Chapter 7
IT Infrastructure for Smart City: Issues and Challenges in Migration from Relational to NoSQL Databases

Amit Kanojia and S. Tanwani

Abstract Smart city applications collect massive amount of data from various types of IoT sensors, engines, and people. Most of the data generated are heterogeneous in nature. However, this data need to be integrated with legacy applications based on SQL. Some of these applications also require migration to NoSQL for improved performance and fault tolerance. This paper addresses challenges of working in hybrid environments and migration issues from SQL to NoSQL databases. Rapid rate of growth in heterogeneous data and characteristics of NoSQL database like easy scalability, high availability, high performance, and low cost are the motivating factors to migrate toward NoSQL from relational databases, especially for applications requiring dealing with unstructured data. NoSQL database gives dynamic schema, adjustable data model, scale out architecture, and allows storage of big data and access to it in an efficient manner. The relational databases store data in form of tables with fixed schema. Relational databases are structured and not capable of handling unstructured and big data. In relational databases, because of normalization, data is spread across multiple tables and expensive join operation is required to integrate data. Due to limitations of relational databases, most of the leading organizations are migrating toward NoSQL. NoSQL databases are analyzed into four categories- Key-value database, Column-oriented database, Document-based database, and Graph database. Key research issues and challenges in migrating from relational to NoSQL include model transformation (including mapping and schema conversion), application integration, strategies related to perform join in different scenarios, use of indexes, storage issues, etc. Massive growth is likely in the area of NoSQL over cloud because of initiatives like Smart City, however, these applications required integration with legacy applications built over SQL. Thus need for migration as well as bridges for connection leading to requirements of our research area. This chapter tends to explore the comparative study of Relational databases and NoSQL

A. Kanojia (✉)
Department of Computer Science, Mata Jijabai Government Girls PG College, Indore, India
e-mail: kamit.ind@gmail.com

S. Tanwani
School of Computer Science & IT, DAVV, Indore, India

V. Bali et al. (eds.), *Decision Analytics for Sustainable Development in Smart Society 5.0*,
Asset Analytics, https://doi.org/10.1007/978-981-19-1689-2_7

databases, classification of NoSQL databases, case study, popular approaches for migration, and some of the key issues and challenges in migrating from Relational to NoSQL databases. This chapter will be fruitful for students and researchers too.

7.1 Introduction

A smart city comprises a major amount and variety of IoT sensors, engines, and people. A smart city endeavor to upgrade everyday operation. The Smart City idea is used to improve various metropolitan services with the use of ICT. In the Smart City, ICTs are used for improving the consistency, performance, and interaction of urban services. It reduces cost and resource utilization. Numerous smart city projects have already been initiated with new ideas, like smart health care, smart transportation, smart waste management, smart parking management, etc [1]. A smart city relies on data and uses large amount of data based on big data collected by IoT devices everywhere. Data generated in such type of applications are unstructured and heterogeneous, which cannot be managed effectively by legacy SQL applications. Thus, there is a need of migration from SQL to NoSQL and integration with legacy applications developed using SQL for sustainable development in smart society. Smart city projects use different ICT tools to make communications effective and innovative [2].

In this section, basic ideas of relational databases, NoSQL databases and their characteristics have been introduced. Carlo Strozzi in year 1998 coined the term NoSQL. NoSQL means "Not Only SQL". The huge amount of heterogeneous information generated from the web applications and business applications is difficult to deal with by RDBMS, which basically manages structured data. An alternative data model handling both structured and unstructured data was felt by business community. The demand led to development of NoSQL. NoSQL data sets are characterized as disseminated, evenly versatile, and open source [3].

RDBMS possess static schema and data are inserted strictly according to schema due to which data accessing rate becomes slower and difficult. RDBMS remains the top data storage technology, however, several industries wish to decrease operating cost to make scalable application that uses cloud computing technologies [4]. NoSQL databases do not have predefined schema that helps in making major application transformations in real time. It makes faster enhancement and more feasible to access data easily. NoSQL systems are high performance, scalable systems [5]. Various reasons for motivation to utilize NoSQL databases are high availability, scalability, distributed architecture, varied data structure, adaptable schema, and consistency and fault tolerance. NOSQL databases have many benefits such as faster read and write access and support massive storage at low cost. The scaling approach easily supports adding multiple servers. Data structure used by NoSQL is in form of key-value pairs [6].

NoSQL has conquered many restrictions of the relational databases [7]. Considering the above, many business applications presently on RDBMS technology intend to migrate over NoSQL. Therefore, in order to get a consistent and effective alternative to change the schema and migrate data to NoSQL from the legacy relational database applications, effective strategies and guidelines are required. The RDBMS is based on structured data model and uses structured query language (SQL). NoSQL offers a adaptable structure and doesn't adhere to standard SQL interface [3]. NoSQL databases are grouped into four most important categories (i) Key-Value Databases (ii) Document-oriented databases (iii) Column-oriented databases and (iv) Graph-oriented databases. Relational Databases follow ACID property whereas CAP theorem is used by NoSQL databases.

Researchers and industrialists exploring and implementing big data provide better opportunity to transform data into an appropriate form. They store and process the data and finally analyze it. With the rapid increase of data, issues occur about the organization, querying, and storage. Relational database management systems have limitations when data is accessible in different formats (such as semi-structured and unstructured data). NoSQL database system provides a streamlined database, supports multiple data models, and provides a schema-free architecture.

Data migration is the technique of relocating data between systems or storage units. For making this efficient, the data extraction, transformation, and loading methods should be implemented. The techniques will help in mapping data from existing system to the latest system which is being implemented. Database migration is the way toward moving the business logic, schema, physical data, and database dependencies from an existing system to a latest system. Database Migration is utilized when there is a necessity to move one database vendor to other. There are various reasons for this, such as cost, capabilities, functionalities, and requirements. Generally data migration occurs during an upgrade of existing hardware or transferring to a totally new system [6].

The rest of the chapter is organized into eight sections. After introducing the main concepts of relational databases and migration from relational to NoSQL database we proceeded further to discuss the related work in Sect. 2. Section 3 describes smart city requirements, Sect. 4 includes a comparative study of relational and NoSQL databases. Section 5 describes the classification of NoSQL databases. Various migration approaches with some examples have been discussed in Sect. 6. The challenges and some of the key issues are described in Sect. 7. Section 8 illustrates case study on MongoDB database. Section 9 concludes the chapter and outlines future work.

7.2 Related Work

The most proficient strategy to migrate a database relies upon the application availability and the computational resources accessible for it [8]. The technique also helps to choose a correct migration strategy to utilize and the metrics that can be gathered to more readily assess the exhibition of a migration. The paper not only

recognizes the current migration methods but also recommends receiving the most effective strategy for migrating relational databases to NoSQL databases. Results show that migration can be successfully performed, if semi-automatic migration approach including manual intervention is the most competent methodology.

Outline of method for mapping relational databases with NoSQL databases is explained in [9]. Different methods for data conversion and middle layer solutions are summarized. In their work, they introduced some research on new research on data migration from relational databases to NoSQL databases.

Methods of Migration the relational model to the document-oriented model is presented in [5]. A migration methodology to MongoDB is proposed, beginning with a technique for extracting a data model from a relational database through metadata. Next, it describes on how to carry on with schema and data migration preserving the integrity constraints. The paper can be further extended to introduce an object modeling in the NoSQL databases, which will affect the best approach for data storage, selection, insertion, and modification time. A prototype is created to show the feasibility and efficiency of the proposed approach.

Reference [8] Gives comparison of the performance of the three migration methods, using metrics that provides with detailed perspective of resource utilization as well as details about migration is being verified. This study shows how techniques of migration from relational databases to NoSQL and their related metrics can be utilized. It is concluded that it is consistently important to approve resources inaccessibility as prerequisite of the projects and not the migration itself [6]. Database migration explains migration process in a phased manner. Migration, the intensive facts of what is expected for the proper design of migration can predict the potential problems that can occur during migration and may decrease the possibility of failure.

Reference [3] Proposed an approach of model transformation and migration of data from RDBMS to MongoDB. The proposed work is separated into four parts (i) considers the query characteristics and data characteristics (ii) designs a model transformation algorithm based on description tags and action tags (iii) automatically migrates the data into MongoDB based on the result of model transformation (iv) develop an useful tool. Experiments demonstrate that utilizing this methodology can accomplish a better performance. They give a tool which enable users to handily perform transformation and migration. Utilizing this tool, users can pick appropriate NoSQL database to replace relational database. The choice of appropriate mid-model is important for the effectiveness of the model transition and data migration between Nosql databases and relational databases [7]. Data and query features are introduced into mid-model. This model, if implemented as a tool can also serve as an interface for DBA.

Reference [10] Girts suggest an approach for faster migration of data from relational database to NoSQL document-oriented databases. Two logical levels using automated means allowing user intervention to further update the logical level are developed over physical data. The data migration function is implemented in the DigiBrowser relational database browser. The proposed method helps to get proof of ideas for a new document-oriented database solution in two to three days.

In [11], Hassan, Rondik J. et al. research show that the Internet of Things has proven to be the biggest promoter and promoter of smart city plans. This is a key step in the transformation of traditional public facilities into smart services and the creation of new services. Big data and cloud computing play a central role in advancing these new projects because they can expand existing work by providing ever-expanding information and computing power. The data generated in smart city projects is heterogeneous and unstructured in nature. NoSQL can easily store unstructured data due to its flexible model. Reference [12] describes numerous NoSQL benchmarks made by the scientific community and mentions HBase's ability to provide citizens with energy bills reinvented in smart city services. The effect confirms that Cassandra and HBase are generally diagnosed as appropriate general actors and can be followed without much concern. The structural design provided has passed ETL verification and predicts big data in the smart city environment. The architecture proposed in this chapter, NoSQL, now plays a vital role in the Internet and mobile applications.

7.3 Smart City Requirements

Smart cities can be defined as developed urban areas that create sustainable economic development and high-quality life through outstanding performance in many key areas such as economy, mobility, environment, people, life, and government. Smart cities use various technologies to improve the infrastructure, energy, learning, well-being, and water supply efficiency of their residents. Generally speaking, a smart city must have:

(i) Qualified supply consisting of public services such as water, electricity, solid waste, sanitation and sewage treatment, and related government services.
(ii) Supply–demand matching system ground transportation services, providing no Congested roads and shortest waiting time for public transportation.
(iii) Active surveillance in the city to provide the public safety that citizens urgently need.
(iv) Provide reliable emergency services as needed, such as ambulances, fire safety, etc.

7.3.1 Smart City Applications

1. ***Smart Traffic Management***

Smart Traffic Management includes modern technologies and services to enhance and innovate the urban traffic experience, solve traffic problems such as traffic congestion, and reduce injuries. The evolution of the scale and trends of urban traffic has raised problems that affect the well-being of cities. This type of transport has a great impact

on the atmosphere of the city: a large number of vehicles that use energy cause noise and air pollution, which causes climate change and damage to the atmosphere.

2. ***Cost-Effective Energy Efficient System***

The clean and green energy digital grid is one of the most attractive prospects for smart cities. For example, smart solar or wind power plants can become an important part of the smart city ecosystem.

3. ***Improving Accessibility of Healthcare Systems***

Smart health care is a combination of technologies designed to maximize the survival rate and quality of life of the population. Smart healthcare networks use mobile, Internet of Things, and computing technologies to ensure accurate diagnosis and transform health care.

4. ***Efficient Parking System***

The parking problem is some worldwide problems that famous urban residents often face, but it is also difficult to solve. Smart city technology can help detecting number of empty parking slots immediately, and reduce the time and confusion that usually comes with parking.

5. ***Smart Irrigation Systems***

In traditional irrigation systems, due to inefficiency, most of the water and other resources are wasted. Smart Irrigation Systems uses IoT sensors to monitor the schedule and run times according to needs. Smart Irrigation sensors monitor plant water, weather, and soil conditions to automatically adjust the watering schedules.

6. ***Smart Homes***

It includes Smart lighting, appliances, gas detectors, etc. Smart lighting systems control lighting of homes remotely via mobile or web applications. Smart appliances give status information of appliances to the users remotely. Smoke detectors use optical detection, ionization, or air sampling techniques to detect the smoke. Gas detectors can detect harmful gases and raise alerts to the users.

7. ***Weather Monitoring***

Weather monitoring systems provide information about weathers. In such systems data collected from several sensors (temperature, humidity, pressure, etc.) and this data is sent to cloud-based applications and storage back-ends.

8. ***Air Pollution Monitoring***

Air pollution monitoring systems are the systems that monitor emission of harmful gases like carbon dioxide, carbon mono oxide, nitrous oxide, etc. Factories and automobiles use gaseous and meteorological sensors. These systems require Integration with a single-chip microcontroller, several air pollution sensors, GPRS-modem, and a GPS module.

7.4 Comparative Study of Relational and NoSQL Databases

NoSQL database is distinctive in numerous perspectives from conventional databases such as complexity, schema, transaction methodology, and dealing with accessing big data. The IoT applications in smart city projects usually have storing big data. With rapid growth in the clients and sensor data, it is not easy for legacy SQL system to handle big users. SQL systems store data in form of static tables and data is spread across multiple tables. To access the relevant information, expensive join operation is needed. NoSQL databases have been introduced to conquer this problem.

7.4.1 Characteristics of NoSQL Systems

Following are the characteristics of NoSQL Systems (Table 7.1).

7.4.2 Advantages and Disadvantages of NoSQL Databases

Advantages of NoSQL Database [13]

- In relational database data is structured and spread across different tables, due to this, there may be a possibility of complex joins and affect the performance of the

Table 7.1 Characteristics of relational and NoSQL databases Source: [32]

Characteristics	Relational databases	NoSQL
Data model	Concept of set and relations in mathematics	Key-value stores, graph, and document data model
Scalability	Vertical scalability	Horizontal Scalability
Transactional reliability	Fully support ACID	Range from BASE to ACID
Handling big data	Complex	Designed to handle big data
Crash recovery	Guarantee crash recovery via recovery manager	Depend on replication as backups to recover from crash
Datawarehouse	Performance degrade due to big data	High performance, scalability, availability in storing big data
Complexity	Rises because structure of DB could be quiet complex, difficult and slow working	Have capabilities to store unstructured, structured, or semi-structured data
Security	Adopted very secure mechanism	Has shortage in security because focus is on other purposes than security

system. NoSQL databases can store structured, semi-structured, and unstructured data format and provides high flexibility and better performance.
- NoSQL databases are well suited for cloud-based application.
- NoSQL has faster speed, easy scalability, high efficiency, and flexibility.
- Regarding performance and retrieving Big Data, NoSQL plays a major role by giving techniques to deal with Big Data.

Disadvantages of NoSQL Database

- Security is one of the important concerns of NoSQL databases. Due to lack of standardization across various vendors and models, it is difficult to provide a unified security solution.
- As compared to relational databases, maintaining NoSQL databases is complex.
- Relational database has SQL as its own standard query language, but NoSQL does not.
- Some NoSQL database vendors and models do not comply with ACID properties.

7.4.3 Advantages and Disadvantages of Relational Databases

Advantages of Relational Databases

- Security methods are well established. However, security methods are in evolving stage in NoSQL database systems.
- When contrasted with NoSQL, RDBMS is very simple to utilize and easy to manipulate data and maintain data integrity with reduced redundancy and replication.
- As compared to NoSQL, the data independence across logical and physical level is better in Relational databases.
- Backup and Recovery mechanisms are sound in relational database system.

7.5 NoSQL Databases Classification

NoSQL databases are classified as

(i) Document-oriented database.
(ii) Key-value database.
(iii) Column-oriented database.
(iv) Graph-oriented database.

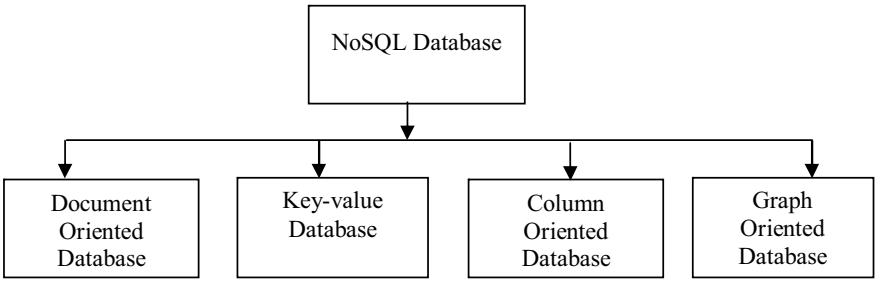

Fig. 7.1 Classification of NoSQL Databases Source: [27]

7.5.1 Document-Oriented Databases

Document-Oriented databases resemble Key-Value databases with the refinement that values are perceptible and can be queried. Document-oriented datasets utilized JSON or XML data format for storing data. Unlike key-value storage, it provides indexing and value-based queries. These databases store your data in the form of documents. The documents here are perceived by a unique set of keys and values, which are relatively the same as those in the key-value database. Document-based databases are schemaless and changeable in nature [10, 14]. The different characteristics of document-oriented storage are horizontal scalability and fragmentation between cluster nodes. Examples of document-oriented storage include MongoDB, Amazon DynamoDB, CouchDB, CouchBase, and so on [10].

7.5.2 Key-Value Databases

The simplicity of Key-value stores gives an incredible and effective methodology contrasted with different types of NoSQL database systems. Key-value database is a grouping of two components: Key and Values. Key is a unique identifier to particular information. The Value is a type of information indicated by the key [14]. Key-value store efficiently performs fundamental CRUD (Create, Read, Update, and Delete) operations. Compared to relational databases, the Key-value data model provides many features, such as highly scalable distributed data, big data support, high concurrency, and data retrieval speed. They also provide sharding across cluster nodes. Fragmentation is a horizontal division procedure used to divide a large amount of information into convenient, small, and effective pieces. However, the key-value database is not very flexible when querying and indexing complex and connected data. Queries in this category are usually based on keys rather than values. Examples of key-value storage are Redis, Memcached, Riak KV, Hazelcast, Ehcached, OrientDB, Aerospike, Amazon basic DB, etc. [10].

7.5.3 Column-Oriented Databases

Column-oriented databases are important when there is a need to deal with massive amount of data. Column stores in NoSQL are basically mixed row-column stores, not at all like pure relational column databases. In column-oriented database, each key is associated with one or more attributes. Column-oriented database stores information in such a way that columns that are needed and accessed at the same time are stored in one location in storage so that they can be accessed quickly with less I/O. It focuses on high scalability in data storage. One of the significant qualities is horizontal scalability. Examples of Column-oriented databases are Hbase, Accumulo, Hypertable, Google Cloud Bigtable, Sqrrl, ScyllaDB, and MapR-DB [10, 14].

7.5.4 Graph-Oriented Databases

Graph databases are based on the Graph theory. It deals with entities and their relationships in form of nodes and edges respectively. The graph is composed of vertex and edges, where nodes are objects and edges are the relationship among the objects. In graph databases, the relational table is replaced with structured relational graphs of interconnected key-value pairs. The graph consists of attributes related to nodes. It uses an index-free adjacency technique.

In this method every node comprises an immediate pointer which points to the adjacent node results in traversing millions of records. Graph databases focus on the relation established between data using pointers and offer schema less and well-organized storage of semi-structured data. Graph databases are faster than relational databases especially when identifying missing links and connections because in Graph databases queries are expressed as traversals. Graph databases support ACID property and support rollback [14].

Every real-world scenario can be corresponding to graphs and can also be modeled on graph database, because graphs have expressive and powerful modeling characteristics. Querying the graph database is more cost-effective because the graph query language does not require dense connections [10].

7.6 Methods of Data Migration

F. Matthews and C. Schulz describe the term data migration as follows: “Tool-supported one-time process which aims at migrating formatted data from a source structure to a target data structure whereas both structures differ on a conceptual and/or physical level”. There are two important steps in data migration: first, restructuring the source data to meet the requirements of the target system, and second, moving the data from the source to the target database. The document provides

several methodologies to manage the following steps: schema transformation, meta model approach, ETL (Extract, Transform, and Load), program transformation, model migration, automatic migration data transfer [10]. Following are some popular methods of data migration.

7.6.1 Mid-Model Approach [7]

The model proposed depends on two basic ideas: Data feature and query feature. The most fundamental element of the model is its generic nature, which can be modeled for any NoSQL database on demand. This model not only migrate data from relational to NoSQL but also maintains the integrity of data which was present in relational databases. This model not only migrates relational data to NoSQL, but also maintains the integrity of data already present in the relational database. Two strong points of Mid-model: (1) The data feature contains some characteristics of the relational database while (2) the query feature specifies the queries The principle challenge to mid-model design was to discover the distinctions that are present in the models of Relational and NoSQL databases. The middle model has an object because the base model is like an entity in a relational database that has a number of properties that describe the characteristics of the entity and its relationships with other objects.

7.6.2 NoSQLayer Approach [15]

NoSQLayer ensures that data is migrated from relational databases to NoSQL without modifying application code. The framework acts as an interface between the database and the application. The proposed solution is based on the use of two phases: migration and mapping. The framework contains two modules; The migration module ensures a consistent migration of data from relational databases to NoSQL databases using the metadata information contained in the data dictionary and the mapping module, providing the basis for running MySQL queries so that the legacy application can continue without modifications. It preserves the semantics of the source database and allows programmers to easily query the database by allowing them to code with a query on a relational database. The queries coming from the application are captured by NoSQLayer and converted to a specific NoSQL database format, the results generated by the NoSQL database are again captured by the framework and converted to tools specific to the application format. Mediator is an open source tool used to communicate between the application and the database. The results of the evaluation show that the framework is viable. However, the cost brought by the layer is very large when low volume data is involved in the operations, but it decreases

as the amount of data increases and thus NoSQLayer gives efficient results. The proposed framework works best with huge volumes of data correlating with MySQL databases. Compare the execution time of different queries with the evolution of data volumes and the proposed system is considered to be an efficient solution for data migration from relational databases to relational.

7.6.3 Data Adapter Approach [9]

Data Adapter system is mainly used for utilizing hybrid database. Data Adapter framework integrate RDBMS and NoSQL databases using Map Reduce process. Result demonstrates that the data adapter takes very less time to parse each query to get required data and information. Test data of Amazon Elastic Map Reduce is utilized as their relational database information source.

7.6.4 Automatic Mapping Framework [9]

It is a framework that provides automatic mapping of relational databases to MongoDB. MongoDB has a key and value pair. To map a relational database to a MongoDB collection, rows from the relational database are mapped as documents. Mapping a relational database to NoSQL involves thinking about the relationships that exist between different database tables. The easiest way to map in a 1:1 relationship includes embedding. Embedding and linking are commonly used in a 1:M relationship, while one-way or two-way embedding is used in an N:M relationship. The framework uses source information metadata from the database whether the relation contains table names, data types to be stored and access privileges, etc. The mapping steps include: creating a MongoDB database, creating a new table in the database using relational database metadata, and relating one table to another through primary keys and foreign keys.

7.7 Issues and Challenges in Migration

The use of Internet of Things (IoT) sensors is dramatically increased in smart city projects because they are cheaper, consume less power, are smaller, and are easier to use. IoT sensors typically collect large amounts of data and send it to a remote server. Data collected by sensors is heterogeneous i.e., contains images, videos, sound, fingerprints, Iris scan, text, etc. This data must be analyzed and accessible for various purposes. Legacy SQL systems are not capable of handling unstructured, heterogeneous, and big data generated in smart city projects. Thus migration as well as bridges for connection with legacy SQL systems is needed. At the moment, it

is difficult to find an appropriate process and strategy for migration from relational databases to non-relational databases effectively, due to following major issues and challenges [3].

7.7.1 Model Transformation

Since no special tools exist, most model transformation techniques depend on the experience of the DBA. Therefore, the physical model of NoSQL is often designed by individuals based on existing relational databases. Very inconvenient when the tables are involved and complex with a large number of attributes. Likewise, it is difficult to choose which tables should be integrated with each other and which tables should use references correctly. From a certain point of view, if we integrate all related tables, it can improve performance while causing data redundancy. Again, NoSQL does not support join operations. If we use a per-table reference, NoSQL will issue multiple queries when reading related documents. Reading related documents through different queries can lead to poor performance. A performance compromise and data redundancy technique works best. In addition, different applications may require different methodologies when converting models. Sometimes manual intervention is required to ensure that the new model is exactly what it is intended to be and that it will meet the requirements.

7.7.2 Data Migration

Currently, data migration tools use basic methodologies, such as moving each database table into a collection. Currently, no existing job can automatically migrate data based on model information. Semi-automatic tools with partial manual intervention are available. These tools provide a straight forward technique to migrate relational databases to NoSQL databases. The Extract-Transform-Load engine implements an interface with several mainstream NoSQL database systems to perform data migrations, such as MongoDB, Hbase, and Cassandra. The only downside is that these ETL systems are therefore unable to map the original database to the target database. ETL tools can only complement the information migration process, forcing users to define their own mapping strategies between different databases.

7.7.3 Schema Conversion

Wu-Chun Chung and Chongxin Li suggest two thoughts on independent modification of the database schema in HBase. The first idea uses the map-reduce framework. This framework converts each table in the database into an HBase table. After the

conversion, each table turns into a column family of HBase. The second reflection uses three rules to handle schema conversions. Related tables are nested and converted to an HBase table. However, it did not take into account multi-level nesting. To convert relational databases to NoSQL databases, some focus on transformation and NoSQL models, while others focus on data migration. Very few studies are based on the combined approach [3].

7.7.4 Strategies to Perform Join

In a relational database, data is stored as static table. Due to normalization, data is spread across multiple tables and while the user wants to access the relevant data as per the requirement, expensive join operation needs to be performed. Join is bulky operation that incurs cost and affects the performance adversely, if not implemented in proper manner. We need to think about how to minimize the cost and maximize the performance when performing join operations. It is evident that different applications and scenarios may require different strategies to execute join operation during migration process. In NoSQL databases, no join operation is required. When migrating from relational to NoSQL, we must define strategies to handle join. There may be a variety of scenario, for each of these, strategy needs to be defined.

7.7.5 Use of Indexes

In any database, the index is an important modifiable performance factor and then an important factor in schema design. NoSQL supports various types of indexes like binary, hash, unique, compound, array, sparse, etc. which are used to increase the data retrieval efficiency. Therefore, while migrating toward NoSQL, a strategy regarding which type of index can be used is a vital consideration. Index selection is based on the type and frequency of the queries used in applications. As with all databases, indexing is not free it incurs overhead cost and forces overhead and causes an overload of writing and resource usage (disk and memory).

MongoDB creates an index on the document's primary key _id field. All indexes are characterized by users as secondary indexes. Any field can be used for the secondary index, including fields inside sub-documents and Arrays. For optimizing performance, query optimizer chooses the index imperially by periodically executing substitute query plans and choosing the arrangement with the best response time [16].

7.8 Case Study on MongoDB

7.8.1 Comparison—SQL and MongoDB

MongoDB is a NoSQL database management system introduced in 2009. It stores data as JSON type documents with dynamic schemas. Its main purpose is to support big data handling. MongoDB is focused on four factors: flexibility, power, speed, and simple use. MongoDB has schemaless document-oriented data model whereas SQL supports relational data model. Relational databases have a standard language SQL though MongoDB supports API calls.

MongoDB has aggregate functions, a built-in map-reduce function that can be used to aggregate large amounts of data. The integrity model used by relational databases is ACID, while MongoDB uses BASE. MongoDB provides consistency, durability, and conditional atomicity. Relational databases provide integrity options that MongoDB does not, such as transactional, isolation, revision management, and referential integrity. On the distribution side, MongoDB and relational information are horizontally scalable and support data replication. While MongoDB supports sharing, relational databases do not. MongoDB and SQL are both cross-platform DBMSs. MongoDB is a free database system, while a license is required to use relational databases [17].

7.8.2 Features of MongoDB

- MongoDB provides high performance.
- Support basic CRUD operations and provide Aggregation.
- MongoDB gives High accessibility and auto-Replication feature. Data is restored through backup (replica) just in case of server failure.
- Provides automatic failover mechanism.
- A database in MongoDB is set of collections, which in turn holds collection of documents. A document is equivalent to record in database.
- Sharding is an important feature supported by MongoDB which provides horizontal scalability.

7.8.3 Advantages of MongoDB

- Installing and Setting up MongoDB is easy.
- MongoDB possesses schemaless structure and provides easy scalability.
- MongoDB supports dynamic queries.
- In MongoDB, no complex joins are required.
- MongoDB provides faster data access.

- As compared to relational databases performance improvement is done easily in MongoDB.
- MongoDB supports horizontal scaling whereas Relational databases support vertical scaling.

7.8.4 CRUD Operations in MongoDB

This section describes the basic CRUD operations in MongoDB. Two databases, one using SQL and one in MongoDB, were created to compare how data will be created, selected, inserted, and deleted in the two databases [18]. MongoDB provides all major CRUD operations and provides aggregation functionality.

Table 7.2 describes the main CRUD operations in MongoDB.

Smart city application stores records in the form of normalized tables. If it is required to retrieve relevant data, expensive join operation needs to be performed.

Table 7.2 CRUD operations in MongoDB Source: [40]

Operations	SQL	MongoDB
Create table	CREATE TABLE Accounts (first_name‘ VARCHAR(50) NULL, ‘last_name‘ VARCHAR(50) NULL, PRIMARY KEY (‘id‘));	db.accounts.insert({ name:”abc”, age:26, address:”indore”})
Delete a table	Drop table accounts;	db.accounts.drop()
Insert	Insert into accounts(name, age, address) VALUES (“abc”, 26, “indore”)	db.accounts.insert({ name:”abc”, age:26, address:”indore”})
Select	Select * from accounts	db.accounts.find()
Select fields	Select first_name, last_name from accounts	db.accounts.find({}, { first_name: 1, last_name: 1})
Conditional select	Select * from Accounts where dep_wid = ”D” and balance > 5000	db.accounts.find({dep_wid:”d”, balance:{$gt:5000}})
Ordered select ascending	Select * from accounts order by user_id asc	db.accounts.find({}).sort({user_id: 1})
Ordered select descending	Select * from accounts order by user_id desc	db.accounts.find({}).sort({user_id: -1})
Select with count	Select count(*) from users	db.users.count()
Update	Uupdate table student set section = “F” where marks <30	db.Student.update({marks:{lt:30}}, {$set:{Section:”F”}})
Delete	delete from Student	db.Student.remove()
Delete with condition	delete from Student where section = ”a”	db.student.delete({section:”a”})

Instead if this application is being migrated to NoSQL MongoDB, one record can be centralized. Other records may be kept on server. As data grows, more servers can be added which provide better scalability and availability. MongoDB supports database sharding feature which allow data packets are stored across different machines to ensure the system does not fail as volume rise. Data stored in smart city application is heterogeneous such as text, image, and biometric data (fingerprints, Iris, face recognition). MongoDB can efficiently store huge volumes of biometric data and images, whereas many other management systems, such as SQL, are less suited for image storage. In MongoDB multiple data can be stored in single collection in the form of embedded documents. It provides better flexibility in accessing data as no join is required. It reduces cost by avoiding multiple joins and also data access speed is increased. In smart city applications, biometric data need to be stored to enhance verification and security mechanism. MongoDB stores fingerprints and takes less time to access it as compared to relational databases.

7.9 Conclusion and Future Work

Relational databases are structured and have limitations when required to handle unstructured and big data. In relational databases, because of normalization, data is spread across multiple tables and expensive join operation is required to integrate data. Due to limitations of relational databases, most of the leading organizations are migrating toward NoSQL. NoSQL databases are well suited for the large volume, reliable and high availability web applications. Scale out approach and performance favors NoSQL.

This chapter tends to explore comparative study of Relational and NoSQL databases, classification of NoSQL databases, case study, popular approaches for migration and some of the key issues and challenges in migrating from Relational to NoSQL databases. The chapter is useful for students and researchers to investigate migration strategies for migrating from Relational to NoSQL databases. Smart city applications collect massive amount of data from various types of IoT sensors, engines, and people. Most of the data generated are heterogeneous in nature. However, this data need to be integrated with legacy applications based on SQL. Some of these applications also require migration to NoSQL for improved performance and fault tolerance. This paper addresses challenges of working in hybrid environments and migration issues from SQL to NoSQL databases.

Key research issues and challenges in migrating from relational to NoSQL include model transformation, data migration, schema conversion, application integration, strategies related to perform join in different scenario, use of indexes, storage issues. In model transformation, we need to consider how to make compromise among performance and data redundancy while transforming the schema. Moreover, various applications may require different strategies at model transformation. Semi-automatic strategies requiring manual intervention are required to tweak the model after it is transformed so as to exactly suit the application requirements. We

need to think about how to minimize the cost and maximize the performance when performing join operations. It is evident that different applications and scenarios may require different strategies in performing join during migration process. There may be a variety of scenario, for each of these, strategy needs to be defined. Indexes are the important performance factor in any database and are therefore crucial for schema design. So while migrating toward NoSQL, strategies regarding which type of index can be used are a vital consideration. On the basis of key issues and challenges addressed in this chapter, strategies for migration from relational to NoSQL for the potential applications of NoSQL databases can be proposed. A fully automatic methodology is proposed and under development using machine learning techniques, which will be based on keeping history data of past manual interventions and selecting one of the best interventions on the basis of application and database parameters. Massive growth is likely in the area of NoSQL over cloud because of initiatives like smart City, smart society however, these applications required integration with legacy applications built over SQL. Thus need for migration as well as bridges for connection leading to requirements of our research area.

References

1. Li F et al (2019) Toward semi-automated role mapping for IoT systems in smart cities. 2019 IEEE international smart cities conference (ISC2). IEEE
2. Sallow AB, Sadeeq M, Zebari RR, Abdulrazzaq MB, Mahmood MR, Shukur HM et al, An investigation for mobile malware behavioral and detection techniques based on android platform. IOSR J Comput Eng (IOSR-JCE) 22:14–20
3. Jia T, Zhao X, Wang Z, Gong D, Ding G (2016) Model transformation and data migration from relational database to MongoDB. In: 2016 IEEE international congress on big data (BigData Congress), pp 60–67. IEEE
4. Kuderu N, Kumari V (2016) Relational database to NoSQL conversion by schema migration and mapping. Int J Comput Eng Res Trends 3(9):506
5. El Alami A, Bahaj M (2017) Migration of relational databases to NoSQL: the way forward. In: 2016 5th international conference on multimedia computing and systems (ICMCS), pp 18–23. IEEE
6. Soni P, Koushal DS, Survey paper on data migration techniques of RDBMS to NoSQL
7. Liang D, Lin Y, Ding G (2015) Mid-model design used in model transition and data migration between relational databases and nosql databases. In: 2015 IEEE international conference on smart city/SocialCom/SustainCom (SmartCity), pp 866–869. IEEE
8. Oliveira F, Oliveira A, Alturas B (2017) Migration of relational databases to NoSQL-methods of analysis. In: 7th international conference on human and social sciences, pp 121–128. Richtmann Publishing
9. Ghotiya S, Mandal J, Kandasamy S (2017) Migration from relational to NoSQL database. In: IOP conference series: materials science and engineering, Vol 263, No 4, p 042055. IOP Publishing
10. Karnitis G, Arnicans G (2015) Migration of relational database to document-oriented database: Structure denormalization and data transformation. In: 2015 7th international conference on computational intelligence, communication systems and networks, pp 113–118. IEEE
11. Hassan RJ et al (2021) State of art survey for iot effects on smart city technology: challenges, opportunities, and solutions. Asian J Res Comput Sci, 32–48

12. Costa C, Santos MY (2016) Reinventing the energy bill in smart cities with NoSQL technologies. Transactions on engineering technologies. Springer, Singapore, pp 383–396
13. Sharma Y, Sharma Y (2019) Case study of traditional RDBMS and NoSQL database system. Int J Res GRANTHAALAYAH 7(7):351–359
14. Abraham SM (2016) Comparative analysis of Mongodb deployments in diverse application areas. Int J Eng Manag Res (IJEMR) 6(1):21–24
15. Rocha L, Vale F, Cirilo E, Barbosa D, Mourão F (2015) A framework for migrating relational datasets to NoSQL. Procedia Comput Sci 51:2593–2602
16. A MongoDB White Paper, RDBMS to MongoDB Migration Guide, considerations and best practices, June 2018
17. Boicea A, Radulescu F, Agapin LI (2012) MongoDB vs Oracle--database comparison. In: 2012 third international conference on emerging intelligent data and web technologies, pp 330–335. IEEE
18. Hecht R, Jablonski S (2011) NoSQL evaluation: a use case oriented survey. In: 2011 international conference on cloud and service computing, pp. 336–341. IEEE
19. Ghule S, Vadali R (2017) Transformation of SQL system to NoSQL system and performing data analytics using SVM. In: 2017 international conference on trends in electronics and informatics (ICEI), pp 883–887. IEEE
20. Hsu JC, Hsu CH, Chen SC, Chung YC (2014) Correlation aware technique for SQL to NoSQL transformation. In: 2014 7th international conference on Ubi-media computing and workshops, pp 43–46. IEEE
21. Potey M, Digrase M, Deshmukh G, Nerkar M (2015) Database migration from structured database to non-structured database. Int J Comput Appl 975:8887
22. https://www.couchbase.com/comparing-couchbase-vs-oracle Moving from relational to NoSQL: How to Get Started
23. Gayathiri NR, Jaspher DD, Natarajan AM (2019) Big Data retrieval techniques based on hash indexing and Mapreduce approach with NoSQL Database. In: 2019 international conference on advances in computing and communication engineering (ICACCE), pp. 1–8. IEEE
24. Das TK, Kumar PM (2013) Big data analytics: a framework for unstructured data analysis. Int J Eng Sci Technol 5(1):153
25. Eckerstorfer F (2011) Performance of NoSQL databases
26. Faraj A, Rashid B, Shareef T (2014) Comparative study of relational and non-relations database performances using Oracle and MongoDB systems. Int J Comput Eng Technol (IJCET) 5(11):11–22
27. Farooq H, Mahmood A, Ferzund J (2017) Do NoSQL databases cope with current data challenges. Int J Comput Sci Inform Secur (IJCSIS) 15(4)
28. Franco M, Nogueira M (2011) Using NoSQL database to persist complex data objects. In: Conference, Instituto de Informá tica Universidade Federal de Goias (UFG)
29. Heripracoyo S, Kurniawan R (2016) Big Data Analysis with MongoDB for decision support system. TELKOMNIKA (Telecommun Comput Electron Control) 14(3):1083–1089
30. Li Y, Manoharan S (2013) A performance comparison of SQL and NoSQL databases. In: 2013 IEEE Pacific Rim conference on communications, computers and signal processing (PACRIM), pp. 15–19. IEEE
31. Mapanga I, Kadebu P (2013) Database management systems: a nosql analysis. Int J Modern Commun Technol Res 1(7):265849
32. Mohamed MA, Altrafi OG, Ismail MO (2014) Relational vs. nosql databases: a survey. Int J Comput Inform Technol 3(03):598–601
33. Nayak A, Poriya A, Poojary D (2013) Type of NOSQL databases and its comparison with relational databases. Int J Appl Inform Syst 5(4):16–19
34. Swaroop P, Gupta V, Singh KR, Rajan SN (2016) NoSQL paradigm and performance evaluation.Scient Soc Adv Res Soc Change 3
35. Zaki AK (2014) NoSQL databases: new millennium database for big data, big users, cloud computing and its security challenges. Int J Res Eng Technol (IJRET) 3(15):403–409

36. Zvarevashe K, Gotora TT (2014) A random walk through the dark side of NoSQL databases in big data analytics. Int J Sci Res 3(6):506–509
37. Priyanka A (2016) A review of nosql databases, types and comparison with relational database. Int J Eng Sci Comput 6(5):4963–4966
38. Győrödi C, Győrödi R, Pecherle G, Olah A (2015) A comparative study: MongoDB vs. MySQL. In: 2015 13th international conference on engineering of modern electric systems (EMES), pp. 1–6. IEEE
39. Simanjuntak HT, Simanjuntak L, Situmorang G, Saragih A (2015) Query response TIME comparison Nosqldb Mongodb with Sqldb Oracle. JUTI: Jurnal Ilmiah Teknologi Informasi 13(1): 95–105
40. Truica CO, Boicea A, Trifan I (2013) CRUD operations in MongoDB. In: 2013 international conference on advanced computer science and electronics information (ICACSEI 2013), pp 347–350. Atlantis Press
41. Cattell R (2011) Scalable SQL and NoSQL data stores. ACM SIGMOD Rec 39(4):12–27
42. Barmpis K, Kolovos DS (2014) Evaluation of contemporary graph databases for efficient persistence of large-scale models. J Object Technol 13(3):3–1
43. Gu Y, Shen S, Zheng G (2011) Application of nosql database in web crawling. Int J Digital Content Technol Appl 5(6):261–266
44. Leavitt N (2010) Will NoSQL databases live up to their promise? Computer 43(2):12–14
45. Padhy S, Kumaran GMM (2019) A quantitative performance analysis between Mongodb and oracle NoSQL. In: 2019 6th international conference on computing for sustainable global development (INDIACom), pp 387–391. IEEE
46. Singh A (2019) Data migration from relational database to MongoDB. Global J Comput Sci Technol