西南科技大学

Southwest University of Science and Technology

Subject: Academic Report

Topic: Advanced Database Technology

Southwest University of Science and Technology

Department of Computer Science and Technology

Student Name: **MD AYNUL ISLAM**

Chinese Name: 叶子

Student ID: **4420190030**

Batch: **Undergraduate 2019**

Contents

1 Abstract

This unit examines advanced database technology. This unit particularly covers three main types of advanced database technologies, including document store, wide column store, and graph database. These three systems represent the broad spectrum of NoSQL databases. The underlying theoretical foundations, such as database modelling, transactions, and graph query processing will also be explored.

In computing, a database is an organized collection of data stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques. Computer scientists may classify database-management systems according to the database models that they support. Relational databases became dominant in the 1980s. These model data as rows and columns in a series of tables, and the vast majority use SQL for writing and querying data. In the 2000s, non-relational databases became popular, referred to as NoSQL because they use different query languages.

2 Introduction

Database technologies take information and store, organize, and process it in a way that enables users to easily and intuitively go back and find details they are searching for. Database technologies come in all shapes and sizes, from complex to simple, large to small. It’s important to think about how the database technology you choose will be able to grow as the size of your data grows, as well as how it will interact with any software you use to query your data.

The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used loosely to refer to any of the DBMS, the database system or an application associated with the database. Formally, a "database" refers to a set of related data and the way it is organized. Access to this data is usually provided by a "database management system" (DBMS) consisting of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data). The DBMS provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.[1]

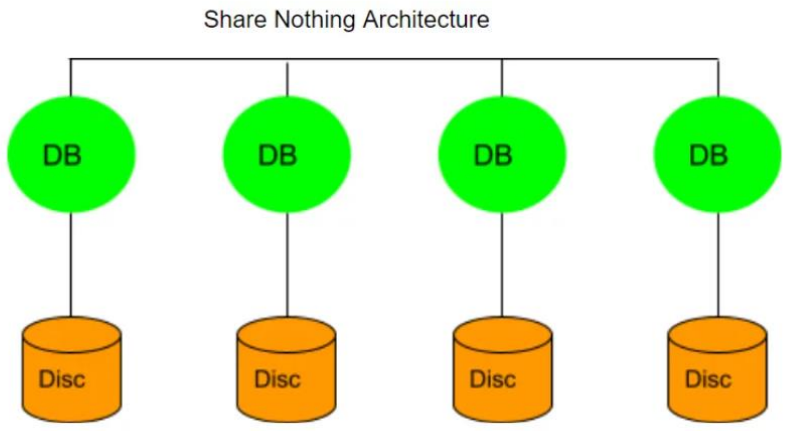
3 Features of NoSQL

3.1 Multi-Model

Unlike relational databases, where data is stored in relations, different data models in NoSQL databases make them flexible to manage data. Each data model is designed for specific requirements. Examples of data models include document, graph, wide-column, and key-value. The concept is to allow multiple data models in a single database. By doing so, the need for deploying and managing different databases for the same data cancels out.

3.2 Distributed

NoSQL databases use the shared-nothing architecture, implying that the database has no single control unit or storage. The advantage of using a distributed database is that data is continuously available because data remains distributed between multiple copies. On the contrary, Relational Databases use a centralized application that depends on the location.



3.3 Flexible Schema

Unlike relational databases where data is organized in a fixed schema, NoSQL databases are quite flexible while managing data. While relational databases were built typically to manage structured data, NoSQL databases can process structured, semi-structured or unstructured data with the same ease, thereby increasing performance.

3.4 Eliminated Downtime

One of the essential features is the eliminated downtime. Since the data is maintained at various nodes owing to its architecture, the failure of one node will not affect the entire system.

3.5 High Scalability

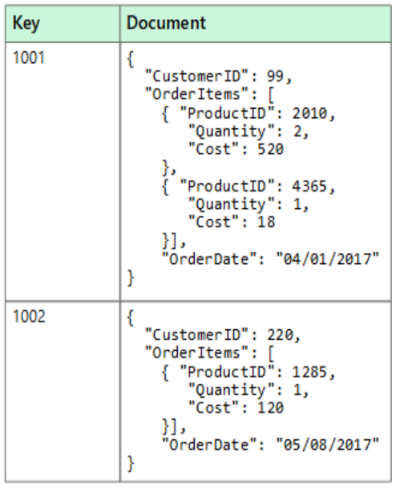
One of the reasons for preferring NoSQL databases over relational databases is their high scalability. Since the data is clustered onto a single node in a relational database, scaling up poses a considerable problem. On the other hand, NoSQL databases use horizontal scaling, and thus the data remains accessible even when one or more nodes go down.[2]

4 Types of non-relational database and NoSQL

A non-relational database is a database that does not use the tabular schema of rows and columns found in most traditional database systems. Instead, non-relational databases use a storage model that is optimized for the specific requirements of the type of data being stored. For example, data may be stored as simple key/value pairs, as JSON documents, or as a graph consisting of edges and vertices. Some of the more popular NoSQL databases are MongoDB, Apache Cassandra, Redis, Couchbase and Apache HBase.

4.1 Document data stores

A document data store manages a set of named string fields and object data values in an entity that's referred to as a document. These data stores typically store data in the form of JSON documents. Each field value could be a scalar item, such as a number, or a compound element, such as a list or a parent-child collection. The data in the fields of a document can be encoded in various ways, including XML, YAML, JSON, BSON, or even stored as plain text. The fields within documents are exposed to the storage management system, enabling an application to query and filter data by using the values in these fields.



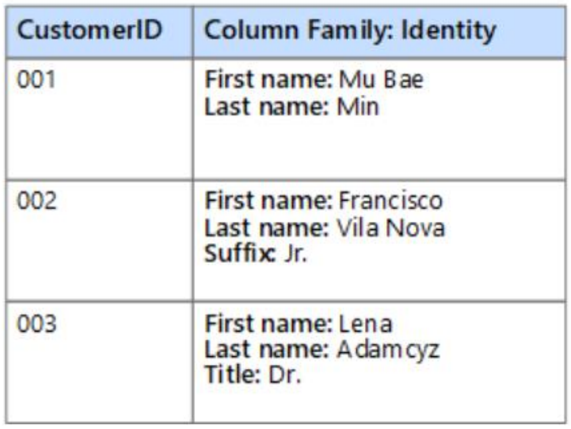
Key	Document
1001	`{ "CustomerID": 99, "OrderItems": [ { "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }`
1002	`{ "CustomerID": 220, "OrderItems": [ { "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }`

Many document databases support in-place updates, enabling an application to modify the values of specific fields in a document without rewriting the entire document. Read and write operations over multiple fields in a single document are typically atomic.

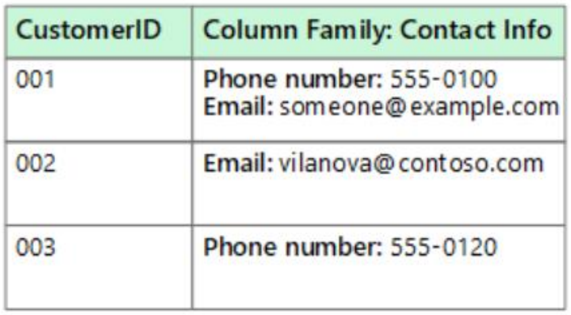
4.2 Columnar data stores

A columnar or column-family data store organizes data into columns and rows. In its simplest form, a column-family data store can appear very similar to a relational database, at least conceptually. The real power of a column-family database lies in its denormalized approach to structuring sparse data, which stems from the column-oriented approach to storing data.

The following diagram shows an example with two column families, Identity and Contact Info. The data for a single entity has the same row key in each column family. This structure, where the rows for any given object in a column family can vary dynamically, is an important benefit of the column-family approach, making this form of data store highly suited for storing data with varying schemas.



CustomerID	Column Family: Identity
001	**First name:** Mu Bae **Last name:** Min
002	**First name:** Francisco **Last name:** Vila Nova **Suffix:** Jr.
003	**First name:** Lena **Last name:** Adamcyz **Title:** Dr.



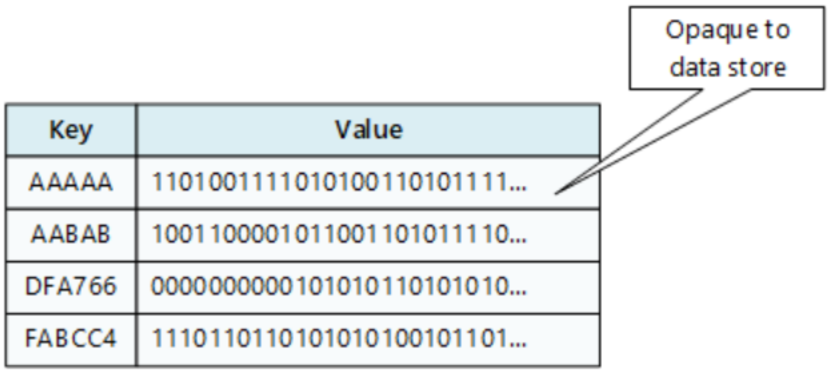
CustomerID	Column Family: Contact Info
001	**Phone number:** 555-0100 **Email:** someone@example.com
002	**Email:** vilanova@contoso.com
003	**Phone number:** 555-0120

On disk, all of the columns within a column family are stored together in the same file, with a specific number of rows in each file. With large data sets, this approach creates a performance benefit by reducing the amount of data that needs to be read from disk when only a few columns are queried together at a time.

4.3 Key/value data stores

A key/value store is essentially a large hash table. You associate each data value with a unique key, and the key/value store uses this key to store the data by using an appropriate hashing function. The hashing function is selected to provide an even distribution of hashed keys across the data storage.

An application can store arbitrary data as a set of values, although some key/value stores impose limits on the maximum size of values. The stored values are opaque to the storage system software. Any schema information must be provided and interpreted by the application. Essentially, values are blobs and the key/value store simply retrieves or stores the value by key.



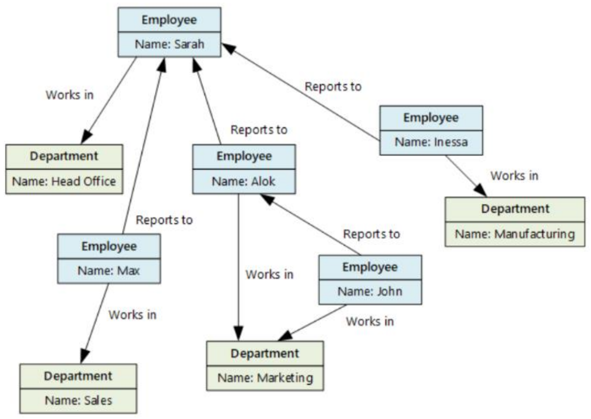
Key	Value
AAAAA	1101001111010100110101111...
AABAB	1001100001011001101011110...
DFA766	0000000000101010110101010...
FABCC4	1110110110101010100101101...

A single key/value store can be extremely scalable, as the data store can easily distribute data across multiple nodes on separate machines.

4.4 Graph data stores

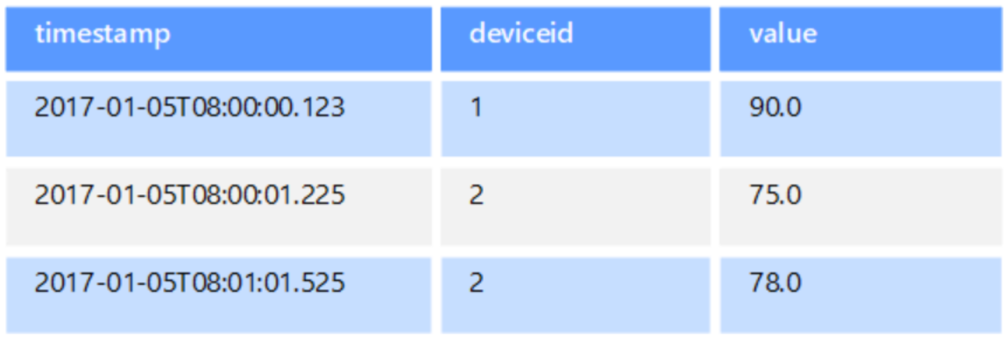
A graph data store manages two types of information, nodes and edges. Nodes represent entities, and edges specify the relationships between these entities. Both nodes and edges can have properties that provide information about that node or edge, similar to columns in a table. Edges can also have a direction indicating the nature of the relationship.

This structure makes it straightforward to perform queries such as "Find all employees who report directly or indirectly to Sarah" or "Who works in the same department as John?" For large graphs with lots of entities and relationships, you can perform complex analyses quickly. Many graph databases provide a query language that you can use to traverse a network of relationships efficiently.



4.5 Time series data stores

Time series data is a set of values organized by time, and a time series data store is optimized for this type of data. Time series data stores must support a very high number of writes, as they typically collect large amounts of data in real time from a large number of sources. Time series data stores are optimized for storing telemetry data. Scenarios include IoT sensors or application/system counters. Updates are rare, and deletes are often done as bulk operations.

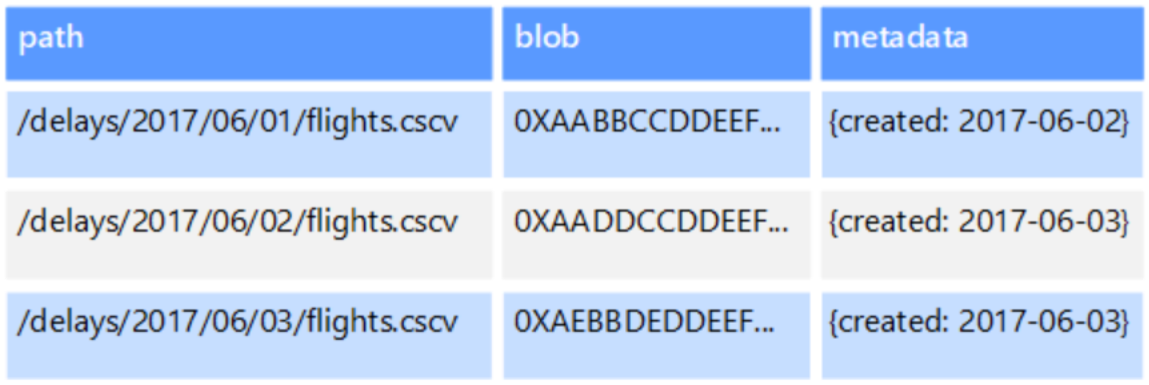


timestamp	deviceid	value
2017-01-05T08:00:00.123	1	90.0
2017-01-05T08:00:01.225	2	75.0
2017-01-05T08:01:01.525	2	78.0

Although the records written to a time series database are generally small, there are often a large number of records, and total data size can grow rapidly. Time series data stores also handle out-of-order and late-arriving data, automatic indexing of data points, and optimizations for queries described in terms of windows of time. This last feature enables queries to run across millions of data points and multiple data streams quickly, in order to support time series visualizations, which is a common way that time series data is consumed.

4.6 Object data stores

Object data stores are optimized for storing and retrieving large binary objects or blobs such as images, text files, video and audio streams, large application data objects and documents, and virtual machine disk images. An object consists of the stored data, some metadata, and a unique ID for accessing the object. Object stores are designed to support files that are individually very large, as well provide large amounts of total storage to manage all files.

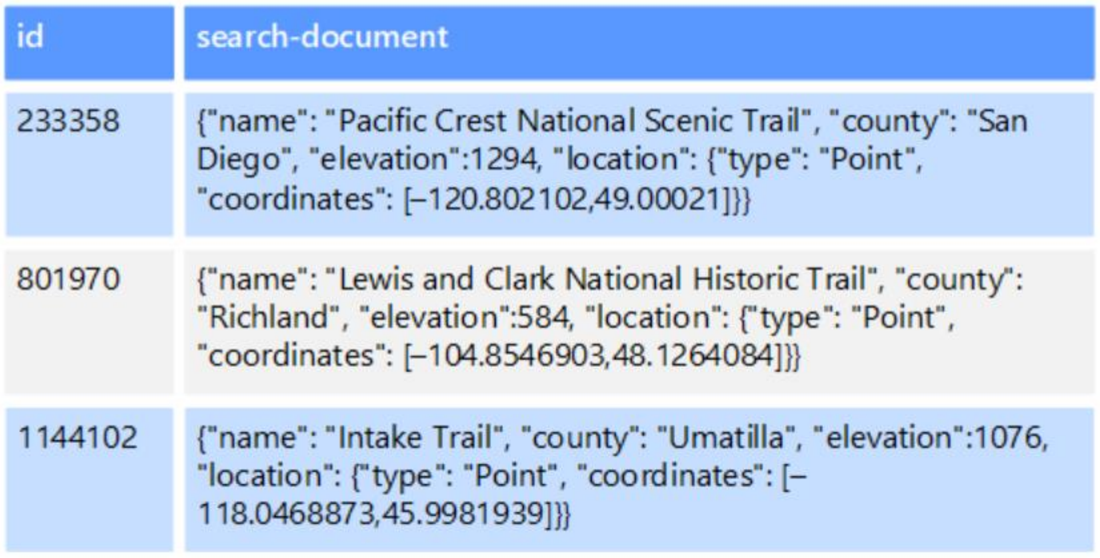


path	blob	metadata
/delays/2017/06/01/flights.cscv	0XAABBCCDDEEF...	{created: 2017-06-02}
/delays/2017/06/02/flights.cscv	0XAADDCCDDEEF...	{created: 2017-06-03}
/delays/2017/06/03/flights.cscv	0XAEBBDEDDEEF...	{created: 2017-06-03}

One special case of object data stores is the network file share. Using file shares enables files to be accessed across a network using standard networking protocols like server message block (SMB). Given appropriate security and concurrent access control mechanisms, sharing data in this way can enable distributed services to provide highly scalable data access for basic, low-level operations such as simple read and write requests.

4.7 External index data stores

External index data stores provide the ability to search for information held in other data stores and services. An external index acts as a secondary index for any data store, and can be used to index massive volumes of data and provide near real-time access to these indexes.

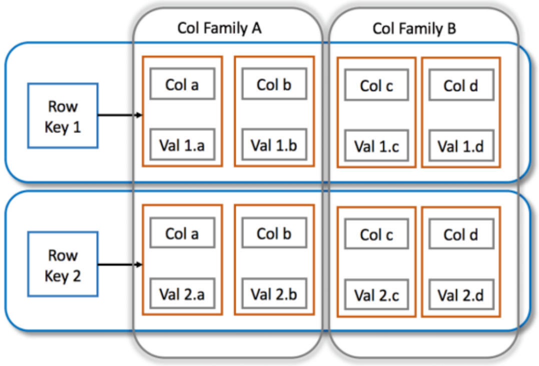


id	search-document
233358	{"name": "Pacific Crest National Scenic Trail", "county": "San Diego", "elevation":1294, "location": {"type": "Point", "coordinates": [–120.802102,49.00021]}}
801970	{"name": "Lewis and Clark National Historic Trail", "county": "Richland", "elevation":584, "location": {"type": "Point", "coordinates": [–104.8546903,48.1264084]}}
1144102	{"name": "Intake Trail", "county": "Umatilla", "elevation":1076, "location": {"type": "Point", "coordinates": [–118.0468873,45.9981939]}}

External index data stores are often used to support full text and web-based search. In these cases, searching can be exact or fuzzy. A fuzzy search finds documents that match a set of terms and calculates how closely they match. Some external indexes also support linguistic analysis that can return matches based on synonyms, genre expansions (for example, matching "dogs" to "pets"), and stemming (for example, searching for "run" also matches "ran" and "running").[3]

5 Design and model document-store and wide column-store databases

A wide-column store (or extensible record stores) is a type of NoSQL database. It uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table. A wide-column store can be interpreted as a two-dimensional key–value store.

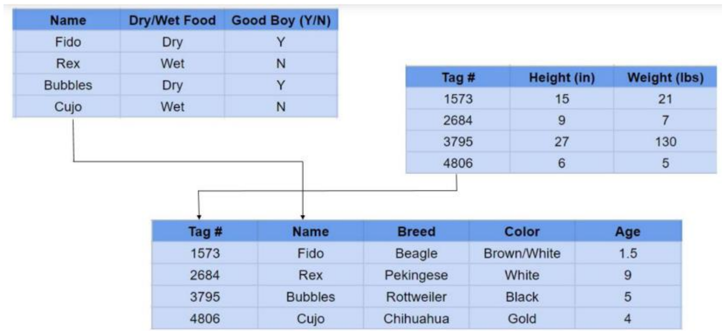


Data is stored in column cells, which gets grouped into column families. A group of families is linked to a row key. Each column family typically contains columns that are used together, so the columns are stored as a contiguous block on disk, enhancing performance. You could see that a wide-column store is like a mix between a relational database and a document store. It still uses rows and columns like that of a relational database. And the column formats can be different for each row in the same table. This combines the regimented tabular structure of the relational model and the flexible data schema of the document model.[4]

6 Compare and contrast between relational and non-relational database modelling

6.1 Relational Databases

A relational database is one that stores data in tables. The relationship between each data point is clear and searching through those relationships is relatively easy. The relationship between tables and field types is called a schema.



Name	Dry/Wet Food	Good Boy (Y/N)
Fido	Dry	Y
Rex	Wet	N
Bubbles	Dry	Y
Cujo	Wet	N

Tag #	Height (in)	Weight (lbs)
1573	15	21
2684	9	7
3795	27	130
4806	6	5

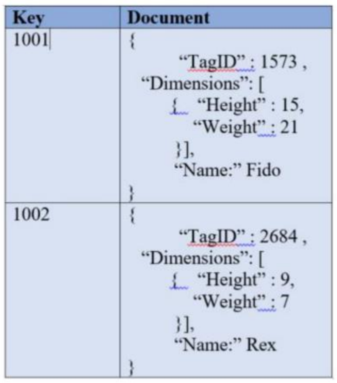
Tag #	Name	Breed	Color	Age
1573	Fido	Beagle	Brown/White	1.5
2684	Rex	Pekingese	White	9
3795	Bubbles	Rottweiler	Black	5
4806	Cujo	Chihuahua	Gold	4

Here we see three tables all providing unique information on a specific dog. A relational database user can then obtain a view of the database to fit their needs. For example, I might want to view or report on all dogs over 100 pounds. Or you may wish to see which breeds eat dry food. Relational databases make answering questions like these relatively easy.

Relational databases are also called SQL databases. SQL stands for Structured Query Language and it’s the language relational databases are written in. SQL is used to execute queries, retrieve data, and edit data by updating, deleting, or creating new records.

6.2 Non-Relational Databases

A non-relational database is any database that does not use the tabular schema of rows and columns like in relational databases. Rather, its storage model is optimized for the type of data it’s storing.



Key	Document

1001	{
	“TagID” : 1573 ,
	“Dimensions”: [
	{ “Height” : 15,
	“Weight” : 21
	}],
	“Name:” Fido
	}

1002	{
	“TagID” : 2684 ,
	“Dimensions”: [
	{ “Height” : 9,
	“Weight” : 7
	}],
	“Name:” Rex
	}

Non-relational databases are also known as NoSQL databases which stands for “Not Only SQL.” Where relational databases only use SQL, non-relational databases can use other types of query language. Document-oriented databases – Also known as a document store, this database is designed for storing, retrieving and managing document-oriented information. Document databases usually pair each key with a complex data structure (called a document). Key-Value Stores – This is a database that uses different keys where each key is associated with only one value in a collection. Think of it as a dictionary. This is one of the simplest database types among NoSQL databases. Wide-Column Stores – this database uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table. Graph Stores – A graph database uses graph structures for semantic queries with nodes, edges, and properties to represent and store data.[5]

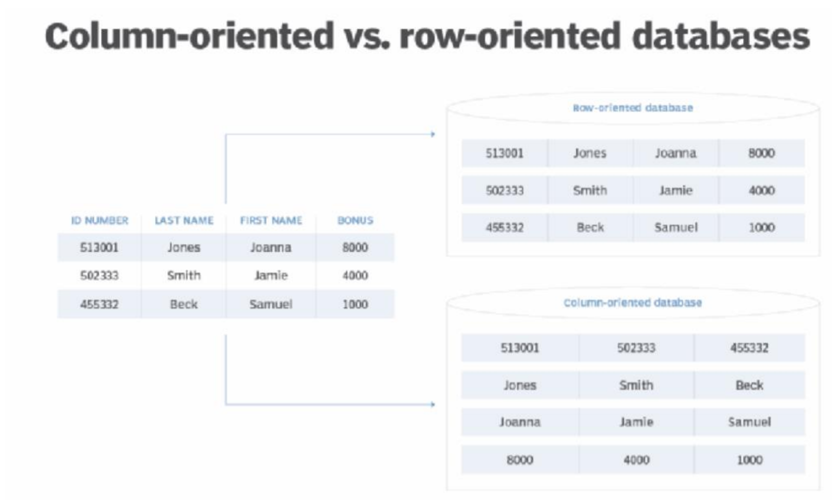
7 Explain the concepts of transactions in non-relational systems

NoSQL databases have come along, in many cases providing a more natural fit from a modeling perspective. In particular, document-oriented databases (e.g. MarkLogic, MongoDB, CouchDB, etc.), with their rich JSON and/or XML persistence models have effectively eliminated this impedance mismatch. And while this has been a boon to developers and productivity, in some cases developers have come to believe that they’d need to sacrifice other features to which they have become accustomed, such as ACID transaction support. The reason is that many NoSQL databases do not provide such capabilities, citing a trade-off to allow for greater agility and scalability not available in traditional relational databases. For many, the rationale for such a tradeoff is rooted in what is known as the CAP theorem.

As a result, transactions have been making their way back into the discussion of NoSQL. This is good news if you’re a Java developer looking for the agility and scale of NoSQL but still want the enterprise expectations of ACID transactions. In this article we will explore one NoSQL database in particular, MarkLogic, and how it provides multi-statement transaction capabilities to the Java developer without sacrificing the other benefits we have come to expect from NoSQL, such as agility and scale-out capability across commodity hardware. Before we do however, let's first take a step back and re-familiarize ourselves with the concepts of ACID.[6]

8 Implement document-store and wide column-store systems

Columnar databases fit this description. These are NoSQL databases built for highly analytical, complex-query tasks. Unlike relational databases, columnar databases store their data by columns, rather than by rows. These columns are gathered to form subgroups. The keys and the column names of this type of database are not fixed. Columns within the same column family, or cluster of columns, can have a different number of rows and can accommodate different types of data and names.



ID NUMBER	LAST NAME	FIRST NAME	BONUS
513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000

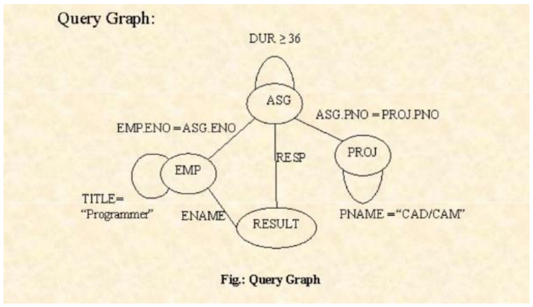
513001	Jones	Joanna	8000
502333	Smith	Jamie	4000
455332	Beck	Samuel	1000

513001	502333	455332
Jones	Smith	Beck
Joanna	Jamie	Samuel
8000	4000	1000

Relational databases have a set schema and they function as tables of rows and columns. Wide-column databases have a similar, but different schema. They also have rows and columns. However, they are not fixed within a table, but have a dynamic schema. Each column is stored separately. If there are similar (related) columns, they are joined into column families and then the column families are stored separately from other column families. The row key is the first column in each column family, and it serves as an identifier of a row. Furthermore, each column after that has a column key (name). It identifies columns within rows and thus enables the querying of the columns. The value and the timestamp come after the column key, leaving a trace of when the data was entered or modified.

9 Demonstrate graph query processing

The first level is a cleverly constructed graph query. Graph queries, for the most part, attempt to identify an explicit pattern within the graph database. Graph queries have an expressive power to return something at the level of an analytic in a normal data processing system. And to be fair, many analytics that you find in the normal world are really just good SQL queries, so this makes sense. What graph queries usually do is find a known subset of the important nodes in the graph haystack.



The next level up in sophistication is the graph algorithm. In this case, the query is perhaps using a function of some sort that runs an algorithm over not just simply selecting and grouping nodes, but also categorizing them or using some other processing technique to sort them out and learn something from them. With a graph query, you're working with a discrete pattern, but with a graph algorithm, the traversal is tightly less bound, but will still begin from a declared subsection or local graph. A query may find you the needles in the haystack you knew you were looking for; an algorithm may tell you which needles are most interesting.[7]

10 Conclusion

Advanced Database Management Systems target a new breed of databases known as NoSQL/newSQL. Advanced database management systems also support new trends in data management fueled by application needs, such as support for advanced analytics, stream processing systems and main memory data processing. In crux, we can say that there are four types of NoSQL Databases: Key-Value (KV) Stores, Document Stores, Column Family Data stores, and Graph Databases. NoSQL data stores provide an alternative to the traditional RDBMS, and you might be not be sure of the NoSQL databases you want to select. The ideal way of identifying the best suitable NoSQL database for your application is to figure out the requirement that is not met by RDBMS.

11 Reference

[1] L. November and T. Databases, “Advanced Databases,” no. November 2014, pp. 1–72, 2015.

[2] “4 Types of NoSQL Databases - Thomas Henson.” https://www.thomashenson.com/4-types-nosql-databases/ (accessed Dec. 31, 2021).

[3] A. Gubichev, “Query Processing and Optimization in Graph Databases,” pp. 1–178, 2015.

[4] “Transactional NoSQL Database.” https://www.infoq.com/articles/MarkLogic-NoSQL-with-Transactions/ (accessed Dec. 21, 2021).

[5] “NoSQL database types explained: Column-oriented databases.” https://searchdatamanagement.techtarget.com/tip/NoSQL-database-types-explained-Column-oriented-databases (accessed Dec. 21, 2021).

[6] Microsoft, “Non-relational data and NoSQL - Azure Architecture Center | Microsoft Docs,” *Microsoft In-Page*, 2020. https://docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data (accessed Dec. 21, 2021).

[7] “What’s the Difference? Relational vs Non-Relational Databases | Logi Analytics.” https://www.logianalytics.com/relational-vs-non-relational-databases/ (accessed Dec. 21, 2021).