A METHOD FOR SELECTING AN EFFECTIVE DATABASE

BY

THOMAS SWED

DEPARTMENT OF COMPUTER SCIENCE

Lewis University
1 University Pkwy,
Romeoville, IL 60446
thomasdswed@lewisu.edu

Romeoville, Illinois
May 2017

I. Abstract

This paper provides an introduction to relational as well as NoSQL databases with the intent of equipping the reading with an adequate understanding to be able to choose an effective model to implement for any given circumstance. Characteristics, strengths and weaknesses will be presented, as well as the various types of NoSQL databases. Conditions are presented that led several companies to consider moving away from a traditional relational database to a NoSQL model. While a NoSQL database has proved beneficial for many companies, the aim of this paper is to give the reader an understanding of when to use a particular type of database. Often times, this will mean combining several different types to create an effective and optimal solution.

II. INTRODUCTION

Data collection has increased exponentially in recent years due to significant advances in smart devices, inexpensive storage, social software and the ability to connect many devices to a network, including vehicles and even homes. An ever-increasing quantity of information is being generated and stored in an unstructured, mixed form that relational databases do not handle well. As the vast amount of data is collected it must be stored in a reliable, efficient, and secure way. Incompatible data formats, incomplete data, non-aligned data structures, and inconsistent data semantics represent significant challenges when interacting and querying these data sets. To handle this diversity, platforms are needed that can integrate and manage structured as well as unstructured data. For many years, data has been stored as a default in structured relational databases while using a declarative language (SQL) to query it for serious data storage and interaction. However, to scale the demands of large datasets and increase productivity, in recent years, the use of a different type of database, namely NoSQL, is on the rise.

Advocates of NoSQL databases claim that they can build systems that perform better, scale easier, and are simpler to develop than traditional databases. Generally, NoSQL databases can be categorized into 4 types based on their particular data model [1]. Often, it is difficult to specify a category for certain databases, as several do not clearly fit into a single category but overlap and may perform functions of multiple models. While this is not a comprehensive list, the following represents some of the more common NoSQL databases in use.

- *Key-Value* model – Examples include ArangoDB, BerkeleyDB, LevelDB, Memcached, Project Voldemort, Redis, *Riak.*
- *Document* model – CouchDB, *MongoDB*, OrientDB, RavenDB, Terrastore
- *Column-Family* model – Amazon SimpleDB, *Cassandra,* HBase, Hypertable
- *Graph* model – FlockDB, HyperGraphDB, Infinite Graph, *Neo4J*, OrientDB

While NoSQL databases have become increasingly popular, relational databases are still widely used, mature, and effective database systems. NoSQL has proved to be beneficial in several cases in dealing with large data sets, but does this mean that it is the paramount solution for a given database? How does an organization determine the most effective type of database? These are difficult questions to answer, and each must be evaluated in its respective context to appropriately asses a company's needs. The aim of this paper is to show the characteristics of NoSQL databases, explain that they are unlikely to replace relational databases, and that the answer in many cases will be combining several different models to suit a given scenario. Rather than being either/or, the answer in many cases is *both*.

III. Relational Databases

A relational database characteristically groups data of the same type and stores it in tables. The information is related and indexed based on a common key or type. Relational databases are best described in terms of a spreadsheet, consisting of collections of tables each containing rows (formally known as tuples) and columns (known as attributes). Information is organized and grouped by common characteristics and stored in rows in a corresponding table. The dominant query language used is SQL and while variations of the language exist between differing software, core principles remain uniform. A relationship is defined when a column in one table references a row in another table. Figure 4.1 gives an example of how an order is stored in a relational database [2].

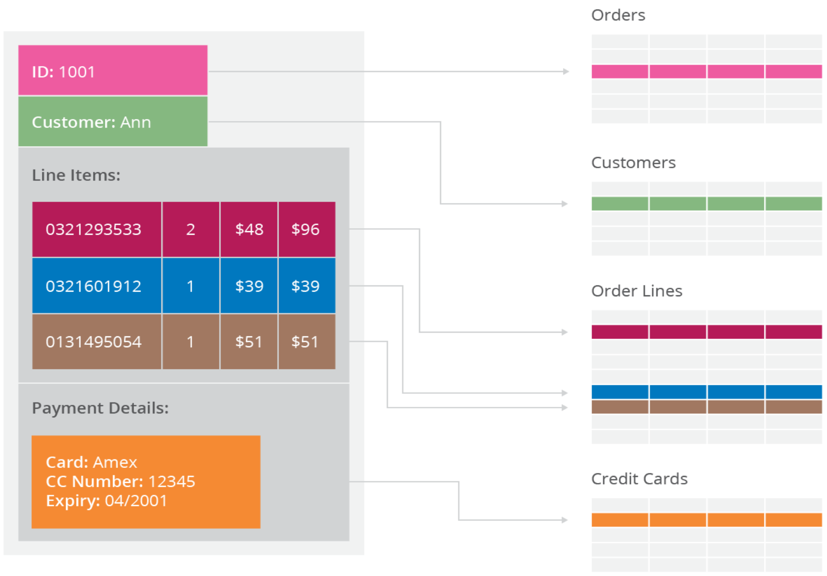


Fig 4.1 Example of an order stored in a relational model

Each line of the order is placed in separate rows in multiple tables based on the type of information each contains. To support running on a cluster, this order would typically be stored in a NoSQL database in an aggregate model as a single entry.

2.1 Benefits

a) Persistent Data

When considering databases, there are essentially two types of memory: volatile (main memory) and non-volatile (backing store). Volatile (RAM) memory is limited in size and does not save data beyond a session; consequently, if power is lost, so is the data. In order to be stored, data must be written to non-volatile memory. This is memory that can retain stored information without power. Examples include hard drives and solid-state drives (SSD). This type of storage allows databases the flexibility and reliability to house data in a convenient way that remains accessible.

b) Concurrency

Concurrency can be defined as the ability for multiple processes to access or change shared data at the same time [3]. Many applications must allow several users to view and edit the same data at once, or concurrently. This poses potential issues when multiple users attempt to modify this information at the same time. Concurrency is a common and extremely difficult problem. Relational databases address this issue by controlling all access to their data through ACID transactions. This allows the database to maintain data consistency while permitting data concurrency. While challenges still exist with this solution, their complexity has been greatly reduced.

c) Integration

Often, applications need to share the same data between various teams developing a common system. A standard solution to this problem is referred to as *shared database integration*, where several applications store data in a single database [4]. This allows users to easily share data while allowing the database's concurrency control to handle the access and editing of data.

d) Standardization

Generally, relational databases became dominant due to their standardized model. Once developers learned the basic relational structure of tables joined by keys, they could apply it to multiple applications. While differences do exist between each individual relational database, the fundamental characteristics remain the same.

2.2 Impedance Mismatch

A major irritation for developers implementing relational databases is commonly referred to as impedance mismatch. Values in a table row must conform to a pre-determined simplistic data type. They may not contain structure, or nested information. This presents a challenge for developers who must translate a more complex data structure into relationships. Conversely, in-memory data structures allow much more complex data sets and prove to be beneficial in some cases. However, in order to use an in-memory data structure a developer must translate it to a relational depiction when storing it.

IV. The Rise of NoSQL

Since the 2000s, the explosion of data collection and storage has continued to rise. Websites began tracking activity and structured it in a very detailed way; huge datasets such as social networks, logs, links and mapping data appeared as well as the emergence of Big Data [5]. As smart devices became mainstream, websites began to experience a growth in users. To account for these new demands, greater resources are required. To "scale up" requires larger machines with more processing and storage capabilities, while “scaling out” means using many small machines grouped in a cluster. A clustered approach proved to be a cost-effective and more reliable method.

The need to scale gave way to a problem: relational databases are designed to be run on a single machine and do lend themselves well to clusters. Technical complications as well as licensing costs related to this incompatibility led powerful organizations to consider a different approach. Google and Amazon each developed database systems designed to operate on clusters, BigTable [6] and Dynamo [7]. Soon many companies began exploring alternatives to their relational database model.

3.1 Characteristics

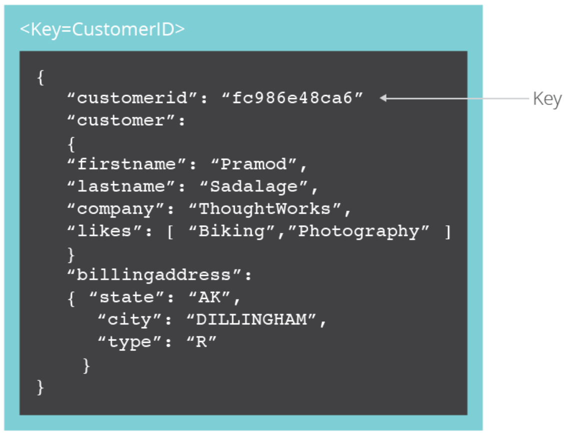
The term “NoSQL” unfortunately has no clear definition and has commonly been known as “non-SQL”, “non-relational” or “not only SQL” [8]. It is therefore more helpful to focus on what the term means, rather than what it stands for. NoSQL databases perhaps most obviously do not use SQL as a query language; they are generally open-source and have been developed after the turn of the century. Another distinguishing feature from relational databases is that they are designed to run on clusters rather than a single machine, with the exception of the graph type. There are four basic types of NoSQL databases: Key-Value stores, Document Store, Column-Family store and Graph [9].

a) Key-Value store

A key-value database stores information using a map or dictionary data model. The data is represented as a collection of key-value pairs, where the key is a unique value in the data set. In order to query the database, searches must be performed on the key which will then return the value associated with it. While this provides a simple and efficient model, it may not be practical in some situations, such as the need to query by a value rather than by the key. In many cases, queries will need to be made on multiple fields rather than only a key.

b) Document Store

As the name implies, this model stores information in the format of a document. A helpful illustration is an order placed to a given company. The individual order, or document is identified by a unique key, often an invoice or order number, while the information about the order is encapsulated. Differing documents are allowed to have completely different fields or may use the same fields. Queries may be made on any encapsulated field in the document, only those with matching fields will be included in the search. Data is most commonly stored in the JSON format, though formats like XML, YAML and BSON are supported. Figure 3.1 gives an example of a document stored in the JSON format [10].



```
{
   “customerid”: “fc986e48ca6”
   “customer”:
   {
   “firstname”: “Pramod”,
   “lastname”: “Sadalage”,
   “company”: “ThoughtWorks”,
   “likes”: [ “Biking”,”Photography” ]
   }
   “billingaddress”:
   { “state”: “AK”,
      “city”: “DILLINGHAM”,
      “type”: “R”
    }
}
```

Figure 3.1. A JSON example document

c) Column-Family Store

Possibly a more difficult concept to grasp for those accustomed to a relational model is that of a column-family model. These databases are similar to key-value models, except instead of having one key-value pair, they have two levels of structure. A key on one level is known as the row identifier while the subsequent values on the second level are referred to as columns.

d) Graph

As the majority of NoSQL databases were designed for use on clusters, the graph model is the exception. It consists of data stored in small records, called nodes, with highly intricate interconnections, called edges. An example is given in figure 3.1 [11]. These databases are designed to model data consisting of complex relationships such as road maps, product preferences or social networks.

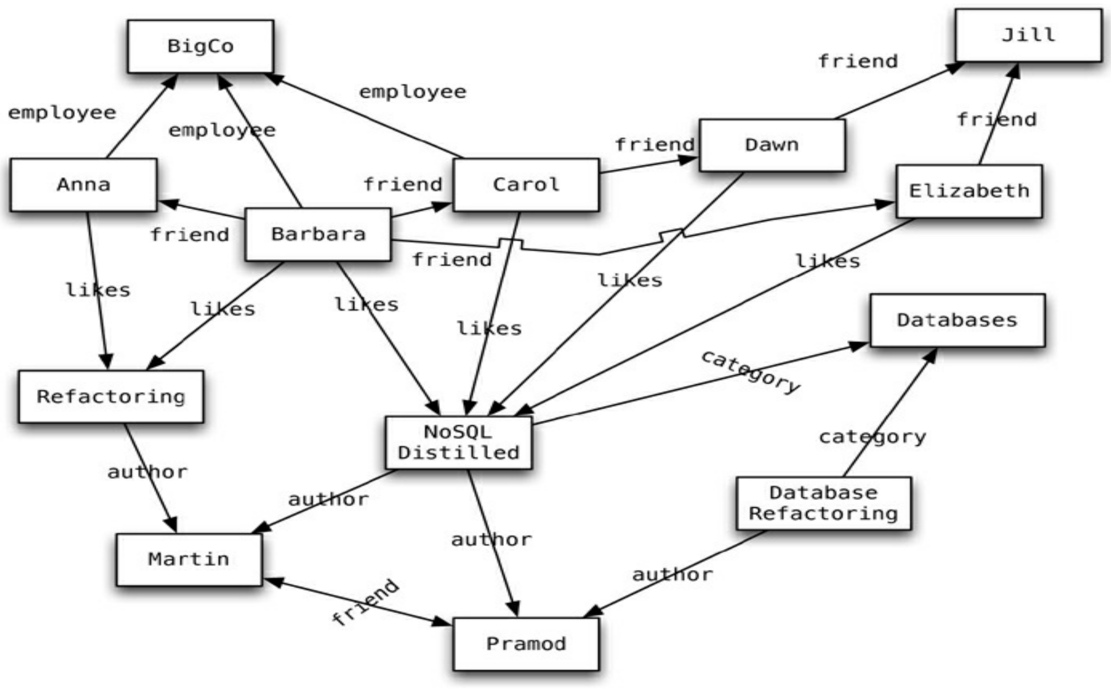


Figure 3.1. A Graph model database

A graph model is applied where information about data interconnectivity or topology is more important than the data itself [12]. The priority is given to the relationships between the stored data. This is a distinct difference from a relational database in that while the relational model has the ability to model relationships between data, the query joins used to retrieve this information severely limits performance times. A graph model has no problem performing many complex joins in a short time period.

V. Data Models

The term data model refers to a collection of conceptual tools used to model representations of real-world entities and the relationships among them [13]. Practically speaking, it is a lens through which we distinguish and interact with data. For many years, the governing data model has been the relational model. Data stored in each column must conform to the data type defined when the table was created; nested values are not allowed. This presents a simple yet powerful model.

By contrast, an aggregate model allows for much more complex data types that permit nested values. This type of model more closely resembles the types of data that application developers use. Data is stored in a record that allows additional information to be kept inside it. Key-value, document, and column-family databases all utilize this model. The primary advantage of aggregate models is that they lend themselves well to running on clusters and readily support sharding and replication by design [14]. While this model provides advantages in certain instances, it is important to think about how the data will be accessed when choosing a particular model. Aggregate models are best used when a specific structure and purpose is in mind when querying and interacting with the data. This is because data is grouped based on how it will be used in addition to its relationship among records. For example, a single aggregate is used where a user needs to access every order a customer places. This is opposed to separate aggregates if a user only needs to view one order at a time.

Data stored in aggregate models is said to be denormalized as opposed to relational models whose information is normalized upon storage. When information is denormalized, redundant copies of the data are duplicated and stored in each aggregate. This greatly increases data retrieval performance but places a greater load on write performance as much more information is recorded. This leads to a source of frustration when updating information in these models. Because the information is duplicated among many aggregates, the challenge lies in updating multiple aggregates at the same time.

In contrast, the data stored in a relational model removes redundancy to provide data integrity while maintaining accessibility. This process ensures that records are not duplicated across or within tables to maintain an ideal structure. Consequently, updates are much more manageable as ACID transactions in this model. ACID (Atomicity, Consistency, Isolation, Durability) refers to a set of properties that are applied when altering a database [15].

- *Atomicity* requires that each transaction be fully completed, otherwise no change is enacted on the database. If an error occurs before the transaction is realized, the system reverts back to the state it was in prior to starting that transaction. This is known as a rollback.
- *Consistency* ensures that a database system is in a valid state upon completion of the transaction, whether it succeeded or failed. The transaction guarantees that any data written to the database has abided by all defined rules, thus enabling a valid state. If an error occurs, the transaction will automatically perform a rollback to return the database to a stable condition.

- The *isolation* property confirms that any execution of a transaction happens successively. So as not to access data simultaneously with another running transaction, isolation maintains data integrity by controlling concurrency.
- *Durability* refers to the permanency of the change that has been enacted upon completion of the transaction. Safeguards are set in place to prevent the loss of information in the event of power loss, crashes or errors. The steps of a transaction are logged in non-volatile memory to guarantee duplication of the systems state at any given point in the transaction process.

4.1 Relationships

Characteristically, aggregate models group data based on the way it is accessed. In several cases, separate aggregates will be required as in the instance of a customer accessing a customer's orders one at a time. However, in this situation the data is stored in separate aggregates while the database is unaware of a relationship between them. Many databases enable a way for developers to define these relationships so that they are visible to the database as they provide certain advantages when accessing the data.

VI. Data Distribution

Aggregate models provide the capability of handling huge amounts of data, faster performance and greater reliability than a relational model. These advantages are influenced by the way in which data is distributed: replication and sharding. Replication duplicates data on multiple servers (nodes), essentially creating copies of the database. It is implemented in two forms: master-slave and peer-to-peer (also known as multi-master). Sharding effectively distributes data across multiple servers (nodes) where each server contains part of the data. Characteristics of each distribution model [16] are outlined below.

- *Single Server* distribution is traditionally how the relational model is implemented. A single server distribution model is not limited to a relational database, however. All data is stored on a single server that controls all interactions with the database. This fits well with the simple design of a relational database. Management of the database is fairly straightforward and isolated. There are instances where it is desirable to use this distribution for a NoSQL database, such as if the data usage will consist of processing aggregates [17]. Where data distribution across clusters is not needed, this is most often the simplest and easiest distribution to use.
- *Sharding* provides the ability to scale content horizontally while allowing the distribution of operations across shards to increase performance [18]. To accomplish this task, data is separated and housed on individual servers (nodes) ideally by how the information will be accessed by users. An individual server in this model is referred to as a shard. It is recommended to use a combination of replication along with sharding to preserve the reliability of the system. This ensures that if a node does fail, the information can still be retrieved from its duplicate. While there are many benefits to sharding, it is important be discerning when implementing a sharding distribution as complexity and challenges are introduced.
- A *master-slave* distribution duplicates data from one server (the master) to one or more different servers (slaves) [19]. This is most useful for reducing the load on reads to the

database as only the master is used for write operations. The slave nodes serve read requests evenly thus reducing the load on the master. Control is strictly one way, coming from the master and going to the slaves. This model increases reliability in the event of the master failing; slaves are still able to field requests. A potential setback with a master-slave model is the possibility of inconsistency among data. In certain cases, changes to data may not be circulated to all slaves. The end result is that slaves contain differences among the same data sets. Different users accessing different slaves may retrieve conflicting results.

- A *Peer-to-Peer (Multi-Master)* model is similar to a master-slave in that the data is replicated across multiple nodes. The difference is that each node has equal weight, effectively eliminating the master. Where in a master-slave model the risk of the master node failing is present, the loss of a node in a peer-to-peer model can easily be overcome as each replica accepts writes as well as reads. Nodes can readily be added to improve the performance of the database. The challenge with this model is the same as with a master-slave. It is possible for two users to attempt to update the same information at the same time, creating a write-write conflict [20].

As indicated previously, sharding and replication can and should be used together to optimize performance and dependability. This allows a dependable distribution of data while optimizing performance and maintaining a reliable database.

VII. Choosing a Database

Up to this point relational and aggregate modeled databases have been defined. Characteristics, advantages and disadvantages of each were reviewed to give a better understanding of when to use a given database. Various data models were presented as a way of learning how data is stored, understood, and interacted with. Distribution models aided in realizing how data is replicated and allocated. This information has been presented as an introduction to the various types of databases as a means of discerning when to use a given system. While the actual implementation of each database type has not been covered, this information gives a starting point and provides a high-level view in choosing a specific design.

In recent years, the explosion of data generation, the rise of Big Data in analytics, and advances in storage hardware have led to the growing demands on database systems. The vast amount of data needing to be stored and queried has presented challenges to the traditional relational database model that has served as a default model for many years. This has led to many companies considering a NoSQL type of database, one that is designed to cope with the demands of large data sets. These databases are generally designed to be run on clusters of machines (servers), effectively distributing vast amounts of data while providing improved performance demands. These models allow a greater amount of flexibility as they are schema-less and have little constraints on how the data is defined. In several instances data can be stored in a loose way while the database automatically defines types. They are not without complexity, however, and should only be considered when a specific intent is in mind and with a firm justification for doing so. There are two primary reasons to consider using a NoSQL database. One is to handle data access with sizes and performance that demand a cluster, the other is to improve the productivity of application development by using a more convenient data interaction style [21].

6.1 When to Use a Relational Database

The relational model has been a standard for many years due to its simplicity, ease of data retrieval, and flexibility while providing data integrity. Relational databases manage the issue of concurrency well through ACID transactions. This ensures that transactions are processed in a dependable way while maintaining data integrity. While the choice of database varies depending on the needs of a given system, a relational model provides several benefits. To define relationships, these models use foreign keys or indexes which enable a user to uniquely identify any given row in a table. In general, they are an ideal choice where an application will need to query multiple transactions, as well as routine complicated queries and data analysis [22].

Relational databases are a mature and well known model to use. They have been around for many years and the principles remain the same across systems. They provide the ability to serve a number of different applications in a reliable and tested way, making them a great choice for a database.

6.2 When to Use a NoSQL Database

- *Programmer Productivity*. As discussed, impedance mismatch occurs when a developer must translate collected data, often in the form of an aggregate model to a relational model in order to store and query it. Since the 2000s many frameworks such as Hibernate, iBATIS, and Rails Active Record have made this a much less cumbersome task, reducing–though not eliminating–the complexity of this issue [23]. NoSQL database naturally allows for collected data to be easily stored in the form it was collected, greatly reducing the load placed on developers to translate this information.

- *Data*-Access *Performance*. Perhaps the primary reason NoSQL databases have gained traction in recent years is that they provide greater performance to vast amounts of data. When the sheer amount of data moves beyond a certain point, relational database cease to cope well as queries become expensive. NoSQL models distribute the data, allowing almost instant access that scale well. Sharding and replication are also easier to perform in a NoSQL database.

Graph databases are often used with data that naturally has a large amount of connections or relationships. They naturally lend themselves to social networks, recommendation generation, and spatial data.

Key-value databases are sound for storing a user's online session information. Things like preferences, profile information, and shopping cart data that does not need to be queried by a developer and can be quickly captured and persisted at a later time.

Document databases adeptly manage web analytics, real-time analytics, ecommerce-applications, and blogging platforms. These models do not perform well in systems that require complex queries spanning across multiple clusters.

Column-family databases generally are used for content management systems, maintaining counters, and heavy write volume such as log aggregation [24]. They are not recommended with systems that require ACID transactions or where queries will need to perform aggregate functions due to the way in which the data is stored.

VIII. Application

I currently work for a book publishing company as a data analyst and interact with at least one of our databases on a daily basis. Our website does provide the option for customers to place orders. We have databases that serve reporting and analytics, an internal intranet, order entry, online activity, and warehousing. To optimize performance needs at my company I would recommend implementing the following types of database models, each one chosen for their designed advantages in solving a specific problem.

- For our website, a suitable option would be to implement a key-value model for shopping cart data as well as session information before an order is placed. This allows quick retrieval of information typically searchable by a user ID. Upon placing an order, it can be stored in a relational model for long term query usage.
- To recommend products for customers to purchase when viewing a given item, a graph model database is a viable option. This data is naturally based on relationships among customer purchase history as well as how the items relate to each other. A common example of this is the “customers who bought this also bought” recommendation on Amazon.
- A document style database will function well to serve our reporting and analytics requirements. This will provide faster performance than a relational model as data can be queried quickly to perform analysis as needed.
- Orders, our intranet, and the database that serves our warehouse are best implemented using a relational database. The benefits of ACID transactions will ensure data integrity and reliability to serve the employees using these systems.

IX. Conclusion

Differing databases are designed for different applications. Each one has a set of strengths and weaknesses tailored to address a specific function. Wood-working tools are designed to be used for shaping wood, but they do not perform their function well if they are used to repair a vehicle. So, too, must a given database be implemented in a situation for which it was designed. This is why it is critical for developers understand all types of database systems. To design an effective and efficient system, a user must not only understand the databases available, but also the specific needs of the application the database will be serving. In many cases, a conglomerate of database types is recommended to build an optimal system. While it is not necessary or ideal to use a single database, this is the default for many companies. Unfortunately, not all problems are solved well using a traditional relational database. The rise of NoSQL databases, however, will not eliminate the relational model but rather, complement its use.

This paper has given a high-level overview of the various NoSQL database options as well as the relational model and various database characteristics with the goal of providing the reader with a better understanding of which database to choose in a specific scenario. The only way to know for sure if a particular type of database will suit a user's needs is to build a sample model that can be fed a sample data set and tested accordingly. Hopefully, a starting point has been given to the reader that can be used to make an informed and effective decision.

REFERENCES

[1] P. J. Sadalage, M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Upper Saddle River: Pearson Education, Inc, 2013, pp. xiii.

[2] Pramod Sadalage, "NoSQL Databases: An Overview", https://www.thoughtworks.com/insights/blog/nosql-databases-overview. Accessed May 10, 2017.

[3] K. Delaney, F. Guerrero, Database Concurrency and Row Level Versioning in SQL Server 2005. https://technet.microsoft.com/en-us/library/cc917674.aspx. Accessed May 8, 2017.

[4] Sadalage, 4.

[5] Sadalage, 8.

[6] F. Chang et al, Bigtable: A Distributed Storage System for Structured Data, Google.Inc. November, 2006, pp. 1-14.

[7] "Amazon DynamoDB", https://aws.amazon.com/dynamodb/. Amazon Web Services, accessed May 8, 2017.

[8] "NoSQL", https://en.wikipedia.org/wiki/NoSQL#cite_note-1. Wikipedia, accessed May 9, 2017.

[9] S. Banerjee, A. Sarkar, Modeling NoSQL Databases: From Conceptual to Logical Level Design. *3rd Intl. Conf. on Applications and Innovations in Mobile Computing (AIMoC)*, February, 2016, pp. 1-8.

[10] "NoSQL Databases: An Overview", https://www.thoughtworks.com/insights/blog/nosql-databases-overview, accessed May 10, 2017.

[11] Sadalage, 27.

[12] R. Angles, C. Gutierrez, Survey of Graph Database Models. ACM Computing Surveys, Vol. 40, No. 1, Article 1, February 2008, pp. 1-39.

[13] Angles, 2.

[14] Pramod Sadalage, "NoSQL Databases: An Overview", https://www.thoughtworks.com/insights/blog/nosql-databases-overview, accessed May 10, 2017.

[15] "ACID Properties", https://msdn.microsoft.com/en-us/library/aa480356.aspx. Microsoft Developer Network, accessed May 10, 2017.

[16] Sadalage, 37.

[17] Sadalage, 38.

[18] "Basic Concepts", www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html. Elasticsearch Reference Guide, accessed May 11, 2017.

[19] "MySQL Master-Slave Replication on the Same Machine", https://www.toptal.com/mysql/mysql-master-slave-replication-tutorial, accessed May 11, 2017.

[20] Sadalage, 43.

[21] Sadalage, 12.

[22] “Relational vs. non-relational databases: Which one is right for you?”, https://www.pluralsight.com/blog/software-development/relational-non-relational-databases, Accessed May 12, 2017.

[23] Sadalage, 146.

[24] “NoSQL Databases: An Overview”, https://www.thoughtworks.com/insights/blog/nosql-databases-overview, accessed May 10, 2017.