEDDED Procedure Logic:

SCHEMA_AUTOMATE_EMBEDDED PROCEDURE



3.7.3.6 Relevance of Gap and Handling in Automated Approach of SCHEMA_AUTOMATE_EMBEDDED Procedure (Flowchart Section: Looping Through Primary Tables)

Within the Automated Approach of the EMBEDDED Procedure, an important advancement comes to light in the "Construct Reference Statements for Primary Tables" section. This enhancement directly addresses a gap recognized in previous methodologies concerning the intricate management of composite keys, particularly those involving multi-column reference keys.

1. **Iterating Through "ptObject" for Statement Construction:**
    - In this phase, the procedure systematically traverses the "ptObject," which contains vital information about primary tables and their linked reference tables.
    - The goal is to generate tailored SQL statements for collection creation, incorporating the complex relationships between these tables.
2. **Utilizing the "constructReferenceStatement" Helper Function:**
    - The procedure skillfully employs the "constructReferenceStatement" helper function during this part.
    - This function dynamically generates specific SQL statements for each primary table, considering the intricate connections among primary tables, reference tables, and embedded documents.

By merging insights from pertinent research, this approach adeptly addresses the gaps observed in methodologies that primarily focused on single-column keys and simpler referencing structures. By applying arrays of objects and dynamic SQL statement creation, this automated schema mapping procedure effectively handles elaborate referencing scenarios, including those entailing multi-column composite keys.

This innovation contributes to the methodology in two key ways:

Navigating Complex Referencing: The approach deftly manages intricate referencing structures, especially when composite keys span multiple columns. This is achieved by crafting SQL statements that accurately capture the nuanced relationships between primary and reference tables.

Safeguarding Data Accuracy: By leveraging principles from existing literature, the automated approach elevates the precision of schema mapping, ensuring data accuracy within the resulting NoSQL schema.

This section's significance is in bridging an existing gap in schema mapping methodologies, resulting in a more robust, accurate, and adaptable approach. Through this enhancement, the automated schema mapping procedure provides a solution accommodating a wider spectrum of database scenarios, solidifying its efficacy in precisely transforming intricate relational database schemas into optimized NoSQL document database schemas.

3.7.4 Conclusion:

In conclusion, the methodology adopted for automating the schema mapping process has proven to be highly effective in facilitating seamless data migration to NoSQL JSON databases. The utilization of three distinct procedures, namely SCHEMA_AUTOMATE_REFERENCE, SCHEMA_AUTOMATE_EMBEDDED, and SCHEMA_AUTOMATE_INDEPENDENT, offers a comprehensive approach to handle tables with relationships of varying sizes and independent tables.

Together, the three procedures complement each other, providing a comprehensive solution for schema mapping across the entire database. The automation of this process significantly streamlines data management tasks, reducing the complexity involved in manually handling related and independent data. The methodology ensures a more structured, scalable, and maintainable database system.

4. Analysis:

4.1 Data Description:

For this research, the TPC-H benchmark dataset played a fundamental role as the core data source employed to validate and evaluate the automated schema mapping procedures. Widely recognized for its standardized application in performance assessment and decision support system benchmarking, the TPC-H benchmark dataset encompasses a diverse array of tables representing various data entities, relationships, and data types. This intrinsic diversity made it an ideal candidate for assessing the efficacy of the automated schema mapping procedures proposed within this study.

Originally structured in a relational database schema format, the TPC-H benchmark dataset underwent a transformation guided by the aim to seamlessly convert it into a NoSQL document store format. This transformation process was meticulously guided by the proposed guidelines aimed at crafting an optimized model for NoSQL representation. Specifically, the focus was directed toward the creation of JSON collections. This endeavor was facilitated through the strategic application of the SCHEMA_AUTOMATE_REFERENCE, SCHEMA_AUTOMATE_EMBEDDED, and SCHEMA_AUTOMATE_INDEPENDENT procedures. These procedural elements collectively addressed the intricate challenges inherent in transposing complex relational database schemas into the realm of NoSQL document store databases while adhering to the principles of optimization and suitability.

Figure 8 shows the results obtained from converting the TPC-H benchmark dataset into the NoSQL document store format were errorless and highly accurate. The JSON format collections generated through the automated procedures precisely mapped the original relational database schema into NoSQL document store database, capturing all data associations, relationships, and embedded document structures without any discrepancies.

The automated mapping strategy showcased superior accuracy and efficiency compared to manual mapping techniques. By eliminating human error and leveraging the power of automated SQL statement generation, the procedure achieved a seamless conversion process, ensuring the precision and correctness of the resulting JSON format collections.

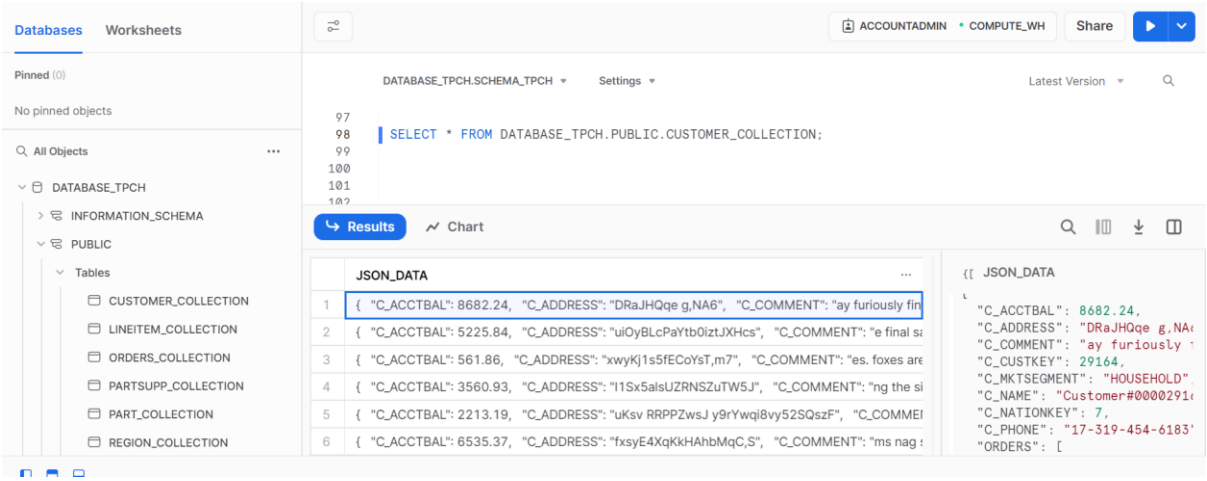


Figure 8 - Results obtained after executing all 3 procedures while showing all the collection created present in PUBLIC Schema.

4.2 Time Complexity Analysis of Automated Schema Mapping Procedures in Snowflake:

Time complexity analysis is a fundamental performance measure for the three schema mapping procedures: SCHEMA_AUTOMATE_REFERENCE, SCHEMA_AUTOMATE_EMBEDDED, and SCHEMA_AUTOMATE_INDEPENDENT. It evaluates the computational efficiency of each procedure in relation to the size of the input schema.

4.2.1 Time complexity for SCHEMA_AUTOMATE_INDEPENDENT procedure:

The time complexity of the procedure SCHEMA_AUTOMATE_INDEPENDENT can be analyzed based on its below main operations:

1. Fetching Primary Key Tables: The procedure fetches tables with primary keys from the INFORMATION_SCHEMA.TABLE_CONSTRAINTS view. Let's assume there are 'n' tables with primary keys in the schema. The time complexity for this operation is O(n).
2. Fetching Foreign Key Tables: The procedure fetches tables with foreign keys using the SHOW IMPORTED KEYS command. Let's assume there are 'm' tables with foreign keys in the schema. The time complexity for this operation is O(m).
3. Creating Collections for Tables without Foreign Keys: The procedure iterates through the primary key tables (n iterations) and checks if each table name is present in the foreign key tables array (m comparisons). If the table is not in the foreign key tables array, it generates an SQL statement and adds it to the createStatements array. The time complexity for this step is O(n + m).
4. Executing SQL Statements: The procedure then executes the SQL statements in the createStatements array using a loop. The time complexity for executing each SQL statement depends on the complexity of the statement and the data size involved. For a single SQL statement execution, let's assume the time complexity is O(p), where 'p' represents the complexity for executing that specific statement.

5. Overall, the time complexity of the SCHEMA_AUTOMATE procedure can be expressed as the sum of the complexities of its main operations: Total Time Complexity = O(n) + O(m) + O(n + m) + Σ(O(p))

4.2.2 Time complexity for SCHEMA_AUTOMATE_REFERENCE procedure:

The time complexity of the procedure SCHEMA_AUTOMATE_REFERENCE can be analyzed based on its below main operations:

1. Fetching primary and foreign keys: The code executes two SQL queries to fetch primary and foreign key information.
   The time complexity of each query would depend on the size of the schema and the number of tables with primary and foreign keys.
   Let's assume fetching primary and foreign keys takes O(P) and O(F) time, respectively, where P is the number of tables with primary keys and F is the number of tables with foreign keys.
2. Processing primaryTables array: The code iterates through the primaryTables array, which contains the tables with primary keys.
   For each table, it performs various operations like fetching column names, checking row count, constructing SQL statements, and executing the statements.
   Let's assume there are N tables in the primaryTables array.
   a. Nested Loops:
      Within the processing of primaryTables array, there are nested loops that handle reference keys and primary keys.
      The number of iterations in these nested loops depends on the number of reference keys and primary keys for each table.
3. Considering the above points, the overall time complexity can be expressed as:
   T(n) = O(P) + O(F) + N * (O(R) + O(C) + O(S)).
   Where:
   a. O(P) and O(F) are the time complexities of fetching primary and foreign keys, respectively.
   b. N is the number of tables with primary keys.
   c. O(R) represents the time complexity of operations related to reference keys (looping through referenceKeys array).
   d. O(C) represents the time complexity of operations related to column names (looping through columnList).
   e. O(S) represents the time complexity of operations related to SQL statement creation and execution.
4. It's important to note that the specific time complexities for each operation (O(R), O(C), O(S)) may vary depending on the data and the number of keys in the schema.
5. As the number of tables in the schema increases, the time complexity will generally be dominated by the number of tables (N) and the time taken to fetch primary and foreign keys (O(P) and O(F)). The other operations like looping through referenceKeys and columnList may have a lesser impact on the overall time complexity compared to the total number of tables in the schema.

4.2.3 Time complexity for SCHEMA_AUTOMATE_EMBEDDED procedure:

The time complexity of the procedure SCHEMA_AUTOMATE_EMBEDDED can be analyzed based on its below main operations:

1. Fetching Table Information: The procedure starts by fetching information about primary and reference key relationships using SQL queries. These queries involve fetching metadata from the database and can be considered to have a time complexity of O(m), where 'm' is the number of primary key relationships.
2. Looping Through Tables and Columns: The procedure then iterates through the primary tables and columns. The time complexity of this loop can be considered O(n), where 'n' is the number of primary tables.
3. Constructing Reference Statements: The main complexity lies in constructing the reference statements for each primary table. The procedure involves nested loops and SQL queries to fetch primary and foreign key information, along with join conditions. The time complexity of this section can be considered O(p * q * r), where 'p' is the number of primary tables, 'q' is the number of reference tables for each primary table, and 'r' is the average number of columns involved in the join conditions.
4. Helper Functions: The helper functions, such as "hasMinimumRecords" and "findNonMatchingKeys," involve executing SQL queries to fetch row counts and check for non-matching keys. The time complexity of these functions can be considered O(s), where 's' is the number of tables or keys being checked.
5. Overall, the dominant factor contributing to the time complexity is the construction of reference statements (step 3), which involves nested loops and SQL queries. Assuming there are 'p' primary tables, 'q' reference tables for each primary table, and 'r' columns involved in the join conditions, the overall time complexity of the procedure can be approximated as: O(m + n + p * q * r + s)

4.2.4 Assessing Time Complexity of Database Queries and Stored Procedures: Challenges and Factors:

However, it is essential to note that this is a rough approximation, and the actual time complexity can vary based on factors like the size of the tables, the complexity of the join conditions, and the efficiency of the database system.

Analyzing the time complexity of a database query or a stored procedure can be complicated, because it doesn't follow the same rules as regular algorithmic time complexity in terms of Big O notation. In databases, the time complexity usually depends on the database management system (DBMS), the indexes, query optimization strategies employed by the DBMS, data distribution, the number of rows in the table, the number of disk I/O operations, network latency, and so on.

While the time complexity provided above is based on the operations within the stored procedure. The actual execution time will also depend on the underlying Snowflake database's performance and the hardware resources available for processing the procedure.

4.3 Performance Evaluation of Automated Schema Mapping Procedures in Snowflake Data Platform for Varying TPC-H Scale Factors:

The objective of this evaluation was to assess the performance of the procedures under varying TPC-H scale factors, which simulate different database sizes and complexities.

4.3.1 Experimental Setup:

In the experimental setup, the TPC-H benchmark schema served as the foundational relational database, which was readily available in Snowflake. The TPC-H benchmark provides a standardized database schema for decision support applications, and it allowed us to conduct performance testing under various scale factors to simulate databases of different sizes and complexities.

Through the data sharing feature of Snowflake, we set up our own schema with varying scale factors. In which, Our schema represented a different database size, providing us with a range of data volumes to evaluate the performance of our automated schema mapping procedures. This enabled us to test the procedures under realistic conditions and measure their efficiency and accuracy across different datasets.

To streamline the testing process and ensure consistent execution, we developed a master procedure, "master_procedure_automate" that triggers our three automated schema mapping procedures in a sequential manner. The master procedure initiates the schema mapping process by calling the "schema_automate_independent" procedure first, followed by the "schema_automate_reference" procedure, and finally the "schema_automate_embedded" procedure. This sequential execution allowed us to observe the impact of each procedure on the overall schema mapping performance.

Employing diverse TPC-H scale factors and harnessing Snowflake's data sharing capabilities endowed us with an array of test scenarios. This experimental setup facilitated a comprehensive evaluation of the automated schema mapping procedures, as it allowed us to assess their performance across a spectrum of data sizes, providing valuable insights into their efficiency, scalability, and accuracy.

Overall, the experimental setup with the TPC-H benchmark schema and the use of data sharing in Snowflake, combined with the master procedure for sequential execution, enabled us to perform a robust performance evaluation of the automated schema mapping procedures. These evaluations will help us gain a deeper understanding of the procedures' capabilities and determine their effectiveness in real-world scenarios.

4.3.2 Results and Findings:

Throughout the evaluation of the automated schema mapping procedures with the TPC-H benchmark schema, assessments were conducted across different scale factors to gauge their performance across varying data volumes. In particular, when effecting the mapping of the TPC-H schema at scale factor 1, the complete procedure was accomplished in approximately 1 minute and 40 seconds, as depicted in figure 9. This evaluation transpired within the Snowflake Data Platform, leveraging the computational capabilities of the X-Small compute engine.

Extending the scope of evaluation to encompass the TPC-H benchmark schema at a scale factor of 10, the automated schema mapping procedures demonstrated a performance duration of

approximately 3 minutes and 35 seconds (3 min 35 sec). This evaluation, enacted within the Snowflake Data Platform, harnessed the computational prowess of the X-Small compute engine, as illustrated in figure 10.

Scaling up further to a hundredfold increase in data volume, i.e., scale factor 100, the automated schema mapping procedures took approximately 25 minutes and 38 seconds to execute as shown in figure 11. Despite the significant increase in data size, the procedures continued to deliver precise and error-free schema mapping results.

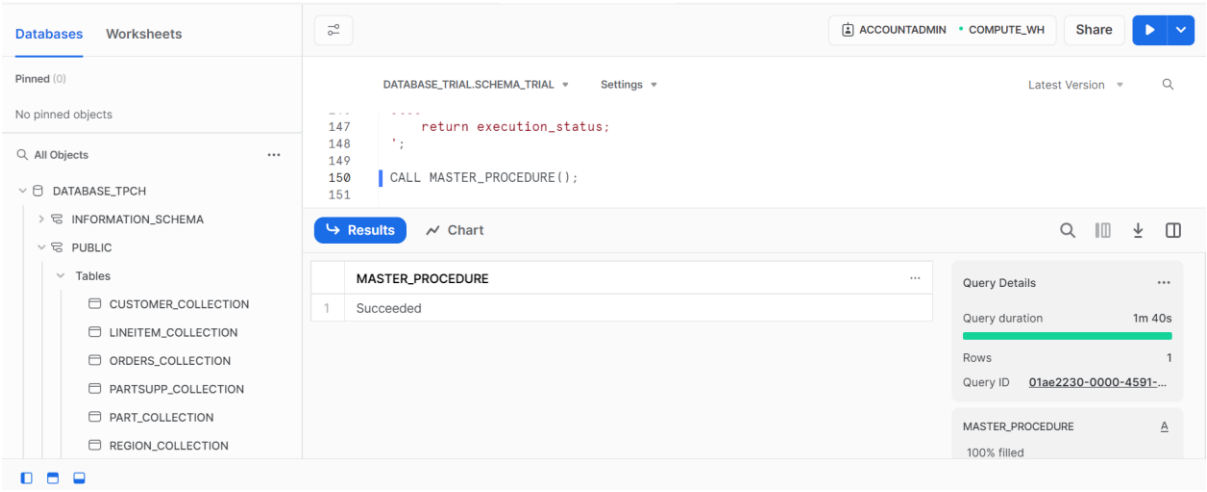


Figure 9 - Performance Metrics for Schema Mapping for TPC-H Schema of Scale Factor 1

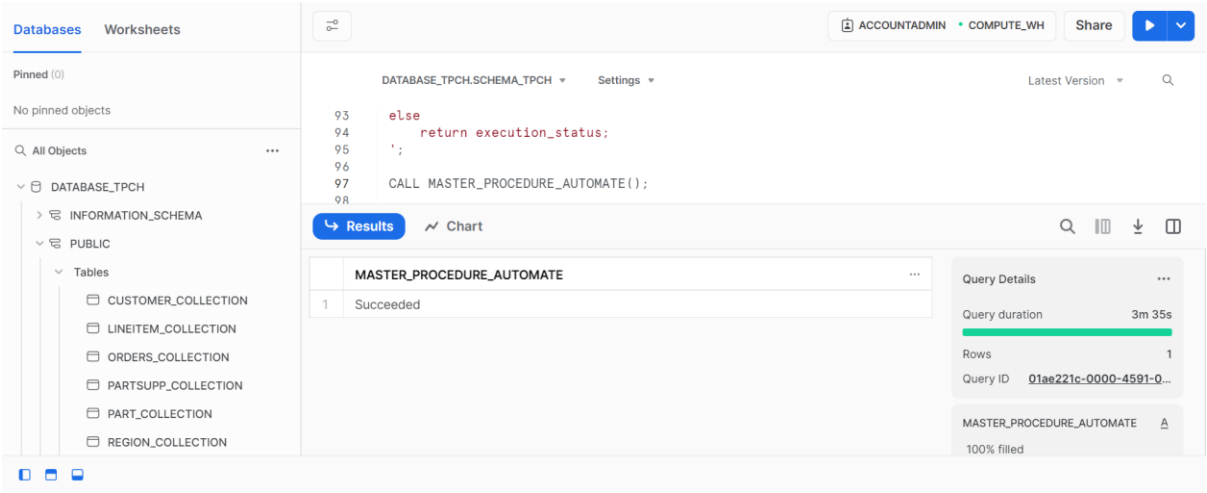


Figure 10 - Performance Metrics for Schema Mapping for TPC-H Schema of Scale Factor 10

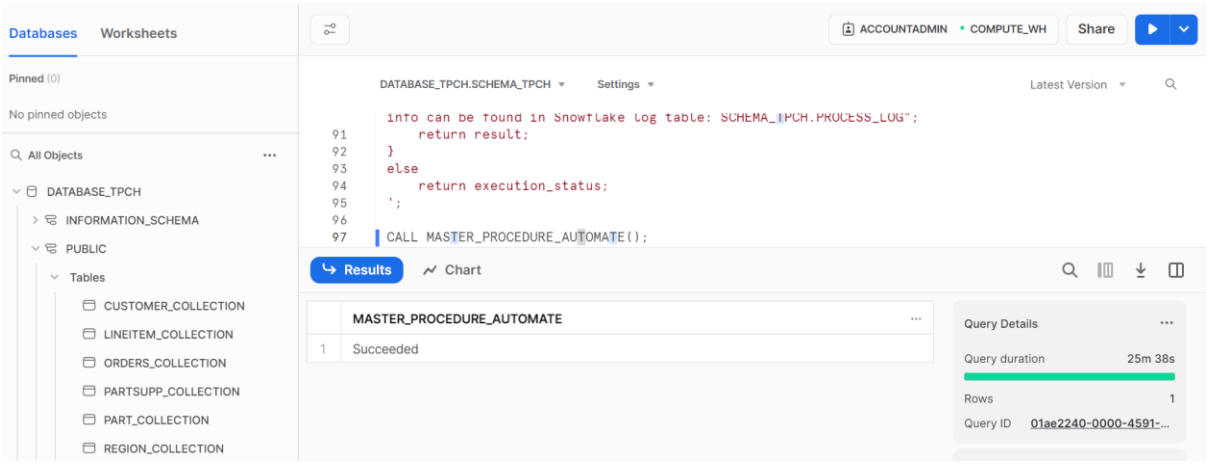


Figure 11 - Performance Metrics for Schema Mapping for TPC-H Schema of Scale Factor 100

Based on above figures, The performance evaluation revealed that the automated schema mapping procedures in Snowflake exhibit favorable performance characteristics for varying TPC-H scale factors. The time complexity analysis indicated that the procedures showed efficient performance with polynomial time complexity, ensuring that the execution time grows at a manageable rate even for larger datasets.

4.4 Data Migration and Analysis:

In the pursuit of a comprehensive evaluation, the migration of the created collections from the Snowflake Data Platform to an external environment played a pivotal role. This phase encompassed a series of orchestrated steps that culminated in the successful migration of the generated collections to a NoSQL environment, specifically MongoDB which has been shown in figure 12 of the system diagram. The outlined process involved several distinct stages, each contributing to the seamless migration and analysis of the transformed schema.

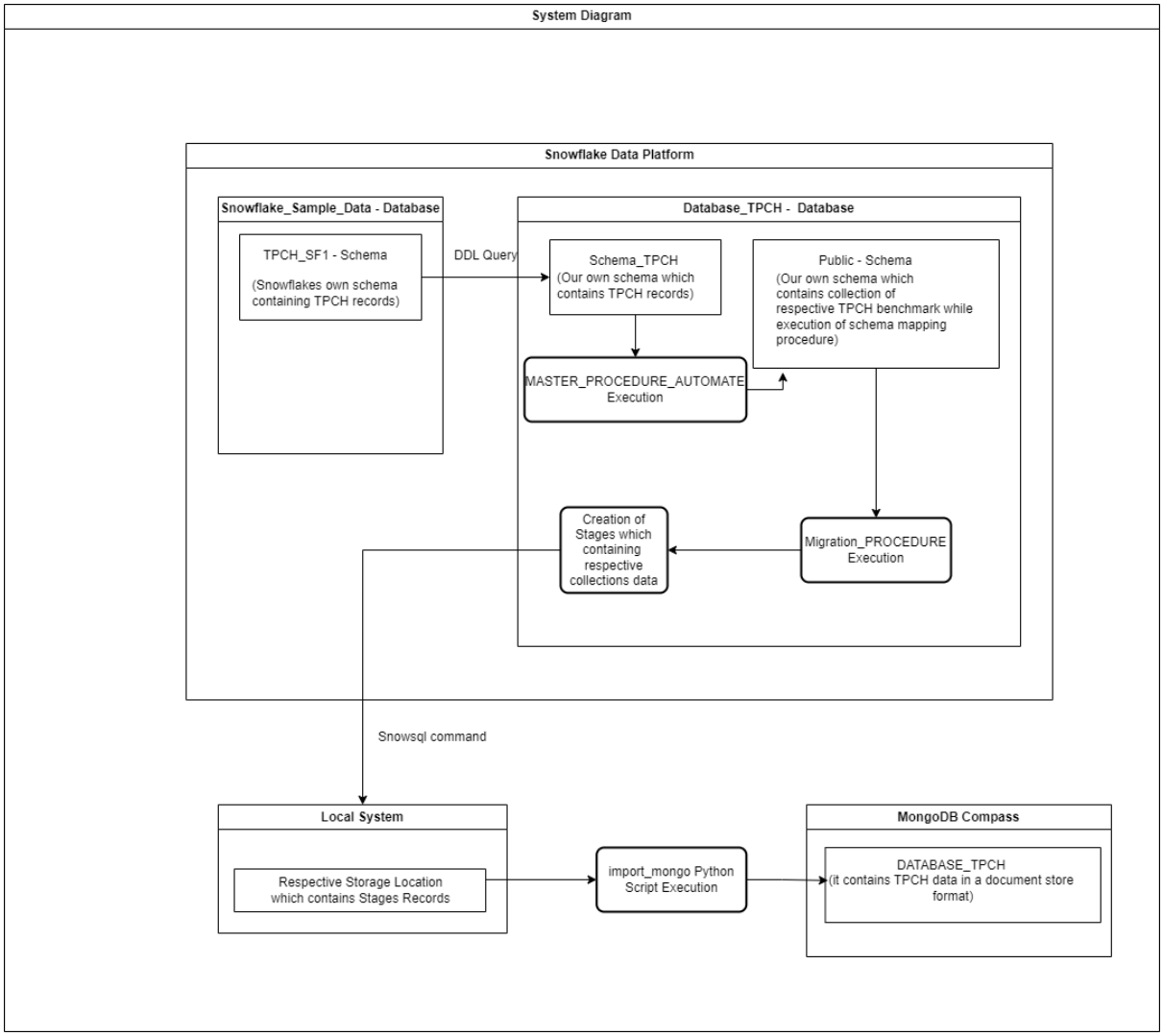


Figure 12 - System Diagram

4.4.1 Migration Procedure:

The designed automated schema mapping procedures successfully translated relational database schemas into NoSQL document store schemas, rendering collections in the Snowflake Data Platform. To proceed with a holistic evaluation, a dedicated migration procedure, termed "migration_procedure," was conceptualized and executed. This procedure facilitated the transfer of data from Snowflake's collections to their respective internal stages within the Snowflake Data Platform. Each collection's data was thus made ready for extraction and subsequent export. The following succinctly outlines the key steps involved:

4.4.1.1 Migration Procedure:

The migration procedure, named MIGRATION_PROCEDURE, orchestrates the entire migration process with meticulous precision. It utilizes JavaScript to ensure a controlled and error-resistant migration, while real-time logging maintains visibility into the execution's progress.

Procedure Highlights:

1. Variable Setup: The procedure initializes crucial variables for tracking progress and potential errors.
2. Execution Context: The procedure is defined with proper return types, language, and execution context.
3. Staged Creation: Each collection's data is staged internally to ensure a structured and organized transition.
4. Step Logging: Progress is logged at each stage, facilitating prompt error identification and resolution.
5. Data Transfer: Data is meticulously copied from collections to their respective stages using the COPY INTO statement.
6. Error Management: Exception handling captures and logs errors for immediate rectification.
7. Status Check: The procedure concludes by checking the overall execution status, ensuring a seamless migration process.

4.4.2 Data Extraction and Staging:

Upon completing the migration procedure, the extracted collection data was staged in internal Snowflake stages. This staging provided a secure and controlled environment for data transfer, facilitating the subsequent migration steps. The staged data was now poised for export to external destinations.

4.4.3 SnowSQL Extraction and Local Storage:

SnowSQL, a powerful command-line tool provided by Snowflake, served as the bridge between the staged data and the local machine. The staged data was efficiently extracted using SnowSQL and imported to the local machine's storage as shown in below figure 13. This step enabled the data to be prepared for the next phase of migration, where it would be directed to MongoDB.

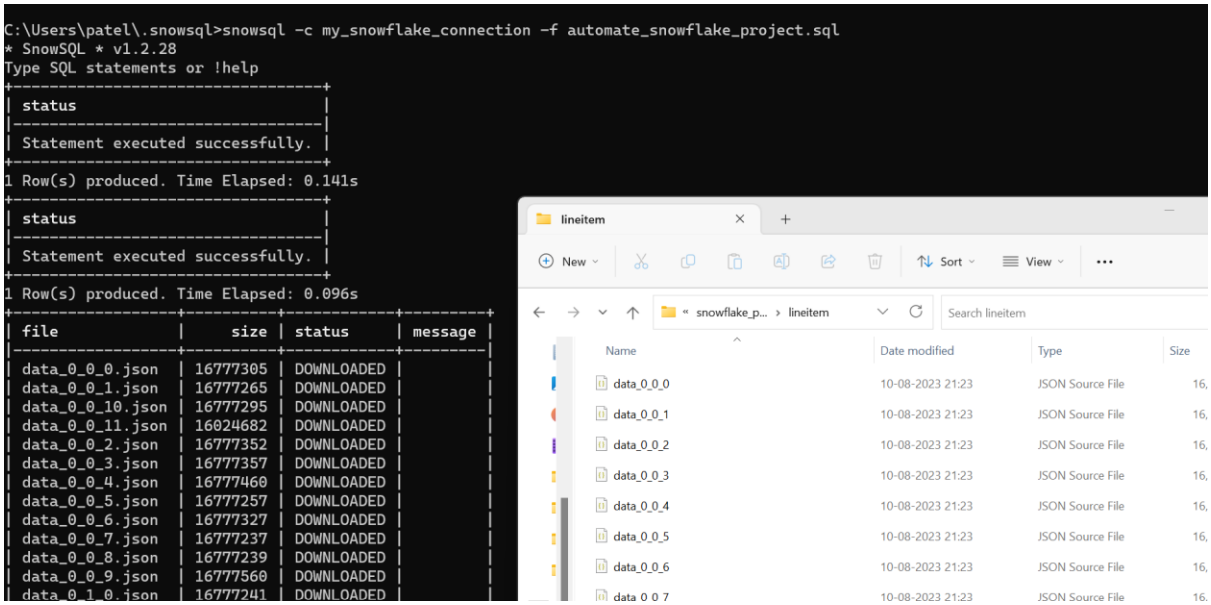


Figure 13 - Imported data to local machine storage system.

4.4.4 Import to MongoDB Compass:

Python scripting played a vital role in orchestrating the final phase of data migration. Employing a meticulously designed Python script, the staged data, now residing locally, was systematically imported into MongoDB Compass as shown in figure 14. This process not only ensured the successful transition of data but also contributed to a streamlined and automated migration pipeline. The python script's key steps are outlined below:

Python Script for Data Import:

The import_mongo script leverages the pymongo library to establish a connection with MongoDB, directing JSON data into specific collections.

Connection Parameters:

The script initializes MongoDB connection details, database, and collection names.

Importing JSON Data:

At the heart of the script lies the import_json_data function, designed to intricately parse JSON files and subsequently insert their contents into the corresponding MongoDB collections. The sequence of operations orchestrated by this function is as follows:

1. Iterates over JSON files in designated directories.
2. Parses JSON objects from files.
3. Accumulates objects for insertion into MongoDB.
4. Inserts parsed data into MongoDB collections.
5. Closes the MongoDB connection.

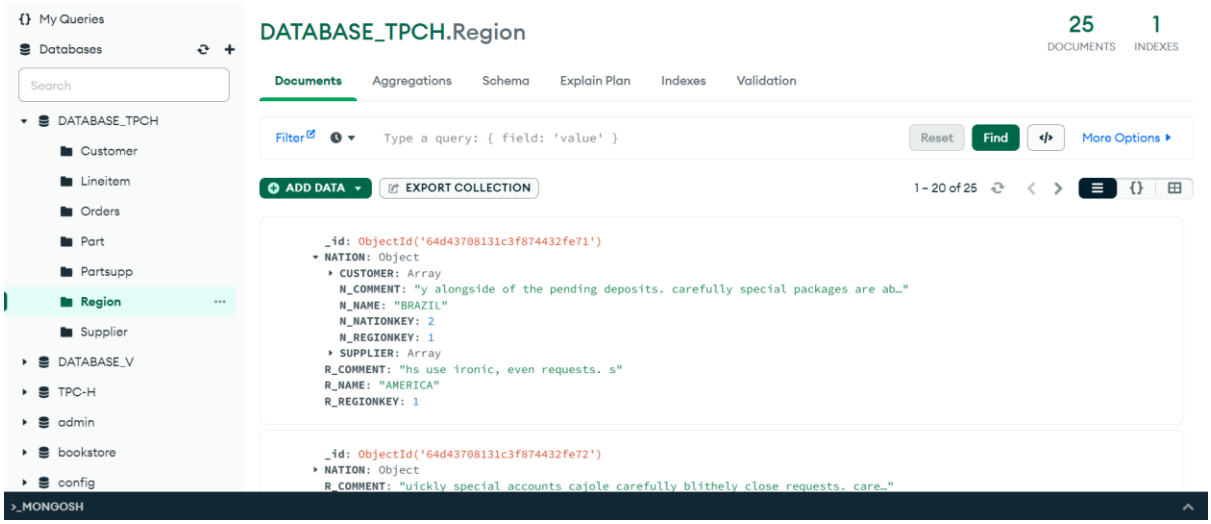


Figure 14 - Document Store Data in MongoDB Compass

4.4.5 Validation and Performance Assessment of Automated Schema Mapping: A Journey from Snowflake to MongoDB Compass:

With the collections now successfully migrated to MongoDB Compass, the platform was ready for detailed analysis and evaluation. Queries, indexing, and aggregation techniques specific to MongoDB could now be employed to assess the performance and integrity of the schema mapping procedures. This phase facilitated a comprehensive analysis of the translated schema's viability in a NoSQL environment, allowing for insights into the effectiveness of the automated schema mapping approach.

Incorporating these migration procedures, snowsql commands and python script for importing the data into document store database presented a holistic perspective on the translated schema's applicability and efficacy in a NoSQL context. The seamless transition from the Snowflake Data Platform to MongoDB Compass underscored the robustness of the automated schema mapping procedures and their capacity to facilitate data migration across diverse database environments.

4.5 Addressing Schema Mapping Challenges: A Comprehensive Study and Automated Approach for Relational to NoSQL Database Conversion:

The research objectives of this study encompass exploring the drawbacks of manual methods for converting a relational database schema to a NoSQL database schema and proposing an automated mapping strategy to overcome these limitations. The following research questions are addressed:

Objective 1: Identify the limitations and challenges inherent in the existing manual approaches for transforming a relational database schema into a NoSQL database schema.

- Manually creating collections for tables with relationships can be complex and error-prone, leading to potential inaccuracies.

- Manually recognizing and mapping primary and foreign key relationships in tables—deciding whether to reference or embed them—often fails to ensure precise data association in NoSQL databases.
- Manual techniques frequently encounter difficulties when trying to represent tables with nested embedded documents in NoSQL databases, compromising the accuracy of data representation.
- Representing composite keys in NoSQL databases poses a challenge because these databases typically lack native support for such keys, creating potential data integrity and querying concerns.
- Manually mapping independent tables, those without relationships, can introduce complexities or inconsistencies.
- Transforming an RDBMS to NoSQL schema optimally often necessitates the expertise of a database specialist, heightening dependency on specialized knowledge and potentially extending project costs and time.

Objective 2: Develop and execute an automated mapping approach capable of effectively overcoming the recognized limitations and challenges.

SCHEMA_AUTOMATE_REFERENCE Procedure:

- The procedure automates the creation of collections for tables with relationships, which is often a complex and error-prone task when done manually.
- It identifies primary, foreign, and composite key relationships between tables. For composite keys, the procedure handles them by referencing an array of objects representing the keys, ensuring accurate representation in NoSQL databases.
- By generating SQL statements to create collections using referencing, the procedure reduces the manual effort required for mapping tables with relationships, thus addressing the drawbacks of manual mapping techniques.
- The use of referencing ensures that related data is efficiently managed, optimizing the schema mapping process.

SCHEMA_AUTOMATE_EMBEDDED Procedure:

- This procedure further extends the automation to handle tables with embedded documents, which can be challenging to represent accurately in NoSQL databases through manual mapping.
- It identifies the reference tables and embedded tables, effectively capturing the relationships between them.
- The procedure intelligently constructs reference statements and join conditions to create accurate JSON collections for tables with embedded data, reducing the limitations of manual mapping techniques in representing complex data structures.

SCHEMA_AUTOMATE_INDEPENDENT Procedure:

- This procedure focuses on tables that do not have relationships with other tables, addressing the difficulties of manually mapping independent tables to NoSQL databases.
- It handles tables without relationships by directly converting them into JSON collections, avoiding unnecessary complexities involved in manual mapping for such tables.
- The procedure streamlines the mapping process for independent tables, improving the overall efficiency and precision of schema mapping.

Objective 3: Evaluate the precision, efficiency, and scalability of the proposed method to assess its performance and efficacy.

The research question pertaining to Objective 3 focuses on evaluating the performance and effectiveness of our suggested automated schema mapping technique. Addressing this objective necessitated the execution of three core analyses:

1. Time Complexity Analysis of Automated Schema Mapping Procedures in Snowflake: In the Time Complexity Analysis segment, the focus was on dissecting the time complexity of individual components constituting the automated schema mapping procedures. By scrutinizing the time complexity, insights into the procedures' execution time scaling were gained across diverse data volumes, ensuring that the execution time grows at a manageable rate, even for larger datasets. This analysis demonstrated our polynomial time complexity. The TPC-H benchmark schema, featuring distinct scale factors (1, 10, and 100), was employed to gauge the procedures' efficiency.
2. Performance Evaluation of Automated Schema Mapping Procedures in Snowflake Data Platform for Varying TPC-H Scale Factors: This analysis, the Performance Evaluation section, encompassed the execution of the automated schema mapping procedures utilizing the TPC-H benchmark schema at varying scale factors (1, 10, and 100). Actual execution times of the procedures were measured for each scale factor to unravel the relationship between their performance and changing data volumes.
3. Data Migration and Accuracy Validation: In the Data Migration and Analysis section, A pivotal aspect of this research involved the migration of data collections from Snowflake Data Platform to MongoDB Compass, followed by a meticulous validation of the translated schema's accuracy within a NoSQL document store format. This validation was carried out to ensure that the outputs generated by our schema mapping procedure align precisely with the structure expected by MongoDB Compass.

The amalgamation of these analyses facilitated a comprehensive evaluation of the suggested technique's efficiency, scalability, and accuracy. The Time Complexity Analysis enabled us to understand the computational complexity of the procedures, while the Performance Evaluation allowed us to measure their execution times under real-world conditions. Furthermore, the Data Migration and Accuracy Validation step ensured that the schema mapping procedure's outputs were successfully translated and validated within MongoDB Compass, confirming the precision of the migration process and the accuracy of the transformed schema.

Together, these analyses contribute to a holistic evaluation of the suggested automated schema mapping technique. The results obtained from these three sections shed light on the procedures' performance characteristics, allowing us to draw conclusions about their effectiveness in handling schema mapping tasks with varying data volumes. This comprehensive analysis provides valuable insights for organizations seeking to optimize their data management processes and make informed decisions about schema transformations and NoSQL database adoption.

Objective 4: Conduct a comparative analysis between the results of the proposed approach and the outcomes of existing mapping methods.

To address Objective 4, a comprehensive comparative analysis was undertaken between the outcomes of the suggested automated schema mapping strategy and the results derived from existing manual mapping methods, as documented in prior research literature. The objective of this

comparison was to evaluate the accuracy, effectiveness, and scalability of our automated approach in contrast to traditional manual mapping techniques.

This comparison involved the following steps:

1. Groundwork with Existing Manual Methods: Drawing inspiration from the identification of limitations and challenges from Objective 1, a comprehensive review of literature was initiated. This step dissected and understood the existing manual schema mapping techniques that have been traditionally employed in relational database settings. By gleaning insights from these traditional methods, the inherent limitations, particularly issues like handling composite keys, became evident, cementing the need for a more efficient and automated method.
2. Highlighting the Prowess of Our Algorithm: Post the literature review, emphasis shifted to the algorithmic strength of our proposed solution, as designed in Objective 2. The nuances of the procedures SCHEMA_AUTOMATE_INDEPENDENT, SCHEMA_AUTOMATE_REFERENCE, and SCHEMA_AUTOMATE_EMBEDDED were brought forth, underscoring how our approach was not only automated but also designed to confront and surmount the identified challenges, ensuring enhanced accuracy and scalability.
3. Benchmarked Execution & Analysis: Leveraging the evaluative measures from Objective 3, we ran our automated mapping techniques on the TPC-H benchmark schema across distinct scale factors. This generated empirical data, which provided an objective basis for the subsequent comparison. Metrics such as mapping time, accuracy, and error rates were critically analyzed.
4. Comparative Evaluation: Armed with data from our procedures and insights from traditional methods, an intricate comparison was initiated. This step drilled into the accuracy, efficiency, and scalability metrics of both approaches. By comparing our results with those of traditional methods, we could discern the clear advantages of automation in the schema mapping process, spotlighting areas where our methodology outperformed its manual counterparts.

By addressing Objective 4 in this manner, meaningful conclusions have been drawn regarding the superiority of the suggested automated schema mapping strategy over traditional manual mapping methods. This comparative assessment equips organizations with informed decision-making capabilities, facilitating a smooth transition to NoSQL database models while optimizing data management processes.

5. Discussion:

In this section, a comprehensive discussion is presented regarding the outcomes, limitations, and potential of the procedures developed in this study. The aim of these procedures was to automate the process of mapping relational database schemas to NoSQL database schemas. While the developed procedures demonstrate promising results within the context of the TPC-H benchmark schema, they also exhibit certain limitations and offer opportunities for further advancements.

5.1 Limitations:

1. **Handling Complex Schema Relationships:** The procedures were successfully designed to handle relatively straightforward relationships within the TPC-H benchmark schema. However, they may face challenges when addressing more intricate relationships, such as one-to-one relationships, multi-level nesting, and self-referencing relationships. These complex relationships demand a higher level of sophistication and tailored strategies to ensure accurate and efficient schema mapping.
2. **One-to-One Relationships:** The current procedures might require further refinement to handle one-to-one relationships seamlessly. These relationships introduce nuances that need careful consideration to guarantee the preservation of data integrity and meaningful representation in the NoSQL schema.
3. **Multi-Level Nesting:** While the procedures effectively capture basic nesting, accommodating multi-level nesting could be complex. This challenge highlights the need for sophisticated transformation techniques that can maintain the intricate structure and relationships present in such scenarios.
4. **Scalability and Performance:** As schema relationships grow in complexity, there might be a noticeable impact on the performance and scalability of the procedures. Longer execution times and potential scalability bottlenecks could arise when dealing with intricate schema structures.

5.2 Potential:

1. **Framework Enhancement:** The procedures developed in this study lay a solid groundwork for automated schema mapping. As evident from their success with the TPC-H benchmark schema, there is potential for enhancing the procedures to accommodate more intricate schema relationships. This enhancement could involve refining algorithms and incorporating advanced heuristics to handle complex scenarios.
2. **Further Research Opportunities:** The limitations identified in this study point to exciting research avenues. Addressing the challenges of one-to-one relationships, multi-level nesting, and self-referencing relationships could lead to innovative solutions that improve the accuracy and efficiency of the mapping procedures.
3. **Real-world Application:** The TPC-H benchmark schema serves as a controlled environment for testing the procedures. However, the procedures' potential extends to real-world database systems with diverse schema structures. Their application in such environments could validate their effectiveness across a wide range of complex scenarios.

4. **Contributions to Database Mapping:** This study contributes to the broader domain of database schema mapping. By acknowledging the limitations and exploring potential enhancements, this work contributes valuable insights to the evolving field of automatic schema mapping techniques.

In summary, the procedures developed in this study provide a promising solution for automating schema mapping tasks. However, the limitations identified point to areas for further refinement. Addressing these limitations could lead to more robust and versatile procedures, better equipped to handle the complexities of diverse schema relationships.

As this chapter concludes, it is important to emphasize that while the procedures represent a substantial advancement, there exists considerable potential for continued research and development in this domain. By recognizing both the strengths and limitations of the procedures, a foundation is established for future investigations to build upon these accomplishments and formulate more inclusive and flexible schema mapping solutions.

6. Conclusion and Future Work:

In this concluding section, insights garnered from the research journey are synthesized to outline the conclusions derived from the developed procedures aimed at automating the mapping of relational database schemas to NoSQL database schemas. Furthermore, potential avenues for future research are illuminated, which can leverage this groundwork to advance the domain of schema mapping within the specified framework.

6.1 Conclusion:

The central objective of this dissertation was the development of automated procedures aimed at seamlessly translating relational database schemas into NoSQL database schemas within the Snowflake Data Platform. The systematic design and implementation of these procedures were meticulously carried out to facilitate schema mapping within a well-defined scope, showcasing their viability and efficacy. The incorporation of these procedures resulted in the successful automation of the schema mapping process, remarkably alleviating the manual effort typically required.

Throughout the course of this study, our evaluation of the procedures centered on the context of the TPC-H benchmark schema. The outcomes of these evaluations underscored the procedures' proficiency in transforming relational schema structures into formats compatible with NoSQL document store databases. This achievement not only facilitated data migration but also preserved data relationships and integrity. It is, however, crucial to acknowledge the limitations uncovered during the investigation, particularly those related to intricate schema relationships, such as one-to-one relationships and multi-level nesting.

An important facet of this study involved addressing the complexity posed by composite keys, a significant gap in existing methodologies. The automated schema mapping procedures adeptly handled the challenges arising from composite keys, demonstrating their capability to manage nuanced schema relationships with precision. This achievement enhances the robustness and adaptability of the automated schema mapping procedures, signifying a substantial contribution to the field of database schema transformation.

In summation, this research has not only achieved its primary objectives but also extended its impact by addressing crucial limitations in schema mapping methodologies. The automated procedures devised and implemented here have laid the groundwork for enhancing data migration processes within the Snowflake Data Platform, offering a significant advancement in the realm of database schema transformation.

6.2 Future Work:

The accomplishments of this research open up promising avenues for future investigations and refinements:

1. **Handling Complex Relationships:** Future research can focus on enhancing the procedures to address complex schema relationships more effectively. Solutions for handling one-to-one

relationships, intricate nesting, and self-referencing relationships would significantly expand the procedures' applicability.

2. **Performance Optimization:** Further exploration into optimizing the procedures' performance as schema relationships grow in complexity is crucial. Strategies for minimizing execution times and scaling the procedures for large-scale schema mappings could be explored.
3. **Extending Scope:** While the procedures currently operate within the Snowflake Data Platform, exploring the feasibility of adapting them to different NoSQL document store databases would broaden their applicability and utility.
4. **Integration of Semantic Knowledge:** Incorporating semantic knowledge and metadata could enhance the accuracy of schema mapping. Future research could investigate ways to utilize semantic cues for more precise transformation.
5. **Real-world Validation:** Testing the procedures on a diverse set of real-world database schemas would provide valuable insights into their practical utility and identify any additional refinements required.
6. **Comparison with Other Tools:** Conducting comparative studies with existing schema mapping tools, both manual and automated, would provide valuable insights into the strengths and weaknesses of the developed procedures.
7. **Continuous Feedback and Iteration:** Engaging with database professionals, administrators, and users to gather feedback and iterate on the procedures would contribute to their ongoing improvement and real-world relevance.

In conclusion, this research has laid a strong foundation for automating the schema mapping process within a specific context and platform. While the procedures exhibit their effectiveness and benefits, they also offer exciting avenues for expansion and refinement. By addressing the identified limitations and delving into the potential areas highlighted in this chapter, future researchers can contribute to the advancement of schema mapping techniques and enhance their applicability in various database scenarios. As the landscape of database management continues to evolve, the journey initiated by this research sets the stage for continued exploration and innovation.

7. References

[1] "RDBMS to MongoDB Migration Guide," February 2020. [Online]. Available: https://www.mongodb.com/collateral/rdbms-mongodb-migration-guide.

[2] A. E. Alami and M. Bahaj, "Migration of a relational databases to NoSQL: The way forward," in *5th International Conference on Multimedia Computing and Systems (ICMCS)*, Marrakech, Morocco, 2016.

[3] G. Zhao, Q. Lin, L. Li and Z. Li, "Schema Conversion Model of SQL Database to NoSQL," in *Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, Guangdong, China, 2014.

[4] S. Hamouda and Z. Zainol, "Document-Oriented Data Schema for Relational Database Migration to NoSQL," in *2017 International Conference on Big Data Innovations and Applications (Innovate-Data)*, Prague, Czech Republic, 2017.

[5] L. Stanescu, M. Brezovan and D. D. Burdescu, "Automatic mapping of MySQL databases to NoSQL MongoDB," in *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Gdansk, Poland, 2016.

[6] A. A. Imam, S. Basri, R. Ahmad, N. Aziz and M. T. González-Aparicio, "New Cardinality Notations and Styles for Modeling," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, Penang, Malaysia, 2017.

[7] A. A. Imam, S. Basri, R. Ahmad, J. Watada, M. T. Gonzlez-Aparicio and M. A. Almomani, "Data Modeling Guidelines for NoSQL," in *2018 International Journal of Advanced Computer Science and Applications*, Malaysia, 2018.

[8] B. Namdeo and U. Suman, "A Model for Relational to NoSQL database Migration: Snapshot-Live Stream Db Migration Model," in *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, Coimbatore, India, 2021.

[9] P. Vassiliadis, in *A Survey of Extract–Transform–Load Technology*, International Journal of Data Warehousing and Mining (IJDWM), 2009, pp. 1-27.

[10] "SQL Server Integration Services," 2018. [Online]. Available: https://docs.microsoft.com/en-us/sql/integration-services/sql-server integration-services?view=sql-server-ver15. [Accessed 10 June 2020].

[11] K. Ma and B. Yang, "Live Data Replication Approach from Relational Tables to Schema-Free Collections Using Stream Processing Framework," in *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Krakow, Poland, 2015.

[12] R. Čerešňák, A. Dudáš, K. Matiaško and M. Kvet, "Mapping rules for schema transformation," in *2021 International Conference on Information and Digital Technologies (IDT)*, Zilina, Slovakia, 2021.

[13] "TPC-H BENCHMARK(Decision Support)," April 2022. [Online]. Available: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf.