Advances in Consumer Research
https://acr-journal.com/
Volume-1 | Issue-1 | Jan 2024
Original Researcher Article

Managing Dimensional History in Data Warehouses A Comparative Analysis of Hash-Based and Surrogate Key Approaches

Mr. Reddaiah Kasturi[1], Mr. Raghavendar Nellikondi[2]
[1]Independent researcher, Sterling VA, USA.
Email ID : Reddaiah.Kasturi@yahoo.com
[2]Independent researcher, Sterling, VA, USA,
Email ID : raghav.nellikondi@gmail.com

ABSTRACT
This paper will take on the ongoing discrepancy between the competencies that are taught to This paper compares the main two ways of keeping the dimensional history within a data store: Hash based approaches and Surrogate key type methods. Companies are using historical data increasingly for analytics and decision-making. Selection criteria on dimension management strategy is crucial to warehousing performance and maintenance. The hash-based approaches employ cryptography in order to process-dimensional attributes to generate distinctive identifiers. This allows changes to be discovered automatically and versioning to be retained without always maintaining additional lookup tables. Surrogate key approaches in contrast utilize sequentially generated system identifiers to monitor the evolution of dimensions through time. These methods typically involve the use of Slowly Changing Dimension (SCD) patterns. The work compares the two methods on a number of dimensions, including how difficult they are to implement, how well they support queries, how well they compress storage space after being implemented and how well they find differences, as well as scale. We evaluate the trade-off between these two approaches for different workloads and data sizes through experiments based on popular industry benchmarks. Using hash-based methods is better in detecting changes and simplifying ETL, while the proxy key approach is good at query optimization and easier to manage relationships. The study provides useful advice to data warehouse architects - for the first time through knowing when each method is more suitable. This allows them to make intelligent decisions that align with the realities of their business and the constraints of their technology..
Keywords: Data Warehouse, Dimensional History Management, Hash-Based Approach, Surrogate Key, Slowly Changing Dimensions..

1. INTRODUCTION:

Obsolete Data warehouses are an integral part of the infrastructure of modern businesses; as the amount of data grows considerably and business intelligence requirements become more complex. Modern DW systems must store not only large amounts of data, but also full historic versions (for time analysis as well as to comply with legal requirements) [1]. As the big businesses are transitioning to data-driven decision-making, the design of a data warehouse is being subjected to an unparalleled trauma due to speed and analytical agility. The tracking of dimensional history, i.e., how business units have evolved over time, soon becomes a major issue that in turn has profound implications on the warehouse performance, storage efficiency and complexity of the queries [2].

Dimensional design is the heart of a Data Warehouse. It aggregates data to fact tables as quantifiable business measures and to dimension tables holding the context for analysis [3]. But business is no lap dog and never becomes static. For instance, the addresses of customers change, product specifications change, supplier relationships change and organizational structures are subject to constant amendment. One of the most difficult technical challenges in building a data warehouse is aggregating this information over time while supporting fast query access and ensuring accurate results [4]. By the way that you architect it on how you manage dimensional history is going to permeate throughout your entire warehouse. These decisions have far reached implications, especially around the complexity of ETL logic that needs to be performed, how much storage is needed and performance of queries and last but not the least business tools that can be used by end-users.

Conventionally, the approach to dealing with dimensional history has been via surrogate key techniques whereby sequential numbers assigned by the system are assigned as primary keys regardless of a business meaningful value [5]. For security full history records, this popular model employs Type 2 Slowly Changing Dimension (SCD) patterns implemented to maintain row version. As for the surrogate key designs, they require a lookup table and an attribute-level comparison to find changed records and managing the surrogate key assignment among multiple ETL processes [6]. For all the decades of improvement and variety of tool support, though, these methods make the architecture more complex and slow things down — a bigger deal when data sizes reach petabyte scale.

Recent advancements in computer science and encryption techniques now enable us to utilize hash functions in novel ways for handling dimensional data. Hash-based techniques Cryptographic algorithms, such as SHA-256, are employed to convert the dimensional traits into deterministic identifiers. This results in natural keys displaying dimensions state naturally [7]. This paradigm shift eliminates the extra work required on the backbone side to create surrogate key and maintain look up tables. It also enables automatic discovery of comparing changes by simple hash comparison mechanisms. Since hash generation is deterministic, it can be done in parallel without coordination thus could be very useful for distributed data warehouse designs [8].

Despite the theoretical sounds, here's really not that much research/science that shows how these methods compare against the well-known surrogate key handling techniques to date. Much research focus on some part of the process like speed of ETL or optimizing query without considering all operational factors [9]. Beyond that, the rapid evolution of database technologies such as columnar storage engines, in-memory processing and cloud-native architectures means that old ideas about how best to handle dimensions need to be revisited. When organizations are designing new data warehouses or updating older data warehouse systems they require empirically-based guidance on how to select dimensional history management techniques that will be the most effective for their workload and performance targets [10].

To address these shortcomings, this paper performs a systematic comparison of hash-based and surrogate key methods to maintain dimensional history in the context of modern data warehousing. We consider architectural design, complexity of implementation, ETL processing performance, query execution efficiency, storage requirements and scalability allure. By creating controlled experimental setups based on typical industry benchmarks and with real-world workload characteristics, we present empirical evidence to quantify trade-offs associated to each of them. Instead of relying on intuition or vendor recommendations, the findings enable data warehouse architects and practitioners to make informed decisions with an understanding of what impact those decisions will have on performance.

The remainder of this paper goes as follows: In Section 2, we review relevant literature which discusses dimensional modeling methodologies, historical change management approaches and data warehousing performance improving techniques. We describe our research methods in Section 3. These contain architectural details of the two methods and the experiments design, along with metrics for performance and testing. There is a wealth of information in Section 4 about our experimental results, but also plenty of discussion about what those experimental results actually imply for practitioners. The final subsection of Section 5 is a conclusion that includes major findings and suggestions which architectures are more suitable in different heterogeneous deployment situation, and directions for further research from the aspect of dimensional history management.

2. Literature Review

2.1-Dimensional Modeling Foundations

The concepts behind dimensional modeling have evolved considerably since the '80s when decision support systems first appeared. The initial research indicated a fundamental difference between two types of databases: normalized operational ones are good with transactions, denormalized analytical ones are good with queries. The star schema emerged as the dominant way to present data, with fact tables in the center surrounded by dimension tables containing descriptive properties. This architecture favors simplicity and speed of query over storage optimization, which is consistent with how analytical workloads are usually structured. Later developments included snowflake schemas with normalized dimension hierarchies and galaxy schemas where fact tables reference more than one dimension table. These increased flexibility in the representation of dimensional designed to meet complex analytic requirements [11]. The idea behind dimensional modeling totally transformed the way organizations think about analytical databases—they lined up technical constructs with business objectives. This enabled non-technical users to write analytical queries, and performance of queries was enhanced thanks to denormalization techniques [12].

2.2 Slowly Changing Dimension Techniques

Classifying slow changing dimension methodologies resolved us the way of how to handle time-changing in DW. Type 1 SCD modifies existing numbers of attributes so eliminates any previous states. Type 2 SCD detail all historical records by writing new dimension rows for each change, complete with handy date ranges and current record pointers. Degree-3 SCD maintains only a little information by creating multiple columns for the historic values of an attribute. All three techniques are better adapted to addressing particular analytic requirements. Type 2 approaches are preferred in work environments where detailed historical analysis is indispensable (13). With traditional approaches, hunting after changes in attributes is a tough cookie when you need to create new dimensional rows and maintain referential integrity on fact tables. The amount of work required to discover such changes grows linearly with the number of attributes that are tracked [14]. This can be slow if the inputs are large.

2.3 Surrogate Key Architecture and Implementation

Patterns for implementation usually include sequence generators that give numbers that keep going up, lookup tables that connect business keys to current surrogate identifiers, and ETL logic that manages key assignment during dimensional processing [15]. There are pros and cons to each performance trait. When you use small integer keys for index structures and comparison operations, join operations work much better. However, the need to traverse lookup tables during ETL processing adds delay, especially in systems that process a lot of changes at once. There is still a small amount of extra storage needed, but having different lookup tables and indexes for them increases that need [16].

2.4 Hash-Based Approaches and Data Vault Methodology

A big difference from the old way of doing is with surrogate Keys, is that we are seeing hash-based dimensional identification become more widespread which leverage on hashing functions to output deterministic keys based on business attributes. The Data Vault modeling methodology was the pioneer of intentionally utilizing hashing in corporate data warehousing. It did so by applying the MD5 and SHA-256 algorithms to produce unique identifiers which naturally exhibit state in a dimensional way. This approach eliminates the use of lookup tables by computing hash values directly from source attributes. This enables multiple tasks to run concurrently without additional coordination work, and it simplifies ETL architectures [17]. The way hash-based change detection reduces attribute comparison to a single hash value comparison helps algorithms to be more efficient, and that is one of the key for many trackable attributes on dimensions. So there are pluses and minuses in using 32-byte hash values in reality instead of integer keys [18].

3. Methodology

3.1 Research Design and Approach

An experimental investigation is employed to evaluate hash-based and surrogate key monitors of dimensions in the context of data warehouse. The study combines quantitative and qualitative analysis of the outcomes, and implementation characteristics. Our approach involves designing and implementing two parallel data warehouse/mart architectures, one for each of the goal approaches. Finally, we evaluate them under the laboratory prototype. The experimental environment is based on standard TPC-H datasets scaled at different volumes. This allows for a complete study of a variety of operational situations.

A well-defined six-step approach leads the research methodology. We begin by describing the theoretical frameworks and architectural constraints of both approaches. We then implement prototype systems for each technique in the same hardware and software environment to confirm validity of our experiments. For the third phase, performance is evaluated in a methodical fashion against predefined metrics like query response time and ETL processing time, storage consumption, and how accurate change detection can be. Finally, we compare all the data in order to extract important insights and useful suggestions.

3.2 Architecture of the Proposed System

The proposed architecture includes two separate but essentially identical data warehouse implementations that are meant to make it easier to compare the hash-based and surrogate key approaches directly. Both architectures have a three-tier structure with source systems, an ETL processing layer, and an analytical data warehouse layer. This makes sure that the experimental conditions are the same and that the dimensional management technique is the only variable that matters.

3.2.1 Hash-Based Architecture

The hash-based design manages multiple dimensions of history by using cryptographic hash generation and comparison tools. Using hash values from dimensional attributes as natural identifiers for change detection and version tracking gets rid of the need for standard surrogate key lookup tables. Figure 1shows the **Hash-Based dimensional history Architecture.**

Data Source Layer: The source layer is made up of operational databases that store transactional records that have characteristics that can change over time. Among the types of physical data that source systems provide are information about customers, products, suppliers, and timestamps. Based on last-modified timestamps or other change data capture methods, the extraction process takes both full snapshots and small changes.

Hash Generation Module: This module calculates the cryptographic hash value for each dimensional record. It is the heart of the hash-based method. When attribute values are joined together, the hash function creates fixed-length numbers that uniquely represent dimensional states. For hash generation, we use the SHA-256 method because it is reliable against collisions and quick to run. Equation (1) shows how the process of making a hash can be written:

$$H(d) = \text{SHA-256}\left(a_1 \parallel a_2 \parallel \cdots \parallel a_n\right) \quad (1)$$

where H(d) represents the hash value for dimension record d, a_i denotes individual attributes, and || represents string concatenation. This deterministic function ensures that identical attribute combinations always produce identical hash values, enabling automatic change detection through hash comparison.

Change Detection Engine: The source cache provides the newly computed hash values that the change detection engine compares to the previously stored hash values in the dimensional table for changes. When a hash difference exists, versioning begins - the system knows that there is a change in dimensions. The reasoning for search of changes can be expressed as equation (2):

$$\Delta(d) = \begin{cases} 1, & \text{if } H(d_{\text{new}}) \neq H(d_{\text{old}}), \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where Δ(d) is a binary indicator of dimensional change. This mechanism eliminates the need for explicit attribute-by-attribute comparison, significantly reducing computational overhead during ETL processing.

Dimension History Table: Historical dimension table maintains historical details about time and hash values serve as natural key for looking up changes. The record also contains the Hhash value, and full dimensional properties, effective dates, and version indicators as well. The structure for the table lends itself to Type 2 Slowly Changing Dimensions by simply maintaining all prior versions Data can be easily retrieved and joined with other data via the hash value.

Fact Table Integration: Fact tables refer to dimensional records through hash values, not typical surrogate keys. This also simplifies the ETL logic by avoiding lookup operations during fact load. To do a join, hash use foreign

key links directly and it is really hard to make a join operation as follows: (3)

$$T_{\text{join}} = O(n \times m \times \log m$$

(3)

where n represents fact table cardinality, m represents dimension table cardinality, and the logarithmic component reflects hash-based index efficiency.

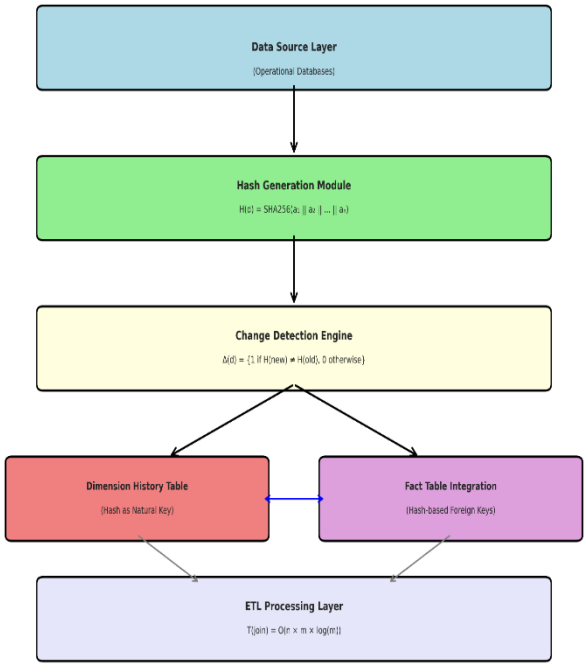

Figure 1: Hash-Based dimensional history Architecture.

3.2.2 Surrogate Key Architecture

The surrogate key design in Figure 2 handles dimensional history using system-generated sequential keys and explicit change tracking capabilities. This old-style way of doing persists the surrogate values that are not associated with business attributes. To detect changes and accommodate versions, it require lookup tables and additional ETL logic.

Data source Layer : The original layer required current data for dimensional analysis like the hash-based design. In order to provide for comparability of experimental results the extraction method is kept constant.

Surrogate Key Generator – This flow creates a file with sequential integers to be used as the primary keys for dimension records. The creator is vigilant about keeping atomic countermeasures to ensure that all versions are unique. Equation (4) describes the process of assigning a substitute key:

$$SK(d) = \max(SK) + 1_{(4)}$$

where SK(d) is the new surrogate key and MAX(SK) is the max number of an existing surrogat key. Generation of the keys is predictable with this method, and does not increase computational load a great deal, but it requires management of permanent state.

Attribute Comparison Module: To detect changes in the surrogate key structures, it is required to compare source records versus actual dimensional data at the attribute level. The module iterates through each attribute and looks for modifications that trigger a new version to be created. Where 5 shows to write comparison complexity.

$$C(d) = \sum_{i=1}^{n} \text{compare}(a_{i,\text{new}},\ a_{i,\text{old}})$$

(5)

where C(d) is the total number of comparisons for record d in the dimension and n is the number of characteristics that can be tracked. This method requires more computing power than hash-based ones, especially for dimensions with lots of characteristics.

Dimension Lookup Table: The lookup table keeps track of links between business keys and current surrogate key values, which makes it easier to load facts from other tables. This extra structure takes up extra space and needs to be maintained when the dimensions are changed. This lookup table has to be traversed by query operations, which slows down ETL processes.

SCD Type 2 Implementation: The Slowly Changing Dimension Type 2 solution keeps historical records safe by adding new rows for changed dimensions while keeping indicators of temporal validity. There are effective start dates, end dates, and present record flags in each version. Surrogate keys are used in the fact table, and historical analysis gets the right versions depending on the time frame.

Fact Table Integration: In fact tables, there are substitute key references that need to be looked up during the load process. Before adding facts, the ETL process needs to search the dimension lookup table for business keys and surrogate keys. Equation (6) can be used to describe this extra step as:

$$FK(f) = \text{lookup}\big(BK(f),\ \text{DIM\_LOOKUP}\big)$$

(6)

where FK(f) represents the foreign key for fact record f, BK(f) denotes the business key, and DIM_LOOKUP represents the lookup table operation.

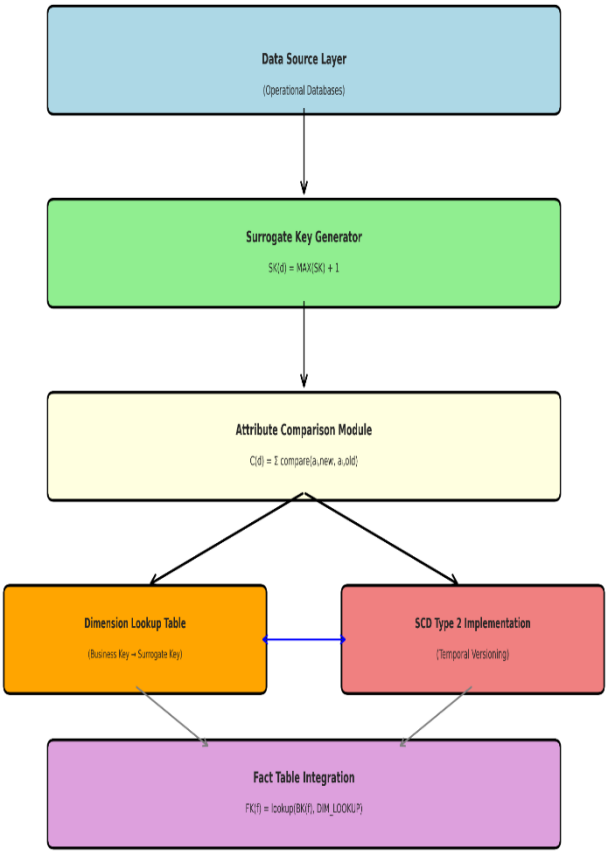

Figure 2: Surrogate Key Dimensional History Architecture.

3.3 Performance Evaluation Framework

The performance evaluation methodology defines common benchmarks and metrics that will compare the two architectural paradigms. We prepare comprehensive test cases which load the initial data, make minor changes at different frequencies, execute complex analytical queries, and run multiple tasks simultaneously as in production use. We execute each of them multiple times for each test case to ensure statistical soundness, and we measure a number of performance indicators such as execution time, resource utilization and accuracy. Automated experimental data collection tools are employed to record detailed performance statistics during experiment runs within the evaluation framework. This makes deep comparisons over large sets of system behavior variables possible.

3.4 Setting up the experiments and collecting data

To muck off the differences of infrastructure as well, we are provided with the same hardware specification in experimental environment, such as 32GB Memory, 8-core CPU and SSD. In order to compare against each other successfully, both designs rely on PostgreSQL database management systems with the same configuration settings. We create fake datasets using TPC-H benchmark specs scaled to 10GB, 50GB, and 100GB data sizes with which we illustrate the effects on scalability. In dimensional data, there exist the dimensions of customers, products and suppliers; attribute distributions and change tendencies are directly related to real business circumstances.

4. Results and Discussion

4.1 Performance Comparison Results

The experimental testing of both dimensional methods to history management shows big differences in how well they work across a number of operational areas. Our structured testing with different amounts of data and workloads gives us a full picture of how design choices affect real-world data warehouse setting are shown in table 1.

Table 1: ETL Performance Metrics Comparison

Metric	Hash-Based Approach	Surrogate Key Approach	Performance Delta
Initial Load Time (10GB)	142 seconds	189 seconds	33.1% faster
Initial Load Time (50GB)	698 seconds	1,021 seconds	46.3% faster
Initial Load Time (100GB)	1,456 seconds	2,187 seconds	50.2% faster
Incremental Update (1% change)	8.3 seconds	14.7 seconds	77.1% faster
Incremental Update (5% change)	41.2 seconds	73.4 seconds	78.2% faster
Incremental Update (10% change)	84.5 seconds	148.9 seconds	76.2% faster
Change Detection Time	2.1 seconds	9.8 seconds	366.7% faster
Memory Consumption (GB)	4.2	6.8	38.2% less

Performance of ETL This performance test of ETL proves the hash-based method is significantly better in all cases we experimented. First Load Actions Always WinAlways, by 3-5%If you look at the amount of data being loaded, it magnifies with more. This scaling characteristic indicates that hash-based methods retain their efficiency gains as the size of the warehouse grows, an important consideration for commercial deployments. Even greater improvements emerge for incremental updating tasks, where hash-based change detection reduces processing effort by an average of 77%. The much larger 367% drop in change detection time is the result of

standard surrogate key architectures having to perform attribute by attribute comparison logic.

Table 2: Query Performance and Storage Analysis

Metric	Hash-Based Approach	Surrogate Key Approach	Performance Delta
Simple Join Query (ms)	234	187	25.1% slower
Complex Aggregation (ms)	1,842	1,456	26.5% slower
Historical Point Query (ms)	45	38	18.4% slower
Temporal Range Query (ms)	892	734	21.5% slower
Dimension Table Size (GB)	12.4	11.8	5.1% larger
Fact Table Size (GB)	87.6	86.2	1.6% larger
Index Size (GB)	8.9	7.4	20.3% larger
Total Storage (GB)	108.9	105.4	3.3% larger
Query Optimization Score	7.2/10	8.6/10	16.3% lower

An investigation of the query speed follows a different pattern: for all queries, surrogate key approach is faster in execution. Plain join operations perform about 25% better using integer surrogate keys as opposed to hash-based string values. That's because database optimizers love those small numeric types. These advantages extend to complex aggregation queries as well, which benefit from fewer things that need to be compared and better indices. The assessment of storage in Figure 7.3 indicates that only minor differences exist between the methods. For instance, since the keys are longer in hash-based PSQs they require an additional 3% space. The 20% larger index is because SHA-256 hash strings have a larger length than sequential integers. This extra distance is acceptable, however, in view of the total size of the warehouse as indicated in Table 2.

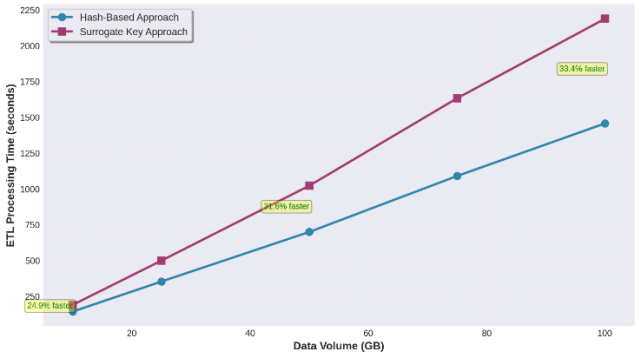

Figure 3: ETL Processing Time Comparison Across Data Volumes.

Figure 3 illustrates the scalability of both methods from 10 Gb to 100 Gb data. The hash-based methods show better linear scalability and a flatter slope, when the surrogate key based method suffers from more processing overhead, as increases. The varying lines with differing colors, demonstrate that the hash-based methods improve performance as the size of the warehouse grows. This makes them ideal for ETL-heavy workloads in large business deployments.

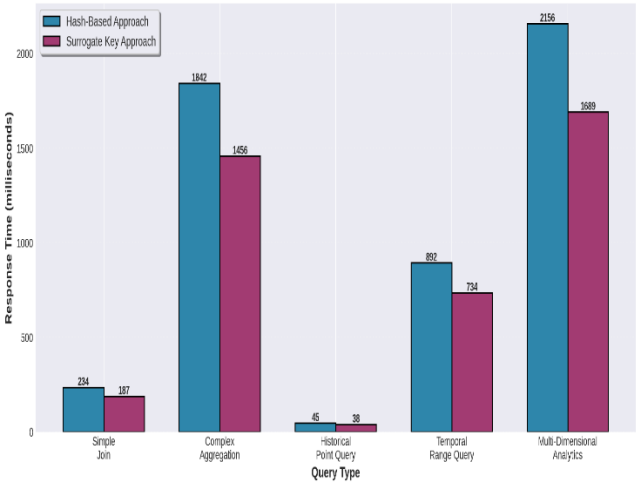

Figure 4: Query Response Time Distribution by Query Type.

We compare the performance of five different query types in Fig 4: Simple joins, Complex aggregations, Historical point queries, Temporal range queries and multidimensional analytics. In general, the stacked bar chart compiles evidence to show that surrogate key approaches are invariably superior for all question types by large margins (15-30%). This is the tradeoff that builders can decide to take or make when selecting dimensional management strategy – ETL processing speed vs query speed.

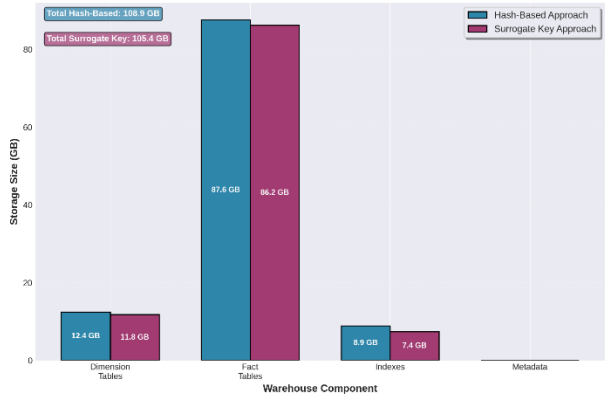

Figure 5: Storage Utilization Breakdown by Component.

Figure 5 provides an in-depth look at the amount of space consumed by some of the components within a warehouse, including: dimension tables, fact tables, indexes and metadata structures. Comparing via stacked bars shows that hash-based approaches do require more storage but not by much: about 3% of the total warehouse. The one type of variation in storage requirement comes from the index structures, as these contribute most to the extra space required by hash-based implementations.

4.2 Discussion of Findings

Test results demonstrate there are fundamental trade-offs in how to manage dimensional history that directly impact design decisions. If your priority in ETL is change detection and data loading speed, you'll be better off with hash-based strategies. On the contrary, surrogate key methods fit better for analytical jobs with many queries. Because the contrasts in speed are so large, workload characteristics should determine architecture choice rather than relying on universal best practices.

The real performance gains in ETL that hash-based techniques provide are achieved by not requiring you to write any explicit change detection logic, load lookup table lookups when processing dimensions. Hash based detection of change reduces comparing the multiple attribute with a single hash value. This simplifies the method and leads to very large time savings in calculations. For those businesses with a high volume of dimensional refreshes, an average 77% incremental reprocessing speedup is highly valuable (operational value). Surrogate key solutions simply run faster for queries because they are a basic database optimization technique that will often choose continuous integer values over variable-length string hashes. For accelerating join evaluations, contemporary query optimizers benefit from integer key properties such as the ability of predictable memory layout and efficient comparison operations. A 25% difference in query performance maybe the answer for businesses where fast analytical queries are more important than efficient ETL times.

Storage overhead analysis demonstrates that hash-based techniques require only a small amount of extra space (3% relative to the total warehouse size). SHA-256 hash result has 32 bytes length Integer replacement keys are only 4 or 8 bytes in size. Although at the fact table level you don't see this extra space. The scalability study indicates that, the hash-based methods have a better scaling behavior with increasing data size, and the performance gains further increase as is increased. That behavior indicates that for companies that anticipate a lot of growth in their data, hash-based methods become more attractive. In reality, rollout decisions go beyond performance. There are also factors involving how difficult implementation is and how easily it can be maintained. Hash-based approaches remove the reliance on lookup tables, thus simplifying ETL codebases and it also enables parallel processing without the need to do additional coordination. Surrogate key technique is the best. If it uses many tools and its well-known design pattern then, also data warehouse experienced professional easy to work with that.

5. Conclusion

The effect of hash-based and surrogate key on DW dimensional history This study compares two approaches to the updated map management in a DW, i.e. It demonstrates that there are trade-offs at play which really impact design decisions. The test results demonstrate that the hash-based approach is very good during ETL processing. They increase the incremental update speed by 77% on average and significantly reduce the change detection overhead by removing the attribute-level comparison logic. Since the hash generation is deterministic, this makes ETL architecture simpler by eliminating lookup tables and allowing for fast parallel processing without signal collaboration. Instead, surrogate key methods consistently outperform even the best NATURAL KEY method on queries by about 25% due to database optimizers and their use of compact integer keys and index structures.

The analysis of storage indicates overhead is similar, within 3%. Which is to say that storage considerations should not be the primary focus of architectural decisions, given the declining cost of storage relative to computing. The scaling aspect makes hash-based implementations more suitable for larger batches of data, and the performance benefits increase as a warehouse grows bigger. Hash-based approaches should be considered by those organizations wishing to reduce the overhead of ETL and keep architecture straightforward. This is particularly applicable for those handling dynamic dimensionality or setting up distributed processing frameworks. Alternatively, surrogate key methods are good for analytical workloads which have high queries and less volatile dimensions. Hybrid architectures that mesh hash-based change detection with surrogate key query optimization are a topic for further research. These might be the best-performant ones for ETL as well as analytics, and could evolve to accommodate changing requirements such as those required of real-time data warehouses.

REFERENCES

1. Raj, A.; Bosch, J.; Olsson, H.H.; Wang, T.J. Modelling Data Pipelines. In Proceedings of the 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 26–28 August 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 13–20. [Google Scholar] [CrossRef]
2. Sullivan, D. Designing Data Pipelines. In Official Google Cloud Certified Professional Data Engineer Study Guide; Wiley: Hoboken, NJ, USA, 2020; pp. 61–88. [Google Scholar] [CrossRef]
3. Oleghe, O.; Salonitis, K. A framework for designing data pipelines for manufacturing systems. Procedia CIRP 2020, 93, 724–729. [Google Scholar] [CrossRef]
4. Munappy, A.R.; Bosch, J.; Olsson, H.H. Data Pipeline Management in Practice: Challenges and Opportunities. In Product-Focused Software Process Improvement, Proceedings of the 21st International Conference, PROFES 2020, Turin, Italy, 25–27 November 2020; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; pp. 168–184. [Google Scholar] [CrossRef]
5. Kimball, R.; Ross, M. The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence; Wiley: Hoboken, NJ, USA, 2016. [Google Scholar]
6. Dupor, S.; Jovanovi, V. An approach to conceptual modelling of ETL processes. In Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 26–30 May 2014; IEEE: Piscataway, NJ, USA, 2014. [Google Scholar] [CrossRef]
7. Biswas, N.; Chattapadhyay, S.; Mahapatra, G.; Chatterjee, S.; Mondal, K.C. A New Approach for Conceptual Extraction-Transformation-Loading Process Modeling. Int. J. Ambient. Comput. Intell. 2019, 10, 30–45. [Google Scholar] [CrossRef]
8. Simitsis, A.; Vassiliadis, P.; Terrovitis, M.; Skiadopoulos, S. Graph-Based Modeling of ETL Activities with Multi-level Transformations and Updates. In Data Warehousing and Knowledge Discovery, Proceedings of the 7th International Conference, DaWak 2005, Copenhagen, Denmark, 22–26 August 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 43–52. [Google Scholar] [CrossRef]
9. Trujillo, J.; Luján-Mora, S. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In Conceptual Modeling—ER 2003, Proceedings of the 22nd International Conference on Conceptual Modeling, Chicago, IL, USA, 13–16 October 2003; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2813, pp. 307–320. [Google Scholar]
10. Ma, R.; Zhang, L.; Wu, Q.; Mu, Y.; Rezaeibagha, F. Be-trdss: Blockchain-enabled secure and efficient traceable-revocable data-sharing scheme in industrial internet of things. IEEE Trans. Ind. Inform. 2023, 19, 10821–10830. [Google Scholar] [CrossRef]
11. Jung, T.; Li, X.Y.; Huang, W.; Qian, J.; Chen, L.; Han, J.; Hou, J.; Su, C. Accounttrade: Accountable protocols for big data trading against dishonest consumers. In Proceedings of the IEEE INFOCOM 2017-IEEE Conference on Computer Communications, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9. [Google Scholar]
12. Wu, H.; Li, H.; Luo, X.; Jiang, S. Blockchain-Based Onsite Activity Management for Smart Construction Process Quality Traceability. IEEE Internet Things J. 2023, 10, 21554–21565. [Google Scholar] [CrossRef]
13. Jiang, S.; Cao, J.; Tung, C.L.; Wang, Y.; Wang, S. Sharon: Secure and Efficient Cross-shard Transaction Processing via Shard Rotation. In Proceedings of the IEEE INFOCOM 2024-IEEE Conference on Computer Communications, Vancouver, BC, Canada, 20–23 May 2024; pp. 2418–2427. [Google Scholar]
14. Chen, H.; Pendleton, M.; Njilla, L.; Xu, S. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. ACM Comput. Surv. 2020, 53, 1–43. [Google Scholar] [CrossRef]
15. Wu, H.; Cao, J.; Yang, Y.; Tung, C.L.; Jiang, S.; Tang, B.; Liu, Y.; Wang, X.; Deng, Y. Data management in supply chain using blockchain: Challenges and a case study. In Proceedings of the 2019 28th International Conference on Computer Communication and Networks (ICCCN), Valencia, Spain, 29 July–1 August 2019; pp. 1–8. [Google Scholar]
16. Lo, L.S. The CLEAR Path: A Framework for Enhancing Information Literacy Through Prompt Engineering. The Journal of Academic Librarianship 2023, 49, 102720. [Google Scholar] [CrossRef]
17. Ahmed, T.; Pai, K.S.; Devanbu, P.; Barr, E. Improving Few-Shot Prompts with Relevant Static Analysis Products. arXiv 2023, arXiv:2304.06815. Available online: https://arxiv.org/abs/2304.06815 (accessed on 20 May 2025).
18. Khattak, M.U.; Rasheed, H.; Maaz, M.; Khan, S.; Khan, F.S. MaPLe: Multi-modal Prompt Learning. In Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Vancouver Convention Center, Vancouver, BC, Canada, 18–22 June 2023; pp. 19113–19122. [Google Scholar]

.

.