EPH - International Journal of Business & Management Science
ISSN (Online): 2208-2190
Volume 05 Issue 03 June 2019
DOI: https://doi.org/10.53555/eijbms.v5i3.161

WORKLOAD QUALIFICATION FRAMEWORK FOR DATA STORES

Jatin Pal Singh

*[*]**Corresponding Author: Jatin Pal Singh***

Abstract

Data Store migration involves data transfer among data storage systems, data formats &/or computer systems. Data Store migration is often considered a difficult subject as it requires careful adaptation of data structures & data models that are appropriate with the target data store without affecting the functionality of application, & correctness and integrity of data. With rapid evolution of cloud services, migrating from legacy data stores to cloud native, or open-source data stores is gaining wide attention. However, there is no standardized approach which addresses the problem of workload categorization, & simplifying the choice of target data stores. This paper presents a comprehensive workload qualification framework & shares an example of choosing between relational & non-relational (key value) data stores. The paper addresses the pervasive challenges and lack of standardization in migrating traditional data stores to open source, cloud native data stores by proposing a unified framework. This framework encompasses algorithm migration, model migration, and migration schemes, tailored to enhance the data migration process across various environments. Through comparative analysis and integration of existing frameworks, it aims to offer a robust solution that aids organizations and developers in navigating the complexities of data migration, thereby contributing to more efficient and effective database management and transition strategies.

Keywords: *workload qualification, data migration, framework, migration, data migration framework*

INTRODUCTION:

Over the last decade, innovation has become a central theme for enterprises. With rapid advancement in technology, public cloud providers have offered services which make it easier for enterprises to innovate their businesses. Enterprises are no longer dependent on legacy software vendors, to innovate & invest on their behalf. This opens up a very competitive environment of 'open source' adoption, & choice of services to cater to specific business. One such scenario is in Data Stores, which ties back to large investment, & very hard to modernize. With technology advancement, enterprises & customers are no longer limited to legacy providers, as they have hundreds of cloud-based Data Stores to choose from. However, there is a lack of a standard methodology to Data Store migration & modernization. The research presented in this paper delves into two aspects of Data Store migration **1)** workload qualification process and **2)** workload migration process, keeping in context a relational (aka RDBMS) and non-relational (aka NoSQL) datastores. In the first part, this research focuses on determining the strategy for your data store, by providing a framework to easily understand the current database workload. It uses relational data stores, which are most common use cases for source, to examine which features you are currently using and what's involved in migrating to other cloud-native data stores. In the second part, this research focuses on addressing the challenges inherent in the structured approach of relational (RDBMS) data stores, as well as the diverse and schema-less nature of non-relational (NoSQL) data stores. For NoSQL, we limit the research scope to Key Value store, & compare it with relational data store. By integrating algorithm migration, model migration, and migration schemes, this study aims to standardize and simplify the process of data migration, ensuring the integrity and effectiveness of data transfer across various database technologies and structures, thereby offering a universal solution to data migration challenges.

Workload Qualification Framework:

The best **migration strategy** for your traditional data store is correlated with its workload & functionality. The first step is often to **analyze** your business requirements to determine what the business application's future is. In some cases, the migration strategy might simply be 'lift & shift' to cloud, as your business might be looking at simply exiting a data center. While, in some other cases, you may be required to dive into specific business requirements like - cost reduction, resilience, scaling etc. In enterprises, often Infrastructure teams are incharge of data store, however, these migrations should not be carried out without a close tie with business & application teams. Once you align, it's time to take a look at the overall data store migration lifecycle, as shown:

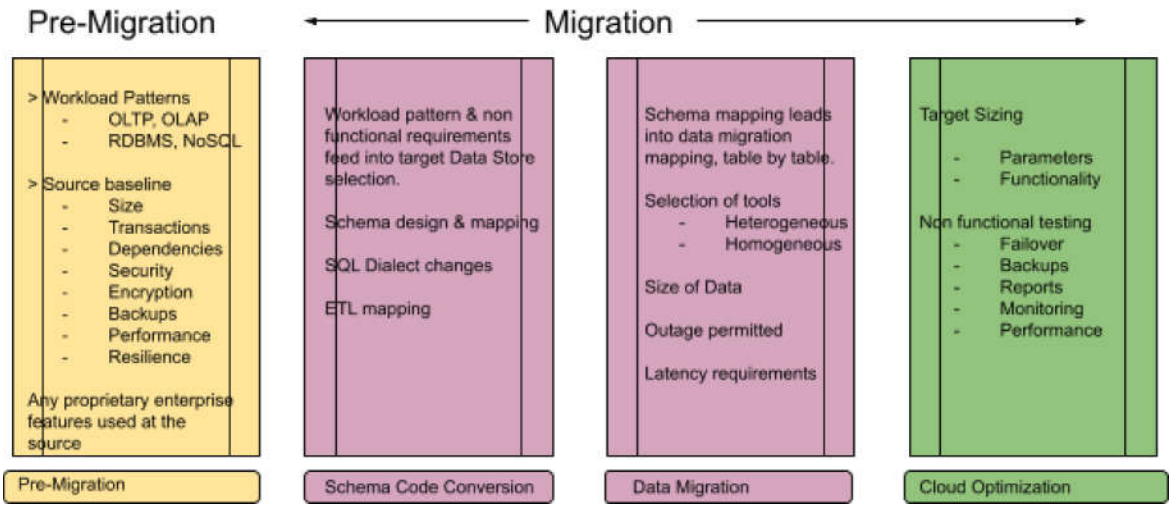


Fig 1: Data Store migration life cycle:

In the above lifecycle, the analysis phase typically consists of a business value discussion to agree on an outcome. Often, the outcomes can be categorized into - functional or non-functional in nature. For **functional** outcomes, the migration is often categorized as modernization, as it involves application functionality re-writes, & follows a typical software lifecycle. **Non-functional** requirements though can be addressed during the migration, such as, scalability, resilience, security, performance, or cost reduction. It's important that each of these non-functional requirements (NFRs) be 'quantified' before we start categorizing & analyzing the workload. However, the question remains, why do we need a workload qualification framework? A Workload Qualification Framework is an essential strategy in database migrations, especially when moving to cloud environments or transitioning between different database technologies. It's a structured approach to evaluating and categorizing various aspects of existing databases and their workloads to ensure a smooth, efficient, and successful migration. In context of data store migrations, here are some aspects which we use to develop a framework (& later propose two approaches):

Components :

Assessment of Source Environment: Documenting the current database landscape, including database size, type, version, and configuration. Understanding the data schema, interdependencies, and any custom features or stored

procedures used. **Workload Analysis:** Profiling workloads to understand query patterns, peak usage times, transaction rates, and read/write ratios. Categorizing workloads into types, such as OLTP, OLAP, batch processing, or real-time analytics. **Performance and Cost Metrics:** Benchmarking current performance metrics to set baseline expectations for the target environment. Estimating the cost of migration and ongoing operations in the new environment.
Evaluation Criteria: Evaluation criteria is a key component of workload qualification & must include - **Compatibility:** Ensuring that features and functions used in the source database are supported in the target environment. **Performance:** Determining if the target environment can meet or exceed current performance levels. **Scalability:** Assessing the ability of the target environment to scale with future growth. **Security and Compliance:** Verifying that the target environment meets all necessary security and regulatory requirements. **Cost:** Evaluating total cost implications, including migration costs, operational expenses, and potential savings.
Migration Strategy Development: Based on the assessment, determining the most appropriate migration method (e.g., retire, rehosting, refactoring, re-platforming, or rebuilding). Planning phased migrations for complex environments, starting with less critical or simpler workloads.
Pilot and Testing: Conducting a pilot migration with a non-critical or representative workload to validate the migration approach. Performing extensive testing in the target environment to ensure functionality, performance, and stability.
Execution and Optimization: Executing the migration according to the planned strategy, monitoring closely for any issues. Post-migration, optimizing the workloads for the new environment, which might include tuning queries, restructuring indexes, or leveraging new features of the target database.
Documentation and Continuous Improvement: Documenting lessons learned, performance improvements, and any issues encountered. Refining the Workload Qualification Framework based on experience to improve future migrations.
As you can understand, Workload Qualification Framework is a detailed and systematic approach that helps organizations understand their database workloads comprehensively and make informed decisions about migration strategies. It emphasizes the importance of preparation, assessment, and continual optimization, ensuring that database migrations enhance performance, reduce costs, and align with business objectives.
Now, let's take a look at some approaches for workload qualification in a relational data store, which is most popular data store:

Approach 01:
- **Category 1:** Workloads that use Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) instead of proprietary drivers to connect to the database. This category typically has simple stored procedures that are used for access controls. The conversion requires fewer than 50 manual changes.
- **Category 2:** Workloads with light use of proprietary features and that don't use advanced SQL language features. This type of workload requires fewer than 200 manual changes.
- **Category 3:** Workloads with heavy use of proprietary features. Workloads in this category are completely driven by advanced stored procedure logic or proprietary features. This type of workload requires more than 200 manual changes that involve database-resident code and features.
- **Category 4:** Engine-specific workloads. Workloads in this category use frameworks that can work only with a specific commercial database engine. For example, these frameworks might include ERP applications, Oracle Forms, Oracle Reports, Oracle Application Express (APEX), or applications that use .NET ActiveRecord extensively.
- **Category 5:** Nonportable, unacceptable risk, or "lift and shift" workloads. Workloads in this category might be implemented on database engines that have no cloud-based equivalent. In some cases, you might not have the source code for these programs.

Category	Type	Description
Category 1	Simple java application, ODBC/JBDC workloads	Business logic is in application
Category 2	Light, proprietary feature workloads	Some business logic is in Data Store, ex: PL/SQL, PL/PgSQL
Category 3	Heavy, proprietary feature workloads	Heavy data processing application, with large functionality in data store, typically hundreds of PL/SQL, PL/PgSQL.
Category 4	Engine specific workloads	Package applications, with 3rd party Independent Software Vendors.
Category 5	Non portable, highly dependant	Legacy software

Approach 02:
A different way to classify your data store's workload can simply be based on its schema's characteristics. A data store's schema can be characterized with - number of objects, size of data, object relational mapping layer, business logic layer, tools used for reporting (ETL), &/or developer's access on code base as well.
This approach is more comprehensive than the previous approach, as it considers factors immediately outside of 'schema' as well. In this approach, you can also assign 'weightage' & complexity to type of schema objects to formulate a numerical approach, for example:

Factor	Weightage	No. of Objects	Comments
Table	0.5	40	Simple
Indexes	0.5	80	Simple
Materialized Views	4	10	Complex
Database Links	2	2	Foreign Data Wrapping
Large Objects	5	5	Complex - Large Objects need special attention
Functions	12	10	Complex - Functions need to be ported, re-written

Migration Strategy - Choosing the Data Store

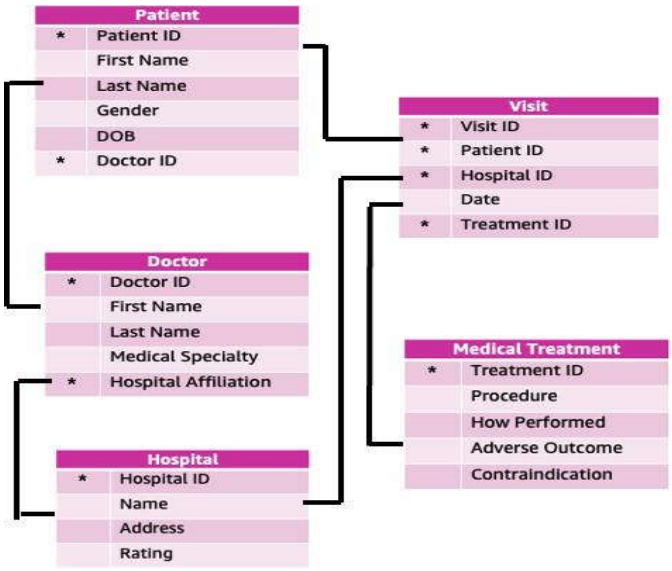
Once the workload qualification is complete, we have a general sense of complexity related to the migration. However, without the knowledge of target data stores, it's not possible, & often inaccurate to assess how you can meet certain non-functional requirements based on application's specific use case. For example, while you qualify workload as category 1 (simple), you may still be required to meet the business requirements of - scalability, or flexible schema, or handling large volume of data, or reduced latency, or fast development speed, which the target system may or may not provide. Here's a general rule to making an informed choice:

1. **Understand Your Data and Workload:**
   - Assess the nature, size, and complexity of your data. Understand if it's structured, semi-structured, or unstructured.
   - Analyze your data access patterns. Consider the types of queries, read/write ratio, and transactional requirements.
2. **Consider Scalability and Performance:**
   - Determine your scalability needs. If you expect rapid growth or large volumes of data, a NoSQL database might be more suitable.
   - Consider the performance of different data stores for your specific workload. Benchmark if necessary.
3. **Evaluate Consistency and Availability Requirements:**
   - Understand the CAP theorem implications for your application. Decide what's more critical: consistency, availability, or partition tolerance.
   - Ensure the database can meet your business's SLAs for uptime and data accuracy.
4. **Assess Data Modeling and Development Implications:**
   - Consider how the data model of the database aligns with your application's needs. Relational databases enforce a structured data model, while NoSQL databases allow for more flexibility.
   - Evaluate the impact on development time and complexity. A database that's easier to work with can significantly speed up development.
5. **Review Maintenance and Operational Aspects:**
   - Look into the operational overhead. Consider aspects like backup, recovery, monitoring, and updates.
   - Check if the database requires specialized skills or if it's well-supported by the community or vendor.
6. **Examine Costs:**
   - Consider both upfront and ongoing costs, including hardware, software, and personnel.
   - Look for hidden costs like licensing fees or costs associated with scaling.
7. **Regulatory Compliance and Security:**
   - Ensure the data store complies with relevant regulations like GDPR, HIPAA, etc.
   - Assess the security features of the database. Consider encryption, access controls, and auditing capabilities.
8. **Compatibility and Integration:**
   - Ensure the new database is compatible with your existing technology stack.
   - Consider the ease of integration with other systems and services you use.
9. **Vendor Support and Community:**
   - Consider the level of support provided by the vendor or the community around the database. Look for documentation, forums, and professional services.
   - Evaluate the longevity and stability of the database technology.

By carefully considering these aspects, you can choose a data store that not only meets your current needs but also supports your application's growth and evolution. Remember, the best choice often involves trade-offs, and what's right for one application may not be suitable for another. Therefore, it's crucial to align the database's characteristics with your specific business requirements and technical context. With this, let's take a look at the popular data store styles:

Relational Stores:

A relational database is a type of database that organizes data into tables, or "relations," consisting of rows and columns. Each row represents a unique record, and each column represents a field in the record. The key feature of relational databases is their ability to efficiently retrieve and manipulate related data across different tables. They use Structured Query Language (SQL) for querying and maintaining the database. Relational databases are known for their reliability, scalability, and flexibility, making them suitable for a wide range of applications. Examples include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. A typical schema & relations are defined as shown:



As such, if you need to lookup with relations, for ex:, number of patient visits each doctor completed last week, you will use queries, with inter-related tables like:

```
SELECT FROM
WHERE
d.first_name, d.last_name, count(*) visit as v,
hospital as h, doctor as d

v.hospital_id = h.hospital_id AND h.hospital_id = d.hospital
AND v.t_date > date_trunc('week',  CURRENT_TIMESTAMP - interval '1 week')
GROUP BY
d.first_name, d.last_name;
```

Pay attention here to the query constructions, as the migration, & choice of data store will be directly determined by the application use case. While there is a great collection of NOSQL stores, the paper doesn't propose one or the other Data Store, but proposes to choose the right Data Store for your application use case. For example, you should consider choosing a relational data store during migration if you have following requirement:

1. **Structured Data Requirements:** Your application strictly requires structured data with a fixed schema. Relational databases are ideal for handling structured data and ensuring that all data adheres to a predefined schema.
2. **Complex Queries and Joins:** Your application needs to perform complex queries, joins, or transactions. Relational databases excel in handling complex queries and ensuring data integrity, especially when the relationships between different data items are complex and need to be precisely managed.
3. **ACID Transactions:** Your application requires strong ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure transactions are processed reliably. Relational databases provide robust transaction support, ensuring that all database transactions are processed reliably and safely.
4. **Data Integrity and Normalization:** You need to enforce data integrity constraints such as foreign keys, unique constraints, or check constraints. Relational databases allow for extensive data integrity rules and normalization techniques to avoid data redundancy and maintain data accuracy.
5. **Reporting and Analysis:** Your application requires extensive reporting and analysis capabilities. Relational databases often provide powerful tools and capabilities for data analysis and reporting, which might be more complex or less efficient in non-relational databases.
6. **Mature Ecosystem and Tools:** You prefer to use a mature ecosystem with well-established tools, support, and community. Relational databases have been around for decades, offering a stable, reliable, and well-understood technology with a wide range of tools for backup, monitoring, performance tuning, and more.

Key Value Stores:

Key-value stores are a type of non-relational database that organize data into a simple key-value pair format. They are designed to be highly scalable, providing a way to store and retrieve vast amounts of data quickly and easily. In key-value stores, data is represented as a collection of key-value pairs, where each key is unique and is used to retrieve the corresponding value. The Key Value store's model is very simple; every single item is stored as a key together with its value. This simplicity can lead to high performance and scalability. Key-value stores often allow for efficient data retrieval and writing as data is accessed via a simple key. This can be particularly beneficial for applications requiring high-speed lookups and minimal latency. They often allow the value to be anything, from simple text or numbers to complex serialized objects. This means you can store a wide variety of data types. Also, Key-value stores can be easily distributed

across clusters and networks, making them inherently scalable. Many are designed to scale out horizontally, providing a way to increase capacity by adding more servers.

Common Uses of Key-Value Stores:

1. **Session Storage:** They are widely used for session management in web applications. The key can be a session identifier, and the value is the user's session data.
2. **Caching:** Key-value stores are ideal for caching frequently accessed data. The fast read and write capabilities make them suitable for caching web pages, API calls, and computational results.
3. **User Profiles & Settings:** Storing user profiles or preferences where each user can be associated with a key, and their profile or preferences can be stored as the value.
4. **Real-time Recommendations:** Key-value stores can be used to quickly retrieve user preferences or past behavior to compute real-time recommendations.

Let's take a look at a use case, consider a user profile table as below:

UserID	Name	Email	Age
1	Alice	alice@example.com	28
2	Bob	bob@example.com	35
3	Charlie	charlie@example.com	40

This table represents user profiles in a relational database with columns for user ID, name, email, and age. Let's see what it takes to p**ort** this to a key value store. In a key-value store, each row of the table can be represented as a key-value pair where the key is the unique identifier (UserID in this case) and the value is the rest of the information associated with the user. The value can be stored as a serialized object, such as JSON, allowing you to retrieve all user information using the key.

Here's how you might represent the 'UserProfiles' table in a key-value store:

1. **Key**: "user:1", **Value**: {"Name": "Alice", "Email": "alice@example.com", "Age": 28}
2. **Key**: "user:2", **Value**: {"Name": "Bob", "Email": "bob@example.com", "Age": 35}
3. **Key**: "user:3", **Value**: {"Name": "Charlie", "Email": "charlie@example.com", "Age": 40}

In this key-value representation:

- Each key is a unique identifier constructed using a pattern like "user:{UserID}". This ensures that the keys are unique and provides a way to access a specific user's information directly.
- Each value is a serialized JSON object containing the name, email, and age of the user. JSON is chosen for its flexibility and ease of use, but other serialization formats can also be used depending on the needs and capabilities of the key-value store being used.

This approach allows you to quickly retrieve all information about a user with a single key lookup in the key-value store, which can be much faster than querying a relational database, especially when the system is designed to scale horizontally across multiple servers or nodes. However, note that while this approach is highly efficient for certain types of access patterns (like retrieving all information about a single user), it may not support other types of queries as easily as a relational database (like finding all users over a certain age). Careful consideration of the access patterns and requirements of your application is essential when deciding how to model your data in a key-value store.

Considerations for Key-Value Stores:

- **Schema Flexibility:** They usually don't enforce a particular schema on the data, which means the structure of data can vary from one key to another. This can be both an advantage and a disadvantage depending on the application requirements.
- **Scalability vs. Complexity:** While key-value stores are designed for scalability, ensuring consistent performance as the system scales can be complex and might require careful design and tuning.
- **Lack of Complex Querying:** They generally don't offer the complex querying capabilities found in relational databases. If your application requires complex queries, joins, or transactions, a key-value store might not be the best choice.

Conclusion:

This paper presents a comprehensive Workload Qualification Framework for Data Store migration, addressing the pervasive challenges of migrating traditional data stores to open-source, cloud-native data stores. It offers a systematic approach to workload categorization and simplification in choosing between relational and non-relational data stores. The paper outlines the importance of careful adaptation of data structures and models in migration, proposes unified

frameworks for algorithm, model migration, and migration schemes, and incorporates comparative analysis for effective database management and transition strategies. Through this framework, organizations and developers are equipped to navigate complexities in data migration, ensuring integrity and efficiency in the transfer process across various database technologies

References

[1]. Abdelsalam Maatuk et al, A Framework for Relational Database Migration
[2]. Yansyah Saputra Wijaya et al, A Framework for Data Migration Between Different Datastore of NoSQL Database
[3]. Leonardo Rocha et al, A Framework for Migrating Relational Datasets to NoSQL